Planning on the Grid: A Status Report
(DRAFT PPDG-20)

Francesco Giacomini, Francesco Prelz, Massimo Sgaravatto,
Igor Terekhov, Gabriele Garzoglio,
and Todd Tannenbaum

francesco.giacomini@mi.infn.it
francesco.prelz@mi.infn.it
massimo.sgaravatto@pd.infn.it
terekhov@fnal.gov
garzoglio@fnal.gov
tannenba@cs.wisc.edu

October 18, 2002

Abstract

Traditional batch job schedulers are a poor fit for a grid computing environment. In order to respond to the grid scheduling challenges, the client must perform planning. In this paper, we expound upon the differences between planning versus scheduling, and present two grid projects which utilize a ClassAd matchmaking framework to enable planning on the grid. Planning for purposes of job placement within the EU DataGrid and DZero SAMGrid projects will be explored.

1 Introduction

A central purpose of grid computing is to enable a community of users to perform tasks upon a pool of shared resources. Because there is rarely a one-to-one correspondence between the number of tasks and the number of resources, a *scheduling model* is required. A scheduling model takes as input a description of both the tasks and the resources, and as output makes decisions that conform to a given *scheduling policy*. These scheduling decisions answer questions such as which resources should be allocated to which tasks and in what order should the tasks be launched. In the context of job resource management systems, the term *scheduling* typically refers to the identification of key metrics in the system and strategies to subsequently optimize them. Much of this work was motivated by the proliferation in the early 1990s of massively parallel processor (MPP) machines, and the desire to use these very expensive resources as efficiently as possible.

1.1 Scheduler Challenges in a Grid Environment

Scheduling in the traditional sense, however, must transform itself when the resources are no longer a collection of CPUs in an MPP machine, or a collection of nodes in a tightly coupled Beowulf cluster [18]. Scheduling in a grid computing environment must contend with challenges that result when the pool of resources spans across multiple administrative domains, is very large, and is widely dispersed. Some of these challenges include:

- *Distributed Ownership* - A traditional scheduler often assumes that it has total control over the resources at all times. But in a grid environment this is not true. Each administrative domain in a virtual organization will desire to enforce its own unique scheduling policy to enforce, and typically require its own scheduling system to control its own resources.

- *Heterogeneous resources* - As grids grow, the variety of resources to manage increases. Compute nodes with varying operating systems, RAM, disk, CPU, and network connectivity is only the beginning. Grids will increasingly require the incorporation and co-scheduling of different types of resources, such as compute nodes alongside electron microscopes alongside software licenses.

- *Dynamic resource pool* - Resources will not be fixed; both the number and type of resources will be in continuous flux as the physical organizations that make up the virtual organization act independently to maintain, upgrade, and retire equipment.

- *Absence of a fixed schema* - Heterogeneous resources have completely disparate attributes. There will be minimal overlap in the schema describing a scanning microscope versus a Linux workstation. Even amongst homogenous resources, different administrative domains may collect different pieces of information. The constant evolution of resources translates into an ever changing schema.

- *External influences* - Significant impact upon the behavior of the resources may result from events external to the scheduler's activity. For instance, assuming the communication behavior of the job is available, a scheduler in a closed environment can predict the impact of job placement upon the connecting network. But this would not be true in a wide-area grid environment where the connecting network involves the Internet. No matter what the scheduler may know about the job, it would be unable to deduce total available Internet network bandwidth in the same fashion.

- *Stale information* - To make decisions in a dynamic environment, schedulers must be continuously updated about task and resource characteristics. With large wide-area grids, latency and bandwidth constraints will require caching this data. Scheduling decisions based upon stale information must be at least detected, and ideally, corrected.

