Argonne National Laboratory preprint, ANL/MCS-P1000-1002

A General Architecture for Scheduling on the Grid

Jennifer M. Schopf
jms@mcs.anl.gov
Mathematics and Computer Sciences Division, Argonne National Laboratory
Department of Computer Science, Northwestern University

Abstract

In this paper we present a general architecture for scheduling on a Grid. A Grid scheduler (or broker) must make resource selection decisions in an environment where it has no control over the local resources, the resources are distributed, and information about the systems is often limited or dated. These interactions are also closely tied to the functionality of the Grid Information Services. The Grid scheduling architecture has three phases: resource discovery, system selection, and job execution. We detail the steps involved in each phase and give examples from current systems.

1. Introduction

More applications are turning to Grid computing to meet their computational and data storage needs. Single sites are simply no longer efficient for meeting the resource needs of high-end applications, and using distributed resources can give the application many benefits. Effective Grud computing is possible, however, only if the resources are scheduled well.

Grid scheduling is defined as the process of making scheduling decisions involving resources over multiple administrative domains. This process can include searching multiple administrative domains to use a single machine or scheduling a single job to use multiple resources at a single site or multiple sites. In this paper we do not address the situation of speculative execution – submitting a job to multiple resources and, when one begins to run, canceling the other submissions. We do, however, discuss resource selection (sometimes termed resource discovery [Berman99], assignment of application tasks to those resources (mapping [Berman99]), and data staging or distribution.

One of the primary differences between a Grid scheduler and a local resource scheduler is that the Grid scheduler does not own the local resources and therefore does not have control over them. The Grid scheduler must make best-effort decisions and then submit the job to the resources selected, generally as the user. Furthermore, the Grid scheduler does not have control over the set of jobs submitted to it, or even know about the jobs being sent to the resources it is considering use of, so decisions that tradeoff one job's access for another's cannot be made in the global sense. This lack of ownership and control is the source of many of the problems to be solved in this area.

We define a *job* to be anything that needs a resource – from a bandwidth request, to an application, to a set of applications (for example, a parameter sweep). We use the term *resource* to mean anything that can be scheduled: a machine, disk space, a QoS network, and so forth. In general, for ease of use, in this paper we refer to resources in terms associated with compute resources; however, nothing about the approach is limited in this way.

Currently, the most common Grid scheduler is the user. Many efforts are under way, however, to change this situation [Nab99, Silver, PBS, Loadleveler, LSF, Condor, EDG01, Apples, LYF+02, Cactus]. We give overviews of several of these in Section 2 and then use them as examples later in the paper, noting that no one system addresses all the needed features yet. In Section 3 we briefly discuss the related Grid Information System interactions expected by a Grid-level scheduler. In Section 4 we describe the three phases of scheduling a job over resources on multiple administrative domains- resource discovery, selection, and job execution. For each step we define the work involved, distinguish it from the work of a common parallel scheduler, and give examples from current systems.

2 Current Grid Scheduling Systems

In this section we give brief overviews of some current Grid scheduling efforts that we use as examples for the later sections of the paper.

2.1 Condor

The goal of the Condor project [Condor] is to develop, implement, deploy, and evaluate mechanisms and policies that support high-throughput computing (HTC) by using distributed environments that can deliver large amounts of processing capacity over long periods of time. The key to HTC is effective management and exploitation of all available computing resources. Recent trends in the cost/performance ratio of computer hardware have placed the control (ownership) over powerful computing resources in the hands of individuals and small groups. These distributed owners will be willing to include their resources in an HTC environment only after they are convinced that their needs will be addressed and their rights protected. It is the owner of each workstation in the collection who defines the conditions under which Condor can allocate the workstation to an external user.

When a user submits a job to Condor, the system finds an available machine on the network and begins running the job on that machine. If Condor detects that a machine running a Condor job is no longer available (perhaps because the owner of the machine came back from lunch and started typing on the keyboard), Condor can checkpoint the job, move (migrate) it to a different machine that would otherwise be idle, and continue the job on the new machine from precisely where that job left off.

Condor provides powerful resource management by matchmaking resource owners with resource consumers. Condor implements ClassAds [RLS00, RLS98], a way to describe jobs and resources in a fashion similar to a newspaper's classified ads. All machines in the Condor pool use a resource offer ad to advertise their resource properties, both static and dynamic, such as available RAM memory, CPU type, CPU speed, virtual memory size, physical location, and current load average. A user specifies a resource request ad when submitting a job. The request defines both the required and a desired set of properties of the resource to run the job. Condor acts as a broker by matching and ranking resource Grid offer ads with resource request ads, making certain that all requirements in both ads are satisfied.

In addition to using Condor as an example of scheduling, we also show examples from two systems derived from the Condor system: its use with Cactus as part of the GrADS project, and its use within the European Data Grid (EDG) project.

2.1.1 Cactus and Condor

Parts of Condor are being used very successfully in the GrADS [Grads] project to schedule applications in cooperation with the Cactus Computational Toolkit [Cactus, AAF+01]. Cactus is an open source problem-solving environment designed for scientists and engineers with a modular structure to enable parallel computation across different architectures and collaborative code development between different groups. This toolkit is being used in GRADS to compute the evolution of gravitational waves according to Einstein's theory of general relativity.

