# IMAGE: An approach to building standards-based enterprise Grids

Gabriel Mateescu[1] and Masha Sosonkina[2]

[1]Research Computing Support Group
National Research Council
Ottawa, ON K1A 0R6, Canada
gabriel.mateescu@nrc.gc.ca

[2]Scalable Computing Laboratory
USDOE Ames Laboratory
Ames, IA 50011, USA
masha@scl.ameslab.gov

## Abstract

*We describe a system for aggregating heterogeneous resources from distinct administrative domains into an enterprise-wide compute grid, such that the aggregated resource provides the services of reliable and flexible queuing, scheduling, execution, and monitoring of batch applications. The system provides scheduling across multiple cluster Grids, user account mapping across domains, and file staging, thereby enabling the consolidation of organization-wide distributed resources into a virtual resource, while preserving local control of resources. The concept of abstract queue, as the unit of aggregating heterogeneous resources, is introduced and instantiated for distributed resource scheduling. The proposed system is an open source, standards-based alternative to similar commercial systems.*

## 1 Introduction

A *compute Grid* is a set of hardware, software and human resources that are used to obtain computational results by running batch applications (or *jobs*) and moving related data. Compute Grids can be divided into four categories [20, 21], based on their geographic and organizational scope and on resource ownership (multiple ownership includes multiple administrative domains and groups of users): *cluster Grids*, *campus Grids*, *enterprise Grids*, and *global Grids*. A cluster Grid (also called *department Grid*) contains resources located at one site within one organization, and belonging to a single owner. A campus Grid differs from a cluster Grid in that its resources belong to multiple owners. Unlike campus Grids, enterprise Grids contain resources located at multiple sites. Finally, global Grids contain resources from multiple organizations.

Distributed Resource Management (DRMS) middleware controls access to, and manages, the resources. DRMS for cluster Grids include PBS Professional [3], Torque and MOAB [5] and Sun N1 Grid Engine [22]. The Globus Toolkit [9, 10] provides services such a job execution, data management, and resource discovery for global Grids. While global Grids have been pioneered by the academic and research communities (e.g., TeraGrid [18] and CoreGrid [6]), enterprise Grids are offered mostly as commercial products ([17, 22]). Thus, to provide a foundation for global Grids, there is a need for open-source enterprise grids solutions.

We describe a system for aggregating heterogeneous resources from distinct administrative domains into an enterprise compute grid, such that the aggregated resource provides the services of reliable and flexible queuing, scheduling, execution, and monitoring of batch jobs. The system presents the client with the image of a cohesive collection of resources, which is uniformly accessed through a single interface, in a location-transparent manner. The system integrates distributed resources managed by a local DRMS, preserving the local control over those resources and adding the benefits of controlled resource sharing across an organization.

The main contributions of this work are: (i) The concept of abstract queue as the unit of aggregating heterogeneous resources; (ii) An effective job queuing and scheduling solution that supports heterogeneous resources located under multiple administrative domains; (iii) An account mapping scheme for authorization across domains. The proposed approach is a step toward creating an environment in which the users can say: "I want to run a job for which I have either the source code of the application or the executable(s) built for one or more platforms. I know the input files needed by the application, but I do not (nor do I want to) know the name of machine that can run the binary,

1-4244-0054-6/06/$20.00 

how much memory or how many CPUs it needs. All I want is to get the job executed as quickly as possible."

Aggregating independently managed resource into an enterprise compute grid provides benefits that include (i) *scalability:* scale out capacity by aggregating individual clusters, preserving the capability of the individual resources and sharing the workload across multiple resources; (ii) *flexibility:* support incremental growth as new resources are added across the organization; (iii) *location transparency:* access enterprise-wide resources regardless of their location; (iv) *preservation of local control:* resources are locally owned and controlled, e.g., access policies are defined locally.

The paper is organized as follows: in Section 2 we present the proposed system, then describe the file staging features in Section 3, the authorization features in Section 4, and the resource brokering features in Section 5. In Section 6 we describe the system's fault tolerance. The most relevant related work is surveyed in Section 7, followed by an evaluation of the system in Section 8 and by concluding remarks in Section 9.

