Resource Manager for Globus-based Wide-area Cluster Computing

Yoshio Tanaka, Mitsuhisa Sato
Real World Computing Partnership
Mitsui bldg. 14F, 1-6-1 Takezono Tsukuba
Ibaraki 305-0032, Japan
{yoshio,msato}@trc.rwcp.or.jp

Motonori Hirano
Software Research Associates, Inc.
1-1-1 Hirakawachou Chiyoda
Tokyo 102-8605, Japan
m-hirano@sra.co.jp

Hidemoto Nakada, Satoshi Sekiguchi
Electrotechnical Laboratory
1-1-4 Umezono Tsukuba
Ibaraki 305-8568, Japan
{nakada,sekiguchi}@etl.go.jp

Abstract

In this paper, we present a new type of Globus resource allocation manager (GRAM) called RMF (Resource Manager beyond the Firewall) for wide-area cluster computing. RMF manages computing resources such as cluster systems and enables utilization of them beyond the firewall in global computing environments. RMF consists of two basic modules, a remote job queuing system (Q system) and a resource allocator. The Q system is a remote job queuing system, which schedules jobs submitted from global computing sites. The resource allocator manages resources and allocates them for requested jobs. For communications beyond the firewall between job processes, we designed the Nexus Proxy, which relays TCP communication links beyond the firewall. RMF and the Nexus Proxy provide a global computing environment in which users can easily utilize such parallel systems as cluster systems and supercomputers beyond the firewall.

1. Introduction

High-speed networks and advances in network infrastructure make a global computing environment an attractive platform for high performance computing[1]. Supercomputers, workstation/PC clusters, and databases, etc. at geographically distributed sites connected by high-speed networks form a networked virtual computer, or a metacomputer. Global computing, that is, parallel/distributed processing on a global computing environment, has great potential for computationally challenging problems. The Globus Metacomputing Toolkit[3, 4] provides basic mechanisms for building software infrastructure for a global computing environment such as communication, authentication, network information, and data access processes. These mechanisms are used to construct various higher-level metacomputing services, such as parallel programming tools and schedulers. The Globus team has reported its experience constructing global computing environments using some testbeds including I-WAY[2] and GUSTO[3]. The I-WAY networking experiment connects supercomputers and other resources at 17 different sites across North America, and saw 60 groups develop application in areas as diverse as large-scale scientific simulation, collaborative engineering, and supercomputer-enhanced scientific instruments.

As cluster systems become more widely available, it becomes feasible to run parallel applications on multiple clusters at different geographic locations (**wide-area cluster systems**[7, 8]) (See Figure 1). To enable wide-area cluster computing, however, many problems have to be solved. For example, a suitable software infrastructure will be expected, which deals with issues like security, heterogeneity, job scheduling, resource management, and accounting. Globus can be one of the best tools to build such infrastructures. We have built a global computing testbed in Japan using Globus toolkit to examine its mechanisms, behavior, and performance. From our experience, we identified the following two issues:

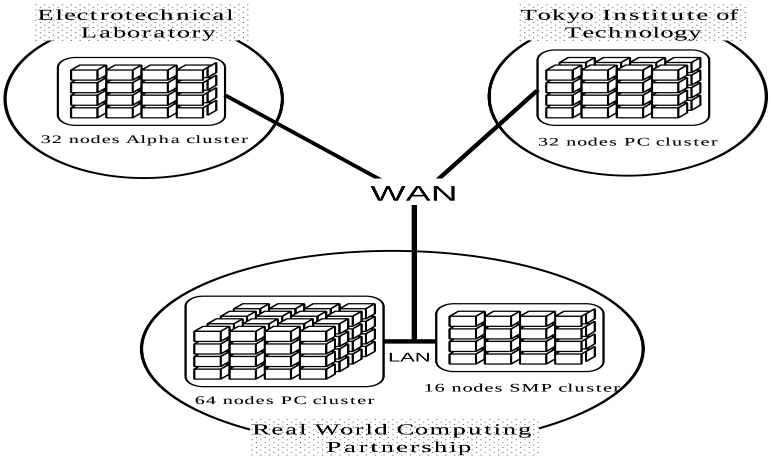

Electrotechnical Laboratory
32 nodes Alpha cluster
Tokyo Institute of Technology
32 nodes PC cluster
WAN
LAN
64 nodes PC cluster
16 nodes SMP cluster
Real World Computing Partnership

Figure 1. Wide-area cluster system.