1.2 Grids: Planning instead of Scheduling

These grid scheduling challenges, and distributed ownership in particular, require a shift in methodology. We have found an approach based upon *planning* on the part of the client or a broker to be more applicable than customary scheduling in such environments. During planning, a logical request is mapped onto physical entities by matching *resource offers* with *resource requests* in a manner that is mindful of any constraints or preferences. Such planning is commonplace in the real world. Reflect on how a consumer, freezing in December because he lives in the middle of Wisconsin, would go about traveling by airline to a lovely and warm location in remote Australia. A phone call would be placed to a travel agent. The travel agent would first ask for the destination, followed by such requirements as arrival date, and finally preferences such as window or aisle seat. Next the agent would consult a timetable of thousands of discrete flights and find an itinerary which is acceptable to both the consumer (acceptable arrival date, fewest number of transfers) and the airline (client agrees to pay the fee, space is still available).

It is important to note that in this process, neither the consumer nor the travel agent has the ability to influence the schedule of the airplanes. That decision is up to the airplane owners – the airlines. But despite the failing of the airlines to schedule a flight from central Wisconsin to central Australia, or even identify the demand for such a flight, the goal can be achieved by planning. Planning on the client-side can produce effective utilization of resources which have been independently scheduled on the server side.

1.3 Grid Planning in Practice

The remainder of this paper will describe two grid projects that utilize planning for job placement. Specifically, the EU DataGrid will be presented in section 3, and the DZero SAMGrid will be discussed in section 4. Both projects utilize the ClassAd Matchmaking Framework, introduced in section 2.4, as the vehicle to perform planning.

Upon incorporating a planning framework into a grid job placement system, the question of *when* does the planning take place is a very important consideration. If a job is mapped to a physical resource upon job submission, we call this *eager planning*. If the mapping process is postponed until some period after the job has been submitted, it is said to be *lazy planning*. Finally, *very lazy planning* occurs if the mapping is postponed right up to the moment when the resource is immediately available.

2 Relevant Technologies

This section will briefly introduce some common underlying technology utilized by both the EU DataGrid and DZero SAMGrid projects: the Globus Toolkit developed by the Globus project, followed by the Condor-G, DAGMan, and Matchmaking services developed by the Condor project.

2.1 Globus Toolkit

The Globus Project [11] provides software tools that make it easier to build computational grids and grid-based applications. These tools are collectively called the Globus Toolkit. The Globus Tookit contains open source implementations of protocols designed to offer features such as uniform access to distributed resources with diverse scheduling mechanisms; information service for resource publication, discovery, and selection; API and command-line tools for remote file management, staging of executables and data; and enhanced performance through multiple communication protocols.

2.2 Condor-G

The Globus Project designed the Grid Resource Access and Management (GRAM) protocol [1] to provide a uniform interface for batch execution. GRAM provides an abstraction for remote process queuing and execution with several powerful features such as strong security and file transfer. The Globus Project provides a server called the jobmanager that speaks GRAM and converts its commands into a form understood by a variety of batch systems.

To take advantage of GRAM, a user still needs a system that can remember what jobs have been submitted, where they are, and what they are doing. If jobs should fail, the system must analyze the failure and re-submit the job if necessary. To track large numbers of jobs, users need queueing, prioritization, logging, and accounting. To provide this service, the Condor project adapted a standard Condor agent to speak GRAM, yielding a system called Condor-G [12]. This required some small changes to GRAM such as adding durability and two-phase commit to prevent the loss or repetition of jobs [13].

Essentially Condor-G represents the marriage of technologies from the Globus and Condor projects. From Globus comes the use of protocols for secure inter-domain communications and standardized access to a variety of remote batch systems. From Condor comes the user concerns of job submission, job allocation, error recovery, and creation of a friendly execution environment.

In its simplest form, Condor-G is an example of eager mapping. Jobs are bound to a compute site at the time of submission. Condor-G can also employ an optional technique called *glide-in*, which allows Condor-G to perform very lazy planning. This technique is explained in [12].