## 2 The proposed system

In this section, we present the concepts on which our resource virtualization solution is based, and describe the proposed system. The system aggregates local resources, where each resource is managed by a DRMS which implements the interface specified by the POSIX standard IEEE 1003.2d-1994 Batch Environment [15, 16], which specifies the interface to a distributed batch queuing environment. This standard has been enacted in 1994 and its updated version is currently incorporated into the IEEE standard 1003.1-2004. Henceforth, we call this standard the *POSIX Batch Interface*.

The resources being aggregated can be computers or clusters, located across an organization, that are locally owned and controlled (via local usage policies), and can be heterogeneous (different processor type, system and application software). We call the aggregate resource, along with the services of access control, job submission, scheduling and control, a *virtual cluster*, and call our approach to building a virtual cluster *Integral Management and Allocation of Grid Elements* (IMAGE). IMAGE provides the following services:

(1) consolidation of multiple clusters into a single virtual cluster whose resources are the union of the resources associated with each cluster;

(2) authentication, authorization and account mapping across multiple domains;

(3) resource brokering and monitoring: selecting a local resource that meets the job's requirements, adhering to local policies of the individual resources, and optimizing the global resource usage;

(4) file staging between the execution machine and the submission machine.

### 2.1 IMAGE Architecture

Here we present the components of an IMAGE system, their functions and relationships, and how IMAGE interfaces to the Globus Toolkit [10, 11] job service.

An IMAGE system consists of four components: (1) one master server; (2) one or more slave servers; (3) one or more job execution machines under the control of each slave server; (4) one or more clients that communicate with the master server.

The master server runs the *virtual cluster service* (VCS) that aggregates the physical clusters. For high availability, multiple instances of VCS may be run (see Section 6). A slave server is a local DRMS that controls either a cluster or a network of computers (the execution machines). The local DRMS together with the controlled machines constitute a *Grid Element*. The clients are either users or software tools that communicate with the master server to submit, query, and control jobs. The IMAGE architecture is depicted in Fig. 1. The VCS implements the POSIX batch interface. Every DRMS also implements the POSIX batch interface. DRMS that implement this interface include PBS Professional [3], OpenPBS, Torque, and MOAB [5]. Different DRMS can be located in distinct domains, as long as there is some trust relation-

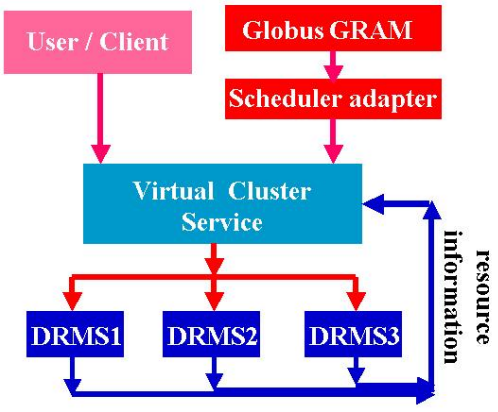


**Figure 1. The architecture of IMAGE**

ship, which is needed for granting VCS users access to the DRMS. Clients can access the VCS either directly, through the POSIX batch interface, or through the Globus GRAM interface in conjunction with an interface adapter (see also Section 2.3). The POSIX command line interface includes commands to submit jobs (*qsub*), get the job status (*qstat*), manage the job (*qalter* and *qsig*), and cancel the job (*qdel*).

The VCS consists of two components: *VCS server* and *VCS scheduler*. The VCS server keeps track of the jobs in persistent queues, assigns to each job a unique identifier, denoted by *$PBS_JOBID*, routes the jobs among the VCS queues and the DRMS queues, performs access control and job monitoring; The VCS scheduler selects the appropriate resource for each job and instructs the VCS server to route the job that resource, as described in Section 5.1.

## 2.2 Abstract queues

Resource management systems may contain two types of queues: *execution queues*, out of which jobs are dispatched to one or more execution hosts, and *routing queues*, out of which job are dispatched to other queues. Routing queues can be used to route jobs to queues located either in the same resource management system or in another system.