1. **Resource managers for cluster systems**
   Under the Globus-based metacomputing systems, Globus Resource Allocation Manager (GRAM) is used for resource management such as process creation on remote resources. According to the site-specific scheduling policy, several types of GRAMs are exist such as fork type GRAM and LSF type GRAM[1]. However, there is no suitable resource managers for resource management of cluster systems. In order to utilize cluster systems as computing servers under Globus-based wide-area cluster systems, several resource managers can be used such as fork type GRAM and LSF type GRAM. If we utilize fork type GRAM, we have to install the Globus system and fork type GRAM on every workstation/PC which composes the cluster system. If the cluster system consists of tens to hundreds of workstations/PCs, installation of the Globus system for all computers would be a burden to administrators of the global computing system. LSF type GRAM is implemented by Load Sharing Facility(LSF), which is a commercial resource management system. LSF provides the facility of application resource management by scheduling, analyzing, and monitoring the processing or distributed application workloads. In LSF, workload scheduling involves taking user jobs and dynamically executing them on the appropriate resource according to policies that cover resource availability, time of job execution, and external events. Since LSF treats a cluster system not a virtual parallel computer but many individual workstations/PCs, LSF is not suitable to manage multiple cluster systems in the LAN environment. The most commercial resource management systems are still expensive and have little flexibility as resource managers on global computing systems.

2. **The firewall problem**
   A firewall is one of the most typical security systems. The typical firewall is a deny-based firewall, which basically denies all TCP/UDP communication links. In order to establish communication links beyond the deny-based firewall, the firewall must be configured to allow the communication link. In this paper, we assume a deny-based firewall. Communication services within the Globus toolkit are provided by the Nexus communication library[6]. Since the Nexus library allocates TCP communication ports dynamically and there is no mechanism in Nexus for specifying the port that the TCP protocol module listens on, the communication link can not be established beyond the firewall. Thus, resources inside the firewall cannot contribute to wide-area cluster systems. In the I-WAY and GUSTO testbeds, supercomputers and workstations have no firewall. In order to spread the global computing environment over the various sites (including universities, laboratories, and companies), the global computing systems should be responsible for firewalls.

In this paper, we present the design and implementation of a new type of Globus Resource Allocation Manager (GRAM) called RMF (Resource Manager beyond the Firewall). RMF is designed to solve the two problems mentioned above, and has the following two features:

[1] Details are described in the next section.

1. RMF manages multiple computing resources such as cluster systems, supercomputers, and workstations and provides them for global computing environments.

2. RMF provides a mechanism to deploy the Globus gatekeeper and the resource manager on individual systems. By deploying the Globus gatekeeper outside the firewall and the resource manager inside the firewall, RMF enables computing resources of clusters inside the firewall to be used for global computing.

We also report the Nexus Proxy, which relays TCP communication link and enables communications beyond the firewall. In the next section, we introduce the basic architecture of the Globus system as background. In section 3, we present the design and implementation of RMF and the Nexus Proxy. We discuss an outline of future directions for global computing environments and also for our research in section 4. Finally, we present our concluding remarks.

2. The Globus Metacomputing Toolkit

Globus is a large U.S. based project which is developing the fundamental technology that is needed to build computational grids – execution environments that enable an application to integrate geographically distributed instruments, displays, and computational and information resources. The Globus metacomputing toolkit is a central element of the Globus system. The toolkit defines the basic services and capabilities required to construct a computational grid. It comprises a set of components that implement basic services for security, resource location, resource management, communication, etc.. Table 1 lists the services currently defined by Globus.