The scheduler used for the GrADS project is the University of Chicago GrADS Resource Selector [Angulo02, LYF+02]. This service extends the basic ClassAd mechanism to support information about sets of resources. The Globus information service, MDS [MDS, FFJ+97], is mined to produce the resource information, which is then conglomerated into data about sets of resources in ClassAds. This information is matched against a ClassAd submitted by the application with detailed usage requirements including an application model. The Condor matchmaking process selects the best match. Job submission is performed by using the standard Cactus job submission tools.

2.1.2 EDG Resource Broker

The Workload Management System (WMS) [EDG, EDG01] is the component of the Data Grid middleware responsible for managing the Grid resources in such a way that applications are conveniently, efficiently, and effectively executed. The core component of the EDG WMS is the Resource Broker (RB). Given a job description, the RB tries to find the best match between the job requirements and the resources available on the Grid, using information from the Globus MDS and Condor matchmaking techniques. The output of the search is a compute element where the job, while running, has access to all the resources specified in the job description, such as data or storage space. The description of a job is expressed in the Job Description Language, which is based on Condor ClassAds. Security, communication, and job submission are layered on top of standard Globus Toolkit components.

2.2 Portable Batch System: PBS

The Portable Batch System, PBS, is a workload management solution for HPC systems and Linux clusters. PBS was originally designed for NASA because existing resource management systems were inadequate for modern parallel/distributed computers and clusters. PBS includes many novel approaches to resource management and job scheduling, such as the extraction of scheduling policy into a single separable, completely customizable module.

The purpose of the PBS system is to provide additional controls over initiating or scheduling execution of batch jobs and to allow routing of those jobs between different hosts. The batch system allows a site to define and implement policy as to what types of resources and how much of each resource can be used by different jobs. The batch system also provides a mechanism with which a user can ensure that a job will have access to the resources required to complete that job.

The batch system comprises several components; typical interaction between the components is based upon the client-server model, with clients making (batch) requests to servers and the servers performing work on behalf of the clients. A batch server is a persistent process or set of processes, such as a daemon. The server manages a number of different objects, such as queues or jobs, each object consisting of a number of data items or attributes. It also provides batch services such as creating, routing, executing, modifying, or deleting jobs for batch clients. A batch server may at times request services of other servers; during such times, the server is acting in the role of a client. User, operator, and administrator commands are batch clients. They allow users of the batch system to request batch services via the command line. While the commands may appear to accomplish certain services, they actually request and obtain the services from a batch server by means of a batch request.

2.3 The KB Metascheduler

The KB Metascheduler is a project , led by Jarek Nabrzyski [Nab99, KNP01], out of the Poznan Supercomputing and Networking Center. This scheduler makes decisions using an AI knowledge-based (KB) multicriteria job-searching technique. It incorporates information about time costs, user preferences, load balancing, memory usage, and cache usage and uses this information with a set of expert techniques, each with its own strengths and weaknesses. AI techniques were chosen because of the complexity of the system, and the success of these techniques in other hard-to-manage environments.

This system is implemented on top of the Globus Toolkit and uses the standard Globus infrastructure with some added high-level services, such as advanced reservations and extensions to the standard information providers.

2.4 AppLeS Parameter Sweep Template

The Application Level Scheduling project (AppLeS) [Apples, BW96, BWF+96] is a high-performance scheduler targeted to multi-user distributed heterogeneous environments. Each Grid application is scheduled by its own AppLeS, which determines and actuates a schedule customized for the individual application and the target computational Grids at execution time. Key to the AppLeS approach is that everything in the system is evaluated in terms of its impact on the application. As a consequence, resources in the system are evaluated in terms of predicted capacities at execution time, as well as their potential for satisfying application resource requirements. AppLeS also interacts very strongly with dynamic information sources such as the Network Weather Service [NWS, WSH99].

The AppLeS project has developed several templates specific to application types. One of these is the AppLeS Parameter Sweep Template [CLZ+00]. Parameter sweep applications (PSAs) are typically structured as sets of "experiments", each of which is executed with a distinct set of parameters. Although parameter sweep applications are independent (i.e. they do not communicate), many PSAs are structured so that distinct experiments share large input files, and produce large output files. In order to achieve efficiency for large-scale runs, shared data files must be co-located with experiments, and the PSA must be scheduled to adapt to the dynamically fluctuating delays and qualities of the service of the shared resources.

The end user or application developer provides its AppLeS agent with application-specific information about current implementation(s) (via the Heterogeneous Application Template) as well as user preferences. This information is combined with dynamic system information (provided by the Network Weather Service) by the AppLeS Coordinator to determine a potentially performance-efficient application schedule. The Coordinator then works with the appropriate resource management systems to implement the schedule on the relevant resources.

2.5 Silver/Maui

The Silver Scheduler [Silver, Silver2, Maui] has been under development for a number of years and is currently being used at the Ohio Supercomputer Center and its eleven satellite sites. Fundamental to this Grid-level scheduler is the idea that a Grid-level module, Silver, interacts with local version of the Maui scheduler on each resource. A user submits a PBS-style job submission locally, which is translated into a meta-job submission and sent to the Silver module. This system has an extremely well developed simulation system that allows administrators to tune their queuing policies of the local Maui installations based on past traces of job submissions. It also has a well-developed framework for use with advance reservations. However, it interacts only with Maui running as a local scheduler.