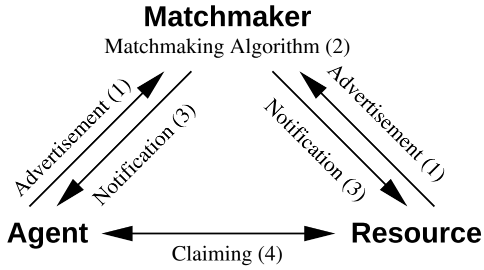

Figure 1: Matchmaking.

2.3 DAGMan

The Directed Acyclic Graph Manager (DAGMan) is a service for executing multiple jobs with dependencies in a declarative form [20, 19]. DAGMan allows the management of a group of jobs, which can be represented via a Directed Acyclic Graph (DAG), where the nodes represent the jobs and the arcs represent the dependencies between them. "Job X depends on job Y" means that X can start only when Y has completed.

DAGMan might be thought of as a distributed, fault-tolerant version of the traditional `make`. Like its ancestor, it accepts a declaration that lists the work to be done and the constraints on its order. Unlike `make`, it does not depend on the file system to record a DAG's progress. Indications of completion may be scattered across a distributed system, so DAGMan keeps private logs, allowing it to resume a DAG where it left off, even in the face of crashes and other failures [23].

2.4 Matchmaking with ClassAds

Matchmaking with ClassAds can be used to bridge the gap between planning and scheduling. Matchmaking creates opportunities for planners and schedulers to work together while still respecting their essential independence.

Matchmaking is typically performed in four steps [15], shown in Figure 1. In the first step, agents and resources advertise their characteristics and requirements in *classified advertisements* (ClassAds), named after brief advertisements for goods and services found in the morning newspaper. In the second step, a *matchmaker* scans the known ClassAds and creates pairs that satisfy each other's constraints and preferences. In the third step, the matchmaker informs both parties of the match. The responsibility of the matchmaker then ceases with respect to the match. In the final step, *claiming*, the matched agent and resource establish contact, possibly negotiate further terms, and then cooperate to execute a job.

Job ClassAd	Machine ClassAd
MyType = “Job”	**MyType** = “Machine”
TargetType = “Machine”	**TargetType** = “Job”
Requirements = ((other.Arch==“INTEL” && other.OpSys==“LINUX”) && other.Disk > my.DiskUsage)	**Machine** = “nostos.cs.wisc.edu”
Rank = (Memory * 10000) + KFlops	**Requirements** = (LoadAvg <= 0.300000) && (KeyboardIdle > (15 * 60))
Cmd = “/home/tannenba/bin/sim-exe”	**Rank** = other.Department==self.Department
Department = “CompSci”	**Arch** = “INTEL”
Owner = “tannenba”	**OpSys** = “LINUX”
DiskUsage = 6000	**Disk** = 3076076
	Department = ”CompSci”
	Memory = 512

Figure 2: Example of two simple ClassAds.

A ClassAd is a set of uniquely named expressions, using a semi-structured data model so no specific schema is required by the matchmaker. 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. After gaining more experience, we created a second ClassAd implementation which introduced richer attribute value types and related operators for records, sets, and tertiary conditional operators similar to C.

Because ClassAds are schema-free, participants in the system may attempt to refer to attributes that do not exist. For example, an job may prefer machines with the attribute `(Owner == ''Fred'')`, yet some machines may fail to define the attribute `Owner`. To solve this, ClassAds use *three-valued logic* which allows expressions to evaluated to either `true`, `false`, or `undefined`. This explicit support for missing information allows users to build robust requirements even without a fixed schema.

The Condor matchmaker assigns significance to two special attributes: `Requirements` and `Rank`. `Requirements` indicates a constraint and `Rank` measures the desirability of a match. 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 ClassAds matching a given customer ClassAd, the matchmaker chooses the one with the highest Rank value (noninteger values are treated as zero), breaking ties according to the provider's Rank value.