These services are distinct and have well-defined interfaces, they can be incorporated into applications or tools in an incremental fashion. Each module defines an interface, which higher-level services use to invoke that module's mechanisms, and provides an implementation, which uses appropriate low-level operations to implement these mechanisms in different environments. Here, we focus the resource management system and proceed to provide a more detailed description of GRAM.

As described before, Globus is a layered architecture in which high-level Globus services are built on top of an essential set of core local services. At the bottom of this layered architecture, GRAM provides the local component for resource management. Each GRAM is responsible for a set of resources operating under a site-specific allocation policy, often implemented by a local resource management system, such as LSF or Condor. Each computing server provides one or more GRAMs, each responsible for a particular local set of resources. For each GRAM, a GRAM gatekeeper (server) must be running and listening at a port on a computing server. For instance, if a server provides two types of GRAMs, LSF and fork, an LSF gatekeeper and a fork gatekeeper must exist on the server before any request is submitted. If the fork gatekeeper receives the request, job processes are created using the $fork$ system call function. If the LSF gatekeeper receives the request, job processes are enqueued via the *bsub* command which is provided by LSF.

The Globus toolkit provides APIs and user commands to utilize the modules listed in Table 1. For example, the globusrun command is provided to submit a job request to a Globus resource. Figure 2 illustrates the resource management architecture and shows how a job request submitted via the globusrun command is dispatched on a remote resource. In the figure, a job request is submitted from a local client to a remote server (gcitest.etl.go.jp).

1. In order to send a job request to the gatekeeper, run the globusrun command on a client. A job request is a request to the gatekeeper to create one or more job processes, expressed in the supplied Resource Specification Language (RSL). In the figure, a job request is sent to the fork gatekeeper. *gcitest.etl.go.jp-*$fork$ is the name of the gatekeeper. The job request is specified by the last argument of the globusrun command.

2. When the globusrun command is invoked, gatekeeper's Distinguished Name (DN) is retrieved from the gatekeeper's name via MDS facilities. As shown in the figure, the gatekeeper's DN contains a hostname, a port number, and a gatekeeper certificate subject.

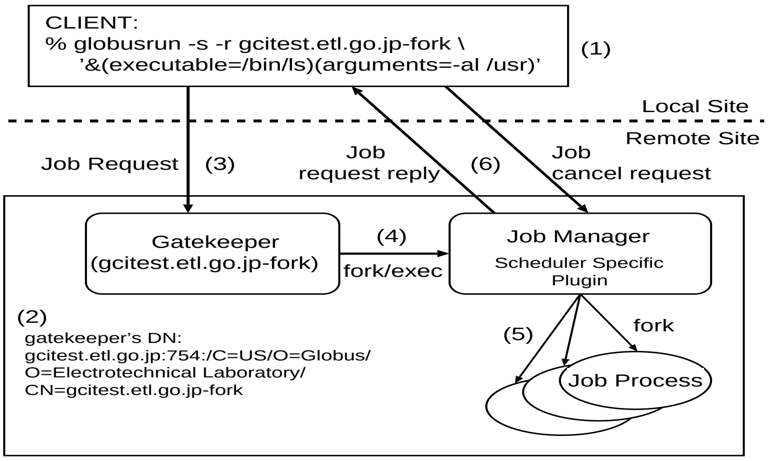
3. According to the gatekeeper's DN, a job request is submitted to the Globus gatekeeper. The gatekeeper prompts the user to enter his pass phrase and checks user authentication.

4. The gatekeeper creates a job manager which starts the job on the local system, and handles all further communication with the client.

5. According to the job manager type, the job manager creates job processes in its own manner. For example, a fork job manager creates job processes using the $fork$ system call function.