3. Grid Information Service

The decisions a scheduler makes are only as good as the information provided to it. Many theoretical schedulers assume one has 100 percent of the information needed, at an extremely fine level of detail, and that the information is always correct. In Grid scheduling, this is far from our experience. In general we have only the highest level of information. For example, it may be known that an application needs to run on Linux, will produce output files somewhere between 20 MB and 30 MB, and should take less than three hours but might take as long as five. Or, it may be known that a machine is running Linux and has a file system located at a certain address that ten minutes ago had 500 MB free, but there is no information about what will be free when one's application runs there.

In general, Grid schedulers get information from a general Grid information system (GIS) that in turn gathers information from individual local resources. Examples of these systems are the Globus Monitoring and Discovery Service (MDS) [CFF+01, MDS] and the Grid Monitoring Architecture (GMA), developed by the Global Grid Forum performance working group [GGF-Perf, TAG+01], which has three reference implementations under development [CTG+01, Smith01, CDF+01], and being deployed as part of the European Data Grid project [EDG-WP3]. These two approaches emphasize different pieces of the monitoring problem although both address it as a whole: MDS concentrates on the resource discovery portion, while GMA concentrates on the provision of data, especially streaming data.

While different in architecture, all Grid schedulers have common features. Each deals with organizing sets of sensors (named either information providers or producers) in such a way that an outside system can have easy access to the data. They recognize that some data is more statically oriented, such as type of operating system, or which file systems are accessible; and this static data is often cached or made more rapidly available. They serve dynamic data in very different ways (streaming versus time-out caches, for example) but recognize the need for a heavier-weight interaction for dealing with data that changes more often. All of these systems are extensible to allow additional monitoring of quantities, as well as higher-level services such as better predictions or quality-of-information metrics [VS02, WSH99, SB99].

Typically, these systems must have an agreed-upon schema, or way to describe the attributes of the systems, in order for different systems to understand what the values mean. This is an area of on-going work and research [Glue, Damed], with considerable debate about how to represent a schema (using LDAP, XML, SQL, CIM, etc.) and what structure should be inherent to the descriptions. More information on the Globus MDS schemas is available at [MDS-Schemas] and for the EDG schemas is available at [EDG-MDS].

4. Stages of Grid Scheduling

Grid scheduling involves three main phase: resource discovery, which generates a list of potential resources; information gathering about those resources and selection of a best set; and job execution, which includes file staging and cleanup. These phases, and the steps that make them up, are shown in Figure 1. We give examples from current systems for each step. We note that no current Grid scheduler implements all of the steps of this architecture.

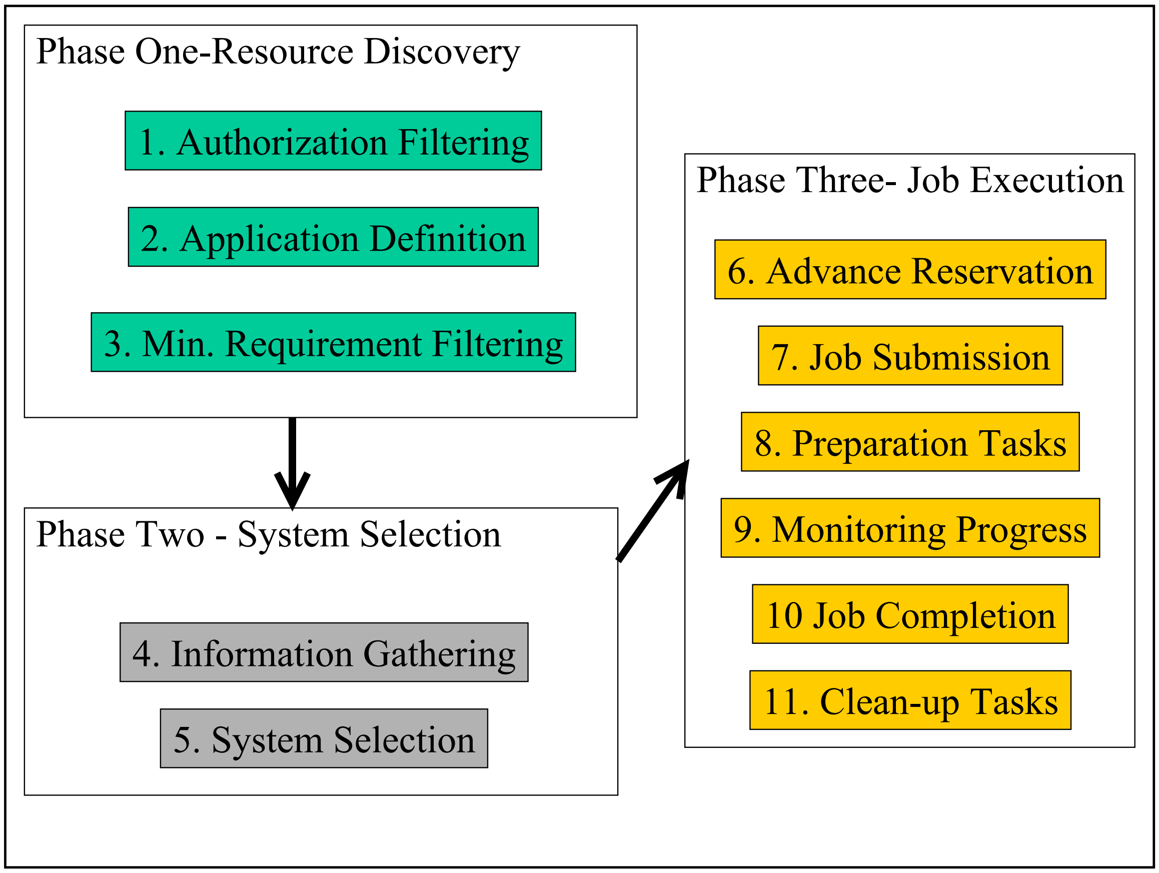
Phase One-Resource Discovery
1. Authorization Filtering
2. Application Definition
3. Min. Requirement Filtering
Phase Two - System Selection
4. Information Gathering
5. System Selection
Phase Three- Job Execution
6. Advance Reservation
7. Job Submission
8. Preparation Tasks
9. Monitoring Progress
10 Job Completion
11. Clean-up Tasks