ClassAds for a job and a machine are shown in Figure 2. The `Requirements` state that the job must be matched with an Intel Linux machine which has enough free disk space (more than 6 megabytes). Out of any machines which meet these requirements, the job prefers a machine with lots of memory, followed by good floating point performance. Meanwhile, the machine ad `Requirements` states that this machine is not willing to match with any job unless its load average is low and the keyboard has been idle for more than 15 minutes. In other words, it is only willing to run jobs when it would otherwise sit idle. When it is willing to run a job, the `Rank` expression states it prefers to run jobs submitted by users from its own department.

3 Planning on the EU DataGrid

3.1 DataGrid Introduction

The European DataGrid project [10] is a three year project (started on January 2001) funded by the European Union with the aim of setting up, by devising and developing scalable software solutions and testbeds, a computation and data-intensive grid of resources, to support globally distributed scientific processing involving multi-PetaByte datasets, tens of thousands of resources, and thousands of simultaneous users. While the project focuses on scientific applications such as High Energy Physics, Earth Sciences and Bio-Informatics, many issues addressed by the project are common to many applications and thus the project has a potential impact on future industrial and commercial activities.

3.2 DataGrid WP1 Architecture

The workload management area (Work Package 1, WP1) [4] of the EU DataGrid project is mandated to define and implement a suitable architecture for distributed scheduling and resource management in the Grid environment.

In the first phase of the project, the main goal was to rapidly implement a working prototype with the fundamental functionalities, providing users with an environment allowing to define and submit jobs to the Grid, and able to find and use the “best” resources for these jobs. This first workload management system, implemented in the first year of the project and deployed in the DataGrid testbed, is described in [2] and [8].

The experience acquired, the feedback received by the users and the additional functionality required to meet the project goals, triggered an update of this architecture, in order to increase the reliability of the system, to simplify the flow of control, to support for new functionalities, to allow the use of WP1 modules (e.g. the Resource Bro-

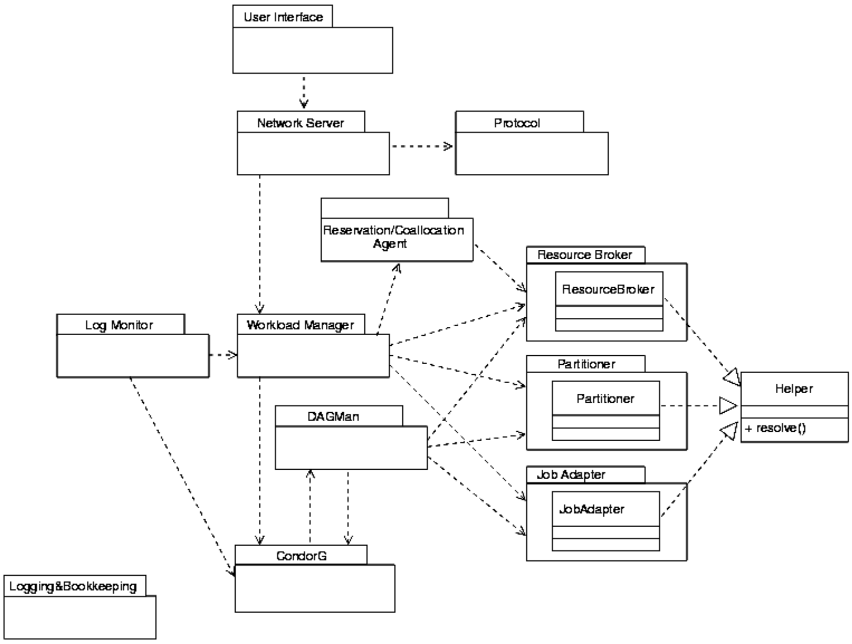

Figure 3: UML diagram describing the new DataGrid WP1 Workload Management System architecture.

ker) also outside the WP1 Workload Management System (WMS).