Table 1. Globus Service

Service	Name	Description
Resource Management	GRAM	Resource allocation and process management
Communication	Nexus	Unicast and multicast communication services
Information	MDS	Distributed access to structure and status information
Security	GSI	Authentication and related security services
Health and status	HBM	Monitoring of health and status of system components
Remote data access	GASS	Remote access to data via sequential and parallel interfaces
Executable management	GEM	Construction, caching, and location of executables



```
CLIENT:
% globusrun -s -r gcitest.etl.go.jp-fork \
    '&(executable=/bin/ls)(arguments=-al /usr)'
```

Figure 2. The architecture of GRAM and the flow of a job request.

6. A job manager translates messages received from the gatekeeper and client into an internal API that is implemented by the machine specific component. It also translates callback requests from the machine specific components through the internal API into messages to the application manager.

Since a gatekeeper invokes a job manager via the *execv* system call and a fork job manager creates job processes using the *fork* system call, the gatekeeper, the job manager, and the job processes are all running on the same resources. In order to utilize different resources from the one on which the gatekeeper is running, other types of GRAMs are necessary such as LSF and Condor.

While Globus services can be used directly by application programmers, they are more commonly accessed via higher-level tools developed by tool developers. One of the tools is MPICH-G[5], a grid-enabled MPI based on the MPICH library, with Nexus used for communication, GRAM services for resource allocation, and GSI services for authentication. MPICH-G is a complete implementation of the MPI-1 standard. The only change to the MPICH startup model is that the contents of the machines file include resource manager (GRAM) names. Once the machines file has been read and resource manager contacts determined, the MPICH-G mpirun implementation calls globusrun to manage the task of job submission and execution.

3. Design and Implementation of RMF

3.1. Resource Management

As described above, we found two problems for building a Globus-based wide-area cluster system. Those are no resource managers suitable for wide-area cluster systems and the firewall problem. Since installation of the Globus system for all computers is a painful work, fork type GRAM can not be used for wide-area cluster systems. On the other hand, LSF type GRAM is also not suitable for building wide-area cluster systems because LSF cannot treats a cluster system as a virtual parallel computers and has little flexibility for changing scheduling policies. Further-more, since LSF dynamically allocates TCP ports for job submission, LSF type GRAM cannot be used for sites which use the firewall.

Aiming to solve the two problems described above (resource management for cluster systems and the firewall problem), we designed and implemented a new experimental type of GRAM called RMF which manages cluster systems and parallel systems inside the firewall and provides them to global computing environments. The most important design issue of RMF is how to "provide computing resources such as cluster systems and supercomputers inside the firewall to global computing environments". RMF consists of two basic modules, a **Job Queuing System (Q system)** and a **Resource Allocator**.

- **Q system**
  The Q system is based on the client-server model. It provides a remote job execution mechanism using job queues. A server of the Q system (Q server) runs on every computing resource inside the firewall. A client of the Q system (Q client) is invoked by a job manager running outside the firewall. A Q client inquires of a resource allocator the best resources on which job processes will run, and it submits job requests to Q servers on the resources specified by the resource allocator. A Q server creates job processes to process jobs received and queues them for later inquiries about job status and cancellation of the jobs. Only a TCP communication port on which Q clients and Q servers communicate must be opened beyond the firewall.

- **Resource Allocator**
  A resource allocator runs as a daemon process on a workstation inside the firewall. It manages resources and waits inquiries from Q clients. When the resource allocator receives an inquiry from a Q client, it selects the most suitable resource on which job processes will run and reports the Q client of the name of the resources.

Figure 3 illustrates the architecture of RMF and the relationships between the modules.

The following steps show the outline of the execution flow of a submitted job request.

0. Run a Globus gatekeeper for an RMF type GRAM outside the firewall. Run a resource allocator inside the firewall. Run a Q server on every computing resource.

1. A job request is submitted to the RMF gatekeeper.

2. The job manager invoked by the gatekeeper creates a Q client process.

3. The Q client inquires of a resource allocator which resources are the best to execute a job. Currently, the Q client sends only the requested number of nodes to the resource allocator as a hint on resource selection.