Figure 1: Three-Phase Architecture for Grid Scheduling

4.1 Phase 1: Resource Discovery

The first stage in any scheduling interaction involves determining which resources are available to a given user. The resource discovery phase involves selecting a set of resources to be investigated in more detail in Phase 2. At the beginning of Phase 1, the potential set of resources is the empty set; at the end of this phase, the potential of resources is some set that has passed a minimal feasibility requirement. Resource discovery is done in three steps: authorization filtering, job requirement definition, and filtering to meet the minimal job requirements.

4.1.1 Step 1: Authorization Filtering

The first step of resource discovery in job scheduling is to determine the set of resources that the user submitting the job has access to. In this regard, computing over the Grid is no different from remotely submitting a job to a single site: without authorization to run on a resource the job will not run. At the end of this step the user will have a list of machines or resources to which he or she has access. The main difference that Grid computing lends to this problem is sheer numbers. It is now easier to get access to more resources, although equally difficult to keep track of them. Also, with current GIS implementations, a user can often find out the status of many more machines than what he or she has accounts on. As the number of resources grows, it simply does not make sense to examine those resources that are not authorized for use.

A number of recent efforts have helped users with security once they have accounts, but very little has been done to address the issues of accounting and account management [SN02]. When a user is performing scheduling at the Grid level, the most common solution to this problem is to simply have a list of account names, machines, and passwords written down somewhere and kept secure. While the information is generally available when needed, this method has problems with fault tolerance and scalability.

Current schedulers that need local permission to run are addressing this requirement in two ways: using the GIS to store this data or using a priori queue management. The KB scheduler keeps this information as part of an information service with other system information and queried as part of the standard resource discovery. The broker for EDG also stores information about every legitimate account on a system in the information index for the resource. The fact that this approach will not scale has been acknowledged, and discussions are ongoing for other implementations. PBS, for example, allows administrators to set up specific queues for authorization groups by setting up execution lists and attaching this information to a specific queue.

Condor does not require an account (login) on machines where it runs a job. Rather, it uses remote system call technology, which traps library calls for such operations as reading or writing from disk files. The calls are then transmitted over the network to be performed on the machine where the job was submitted.

4.1.2 Step 2: Application Requirement Definition

To proceed in resource discovery, the user must be able to specify some minimal set of job requirements in order to further filter the set of feasible resources (see Step 3).

The set of possible job requirements can be very broad and will vary significantly between jobs. It may include static details (the operating system or hardware for which a binary of the code is available, or the specific architecture for which the code is best suited) as well as dynamic details (for example, a minimum RAM requirement, connectivity needed, /tmp space needed). Some schedulers are at least allowing for better coarse-grained information about the applications, for example, the fact that the EDG broker allows for specification of certain software packages or the fact that AFS is available. The more details that are included, the better the matching effort can be.

Currently, the user specifies this data as part of the command line or submission script (PBS, LSF), or as part of the submitted ClassAd (in approaches using Condor's matchmaking such as the Cactus work or the EDG broker). Many projects have emphasized the need for this data as well, for example with AppLeS and the Network Weather Service. It is generally assumed in most system work that the information is simply available.

On a Grid system this situation is complicated by the fact that application requirements will charge with respect to the systems they are matched to. For example, depending on the architecture and the algorithm, memory requirements may change, as may libraries needed, or assumptions on available disk space.

Very little work has been done to automatically gather this data, or to store it for future use. This is in part because the information may be hard to discover. Attempts to have users supply this information on the fly has generally resulted in data that has dubious accuracy - for example, notice how almost every parallel scheduler requires an expected execution time, but almost every system administration working with these schedulers compensates for the error in the data, by as much as 50% in some [ASW99]. Attempts to gather the information, for example by using forms, has been found to be extremely time consuming as well [GFUWG99]. In the future, one can envision compilers that could aid in supplying basic requirements for applications [KMM+02], or monitoring, but generally this information, so closely tied to the applications are considered the responsibility of some package other than the schedulers, especially a Grid-level scheduler.