The unit of aggregating resources in IMAGE is an *abstract queue*. An abstract queue represents a collection of resources (CPU, memory, software licenses), along with a persistent queue of jobs using – or waiting to use – those resources, and a set policies governing the use of the resources by the jobs. The policies may include access control list of authorized users, scheduling policies (e.g., job sort-order), as well as job-level, user-level, and queue-level limits on the amount of resources that can be assigned to a job, user, or queue.

An abstract queue differs from a concrete queue in that there can be several concrete queues (routing or execution queues) implementing a single abstract queue. For example, in IMAGE an abstract queue is implemented by two concrete queues: (i) a local queue defined at the DRMS level, out of which jobs are dispatched to the hosts managed by the DRMS; (ii) a routing queue defined at the VCS level, through which jobs are routed to the corresponding local queue in the DRMS; this routing queue acts as a *stub queue* for the DRMS queue. Each grid element can be thought of as a collection of queues, where each queue has its access control and resource policies. Thus, the virtual cluster is the union of all the abstract queues, along with the aggregation services mentioned above.

While other work on grid scheduling considers queue hierarchies, e.g., in [19] a network of *grid queues* and *local queues* is considered, the abstract queue is a novel concept which allows to have a logical queue divided into several concrete queues. In IMAGE, the abstract queue is divided into a part located in the VCS (the stub queue) and one located in the DRMS (the local queue). Having the VCS scheduler dispatch jobs to the stub queue rather than to the DRMS execution queue has benefits for reliability and performance: sending a job from the VCS to a DRMS involves network communication, while the stub queue is local to the system running the VCS scheduler. By communicating with the stub queue rather than the DRMS local queue, the VCS scheduler takes a shorter time to dispatch jobs and is insulated from transient communication problems that may occur between the VCS and DRMS. The stub queue acts like a "buffer" for the DRMS local queue. The abstract queue enables efficient and reliable scheduling and does not affect the scheduling decisions. The VCS scheduler makes decisions as to what local queue to dispatch the job, but sends the job to the stub queue rather than the DRMS local queue.

Resource properties, resource usage, and scheduling policies are defined for the abstract queues. The VCS server maintains the input queue of jobs. The VCS scheduler fetches jobs from the input queue and makes scheduling decisions in terms of abstract queues, then dispatches each job to the VCS stub queue corresponding to the abstract queue; the stub queue then routes the job to the DRMS local queue.

Consider the example given in Fig. 2. Two DRMS, called DRMS1 and DRMS2, have two queues called Qa and Qb, and these queues are accepting jobs from the VCS. The VCS is then configured such that the routing

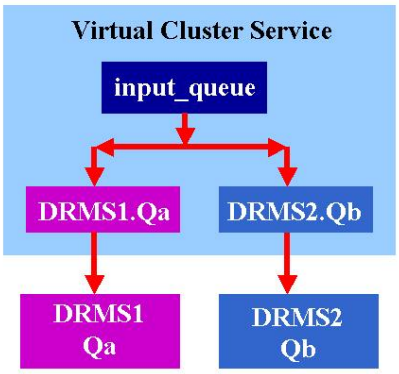


**Figure 2. Job routing through queues**

queue *input_queue* has a destination for each DRMS. In Fig. 2, there are two destination queues, DRMS1.Qa and DRMS2.Qb, which are in turn routing queues whose destinations are respectively Qa in DRMS1 and Qb in DRMS2. The VCS scheduler routes jobs to either DRMS1.Qa or DRMS2.Qb, then DRMS1.Qa routes the jobs to Qa and DRMS2.Qb to Qb. The abstract queue Qa is thus realized by way of the local queue Qa in DRMS1 and the stub queue DRMS1.Qa in the VCS.