4. A resource allocator selects resources and reports their names to the Q client.

5. The Q client submits a job request to the Q server running on the resources reported by the resource allocator. The job request includes information necessary for job process creation, such as user information, job type (the same as the job type used in Globus toolkit), the number of nodes, the executable file name and its arguments, and shell environment variables.

6. The Q server receives the job request from the Q client and creates job processes according to the job type. The MPI job type shows that the job is requested by the MPICH-G mpirun command. The MPI job type implies that the executable is written using MPICH-G and the MPI processes will be running over the wide-area cluster system. If the job type is MPI, the Q server creates a job process to execute the job directly, meaning that the Q server creates the process via an *execv* system call. Other job types indicate that the job is directly requested via the globusrun command. The other job types indicate that the MPI processes will be running on resources in one site. If the job type is not MPI, the Q server creates a machines file which includes the names of machines reported by the resource allocator and runs an mpirun command with the machines file as a -machinefile argument. Here, the mpirun command is not MPICH-G based but cluster specific MPICH based such as MPICH on ch_p4 device and MPICH on ch_shmem device.

The Q system provides a mechanism to create job processes on different resources from the one on which the gatekeeper and the job manager are running. Since the Globus GASS facility uses files for input/output, the Q system also transfers the files to remote resources.

The resource allocator reads a configuration file of the resources. Figure 4 shows a sample configuration file.
'#'indicates a comment line. *Name* is the name of the resource, *type* is either 'c' or 'p'. 'c' means that the resource is a cluster system, and 'p' means that the resource is a supercomputer. *procs* is the number of processors in a node, and *nnodes* is the num-

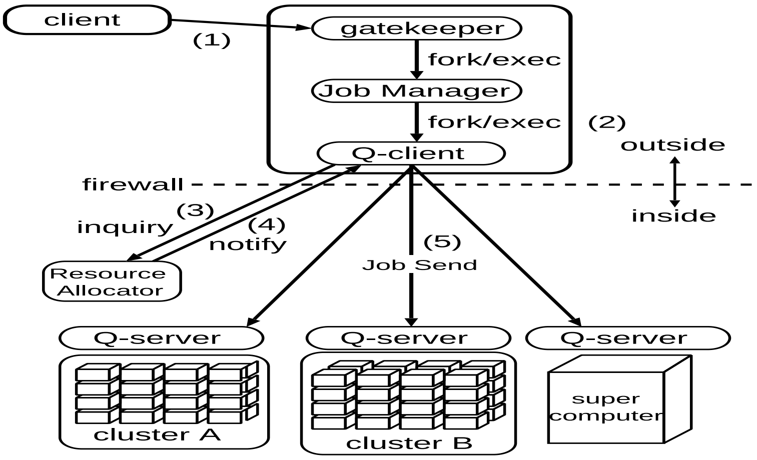


Figure 3. The architecture of RMF.

```
# Name       type   procs   nnodes   clock    prefix
COMPaS          c       4        8     200   pdpsmp
COMPaS II       c       4        4     450   pdsmpc
SR2201          p       1      256     150
```

Figure 4. A Sample Configuration File.

ber of nodes in the resource. *clock* is the CPU clock in MHz, and *prefix* is a hint for the name of the workstations/PCs in the cluster system. In Figure 4, for instance, the names of the nodes of COMPaS are pdpsmp0, pdpsmp1, ..., pdpsmp7. The resource allocator selects resources using several hints such as the number of running jobs, available nodes, and the CPU clock.

3.2. The Nexus Proxy

By submitting jobs to the RMF gatekeeper, users can execute jobs on remote resources protected by the firewall. However, there is still a restriction. Jobs running inside the firewall cannot communicate with processes which are running outside the firewall. For example, let us consider the case where we try to run two MPICH-G processes on two sites, each process running on an individual site. These processes use the Nexus communication library and the Nexus tries to establish TCP communication links on a standard WAN environment. Since the Nexus library allocates TCP communication ports dynamically and there is no mechanism in Nexus for specifying the port that the TCP protocol module listens on, the communication link between the processes cannot be established beyond the firewall so that the processes can not communicate with each other.