4.1.3 Step 3: Minimal Requirement Filtering

Given a set of resources to which a user has access and at least a small set of job requirements, the third step in the resource discovery phase is to filter out the resources that do not meet the minimal job requirements. At the end of this step, a Grid scheduler will have a reduced set of resources to investigate in more detail.

Current Grid Information Services are set up to contain both static and dynamic data, described in Section 3. Many of them cache data with an associated time-to-live value to allow quicker response times of long-lived data, including basic resource information such as operating system, available software, and hardware configuration. Since resources are distributed and getting data to make scheduling decisions can be slow, this step uses basic, mostly static, data to evaluate whether a resource meets some basic requirements. This is similar to the discovery stage in a monitoring and discovery service.

A user doing his or her own scheduling will simply go through the list of resources and eliminating the ones that do not meet the job requirements (as much as they are known), for example, ruling out all the non-AFS-accessible resources for applications requiring AFS.

Because the line between static and dynamic data is often one drawn for convenience and not for science, most automatic systems incorporate this feasibility searching into Step 4, where full-fledged queries are made on the system. PBS is one exception. It does a first pass on the available jobs and sorts them according to some administrator-defined policy, but also d then does a second-tier evaluation using dynamic filters for which those that should be run soonest. A high-level filter is then used on the “most deserving” job to determine which of the available resources can be used based on static criteria. The second stage matching for PBS, our Step 4 below, assigns the job to a specific resource based on a more detailed, dynamic policy. The Maui/Silver scheduler also does static-level information filtering at the higher Silver level, including account feasibility, aninformation at the local level using the Maui components. Condor also does an initial matching (using ClassAds and the matchmaker) and then a feasibility evaluation upon claiming the actual resources.

We maintain that as systems grow, this stage will be an important one for continued scalability of other Grid-level schedulers.

4.2 Phase :2 System Selection

Given a group of possible resources (or a group of possible resource sets), all of which meet the minimum requirements for the job, a single resource (or single resource set) must be selected on which to schedule the job. This selection is generally done in two steps: gathering detailed information and making a decision. We discuss these two steps separately, but they are inherently intertwined, as the decision process depends on the available information.

4.2.1 Step 4: Dynamic Information Gathering

In order to make the best possible job/resource match, detailed dynamic information about the resources is needed. Since this information may vary with respect to the application being scheduled and the resources being examined, no single solution will work in all, or even most, settings. The dynamic information gathering step has two components: what information is available and how the user can get access to it.

In general, most schedulers interact with a Grid information service, as described in Section 3. Additionally, some scheduler-specific information may be available. PBS runs a daemon on each execution host to gather this data, which is then sent back to the central Grid-level server. This data is collected at configurable intervals. The basic Condor approach generates the information on each machine, sends this data to the central manager, and updates the dynamic information every five minutes. A user (or scheduling service) would ask, If I give this job to a resource (or set of resources) now, when will it finish? However, no current LRMS can answer that question, even partially (e.g., with an answer such as, It should take one hour but might be as long as three).

In general on the Grid, scalability issues as well as consistency concerns significantly complicate the situation. Not only must more queries be made, but also if resources are inaccessible for a period of time, the system must evaluate what to do about the data needed for those sites. Currently, this situation is generally avoided by assuming that if dynamic data is not available, the resources should be ignored; however, in larger systems, another approach should be considered.

A large body of predictive work exists in this area, but most of it requires additional information not available on current systems. And even the more applied work [Wolski98, Netlogger, SFT98, Downey97] has not been deployed on current systems.

4.2.2 Step 5: System Selection

With the detailed information gathered in Step 4, the next step is to decide which resource (or set of resources) to use. Various approaches are possible.

The most commonly used system for matching application requirements to resources is the Condor Matchmaker/ClassAd system. A ClassAd can contain an expression that evaluates how well it matches to the corresponding ClassAds. For example, a resource owner can specify that member of the research group get first priority, those in the university second priority, and then anyone else from a given set. This simple concept can be extended as needed. Additional work has been done in the GrADS project to match sets of resources to applications instead of just single resources.

The Silver/Maui scheduler submits a full job description to each of the local schedulers, each of which individually returns feasible time ranges, including estimated execution times, cost, and resources used. The higher-level Silver daemon then performs range-based calculus to select the best resource based on user-submitted criteria.

The KB scheduler uses a novel multi-objective schedule evaluation that finds compromise schedules based on a set of AI techniques, thereby avoiding some of the more common scalability issues with more statistical-based approaches. This should allow more flexibility and a better scaling effect as the number of local resources considered grows.

4.3 Phase 3: Job Execution

The third phase of Grid scheduling is running a job. This involves a number of steps, few of which have been defined in a uniform way between resources.

4.3.1 Step 6: Advance Reservation

In order to make the best use of a given system, part or all of the resources may have to be reserved in advance. Depending on the resource, advance reservation can be easy or hard to do and may be done with mechanical means or human means. Moreover, the reservations may or may not expire with or without cost.

One issue in having advance reservations become more common is the need for the lower-level resource to support the fundamental services on the native resources. Currently, such support is not implemented in many resources.

Advance reservation work has been done in the Globus Project by Roy [GARA, FRS00] in reserving resources other than machines or networks. PBS has advance reservations available on its resources. It is configurable as to who can make an advance reservation; if an advance reservation can be made, PBS will make one. The Maui/Silver system goes one step further in that it can be configured to try to obtain a better (sooner) advanced reservation on the available resources until the originally set reservation is claimed.

