Condor-G: A Computation Management Agent for Multi-Institutional Grids

James Frey, Todd Tannenbaum, Miron Livny
Department of Computer Science
University of Wisconsin
Madison, WI 53706
{ jfrey, tannenba, miron }@cs.wisc.edu

Ian Foster, Steven Tuecke
Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, IL 60439
{ foster, tuecke }@mcs.anl.gov

Abstract

In recent years, there has been a dramatic increase in the amount of available computing and storage resources. Yet few have been able to exploit these resources in an aggregated form. We present the Condor-G system, which leverages software from Globus and Condor to allow users to harness multi-domain resources as if they all belong to one personal domain. We describe the structure of Condor-G and how it handles job management, resource selection, security, and fault tolerance.

1. Introduction

In recent years the scientific community has experienced a dramatic pluralization of computing and storage resources. The national high-end computing centers have been joined by an ever-increasing number of powerful regional and local computing environments. The aggregated capacity of these new computing resources is enormous. Yet, to date, few scientists and engineers have managed to exploit the aggregate power of this seemingly infinite Grid of resources. While in principle most users could access resources at multiple locations, in practice few reach beyond their home institution, whose resources are often far from sufficient for increasingly demanding computational tasks such as simulation, large scale optimization, Monte Carlo computing, image processing, and rendering. The problem is the significant "potential barrier" associated with the diverse mechanisms, policies, failure modes, performance uncertainties, etc., that inevitably arise when we cross the boundaries of administrative domains.

Overcoming this potential barrier requires new methods and mechanisms that meet the following three key user requirements for computing in a "Grid" that comprises resources at multiple locations:

- They want to be able to discover, acquire, and reliably manage computational resources dynamically, in the course of their everyday activities.
- They do not want to be bothered with the location of these resources, the mechanisms that are required to use them, with keeping track of the status of computational tasks operating on these resources, or with reacting to failure.
- They do care about how long their tasks are likely to run and how much these tasks will cost.

In this article, we present an innovative distributed computing framework that addresses these three issues. The *Condor-G* system leverages the significant advances that have been achieved in recent years in two distinct areas: (1) security, resource discovery, and resource access in *multi-domain* environments, as supported within the Globus Toolkit [12], and (2) management of computation and harnessing of resources within a *single* administrative domain, specifically within the Condor system [20, 22]. In brief, we combine the inter-domain resource management protocols of the Globus Toolkit and the intra-domain resource management methods of Condor *to allow the user to harness multi-domain resources as if they all belong to one personal domain*. The user defines the tasks to be executed; Condor-G handles all aspects of discovering and acquiring appropriate resources, regardless of their location; initiating, monitoring, and managing execution on those resources; detecting and responding to failure; and notifying the user of termination. The result is a powerful tool for managing a variety of parallel computations in Grid environments.

Condor-G's utility has been demonstrated via record-setting computations. For example, in one recent computation a Condor-G agent managed a mix of desktop workstations, commodity clusters, and supercomputer processors at ten sites to solve a previously open problem in numerical optimization. In this computation, over 95,000 CPU hours were delivered over a period of less than seven days, with an average of 653 processors being active at any one time. In another case, resources at three sites were used to simulate and reconstruct 50,000 high-energy physics events, consuming 1200 CPU hours in less than a day and a half.

In the rest of this article, we describe the specific problem we seek to solve with Condor-G, the Condor-G architecture, and the results obtained to date.

2. Large-scale sharing of computational resources

We consider a Grid environment in which an individual user may, in principle, have access to computational resources at many sites. Answering why the user has access to these resources is not our concern. It may be because the user is a member of some scientific collaboration, or because the resources in question belong to a colleague, or because the user has entered into some contractual relationship with a resource provider [14]. The point is that the user is authorized to use resources at those sites to perform a computation. The question that we address is *how to build and manage a multi-site computation* that uses those resources.

Performing a computation on resources that belong to different sites can be difficult in practice for the following reasons:

- Different sites may feature different authentication and authorization mechanisms, schedulers, hardware architectures, operating systems, file systems, etc.
- The user has little knowledge of the characteristics of resources at remote sites, and no easy means of obtaining this information.
- Due to the distributed nature of the multi-site computing environment, computers, networks, and subcomputations can fail in various ways.
- Keeping track of the status of different elements of a computation involves tedious bookkeeping, especially in the event of failure and dependencies among subcomputations.

Furthermore, the user is typically not in a position to require uniform software systems on the remote sites. For example, if all sites to which a user had access ran DCE and DFS, with appropriate cross-realm Kerberos authentication arrangements, the task of creating a multi-site computation would be significantly easier. But it is not practical in the general case to assume such uniformity.