To overcome this restriction, we designed the Nexus Proxy which relays TCP communications to provide a communication mechanism beyond the firewall. The Nexus Proxy server runs outside the firewall and it receives relay requests from clients. When a client tries to connect to remote hosts or bind a socket to listen for requests, the client sends a relay request (either a connect request or a bind request) to the Nexus Proxy server instead of call *connect*() and *bind*() functions. A connect request includes the destination address and the port number to which the client intends to connect. A bind request includes the address and the port number on which the client listens for the request. Some library functions are provided to utilize the Nexus Proxy mechanism. The most important functions are $NXProxyConnect()$ and $NXProxyBind()$. $NXProxyConnect()$ sends a connect request to the Nexus Proxy server and returns a file descriptor on which the client can communicate with the remote resources. A bind request is sent to the Nexus Proxy server via $NXProxyBind()$. Figure 5 illustrates the communication mechanism via the Nexus Proxy when a process running inside the firewall intends to connect to a process running outside the firewall.

0. The Nexus Proxy server is running outside the firewall and listens requests from clients. process A

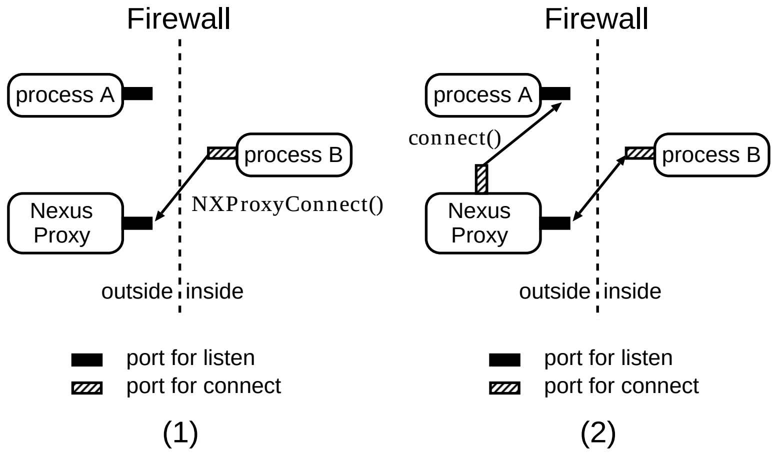


Figure 5. The communication mechanism via the Nexus Proxy.

is running outside the firewall and waits communication requests from other processes.

1. When process B intends to connect to process A, it calls $NXProxyConnect()$ instead of *connect*(). Then, a connect request is sent to the Nexus Proxy server.
2. When the Nexus Proxy server accepts the request, a communication link between the process B and the Nexus Porxy is established. Then, the Nexus Proxy server calls the *connect*() function and sends a connect request to the process A, which is specified in the connect request from process B.
3. When the process A accepts the connect request from the Nexus Proxy server, the communication link between the Nexus Proxy and the process A is established. The Nexus Proxy server relays the communication between the client and the destination.

If the Nexus Proxy server receives a bind request, it creates a new socket and listens for connect requests on it. When the server accepts connect requests from remote resources, it connects to the client and establishes the communication link between the client and the remote resources. The firewall must be configured to allow a communication link to the Nexus Proxy. The architecture of the Nexus Proxy is similar to other Proxy servers such as SOCKS. However, since SOCKS can not be utilized for initial passive open socket, it can not incorporated into the Nexus communication library. We modified the source code of Globus to use $NXProxyConnect()$ and $NXProxyBind()$ instead of *connect*() and *bind*(). Address information for the communication startpoint/endpoint should also be changed to indicate the Nexus Proxy server. As a result, MPICH-G processes are able to communicate with each other beyond the firewall.