4.3.2 Step 7: Job Submission

Once resources are chosen, the application can be submitted to the resources. Job submission may be as easy as running a single command or as complicated as running a series of scripts and may or may not include setup or staging (see Step 8).

A typical example is running the qsub through PBS. For example, in PBS

```
% qsub -l ncpus=16 -l walltime=4:00:00 myjobscript
> 16388.cluster.pbspro.com
```

will submit a 16-node job with a maximum wallclock time of 4 hours, using the setup script myjobscript, and will tag it with the PBS identifier 16388.cluster.pbspro.com.

In a Grid system, the simple act of submitting a job can be made very complicated by the lack of any standards for job submission. Some systems, such as the Globus GRAM approach [GRAM], wrap local scheduling submissions but rely heavily on local-parameter fields. Ongoing efforts in the Global Grid Forum [GGF, GGF-Sched] address the need for common APIs [GGF-DRMMA], languages [GGF-dict], and protocols [GGF-GRM], but much work is still to be done.

4.3.3 Step 8: Preparation Tasks

The preparation stage may involve setup, staging, claiming a reservation, or other actions needed to prepare the resource to run the application. One of the first attempts at writing a scheduler to run over multiple machines at NASA was considered unsuccessful because it did not address the need to stage files automatically, for example.

Most often, a user will run scp, ftp or a large file transfer protocol such as GridFTP [ABF+02, ABB+01] to ensure that the data files needed are in place. Condor also supports a variation of this stage, in which it runs an extra process before and after the application is run to do staging and clean-up. Unicore [Unicore] can also support automated prestaging. PBS supports file staging as well using RCP or SCP currently and GridFTP in the near future; the files to be staged are specified as part of the input command line or script. In a Grid setting, authorization issues, such as having different user names at different sites or storage locations, as well as scalability issues, can complicate this process.

4.3.4 Step 9: Monitoring Progress

Depending on the application and its running time, users may monitor the progress of their application and possibly change their mind about where or how it is executing.

Today, such monitoring is typically done by repetitively querying the resource for status information. PBS has a CPU time or walltime maximum only, but it supports basic status commands, as do Condor, using the Q command, and LSF, using its status command.

If a job is not making sufficient progress, it may be rescheduled (i.e., returning to Step 4). Such rescheduling is significantly harder on a Grid system than on a single parallel machine because of the lack of control involved – other jobs may be scheduled and the one of concern pre-empted, without warning or notification. In general, a Grid scheduler will not be able to address this situation. It may be possible to develop additional primitives for interactions between local systems and Grid schedulers, but no current work is taking place in this area.

4.3.5 Step 10: Job Completion

When the job is finished, the user needs to be notified. Often, submission scripts for parallel machines will include an e-mail notification parameter. This is what PBS does, for example.

For fault-tolerant reasons, however, such notification can prove surprisingly difficult. Moreover, with so many interacting systems one can easily envision situations in which a completion state cannot be reached. Much further work is needed.

4.3.6 Step 11: Cleanup Tasks

After a job is run, the user may need to retrieve files from that resource in order to do data analysis on the results, remove temporary settings, and so forth. Any of the current systems that do staging (Step 8) also handle cleanup.

5. Conclusion

This document defines the steps a user currently follows to make a scheduling decision across multiple administrative domains. Scheduling on a Grid comprises three main phases: (1) resource discovery, which generates a list of potential resources; (2) information gathering and choosing a best set of resources; and (3) job execution, which includes file staging and cleanup. We also review current approaches in these areas.

While many schedulers have begun to address the needs of a true Grid-level scheduler, none of them currently supports the full range of actions required. Throughout this paper we have directed attention to complicating factors that must be addressed for the next generation of schedulers to be more successful in a complicated Grid setting.

Acknowledgments

Thanks to the Grid Forum Scheduling Area participants for initial discussions, especially the co-chair, Bill Nitzberg (PBS, Veridian Systems). Many thanks also to Alan Su (UCSD, AppLeS), Dave Jackson (Ames Lab, Maui/Silver), Alain Roy (University of Wisconsin, Condor), and Dave Angulo (University of Chicago, Cactus). This work was supported in part by the Mathematical Information and Computational Sciences Division Subprogram of the Office of Advanced Scientific Computing Research, U.S. Department of Energy, under contract W-31-109-Eng-38.

References

[AAF+01] Gabrielle Allen, David Angulo, Ian Foster, Gerd Lanfermann, Chang Liu, Thomas Radke, Ed Seidel, and John Shalf, "The Cactus Worm: Experiments with Dynamic Resource Discovery and Allocation in a Grid Environment", International Journal of High Performance Computing Applications, Volume 15, Number 4, 2001 (also available from http://www.cactuscode.org/Papers/IJSA_2001.pdf).

[ABB+01] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnel, and S. Tuecke, "Data Management and Transfer in High-Performance Computational Grid Environments", Parallel Computing*,* 2001.

[ABF+02] W. Allcock, J. Bresnahan, I. Foster, L. Liming, J. Link, and P. Plaszczac**,** GridFTP Update, http://www.globus.org/datagrid/deliverables/GridFTP-Overview-200201.pdf, January 2002.