The updated Workload Management System architecture [3] is illustrated in Figure 3.

The *User Interface (UI)* is the component that allows users to access the functionality offered by the Workload Management System, in particular to submit, control and monitor jobs on the Grid. Users are provided with a specific, ClassAd-based Job Description Language, JDL ([5], [6]) to specify the characteristics of their jobs and the requirements and preferences to be matched against available Grid resources.

The *Network Server* is a generic network daemon, responsible for accepting incoming requests from the UI (e.g. job submission, job removal), which, if valid, are then passed to the Workload Manager. For this purpose the Network Server uses *Protocol*, to check if the incoming requests conform to the agreed protocol.

The *Workload Manager*, given a valid request, takes the appropriate actions to satisfy it. To do so, it may need support from other components, which are specific to the different request types. All these components that offer support to the Workload Manager provide a class whose interface is inherited from a *Helper* class. Essentially the Helper, given a JDL expression, returns a modified one, which represents the output of the required action. For example, if the request was to find a suitable resource for a job, the input JDL expression will be the one specified by the user, and the output will be the JDL expression augmented with the computing resource choice.

The *Resource Broker* is one of these classes offering support to the Workload Manager. It provides a matchmaking service: given a JDL expression (e.g. for a job submission, or for a resource reservation request), it finds the resource(s) that best match the request.

The *Job Adapter* is responsible for arranging the final cosmetics to the JDL expression for a job, before it is passed to Condor-G for the actual submission.

Condor-G is the module responsible for performing the actual job management operations, issued on request of the Workload Manager.

The *Log Monitor* task is to intercept interesting events about active jobs out of Condor-G and to trigger appropriate actions.

The *Reservation Agent* and the *Co-Allocation Agent*, represented in the same block to simplify the figure, are the core components of the resource reservation and resource co-allocation frameworks, discussed in [3].

For what concerns the *Logging and Bookkeeping service*, it stores logging and bookkeeping information concerning events generated by the various components of the WMS. Using this information, the LB service keeps a state machine view of each job. The dependencies between this component and the other modules of the Workload Management System are not represented in the figure, again for increased simplicity.

The *Partitioner* is a Helper class used to support partitionable jobs, that is jobs which can be "decomposed" into sub-jobs; job partitioning is discussed in [3] and [7].

3.3 The Resource Broker

The Resource Broker (RB) can be considered the core component of the workload management system. Its main task is to find a computing resource that best matches the requirements and preferences of a submitted job.

This task is composed of three main functional units:

- a matchmaking unit, returning all the resources suitable for a given JDL expression;
- a ranking unit, returning the "best" resources for a given JDL expression, out of a set of matches;
- a planning strategy unit: the RB must take into account different parameters, such as the resource access policies, the availability of the input data set, the characteristics and status of resources, the availability of the required application environments. This resource selection code is kept modular enough to easily implement new, more clever strategies. As first and basic optimization strategy, jobs are submitted "close" to data.

The matchmaking is performed when the RB receives it, and therefore when a job is ready to be submitted to the Grid: a very lazy planning approach (the job is bound to a resource where the resource is already acquired; therefore when there is an available resource, the best-matching job among those that are in the queue is found) will be investigated in the future.

To achieve its goal the Resource Broker interacts with other Grid services, in particular it obtains information about data location from the Replica Catalog (which maps logical files to their physical entities), and information about Grid resources from the Information Services.

The implementation of the RB is based on the Condor matchmaking library: the required and preferred resources (specified in the JDL expression passed to the RB by the UI) are matched with a ClassAd view of the available resources (obtained from the Information Services and the Replica Catalog).

The Resource Broker is designed so that it can be used at different hierarchical levels: both at a *community* level (one RB serving a group of users) and possibly even at a *personal* level (one RB per submitting machine).