The Condor-G system addresses these issues via a separation of concerns between the three problems of remote resource access, computation management, and remote execution environments:

- *Remote resource access* issues are addressed by requiring that remote resources speak standard protocols for resource discovery and management. These protocols support secure discovery of remote resource configuration and state, and secure allocation of remote computational resources and management of computation on those resources. We use the protocols defined by the Globus Toolkit [12], a de facto standard for Grid computing.
- *Computation management* issues are addressed via the introduction of a robust, multi-functional *user computation management agent* responsible for resource discovery, job submission, job management, and error recovery. This Condor-G component is taken from the Condor system [20].
- *Remote execution environment* issues are addressed via the use of *mobile sandboxing* technology that allows a user to create a tailored execution environment on a remote node. This Condor-G component is also taken from the Condor system.

This separation of concerns between remote resource access and computation management has some significant benefits. First, it is significantly less demanding to require that a remote resource speak some simple protocols rather than to require it to support a more complex distributed computing environment. This is particularly important given that the deployment of production Grids [4, 18, 27] has made it increasingly common that remote resources speak these protocols. Second, as we explain below, careful design of remote access protocols can significantly simplify computation management.

3. Grid protocol overview

In this section, we briefly review the Grid protocols that we exploit in the Condor-G system: GRAM, GASS, MDS-2, and GSI. The Globus Toolkit provides open source implementations of each.

3.1. Grid security infrastructure

The Globus Toolkit's Grid Security Infrastructure (GSI) [13] provides essential building blocks for other Grid protocols and for Condor-G. This authentication and authorization system makes it possible to authenticate a user just once, using public key infrastructure (PKI) mechanisms to verify a user-supplied "Grid credential." GSI then handles the mapping of the Grid credential to the diverse local credentials and authentication/authorization mechanisms that apply at each site. Hence, users need not re-authenticate themselves each time they (or a program acting on their behalf, such as a Condor-G computation management service) access a new remote resource.

GSI's PKI mechanisms require access to a *private key* that they use to sign requests. While in principle a user's private key could be cached for use by user programs, this approach exposes this critical resource to considerable risk. Instead, GSI employs the user's private key to create a *proxy credential*, which serves as a new private-public key pair that allows a proxy (such as the Condor-G agent) to make remote requests on behalf of the user. This proxy credential is analogous in many respects to a Kerberos ticket [26] or Andrew File System token.

3.2. GRAM protocol and implementation

The Grid Resource Allocation and Management (GRAM) protocol [10] supports remote submission of a computational request ("run program P") to a remote computational resource, and subsequent monitoring and control of the resulting computation. Three aspects of the protocol are particularly important for our purposes: security, two-phase commit, and fault tolerance. The latter two mechanisms were developed in collaboration with the UW team and are not yet part of the GRAM version included in the Globus Toolkit. They will be in the GRAM-2 protocol revision scheduled for later in 2001.

GSI security mechanisms are used in all operations to authenticate the requestor and for authorization. Authentication is performed using the supplied proxy credential, hence providing for single sign-on. Authorization implements local policy and may involve mapping the user's "Grid id" into a local subject name; however, this mapping is transparent to the user. Work in progress will also allow authorization decisions to be made on the basis of capabilities supplied with the request.

Two-phase commit is important as a means of achieving "exactly once" execution semantics. Each request from a client is accompanied by a unique sequence number, which is also included in the associated response. If no response is received after a certain amount of time, the client can repeat the request. The repeated sequence number allows the resource to distinguish between a lost request and a lost response. Once the client has received a response, it then sends a "commit" message to signal that job execution can commence.

Resource-side fault tolerance support addresses the fact that a single "resource" may often contain multiple processors (e.g., a cluster or Condor pool) with specialized "interface" machines running the GRAM server(s) that maintain the mapping from submitting client to local process. Consequently, failure of an interface machine may result in the remote client losing contact with what is otherwise a correctly queued or executing job. Hence, our GRAM implementation logs details of all active jobs to stable storage at the client side, allowing this information to be retrieved if a GRAM server crashes and is restarted. This information can include details of how much standard output and error data has been received, thus permitting a client to request resending of this data after a crash of client or server.

3.3. MDS protocols and implementation

The Globus Toolkit's MDS-2 provides basic mechanisms for discovering and disseminating information about the structure and state of Grid resources [9]. The basic ideas are simple. A resource uses the Grid Resource Registration Protocol (GRRP) to notify other entities that it is part of the Grid. Those entities can then use the Grid Resource Information Protocol (GRIP) to obtain information about resource status. These two protocols allow us to construct a range of interesting structures, including various types of directories that support discovery of interesting resources. GSI authentication is used as a basis for access control.