[Angulo02] Dave Angulo, personal communication, 2002.

[Apples] Application Level Scheduling (AppLeS), http://apples.ucsd.edu/.

[ASW99] Alliance Scheduling Workshop, UIUC, February 1999.

[Berman99] Francine Berman, "High Performance Schedulers", Chap 12, in I. Foster and C. Kesselman, ed., The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann Publishers, Inc, 1999.

[BW96] Fran Berman and Rich Wolski, "Scheduling from the Perspective of the Application", Proceedings of the Symposium on High Performance Distributed Computing, 1996 (also available at http://apples.ucsd.edu/pubs/hpdc96.ps).

[BWF+96] Fran Berman, Richard Wolski, Silvia Figueira, Jennifer Schopf, and Gary Shao, "Application-Level Scheduling on Distributed Heterogeneous Networks", Proceedings of Supercomputing 1996 (also available at http://apples.ucsd.edu/pubs/sup96.ps and UCSD CS Tech Report #CS96-482).

[Cactus] Cactus, http://www.cactuscode.org/, 2002.

[CDF+01] B. Coghlan, A. Djaoui, S. Fisher, J. Magowan, and M. Oevers, "Time, Information Services and the Grid", Proceedings of the British National Conference on Databases, 2001.

[CFF+01] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman, "Grid Information Services for Distributed Resource Sharing", Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10), IEEE Press, August 2001.

[CLZ+00] Henri Casanova, Arnaud Legrand, Dmitrii Zagorodnov, and Francine Berman, "Heuristics for Scheduling Parameter Sweep applications in Grid Environments", Proceedings of the Heterogeneous Computing Workshop, May 2000.

[Condor] Condor, http://www.cs.wisc.edu/condor/publications.html.

[CTG+01] B. Crowley, B. Tierney, D. Gunter, J. Lee, and M. Stouffer, Python GMA, http://sourceforge.net/projects/py-gma/, 2001.

[Damed] Discovery and Monitoring Event Data Working Group, http://www-didc.lbl.gov/damed/.

[Downey97] A. Downey, "Queue Times on Space-Sharing Parallel Computers", Proceedings of the 11th International Parallel Processing Symposium, 1997.

[EDG] European Data Grid Project, http://eu-datagrid.web.cern.ch/eu-datagrid/default.htm.

[EDG01] European Data Grid Project WP1, "Definition of Architecture, Technical Plan and Evaluation Criteria for Scheduling, Resource Management, Security and Job Description", Datagrid document DataGrid-01-D1.2-0112-0-3, 14/09/2001 (also available from http://server11.infn.it/workload-grid/docs/DataGrid-01-D1.2-0112-0-3.pdf).

[EDG-MDS] European Data Grid Project, "MDS Deployment for Testbed 1", Datagrid Document MDS-Deployment-PM9-WP3, January 2002 (also available from marianne.in2p3.fr/datagrid/documentation/MDS-Deployment-PM9-WP3.pdf).

[EDG-WP3] European DataGrid Work Package 3 - Grid Monitoring Services, http://hepwww.rl.ac.uk/DataGridMonitoring/, 2001.

[FFK+97] S. Fitzgerald, I. Foster, C. Kesselman, G. von Laszewski, W. Smith, and S. Tuecke, "A Directory Service for Configuring High-Performance Distributed Computations", Proceedings of the 6th IEEE Symp. on High-Performance Distributed Computing, pp. 365-375, 1997 (also available at http://ftp.globus.org/pub/globus/papers/hpdc-97-mds.pdf).

[FRS00] I. Foster, A. Roy, and V. Sander, "A Quality of Service Architecture That Combines Resource Reservation and Application Adaptation", 8th International Workshop on Quality of Service, 2000 (also available at http://www.globus.org/documentation/incoming/iwqos_adapt1.pdf).

[GARA] Globus Advance Reservation Architecture, http://www-fp.mcs.anl.gov/qos/papers/gara_admin_guide.pdf.

[GFUWG99] Grid Forum Users Working Group, www.gridforum.org, 2000.

[GGF] Global Grid Forum, www.gridforum.org.

[GGF-dict] Global Grid Forum Scheduling Dictionary Working Group, http://www-unix.mcs.anl.gov/~schopf/ggf-sched/WG/sd-wg.html.

[GGF-DRMMA] Global Grid Forum Distributed Resource Management Application API Working Group, http://www-unix.mcs.anl.gov/~schopf/ggf-sched/WG/drmaa-wg.html.

[GGF-GRM] Global Grid Forum Grid Resource Management Working Group, http://www-unix.mcs.anl.gov/~schopf/ggf-sched/WG/grm-wg.html.

[GGF-Sched] Global Grid Forum Scheduling Area, http://www-unix.mcs.anl.gov/~schopf/ggf-sched/.

[GGF-Perf] Global Grid Forum Performance Area, http://www.gridforum.org/4_GP/Perf.htm, 2001.

[Glue] Glue-schema, http://www.hicb.org/glue/glue-schema/schema.htm.

[Grads] Grid Application Development Software Project, http://nhse2.cs.rice.edu/grads/

[GRAM] Globus Resource Allocation Manager, http://www.globus.org/gram/.