3.4 Integration with DAGMan

The first version of the Workload Management System supported only submission of independent jobs. Although this certainly allows the accomplishment of significant work, it became evident that providing support for managing jobs with dependencies between them would have better addressed some important application needs.

The reviewed version of the WMS was then designed with the integration of DAGMan in mind. When a DAG is submitted to Condor-G, this starts a local DAGMan process, whose purpose is to iterate through all the jobs contained in the DAG and start them in the right order: a job is passed to Condor-G when freed from any dependency. The whole processing is done in a fail-safe way.

Due to the lazy planning scenario that we are considering (each node of the DAG is bound to a resource when the job is ready to be submitted, that is when it is free from dependencies), some work was necessary in order to force the DAGMan process to call some scheduling functionality before the hand-over of a job to Condor-G, to find an appropriate resource for its execution. Aiming at maximizing the software reuse, it was considered an advantage if the solution could be shared with the Workload Manager implementation. The common solution was the adoption of the Helper approach, that easily allows calling the EDG Resource Broker and any other functionality exported via the Helper interface.

Additional work was devoted to bridge the different format of requests used by Condor-G and DAGMan on one side and the EDG software on the other.

4 Planning on the DZero SAMGrid

The D0 project approaches the topic of Job Planning from two directions. Logical job management involves understanding of what a D0 job is, and how it is viewed by the user and by the scheduler. Physical job management is centered at resource management and involves brokering, scheduling and planning. At D0, both aspects of job planning are impacted strongly by the *data handling* system, because most D0 jobs are fundamentally data-intensive and because data handling (SAM, [17]) has historically been the strongest part of the D0 meta-computing[21].

4.1 Job Definition and Structure

In the Grid world, it is widely known that jobs are composite entities represented by a DAG (Directed Acyclic Graph), and tools such as DAGMan emerge that can represent dependencies among the job's parts. In general, however, *job definition* is more than the mere specification of a DAG; the purpose of this section is to reflect the ongoing work to deeper understand and better describe the D0 job.

At D0, we say that a job is *unstructured*, if its details are unknown to the scheduler, the status monitoring service, and the metadata catalogue. The user submits a single script to the system, even though it may be quite complicated internally, and contain distinct steps from the user's point of view. The more job details are known to the system, the more *structured* we say it is.

Work to define the job structure started in [9]. In summary, a job is a collection of synchronized *phases*. Within each phase, different *lines* of execution take place, with varying degrees of parallelism and each consisting of a list of *packages* to be executed sequentially at each machine.

Implicit for each phase is the *input dataset*. A phase boundary (as opposed to a boundary between packages in a line) has results of independent importance, i.e., the user may wish to reuse them later. Furthermore, it is a high-level *checkpoint*, a place where the job may be suspended and resumed later, perhaps in a different location. One shouldn't confuse this checkpoint with the low-level checkpoint mechanism offered by Condor, which checkpoints a running binary using a memory dump of the process. Rather, since we deal with datasets, our inter-phase checkpointing makes interim data persistent with the data handling system so that it can be made available again later. Thus, defining and perhaps storing of the datasets is an *integral part* of job handling and our final job definition scheme will reflect this concept.

Another key point that cannot yet be fully represented using tools such as DAGMan is the degree of parallelism at each phase (or stage). Specifically, an optimal DAG can be specified if the number of nodes is known statically, i.e. prior to job submission. In the SAMGrid project, we believe, however, that once a phase is complete, the next phase's desired degree of parallelism is determined by the output of the preceding phase, such as the size of the dataset in terms of e.g. the number of files created by that phase. Thus, a highly optimized scheduler will obtain this information dynamically with the help of the data handling system. At present, however, we are uncertain whether such a functionality is required in the Grid planner or is best left to negotiation between the data handling and the local batch system (by means of the "sam submit" command[22]).