3.4. GASS

The Globus Toolkit's Global Access to Secondary Storage (GASS) service [7] provides mechanisms for transferring data between a remote HTTP, FTP, or GASS server. In the current context, we use these mechanisms to stage executables and input files to a remote computer. As usual, GSI mechanisms are used for authentication.

4. Computation management: the Condor-G agent

Next, we describe the Condor-G computation management service (or Condor-G agent).

4.1. User interface

The Condor-G agent allows the user to treat the Grid as an entirely local resource, with an API and command line tools that allow the user to perform the following job management operations:

- submit jobs, indicating an executable name, input/output files and arguments;
- query a job's status, or cancel the job;
- be informed of job termination or problems, via callbacks or asynchronous mechanisms such as email;
- obtain access to detailed logs, providing a complete history of their jobs' execution.

There is nothing new or special about the semantics of these capabilities, as one of the main objectives of Condor-G is to preserve the look and feel of a local resource manager. The innovation in Condor-G is that these capabilities are provided by a personal desktop

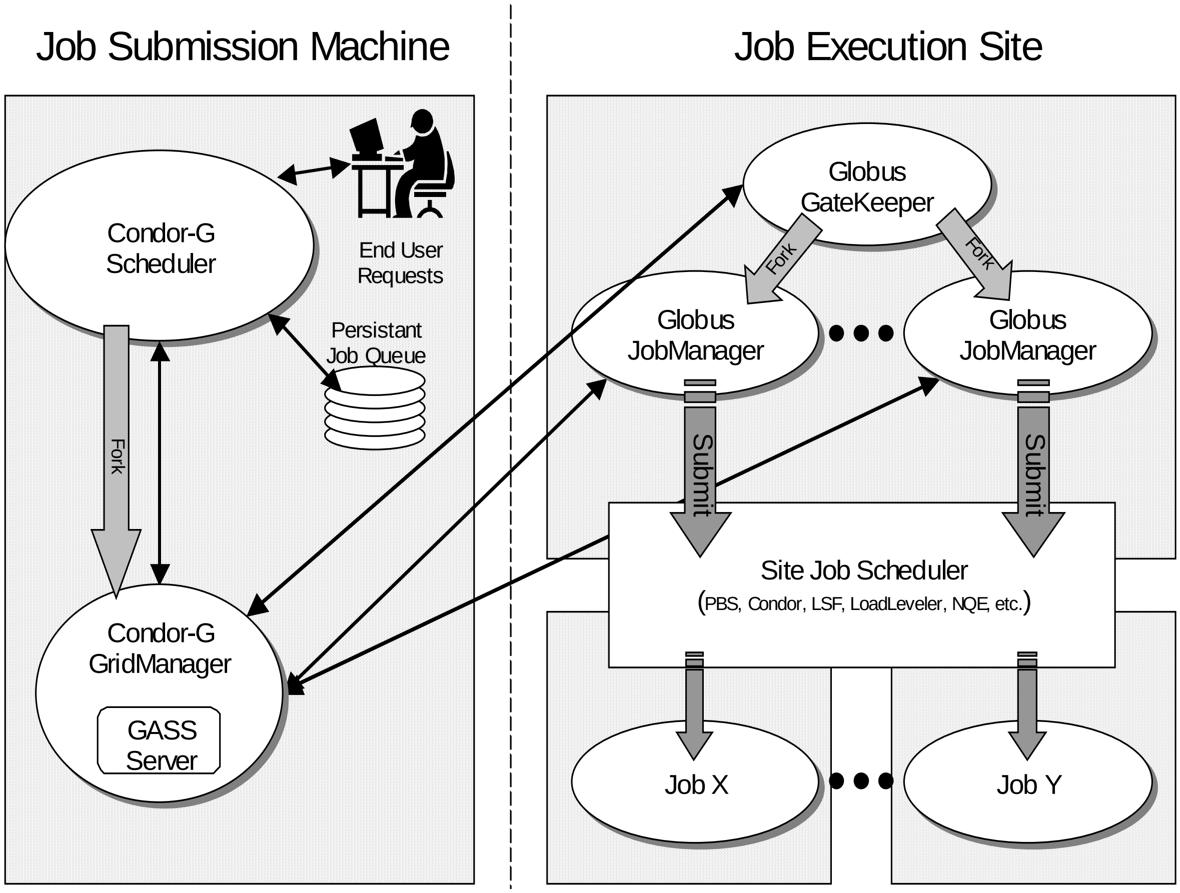

Figure 1. Remote execution by Condor-G on Globus-managed resources

agent and supported in a Grid environment, while guaranteeing fault tolerance and exactly-once execution semantics. By providing the user with a familiar and reliable single access point to all the resources he/she is authorized to use, Condor-G empowers end-users to improve the productivity of their computations by providing a unified view of dispersed resources.