The VCS provides the following functionality:

- maintains a persistent queue of the jobs submitted to the collection of managed clusters;
- tracks the status of the jobs across the entire job life cycle (from pending to running and to the exit status);
- monitors the status of the grid elements and fetches their access policy;
- schedules jobs to run on a grid element that meets the job requirements; scheduling is done such that it obeys global policies (defined at VCS level) as well as local policies;
- stages files between the VCS machine and the execution hosts;
- maps the *job owner* account to an appropriate *effective user* account on the grid element.

### 2.3 Integration with the Globus Toolkit

IMAGE can be integrated in a Grid environment by presenting it as a job execution service for the aggregated resources. For example, IMAGE is easily integrated with the Globus Toolkit's Grid Resource Allocation and Management [8] component (GRAM), either the pre-web services and the web-services version [14] of GRAM. This is done using a *scheduler adapter* module that converts the GRAM protocol to the POSIX Batch interface (see Fig. 1). The module provides the functions of submitting, querying, and deleting a job.

## 3 File staging

File staging occurs along two paths: (i) between the client machine and VCS; (ii) between VCS and grid element. For the first path one can use file transfer frameworks such as GridFTP [2, 1], while for the second path (which traverses a LAN) one can use SSH-based file copying (*scp*), or a distributed file system that includes the VCS and the local resource. File transfer frameworks that use multi-stream protocols (such as GridFTP) do not necessarily offer significantly superior performance in a LAN environment, since the LAN bandwidth does not grow linearly with the number of streams ([4]). Therefore, we use the *scp* tool for staging between the VCS and the Grid Elements.

### 3.1 Specifying staging

File staging is specified by *VCS directives* included in the job submission file. The directives specify the files to be staged between VCS and the grid element. For example, assume that a job with the identifier *$PBS_JOBID* is owned by *user1* who is mapped to the effective user *grid1*, and the job has the input files in the *in* subdirectory and creates the result files in the *out* subdirectory. The following directive describes the file staging for this job:

```
#VCS -v  VCS_STAGING=1,
     VCS_JOB_DIR=/home/user1/in,
     VCS_STAGEOUT=out, VCS_JOB_OUT=../
```

VCS translates this directive into DRMS-specific directives that perform two actions: (i) file staging from the VCS directory /home/user1/in to the grid element directory $HOME/$PBS_JOBID/in, where $HOME is the home directory of *grid1*; (ii) file staging from the grid element directory $HOME/$PBS_JOBID/out to the VCS directory /home/user1/out. These directives are inserted by the scheduler adapter for jobs submitted via Globus GRAM.

## 4 Authorization and account mapping

IMAGE supports multi-domain authorization, whereby each local DRMS defines an access list for the queue accessible from the the VCS. The VCS defines its own list of authorized users, which is related to the Globus Toolkit *grid map-file*. In other words, each Grid user has her/his Grid identity mapped to an account on the VCS, by way of the grid map-file, such that every account defined by this mapping is also authorized by the VCS access list. Users from other organizations are required to present a Grid credential, while users from the same organization have the option of authenticating based on user name and password. VCS manages for each job the attributes: (i) *owner*: the user account on the VCS who submits the job, (ii) *effective user*: the user account under which the job runs on the grid element, as determined by the VCS and the DRMS access control policy. Unlike the owner, the effective user depends on the grid element selected for the job. IMAGE determines the effective user in accordance with the DRMS policy, as follows:

- for each job owner, VCS maintains a map like this:

  $Map(user1) = ($
  $grid1@DRMS1.Qa, grid2@DRMS2.Qb, \ldots).$

  indicating that owner *user1* may run on queue Qa of DRMS1 as user *grid1* and on queue Qb of DRMS2 as *grid2*.

- Every DRMS maintains for each queue a list of authorized users, e.g., DRMS1 maintains for the queue Qa a list like this:

  $$ACL(DRMS1.Qa) = (user1@VCS, ...).$$

- VCS scans the entries in $Map$ and looks up *user1* in the corresponding $ACL$ lists, until a DRMS and queue containing *user1* is found. VCS then sets the *effective user* attribute of the job. For the above example, VCS sees *grid1@DRMS1.Qa* in *Map* then finds *user1* in *ACL(DRMS1.Qa)*, so it maps *user1* to *grid1@DRMS1.Qa*.