4.2 Physical Job Management

The most common Grid infrastructure for the fundamental service of remote job execution and monitoring is provided by the GRAM protocol and jobmanager service from the Globus Toolkit. The Condor-G software provides the higher level service of reliable job submission. For our purposes, we need to provide the *request brokering* service (referred to as resource brokering in [10]).

Having reviewed the relevant Grid technologies, we have chosen to use Condor's Match Making Service (MMS), [16] as the request broker. The novelty of our approach is that the properly configured MMS will be the whole request broker, rather than its part, base, or advisor. Our choice is driven by the collaboration of D0 with the University of Wisconsin under PPDG, as well by the record of success of MMS as part of the Condor system.

Such a utilization of MMS constitutes a paradigm shift for Condor-G from a personal grid manager to a whole system job manager. This approach gives us unlimited opportunities to implement various policies, as well as resource management considerations (together, they form what we call *computational economy*), by means of redefining the *ranking functions* for the jobs.

The contents of the ranking function will be determined hopefully as a result of a research project; at present, we are confident about one issue - that the data handling system will play a key role in defining such ranks. We believe the data handler will write part of the ClassAds, and will enter the job management area by means of more than mere publishing of its *replica catalogue* (a well established idea in the Grid world). Given the availability of SAM and experience using it, we believe that we are well positioned to provide a higher degree of sophistication in our computational economy. Examples of factors that may affect scheduling include:

- The utilization, current and projected, of the disk caches managed by SAM stations. The rate of data consumption by the existing tasks.
- The throughput of the data transfers to/from stations (this is different from network conditions as it is affected by the previous item).
- The near term predictions, from the Data Handling Global Resource manager, on the availability of the remotely cached datasets.

Note that these, and other factors may well change after the job is submitted and before it is successfully matched, thus precluding their full specification in the submitted or the advertised ClassAds. We have therefore extended the

ClassAd mechanism to include the capability to call, in the course of the match-making, an *externally supplied function*. In the near future, we plan to use these external functions to consult the data handling system.

The Condor MMS is a rather fertile ground for job management. Note that our idea to have the MMS use external function is not restricted to the data handling considerations. It gives the system designer a wide opportunity to include virtually arbitrary criteria for scheduling — a promising framework for building the prototype for SAM and other Grids.

4.3 D0 Summary and Further Information

In summary, we envision the following picture [14]. The user defines a job in some user-friendly language (the *job definition language*) whereby he specifies the various parts of the job, their dependencies, their desired or possible degree of parallelism and the significance (and ways of handling) of its interim datasets. This *job description file* (JDF) is processed by the client (*user interface* in the EDG terminology) so as to incorporate the appropriate ranking functions etc, and another job description file, this time in the format of ClassAds plus DAGMan. This JDF is given to the request broker that rejects or matches the job and further (although to a smaller extent) re-writes the JDF so as to send it to the job submission service. It is executed in the well-known fashion by the lower layers. For more details, please see [14] and references therein.

5 Conclusions

Two applications of existing Grid planning technology were presented. Both see the active participation of experimental/scientific groups whose computing needs are most appropriately addressed not just in terms of computing cycles, but also considering the nature, size and distribution of the data to be processed. Existing technologies now provide a firm ground where the Grid distributed computing paradigm can be matched to this kind of requirements. Extensions to the Class-ad match-making services are being worked on to make data, storage and other application-specific information available for symmetric match-making and simplify the procedures that are currently needed to map to the existing sources of Grid information. The advantage of applying (*lazy* or *very lazy*) planning choices to the individual stages that a single computation or data-handling job can be partitioned into is also becoming clearer. Therefore, the necessary analysis to extend the current support for declarative DAGs and provide handling for different classes of failure conditions and other events that may occur dynamically at DAG execution time is also in progress. The constant reality check provided by the needs of our end-users is the main catalyst allowing good progress for these developments.

ACKNOWLEDGEMENTS

The US work is partially sponsored by the Particle Physics Data Grid SciDac Collaboratory Pilot. This paper is PPDG-20.; Fermilab work is partially sponsored by DOE contract No. DE-AC02-76CH03000; WP1 work is funded by the European DataGrid Project (IST-2000-25182).