4.2. Supporting remote execution

Behind the scenes, the Condor-G agent executes user computations on remote resources on the user's behalf. It does this by using the Grid protocols described above to interact with machines on the Grid and mechanisms provided by Condor to maintain a persistent view of the state of the computation. In particular, it:

- stages a job's standard I/O and executable using GASS,
- submits a job to a remote machine using the revised GRAM job request protocol, and
- subsequently monitors job status and recovers from remote failures using the revised GRAM protocol and GRAM callbacks and status calls, while
- authenticating all requests via GSI mechanisms.

The Condor-G agent also handles resubmission of failed jobs, communications with the user concerning unusual and erroneous conditions (e.g., credential expiry, discussed below), and the recording of computation on stable storage to support restart in the event of its failure.

We have structured the Condor-G agent implementation as depicted in Figure 1. The Scheduler responds to a user request to submit jobs destined to run on Grid resources by creating a new *GridManager* daemon to submit and manage those jobs. One GridManager process handles all jobs for a single user and terminates once all jobs are complete. Each GridManager job submission request (via the modified two-phase commit GRAM protocol) results in the creation of one Globus JobManager daemon. This daemon connects to the GridManager using GASS in order to transfer the job's executable and standard input files, and subsequently to provide real-time streaming of standard output and error. Next, the JobManager submits the jobs to the execution site's local scheduling system. Updates on job status are sent by the JobManager back to the GridManager, which then updates the Scheduler, where the job status is stored persistently as we describe below. When the job is started, a process environment variable points to a file containing the address/port (URL) of the listening GASS server in the GridManager process. If the address of the GASS server should change, perhaps because the submission machine was restarted, the GridManager requests the JobManager to update the file with the new address. This allows the job to continue file I/O after a crash recovery.

Condor-G is built to tolerate four types of failure: crash of the Globus JobManager, crash of the machine that manages the remote resource (the machine that hosts the GateKeeper and JobManager), crash of the machine on which the GridManager is executing (or crash of the the GridManager alone), and failures in the network connecting the two machines.

The GridManager detects remote failures by periodically probing the JobManagers of all the jobs it manages. If a JobManager fails to respond, the GridManager then probes the GateKeeper for that machine. If the GateKeeper responds, then the GridManager knows that the individual JobManager crashed. Otherwise, either the whole resource management machine crashed or there is a network failure (the GridManager cannot distinguish these two cases). If only the JobManager crashed, the GridManager attempts to start a new JobManager to resume watching the job. Otherwise, the GridManager waits until it can reestablish contact with the remote machine. When it does, it attempts to reconnect to the JobManager. This can fail for two reasons: the JobManager crashed (because the whole machine crashed), or the JobManager exited normally (because the job completed during a network failure). In either case, the GridManager starts a new JobManager, which will resume watching the job or tell the GridManager that the job has completed.

To protect against local failure, all relevant state for each submitted job is stored persistently in the scheduler's job queue. This persistent information allows the GridManager to recover from a local crash. When restarted, the GridManager reads the information and reconnects to any of the JobManagers that were running at the time of the crash. If a JobManager fails to respond, the GridManager starts a new JobManager to watch that job.

4.3. Credential management

A GSI proxy credential used by the Condor-G agent to authenticate with remote resources on the user's behalf is given a finite lifetime so as to limit the negative consequences of its capture by an adversary. A long-lived Condor-G computation must be able to deal with credential expiration. The Condor-G agent addresses this requirement by periodically analyzing the credentials for all users with currently queued jobs. (GSI provides query functions that support this analysis.) If a user's credentials have expired or are about to expire, the agent places the job in a hold state in its queue and sends the user an e-mail message explaining that their job cannot run again until their credentials are refreshed by using a simple tool. Condor-G also allows credential alarms to be set. For instance, it can be configured to e-mail a reminder when less than a specified time remains before a credential expires.

Credentials may have been forwarded to a remote location, in which case the remote credentials need to be refreshed as well. At the start of a job, the Condor-G agent forwards the user's proxy certificate from the submission machine to the remote GRAM server. When an expired proxy is refreshed, Condor-G not only needs to refresh the certificate on the local (submit) side of the connection, but it also needs to re-forward the refreshed proxy to the remote GRAM server.