With this scheme, users can be added to VCS without having to reconfigure the local DRMS.

Data integrity and privacy are supported using three techniques: (i) For some job owners, VCS makes sure that the effective user to whom these owners are mapped is not shared with another job owner; (ii) For the other job owners, for which the effective user can be shared by several owners, VCS makes sure that on every Grid Element there is only one owner at a time using the effective user account; (ii) For all owners, each job executes on the grid elements under a directory name derived from the unique job identifier *$PBS_JOBID* and the files staged for a job are removed at the end of the job from the grid element.

### 4.1 Authorization for file staging

File staging between the VCS machine and the execution machine is done with the secure copy tool *scp*. Due to the account mapping between the job owner on VCS and the effective user on the grid element, authorization needs to be granted to the effective user to *scp* to/from VCS. For example, if the job owner *user1* is mapped to the effective user *grid1* on the grid element GE1, then, in order to allow staging of job files between VCS and GE1, the file *authorized_keys* of *user1* on VCS contains the public key of user *grid1*. To protect the account *user1* on VCS, the VCS is configured such that that *grid1* cannot execute any command as *user1* except for *scp* to/from the directories allocated to the job running as *grid1*.

## 5 Resource brokering and monitoring

Resource brokering includes: (i) maintaining a persistent queue of the jobs; (ii) maintaining the state of the grid elements; (iii) scheduling jobs queued by the VCS. Next we describe these operations.

### 5.1 Job scheduling

Job scheduling maps jobs to resources. Scheduling is performed by VCS in cooperation with the local DRMS. When jobs enter IMAGE, they are placed in an *input queue* which is organized as a *priority queue*. The rank of a job in the priority queue is determined by several attributes, some of which are dynamic, including the priority of the job owner, the time the job has been pending in the queue, and the priority assigned to the job by the owner. Jobs whose pending time exceeds a configurable threshold are marked as *urgent jobs*. Job scheduling occurs as follows:

- The VCS scheduler sorts the jobs in priority order and schedules them in order of decreasing priority. If there are no urgent jobs, jobs of lower priority can backfill; otherwise, strict priority order is enforced.

- Resource selection: for each selected job, the VCS scheduler tries to find a suitable grid element, based on the job's requirements, the resource state, and the access control policy.

- If a grid element is found, the VCS scheduler maps the job owner to an effective user authorized by the grid element, as described in Section 4, and parses the job's file staging directives as described in Section 3.1.

- The VCS scheduler dispatches the job to the selected DRMS, by way of the stub queue on the VCS. The local DRMS, upon receiving the job, selects one or more execution hosts (based on the job's requirement) to which it maps the job, and runs the job.

- The VCS scheduler re-queues the jobs that fail because the resources to which they are dispatched become unavailable.

Resource selection is important in a Grid environment, where resources are heterogeneous and the scheduler must find which of the resources have the properties required by the job. In IMAGE, this issue is addressed as follows. Jobs have a *requires* attribute

which is a boolean expression whose terms define properties of the resource (similar to Condor classads [24]). For example,

$$requires = ((OpSys == \text{“}Linux''}) \;\&\&\; (Kernel => \text{“}2.6.5'') \;\&\&\; (CPU == \text{“}i686'') )$$

In case the user submits the source code of an application instead of the executable, *requires* specifies also compilers and other tools. In the resource selection step, the job scheduler evaluates the job's *requires* expression for the grid elements (which publish resource information with the VCS) and screens out those grid elements that do not satisfy it.

### 5.2 Resource Monitoring

The status of all grid elements is tracked by the VCS to determine the following: (i) the availability of the resources (nodes that are up and for which remote submission is enabled); (ii) the status of the jobs, the resources that are assigned to jobs, and those that are free for new jobs; (iii) the access control associated with the queues on each grid element. All jobs submitted to the VCS can be queried to determine the current status, the trace of the events underwent by the job, and the exit status (if the job has completed).