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] DataGrid - Definition of Architecture, Technical Plan and Evaluation Criteria for Scheduling, Resource Management, Security and Job Description. http://www.infn.it/workload-grid/docs/DataGrid-01-D1.2-0112-0-3.pdf.

[3] DataGrid - Definition of Architecture, Technical Plan and Evaluation Criteria for the Resource Co-Allocation Framework and Mechanisms for Parallel Job Partitioning. todo.

[4] Home page for the Grid Workload Management workpackage of the DataGrid project. http://www.infn.it/workload-grid.

[5] DataGrid - Job Description Language HowTo. http://www.infn.it/workload-grid/docs/DataGrid-01-TEN-0102-0_2.pdf.

[6] DataGrid - JDL Attributes. http://www.infn.it/workload-grid/docs/DataGrid-01-NOT-0101-0_6-Note.pdf.

[7] DataGrid - Job partitioning and checkpointing. http://www.infn.it/workload-grid/docs/DataGrid-01-TED-0119-0_3.pdf.

[8] C. Anglano et al. Integrating Grid tools to build a Computing Resource Broker: Activities of DataGrid WP1. In *Proceedings of the Conference on Computing in High Energy Physics 2001 (CHEP01)*, Beijing, September 2001.

[9] D. Meyer et al. D0 Job Components. internal D0 document.

[10] Home page for European DataGrid Project. http://www.eu-datagrid.org.

[11] Ian Foster and Carl Kesselman. Globus: A metacomputing intrastructure toolkit. *International Journal of Supercomputer Applications*, 11(2):115–128, 1997.

[12] James Frey, Todd Tannenbaum, Ian Foster, Miron Livny, and Steve Tuecke. Condor-G: A computation management agent for multi-institutional grids. In *Proceedings of the Tenth IEEE Symposium on High Performance Distributed Computing (HPDC)*, pages 7–9, San Francisco, California, August 2001.

[13] James Frey, Todd Tannenbaum, Ian Foster, Miron Livny, and Steve Tuecke. Condor-G: A computation management agent for multi-institutional grids. *Cluster Computing*, 5:237–246, 2002.

[14] G. Garzoglio. The SAM-GRID Project: Architecture and Plan (plenary talk). In *Proceedings of the VIII International Workshop on Advanced Computing and Analysis Techniques in Physics Research (ACAT'2002)*, June 2002 (to appear).

[15] Miron Livny and Rajesh Raman. High-throughput resource management. In Ian Foster and Carl Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1998.

[16] Rajesh Raman, Miron Livny, and Marvin 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.

[17] Home page for the SAM project. http://d0db.fnal.gov/sam.

[18] T. Sterling, D. Savarese, D. J. Becker, J. E. Dorband, U. A. Ranawake, and C. V. Packer. BEOWULF: A parallel workstation for scientific computation. In *Proceedings of the 24th International Conference on Parallel Processing*, pages 11–14, Oconomowoc, Wisconsin, 1995.

[19] Condor Team. Condor Manual. Available from `http://www.cs.wisc.edu/condor/manual`, 2001.

[20] Condor Team. The directed acyclic graph manager. http://www.cs.wisc.edu/condor/dagman, 2002.

[21] I. Terekhov. The SAM-GRID Project: Architecture and Plan (plenary talk). In *Proceedings of the VIII International Workshop on Advanced Computing and Analysis Techniques in Physics Research (ACAT'2002)*, June 2002 (to appear).

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

[23] Douglas Thain, Todd Tannenbaum, and Miron Livny. Condor and the grid. In Fran Berman, Geoffrey Fox, and Tony Hey, editors, *Grid Computing: Making the Global Infrastructure a Reality*. John Wiley & Sons Inc., December 2002.