To reduce user hassle in dealing with expired credentials, Condor-G could be enhanced to work with a system like MyProxy [23]. MyProxy lets a user store a long-lived proxy credential (e.g. a week) on a secure server. Remote services acting on behalf of the user can then obtain short-lived proxies (e.g. 12 hours) from the server. Condor-G could use these short-lived proxies to authenticate with and forward to remote resources and refresh them automatically from the MyProxy server when they expire. This limits the exposure of the long-lived proxy (only the MyProxy server and Condor-G have access to it).

4.4. Resource discovery and scheduling

We have not yet addressed the critical question of how the Condor-G agent determines where to execute user jobs. A number of strategies are possible.

A simple approach, which we used in the initial Condor-G implementation, is to employ a user-supplied list of GRAM servers. This approach is a good starting point for further development.

A more sophisticated approach is to construct a personal *resource broker* that runs as part of the Condor-G agent and combines information about user authorization, application requirements and resource status (obtained from MDS) to build a list of candidate resources. These resources will be queried to determine their current status, and jobs will be submitted to appropriate resources depending on the results of these queries. Available resources can be ranked by user preferences such as allocation cost and expected start or completion time. One promising approach to constructing

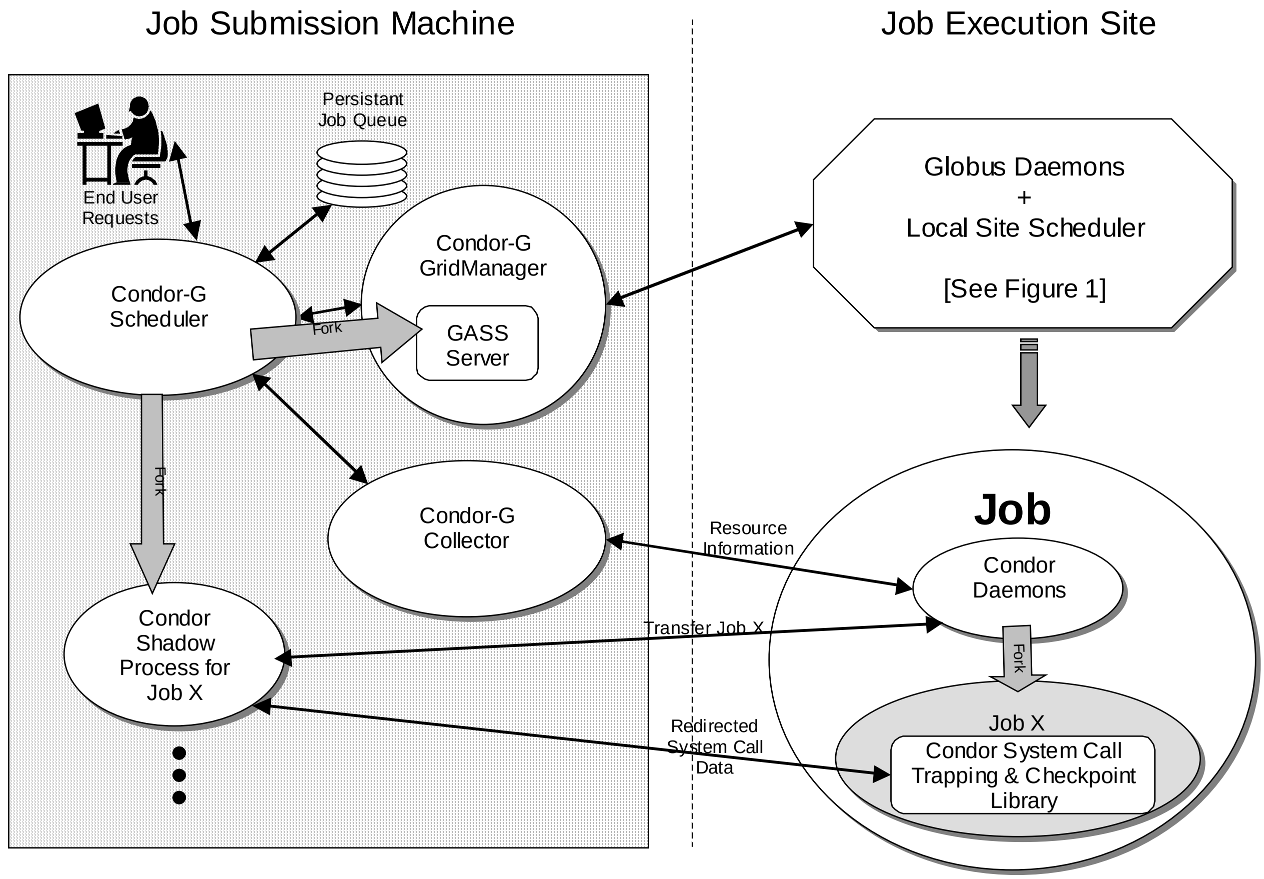

Figure 2. Remote job execution via GlideIn

such a resource broker is to use the Condor Matchmaking framework [25] to implement the brokering algorithm. Such an approach is described by Vazhkudai et al. [28]. They gather information from MDS servers about Grid storage resources, format that information and user storage requests into ClassAds, and then use the Matchmaker to make brokering decisions. A similar approach could be taken for computational resources for use with Condor-G.