## 6 Fault tolerance

The Virtual Cluster Service is a mission-critical component of an enterprise Grid, thus its services must survive the occurrence hardware and software failures. The IMAGE system is designed to be fault-resilient. Specifically, the master server is implemented as a logical server consisting of multiple physical servers. Typically, the logical server is implemented using a primary server along with a secondary server, with failover from the primary to the secondary server, when the primary server stops responding to heartbeat queries. To ensure that the physical servers work in a mutually-exclusive mode, the heartbeat can use a redundant connection between the servers.

Clients tools can access a primary/secondary server in two ways: (i) by explicitly specifying the physical server name, corresponding to the primary and secondary sever; (ii) by using the *round-robin DNS* technique, which allows mapping the name of the logical server to two IP addresses, one for the primary server and one for the secondary server. With round-robin DNS, the DNS maps a host name to a list of IP addresses, whose order rotates with subsequent queries. Under method (ii), the client scans the list of IP addresses obtained from DNS until it finds one that it can connect to. In contrast, under method (i), the client needs to be enhanced to translate the logical server name into a list of physical server names (each with a single IP address) and traverse the list until an up-and-running server is found.

## 7 Related Work

In this section we survey related work on managing distributed resources. We consider four products or projects: Platform LSF multicluster, the Condor system, Sun N1 Grid Engine, and the GridWay project.

### 7.1 Platform LSF Multicluster

The Platform LSF Multicluster [17] is a mature, powerful and stable cluster aggregation solution. Unlike IMAGE, where there is a master server dispatching jobs to the local resource managers, LSF multicluster has a decentralized architecture, where every cluster can potentially communicate with every other cluster. While this allows for more balanced job submission, it creates management problems, since a change made to the configuration or policies on a cluster needs to be communicated to, or discovered by, all the other clusters. LSF supports user account mapping across clusters and failover of the scheduler component. It does not implement the POSIX batch interface, and support for parallel applications is distributed separately. Overall, LSF Multicluster is a robust solution for aggregating enterprise-wide resources, and IMAGE is an open source alternative to it.

### 7.2 Condor

The Condor [24] system manages resources within a single administrative domain, being focused toward high throughput computing and networks of computers that are not dedicated to running batch jobs. A set of machines is combined to form a pool, under the control of a *central manager*, and whose resources are matched by the central manager against the requirements of the job requests from clients. Condor allows to combine local access policies (defined by the owner of each machine) with global policies (defined in the central manager). Condor supports both serial and parallel jobs, using the concept of *universe* (which denotes a job execution environment) but requires a dedicated scheduler and dedicated resources for the latter. Condor-G [23] is a Condor universe that combines the intra-domain resource and job management capability of Condor with the inter-domain resource sharing and security capability of the Globus Toolkit. The main limitation of

Condor-G as a multi-domain resource management system is that the *negotiator*, the component of the central manager that selects a resource for a job, is stateless, which makes it extremely difficult to implement scheduling policies such as "before scheduling the next job to a particular resource, wait for that resource to update its state to reflect the reception of the previous job". In the absence of such a rule, it is likely that a resource will be either over- or under-subscribed.

### 7.3 Sun N1 Grid Engine

Sun N1 Grid Engine [22] is a scalable and robust distributed resource management system, supporting both high throughput and flexible scheduling policies. In addition, it supports the Distributed Resource Management Application API (DRMAA) [12], a proposed Global Grid Forum standard for integrating applications into a DRMS. However, the usage of the N1 Grid Engine in a multi-domain setting is hindered by two issues: (i) it does not support user account mapping; (ii) it has non-orthogonal support for parallel jobs. Because of (i), users must have the same account on all the hosts, which is problematic to achieve across administrative domains. Moreover, while it is possible to run parallel jobs under the N1 Grid Engine, this requires the administrator to define a *parallel environment* under which parallel jobs execute. This is unlike systems such as PBS Professional and Torque, which treat parallel jobs simply as jobs requiring more than one processor and do not require a special environment for parallel jobs. Treating parallel jobs as a special case limits the types of jobs that can use it.