[KMM+02] Ken Kennedy, Mark Mazina, John Mellor-Crummey, Keith Cooper, Linda Torczon, Fran Berman, Andrew Chien, Holly Dail, Otto Sievert, Dave Angulo, Ian Foster, Dennis Gannon, Lennart Johnsson, Carl Kesselman, Ruth Aydt, Daniel Reed, Jack Dongarra, Sathish Vadhiyar, and Rich Wolski, "Towards a Framework for Preparing and Executing Adaptive Grid Programs", Proceedings of NSF Next Generation Systems Program Workshop (International Parallel and Distributed Processing Symposium 2002), Fort Lauderdale, FL, to appear (also available from http://nhse2.cs.rice.edu/grads/publications/GrADSIPDPS02.pdf).

[KNP01] Krysztof Kurowski, Jarek Nabrzyski, and Juliusz Pulacki, "User Preference Driven Multiobjective Resource Management in Grid Environments", Proceedings of CCGrid 2001, May 2001.

[LSF] LSF, http://www.platform.com/products/LSF/.

[Loadleveler] Loadleveler, http://www-1.ibm.com/servers/eserver/pseries/software/sp/loadleveler.html.

[LYF+02] Chuang Liu, Yingyun Yang, Ian Foster, and Dave Angulo, "Design and Evaluation of a Resource Selection Framework for Grids" (available from http://people.cs.uchicago.edu/~dangulo/papers/hpdc-resource-selector.pdf).

[Maui] Maui Scheduler, http://supercluster.org/maui.

[MDS] Globus Monitoring and Discovery Service, MDS 2.1 Features in the Globus Toolkit 2.0 Release, http://www-fp.globus.org/gt2/mds2.1/, 2001.

[MDS-Schemas] Globus Monitoring and Discovery Service (MDS) 2.1 Schemas, http://www.globus.org/gt2/mds2.1/Schema.html.

[Nab99] Jarek Nabrzyski, "Knowledge-based Scheduling Method for Globus", Globus Retreat, Redondo Beach, 1999, http://www.man.poznan.pl/metacomputing/ai-meta/globusnew/index.html.

[Netlogger] NetLogger: A Methodology for Monitoring and Analysis of Distributed Systems, http://www-didc.lbl.gov/NetLogger.

[NWS] The Network Weather Serrvice, http://nws.npaci.edu/NWS/.

[PBS] Portal Batch System (PBS), http://pbspro.com.

[RLS00] Rajesh Raman, Miron Livny, and Marvin Solomon, "Resource Management through Multilateral Matchmaking", Proceedings of the 9th IEEE Symposium on High Performance Distributed Computing (HPDC9), Pittsburgh, Pennsylvania, August 2000, pp. 290-291 (also available at http://www.cs.wisc.edu/condor/doc/gangmatching.ps).

[RLS98] Rajesh Raman, Miron Livny, and Marvin Solomon, "Matchmaking: Distributed Resource Management for High Throughput Computing", Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing, July 28-31, 1998, Chicago, IL (also available at http://www.cs.wisc.edu/condor/doc/hpdc8.ps).

[SB99] J. M. Schopf and F. Berman, "Stochastic Scheduling", Proceedings of SuperComputing'99 , 1999 (also available at http://www.cs.nwu.edu/~jms/Pubs/TechReports/sched.ps).

[SFT98] W. Smith, I. Foster, and V. Taylor, "Predicting Application Run Times Using Historical Information", Proceedings of the IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing, 1998.

[Silver] Silver, http://www.supercluster.org/projects/silver/.

[Silver2] Silver Design Overview, http://supercluster.org/projects/silver/designoverview.html.

[Smith01] W. Smith, "A Framework for Control and Observation in Distributed Environments", NASA Ames Technical Report NAS-01-006, June 2001.

[SN02] Jennifer M. Schopf and Bill Nitzberg, "Grids: The Top Ten Questions", Scientific Programming, Special Issue on Grid Computing, to appear (also available from http://www-unix.mcs.anl.gov/~schopf/Pubs/topten.final.pdf).

[TAG+01] B, Tierney, R. Aydt, D. Gunter, W. Smith, V. Taylor, R. Wolski, and M. Swany, "A Grid Monitoring Architecture", GWD-Perf-16-1, 2001 (also available at http://www-didc.lbl.gov/GGF-PERF/GMA-WG/papers/GWD-GP-16-1.pdf).

[Unicore] Unicore, http://www.unicore.de/

[VS02] Sudharshan Vazhkudai and Jennifer M. Schopf, "Predicting Sporadic Grid Data Transfers", Preprint ANL/MCS-P949-0402, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, April 2002 (also available from http://www-unix.mcs.anl.gov/~schopf/Pubs/hpdc-prediction-submitted.pdf).

[WSH99] Rich Wolski, Neil T. Spring, and Jim Hayes, "The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing", Journal of Future Generation Systems, 1999 (also available at http://www.cs.ucsd.du/users/rich/papers/nws-arch.ps.gz or as UCSD Technical Report Number TR-CS98-599, September, 1998).

[Wolski98] R. Wolski, "Dynamically Forecasting Network Performance Using the Network Weather Service", Journal of Cluster Computing, Volume 1, pp. 119-132, January 1998.