In the case of high throughput computations, a simple but effective technique is to "flood" candidate resources with requests to execute jobs. These can be the actual jobs submitted by the user or Condor "GlideIns" as discussed below. Monitoring of actual queuing and execution times allows for the tuning of where to submit subsequent jobs and to migrate queued jobs.

5. GlideIn mechanism

The techniques described above allow a user to construct, submit, and monitor the execution of a task graph, with failures and credential expirations handled seamlessly and appropriately. The result is a powerful management tool for Grid computations. However, we still have not addressed issues relating to what happens when a job executes on a remote platform where required files are not available and local policy may not permit access to local file systems. Local policy may also impose restrictions on the running time of the job, which may prove inadequate for the job to complete. These additional system and site policy heterogeneities can represent substantial barriers.

We address these concerns via what we call *mobile sandboxing*. In brief, we use the mechanisms described above to start on a remote computer not a user job, but a daemon process that performs the following functions:

- It uses standard Condor mechanisms to advertise its availability to a Condor Collector process, which is queried by the Scheduler to learn about available resources. Condor-G uses standard Condor mechanisms to match locally queued jobs with the resources advertised by these daemons and to remotely execute them on these resources [25].
- It runs each user task received in a "sandbox," using system call trapping technologies provided by the Condor system [20] to redirect system calls

issued by the task back to the originating system. In the process, this both increases portability and protects the local system.

- It periodically checkpoints the job to another location (e.g., the originating location or a local checkpoint server) and migrates the job to another location if requested to do so (for example, when a resource is required for another purpose or the remote allocation expires) [21].

These various functions are precisely those provided by the daemon process that is run on any computer participating in a Condor pool. The difference is that in Condor-G, these daemon processes are started not by the user, but by using the GRAM remote job submission protocol. In effect, the Condor-G GlideIn mechanism uses Grid protocols to dynamically create a personal Condor pool out of Grid resources by "gliding-in" Condor daemons to the remote resource. Daemons shut down gracefully when their local allocation expires or when they do not receive any jobs to execute after a (configurable) amount of time, thus guarding against runaway daemons. Our implementation of this "GlideIn" capability submits an initial GlideIn executable (a portable shell script), which in turn uses GSI-authenticated GridFTP to retrieve the Condor executables from a central repository, hence avoiding a need for individual users to store binaries for all potential architectures on their local machines.

Another advantage of using GlideIns is that they allow the Condor-G agent to delay the binding of an application to a resource until the instant when the remote resource manager decides to allocate the resource(s) to the user. By doing so, the agent minimizes queuing delays by preventing a job from waiting at one remote resource while another resource capable of serving the job is available. By submitting GlideIns to *all* remote resources capable of serving a job, Condor-G can guarantee optimal queuing times to its users. One can view the GlideIn as an empty shell script submitted to a queuing system that can be populated once it is allocated the requested resources.

6. Experiences

Three very different examples illustrate the range and scale of application that we have already encountered for Condor-G technology.

An early version of Condor-G was used by a team of four mathematicians from Argonne National Laboratory, Northwestern University, and University of Iowa to harness the power of over 2,500 CPUs at 10 sites (eight Condor pools, one Cluster managed by PBS, and one supercomputer managed by LSF) to solve a very large optimization problem [3]. In less than a week the team logged over 95,000 CPU hours to solve more than 540 billion Linear Assignment Problems controlled by a sophisticated branch and bound algorithm. This computation used an average of 653 CPUs during that week, with a maximum of 1007 in use at any one time. Each worker in this Master-Worker application was implemented as an independent Condor job that used Remote I/O services to communicate with the Master.