4. Other Issues for Wide-area Cluster Computing

In the Globus-based wide-area cluster computing environment, RMF and the Nexus Proxy provide a mechanism to utilize parallel systems such as workstation/PC clusters and supercomputers beyond the firewall. Although RMF is a suitable GRAM for wide-area cluster systems based on the Globus toolkit, there are still several issues for wide-area cluster computing:

- **Optimization of a resource allocation algorithm**
  The current implementation of the resource allocator uses only static information including the resource configuration listed in Figure 4, the number of requested nodes, and the number of running jobs on each resource as hints for resource allocation scheduling. Dynamic information such as the processor's load can be introduced to the allocation scheduling.

  On the other hand, it will be convenient to provide methods to specify resource requirements for users (currently, only the number of required nodes can be specified). For example, a user may specify the computational requirements in terms of floating point performance (MFLOPs) or network bandwidth. As used in the Globus system, Resource

Specification Language (RSL) can be used to describe these resource requirements.

- **Executable management on remote resources**
When users submit jobs to remote resources, the corresponding executable must be on each remote resource in advance. Manual staging of executables is a painful activity. Globus Executable Management (GEM) service is intended to support the identification, location, and creation of executables in heterogeneous environments. GEM can be used in conjunction with other Globus services to implement a variety of distributed code management strategies, based on online executable archives and compile servers. We are designing automated mechanism for identifying and generating appropriate executables.

- **Programming environments**
MPICH-G enables us to run any arbitrary MPI application on global computing environments simply by running the mpirun command. As described in [9], however there are several performance problems such as collective communication performance on wide-area cluster systems. Further research is necessary, for example, research related to collective operation performance and network topology information.

5. Concluding Remarks

We have presented a new type of Globus resource allocation manager, RMF, which intends to utilize cluster systems and supercomputers beyond the firewall. The Nexus Proxy relays TCP communications beyond the firewall and can be used from the Nexus communication library. RMF and the Nexus Proxy enable us to construct a wide-area cluster system based on the Globus system in which users can utilize computing resources such as cluster systems beyond the firewall easily. Since the firewall is a severe obstacle to constructing global computing environments, RMF type GRAM is suitable for building Globus-based wide-area cluster system.

References

[1] I. Foster and Carl Kesselman, “The GRID: Blueprint for a New Computing Infrastructure”, Morgan Kaufmann Publishers (1998).

[2] I. Foster, Jonathan Geisler, Bill Nickless, Warren Smith, and Steven Tuecke, “Software Infrastructure for the I-WAY high performance distributed computing experiment”, Proc. 5th IEEE Symp. on High Performance Distributed Computing, pp. 562–572 (1996).

[3] I. Foster and Carl Kesselman, “The Globus Project: A status report”, Proc. Heterogeneous Computing Workshop, pp. 4–18 (1998).

[4] http://www.globus.org/

[5] I. Foster and Nicholas T. Karonis, “A Grid-Enabled MPI: Message Passing in Heterogeneous Distributed Computing Systems”, Proc. Supercomputing (1998).

[6] I. Foster, Carl Kesselman, and Steven Tuecke, “The Nexus approach to integrating multithreading and communication”, Journal of Parallel and Distributed Computing, Vol. 37, pp. 70–82 (1996).

[7] http://www.cs.vu.nl/albatross/.

[8] H. E. Bal, Aske Plaat, Thilo Kielmann, Jason Maassen, Rob van Nieuwpoort, and Ronald Veldema, “Parallel Computing on Wide-Area Clusters: the Albatross Project”, Proc. Extreme Linux Workshop, pp. 20–24 (1999).

[9] T. Kielmann, R. F. H. Hofman, H. E. Bal, A. Plaat, and R. A. F. Bhoedjang, “MagPIe: MPI’s Collective Communication Operations for Clustered Wide Area Systems”, Proc. Seventh ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, pp. 131–140 (1999).