### 7.4 GridWay

GridWay [13] is a user-level framework aimed at facilitating the execution of applications on Grids based on the Globus Toolkit. A *submission agent* manages the jobs on the submission machine: using resource information extracted from the Monitoring and Discovery Service (MDS) [7, 25], the agent selects a resource for each job and submits the job to the selected resource. Each application communicates its *requirements* (as required resource attributes) and its *preferences* about resources to the framework. The submission agent selects a resource by matching the application's requirements against the resource attributes.

Unfortunately, the GridWay framework relies on excessively strong assumptions about the applications being run, the information provided by MDS, and the version of Globus Toolkit installed on the resources. The application is assumed to generate performance profile data to help the submission agent with migrating the jobs when the performance on a resource degrades. GridWay also assumes that all the information needed by the submission agent to select a resource is available from Globus MDS. This is questionable, given that different sites may publish different information. Moreover, it assumes that the list of authorized users on a grid resource is available through MDS, which may not be consistent with the policy of the local resource.

GridWay is tightly integrated with Globus, and relies on components that have either been phased out, such as the GASS server (for file staging), or have been changed and are not backward-compatible, such as MDS (version 4.0 of Globus does not include LDAP-based MDS, only web-services-based based MDS). GridWay scheduling decisions are likely to be imprecise because it employs multiple submission agents that work in an uncoordinated manner creating the risk of conflicting decisions, e.g., two submission agents deciding to submit a job to the same resource.

## 8 Experimental Evaluation

We have evaluated the efficacy of IMAGE on a real-life simulation performed at the request of a computational biologist. The problem has been a simulation involving a genome-wide computation on an input containing 15 million lines of tagged DNA sequences. For each input line, the simulation generates an output line. Lines can be processed independently and processing a line takes about $T_0 = 1.1$ seconds on a resource GE1 with a queue $Qa$ with $P_1 = 64$ CPUs and $4/3T_0$ on a resource GE2 with a queue $Qb$ with $P_2 = 36$ CPUs. Running this program serially on GE1 will require $15 \times 10^6 \times 1.1$ seconds $\approx 191$ days.

We have reduced the time from 191 days to 55.5 hours using two techniques. First, we have divided the input into chunks of $n = 10,000$ lines, and defined a job for each chunk, so there are $N = 1,500$ jobs and each job takes the time $t_0$ on GE1 and $4/3t_0$ on GE2, where $t_0 = n \times T_0 = 10,000 \times 1.1 = 11,000$ seconds i.e., 3.05 hours. Had we solved the problem only on GE1, the time would have been $t_0 \times (N/P_1) = 71.5$ hours.

The second technique for reducing the execution time has been to use IMAGE to run the jobs concurrently on GE1 and GE2. Assuming perfect load balancing, and letting $N_1$ and $N_2$ be respectively the number of jobs executed by GE1 and GE2, thus $N = N_1 + N_2$, we obtain the execution time $T$ on GE1 and GE2:

$$T = \max\left\{t_0 \times \frac{N_1}{P_1}, \frac{4t_0}{3} \times \frac{N_2}{P_2}\right\}.$$

The optimum time $T_{min}$ is reached when $t_0 \times N_1/P_1$

equals $4/3t_0 \times (N - N_1)/P_2$. Therefore,

$$T_{min} = \frac{4\,N\,t_0}{4P_1 + 3P_2} = \frac{4 \times 1,500 \times 3.05}{4 \times 64 + 3 \times 36} = 50.27 \text{ hours}$$

The time measured for IMAGE has been 55.5 hours, the 10% difference with respect to the ideal time being due to the combined contribution of the time to stage the files, the time to rerun some jobs due to resource failure, imperfect load balancing, the VCS server delay in sensing the resource status, and the time taken by the VCS scheduler.

## 9 Conclusions

We have presented a system for aggregating heterogeneous clusters from multiple administrative domains into a virtual cluster. Attributes of IMAGE include scale, safety, small footprint, and reliability. Resource scale-out stems from scheduling applications across resources from separate administrative domains; Safety is due to the lack of privilege-elevation vulnerabilities (e.g., set-uid programs) and running the VCS scheduler as a non-privileged user. The failover capability and the persistent state of the VCS server provide reliable service. In addition, IMAGE is a standards-based solution, which allows it to work with multiple DRMS.