A group at Caltech that is part of the CMS Energy Physics collaboration has been using Condor-G to perform large-scale distributed simulation and reconstruction of high-energy physics events. A two-node Directed Acyclic Graph (DAG) of jobs submitted to a Condor-G agent at Caltech triggers 100 simulation jobs on the Condor pool at the University of Wisconsin. Each of these jobs generates 500 events. The execution of these jobs is also controlled by a DAG that makes sure that local disk buffers do not overflow and that all events produced are transferred via GridFTP to a data repository at NCSA. Once all simulation jobs terminate and all data is shipped to the repository, the Condor-G agent at Caltech submits a subsequent reconstruction job to the PBS system that manages the reconstruction cluster at NCSA.

Condor-G has also been used in the GridGaussian project at NCSA to prototype a portal for running Gaussian98 jobs on Grid resources. This Portal uses GlideIns to optimize access to remote resources and employs a shared Mass Storage System (MSS) to store input and output data. Users of the portal have two requirements for managing the output of their Gaussian jobs. First, the output should be reliably stored at MSS when the job completes. Second, the users should be able to view the output as it is produced. These requirements are addressed by a utility program called G-Cat that monitors the output file and sends updates to MSS as partial file chunks. G-Cat hides network performance variations from Gaussian by using local scratch storage as a buffer for Gaussian's output, rather than sending the output directly over the network. Users can view the output as it is received at MSS using a standard FTP client or by running a script that retrieves the file chunks from MSS and assembles them for viewing.

7. Related work

The management of batch jobs within a single distributed system or domain has been addressed by many research and commercial systems, notably Condor [20], DQS [17], LSF [29], LoadLeveler [16], and PBS [15]. Some of these systems were extended with restrictive and ad hoc capabilities for routing jobs submitted in one domain to a queue in a different domain. In all cases, both domains must run the same resource management software. With the exception of Condor, they all use a resource allocation framework that is based on a system-wide collection of queues–each representing a different class of service.

Condor flocking [11] supports multi-domain computation management by using multiple Condor flocks to exchange load. The major difference between Condor flocking and Condor-G is that Condor-G allows inter-domain operation on remote resources that require authentication, and uses standard protocols that provide access to resources controlled by other resource management systems, rather than the special-purpose sharing mechanisms of Condor.

Recently, various research and commercial groups have developed software tools that support the harnessing of idle computers for specific computations, via the use of simple remote execution agents (workers) that, once installed on a computer, can download problems (or, in some cases, Java applications) from a central location and run them when local resources are available (i.e. SETI@home [19], Entropia, and Parabon). These tools assume a homogeneous environment where all resource management services are provided by their own system. Furthermore, a single master (i.e., a single submission point) controls the distribution of work amongst all available worker agents. Application-level scheduling techniques [5, 6] provide "personalized" policies for acquiring and managing collections of heterogeneous resources. These systems employ resource management services provided by batch systems to make the resources available to the application and to place elements of the application on these resources. An application-level scheduler for high-throughput scheduling that takes data locality information into account in interesting ways has been constructed [8]. Condor-G mechanisms complement this work by addressing issues of uniform remote access, failure, credential expiry, etc. Condor-G could potentially be used as a backend for an application-level scheduling system.

Nimrod [2] provides a user interface for describing "parameter sweep" problems, with the resulting independent jobs being submitted to a resource management system; Nimrod-G [1] generalizes Nimrod to use Globus mechanisms to support access to remote resources. Condor-G addresses issues of failure, credential expiry, and interjob dependencies that are not addressed by Nimrod or Nimrod-G.

8. Acknowledgment

This research was supported by the NASA Information Power Grid program.

9. References

[1] Abramson, D., Giddy, J., and Kotler, L., "High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid?", *IPDPS'2000*, IEEE Press, 2000.

[2] Abramson, D., Sosic, R., Giddy, J., and Hall, B., "Nimrod: A Tool for Performing Parameterized Simulations Using Distributed Workstations", *Proc. 4th IEEE Symp. on High Performance Distributed Computing*, 1995.

[3] Anstreicher, K., Brixius, N., Goux, J.-P., and Linderoth, J., "Solving Large Quadratic Assignment Problems on Computational Grids", *Mathematical Programming*, 2000.

[4] Beiriger, J., Johnson, W., Bivens, H., Humphreys, S., and Rhea, R., "Constructing the ASCI Grid", *Proc. 9th IEEE Symposium on High Performance Distributed Computing*, IEEE Press, 2000.

[5] Berman, F., "High-Performance Schedulers", Foster, I. and Kesselman, C. eds., *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, 1999, pp. 279-309.

[6] Berman, F., Wolski, R., Figueira, S., Schopf, J., and Shao, G., "Application-Level Scheduling on Distributed Heterogeneous Networks", *Proc. Supercomputing '96*, 1996.

[7] Bester, J., Foster, I., Kesselman, C., Tedesco, J., and Tuecke, S., "GASS: A Data Movement and Access Service for Wide Area Computing Systems", *Sixth Workshop on I/O in Parallel and Distributed Systems*, May 5, 1999.

[8] Casanova, H., Obertelli, G., Berman, F., and Wolski, R., "The AppLeS Parameter Sweep Template: User-Level Middleware for the Grid", *Proc. SC'2000*, 2000.

[9] Czajkowski, K., Fitzgerald, S., Foster, I., and Kesselman, C., "Grid Information Services for Distributed Resource Sharing", 2001.

[10] Czajkowski, K., Foster, I., Karonis, N., Kesselman, C., Martin, S., Smith, W., Tuecke, S., "A Resource Management Architecture for Metacomputing Systems", *Proc. IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing*, 1998.

[11] Epema, D.H.J., Livny, M., Dantzig, R.v., Evers, X., and Pruyne, J., "A Worldwide Flock of Condors: Load Sharing among Workstation Clusters", *Future Generation Computer Systems, 12*, 1996.

[12] Foster, I. and Kesselman, C., "Globus: A Toolkit-Based Grid Architecture", Foster, I. and Kesselman, C. eds., *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, 1999, pp. 259-278.

[13] Foster, I., Kesselman, C., Tsudik, G., and Tuecke, S., "A Security Architecture for Computational Grids", *ACM Conference on Computers and Security*, 1998, pp. 83-91.

[14] Foster, I., Kesselman, C., and Tuecke, S., "The Anatomy of the Grid: Enabling Scalable Virtual Organizations", *Intl. J. Supercomputer Applications, (to appear)*, 2001. http://www.globus.org/research/papers/anatomy.pdf.

[15] Henderson, R. and Tweten, D., "Portable Batch System: External Reference Specification", 1996.

[16] IBM, "Using and Administering IBM LoadLeveler, Release 3.0", IBM CorporationSC23-3989, 1996.

[17] Institute, S.C.R., "DQS 3.1.3 User Guide", Florida State University, Tallahassee, 1996.

[18] Johnston, W.E., Gannon, D., and Nitzberg, B., "Grids as Production Computing Environments: The Engineering Aspects of NASA's Information Power Grid", *Proc. 8th IEEE Symposium on High Performance Distributed Computing*, IEEE Press, 1999.

[19] Korpela, E., Werthimer, D., Anderson, D., Cobb, J., and Lebofsky, M., "SETI@home: Massively Distributed Computing for SETI", *Computing in Science and Engineering*, *3*(1), 2001.

[20] Litzkow, M., Livny, M., and Mutka, M., "Condor - A Hunter of Idle Workstations", *Proc. 8th Intl Conf. on Distributed Computing Systems*, 1988, pp. 104-111.

[21] Litzkow, M., Tannenbaum, T., Basney, J., and Livny., M., "Checkpoint and Migration of UNIX Processes in the Condor Distributed Processing System", University of Wisconsin-Madison Computer Sciences, Technical Report 1346, 1997.

[22] Livny, M., "High-Throughput Resource Management", Foster, I. and Kesselman, C. eds., *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, 1999, pp. 311-337.

[23] Novotny, J., Tuecke, S., and Welch, V., "An Online Credential Repository for the Grid: MyProxy", *to appear in HPDC10*.

[24] Papakhian, M., "Comparing Job-Management Systems: The User's Perspective", *IEEE Computational Science & Engineering*, (April-June) 1998. See also http://pbs.mrj.com.

[25] Raman, R., Livny, M., and Solomon, M., "Resource Management through Multilateral Matchmaking", *Proceedings of the Ninth IEEE Symposium on High Performance Distributed Computing (HPDC9)*, Pittsburgh, Pennsylvania, August 2000, pp. 290-291.

[26] Steiner, J., Neuman, B.C., and Schiller, J., "Kerberos: An Authentication System for Open Network Systems", *Proc. Usenix Conference*, 1988, pp. 191-202.

[27] Stevens, R., Woodward, P., DeFanti, T., and Catlett, C., "From the I-WAY to the National Technology Grid", *Communications of the ACM, 40*(11), 1997, pp. 50-61.

[28] Vazhkudai, S., Tuecke, S., and Foster, I., "Replica Selection in the Globus Data Grid", *Proc. Of the First IEEE/ACM International Conference on Cluster Computing and the Grid (CCGRID 2001)*, IEEE Computer Society Press, May 2001, pp. 106-113.

[29] Zhou, S., "LSF: Load Sharing in Large-Scale Heterogeneous Distributed Systems", *Proc. Workshop on Cluster Computing*, 1992.