## References


[1] W. Allcock. GridFTP Protocol Specification. http://www.gridforum.org/documents/GFD.20.pdf.

[2] B. 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*, 28(5):749–771, 2002.

[3] Altair Engineering. PBS Professional Overview. http://www.altair.com/software/pbspro.htm.

[4] H. Casanova. Modeling large-scale platforms for the analysis and the simulation of scheduling strategies. In *Procs. 6th Workshop on Advances in Parallel and Distributed Computational Models*, 2004.

[5] Cluster Resources. MOAB and Torque Documentation. http://clusterresources.com.

[6] CoreGRID. European Network of Excellence. http://www.coregrid.net.

[7] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid Information Services for Distributed Resource Sharing. In *Procs. 10th IEEE Symp. on High-Performance Distributed Computing (HPDC-10)*. IEEE Press, Aug. 2001.

[8] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. A Resource Management Architecture for Metacomputing Systems. In D. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, volume 1459 of *Lecture Notes in Computer Science*, pages 62–82. Springer-Verlag, 1998.

[9] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, Summer 1997.

[10] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the Grid: enabling scalable virtual organizations. *Intl. Journal of High Performace Computing Applications*, 15(3):200–222, 2001.

[11] Globus Alliance. Globus Toolkit Documentation. http://www.globus.org, 2006.

[12] H. Rajic, (ed.). Distributed Resource Management Application API Specification 1.0. *Global Grid Forum Recommendation GFD.022*, June 2004.

[13] E. Huedo, R. S. Montero, and I. M. Llorente. A framework for adaptive execution in grids. *Software - Practice and Experience*, 34(7):631–651, 2004.

[14] IBM. The WS-Resource Framework. http://www-128.ibm.com/developerworks/webservices/library/specification/ws-resource/ws-wsrfpaper.html, 2006.

[15] IEEE. IEEE 1003.2d-1994 Standard for Information Technology. http://standards.ieee.org/reading/ieee/std_public/description/posix/1003.2d-1994_desc.html.

[16] IEEE. Portable Operating System Interface (POSIX) Standards. http://standards.ieee.org/ catalog/olis/arch_posix.html, 2003.

[17] Platform Computing. Using Platform LSF MultiCluster. http://www.platform.com/products/LSF/, 2006.

[18] D. A. Reed. Grids, the TeraGrid, and Beyond. *Computer*, 36(1):62–68, 2003.

[19] H. Shan, L. Oliker, and R. Biswas. Job superscheduler architecture and performance in computational grid environments. In *Supercomputing 2003 Conference (SC'03), November 15-21, Phoenix USA*, 2003.

[20] Sun Microsystems. Sun Powers the Grid, 2003.

[21] Sun Microsystems. Understanding datagrid architecture for higher education. http://www.sun.com/products-n-solutions/edu/whitepapers/pdf/datagrid.pdf, February 2005.

[22] Sun Microsystems. Sun N1 Grid Engine. http://www.sun.com/grid, 2006.

[23] D. Thain, T. Tannenbaum, and M. Livny. Condor and the Grid. In F. Berman, G. Fox, and T. Hey, editors, *Grid Computing: Making the Global Infrastructure a Reality*. John Wiley & Sons Inc., 2003.

[24] D. Thain, T. Tannenbaum, and M. Livny. Distributed Computing in Practice: The Condor Experience. *Concurrency and Computation: Practice and Experience*, 17(2–4):323–356, 2005.

[25] G. von Laszewski, S. Fitzgerald, I. Foster, C. Kesselman, W. Smith, and S. Tuecke. A Directory Service for Configuring High-Performance Distributed Computations. In *Proceedings of the 6th IEEE Symposium on High-Performance Distributed Computing*, pages 365–375, Portland, OR, 5-8 Aug. 1997.