NASA/CR-2002-211960
ICASE Report No. 2002-43

An Experimental Framework for Executing Applications in Dynamic Grid Environments

Eduardo Huedo
Centro de Astrobiología, CSIC-INTA, Torrejón de Ardoz, Spain

Ruben S. Montero and Ignacio M. Llorente
Universidad Complutense, Madrid, Spain

November 2002

The NASA STI Program Office . . . in Profile

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA Scientific and Technical Information (STI) Program Office plays a key part in helping NASA maintain this important role.

The NASA STI Program Office is operated by Langley Research Center, the lead center for NASA's scientific and technical information. The NASA STI Program Office provides access to the NASA STI Database, the largest collection of aeronautical and space science STI in the world. The Program Office is also NASA's institutional mechanism for disseminating the results of its research and development activities. These results are published by NASA in the NASA STI Report Series, which includes the following report types:

- TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA's counterpart of peer-reviewed formal professional papers, but having less stringent limitations on manuscript length and extent of graphic presentations.

- TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.

- CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.

- CONFERENCE PUBLICATIONS. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or cosponsored by NASA.

- SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.

- TECHNICAL TRANSLATION. English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services that complement the STI Program Office's diverse offerings include creating custom thesauri, building customized data bases, organizing and publishing research results . . . even providing videos.

For more information about the NASA STI Program Office, see the following:

- Access the NASA STI Program Home Page at ***http://www.sti.nasa.gov***

- Email your question via the Internet to help@sti.nasa.gov

- Fax your question to the NASA STI Help Desk at (301) 621-0134

- Telephone the NASA STI Help Desk at (301) 621-0390

- Write to:
  NASA STI Help Desk
  NASA Center for AeroSpace Information
  7121 Standard Drive
  Hanover, MD 21076-1320

NASA/CR-2002-211960
ICASE Report No. 2002-43

An Experimental Framework for Executing Applications in Dynamic Grid Environments

Eduardo Huedo
Centro de Astrobiología, CSIC-INTA, Torrejón de Ardoz, Spain

Ruben S. Montero and Ignacio M. Llorente
Universidad Complutense, Madrid, Spain

ICASE
NASA Langley Research Center
Hampton, Virginia

Operated by Universities Space Research Association

Prepared for Langley Research Center
under Contract NAS1-97046

November 2002

Available from the following:

NASA Center for AeroSpace Information (CASI)
7121 Standard Drive
Hanover, MD 21076-1320
(301) 621-0390

National Technical Information Service (NTIS)
5285 Port Royal Road
Springfield, VA 22161-2171
(703) 487-4650

AN EXPERIMENTAL FRAMEWORK FOR EXECUTING APPLICATIONS IN DYNAMIC GRID ENVIRONMENTS*

EDUARDO HUEDO†, RUBEN S. MONTERO‡, AND IGNACIO M. LLORENTE§

Abstract. The Grid opens up opportunities for resource-starved scientists and engineers to harness highly distributed computing resources. A number of Grid middleware projects are currently available to support the simultaneous exploitation of heterogeneous resources distributed in different administrative domains. However, efficient job submission and management continue being far from accessible to ordinary scientists and engineers due to the dynamic and complex nature of the Grid. This report describes a new Globus framework that allows an easier and more efficient execution of jobs in a "submit and forget" fashion. Adaptation to dynamic Grid conditions is achieved by supporting automatic application migration following performance degradation, "better" resource discovery, requirement change, owner decision or remote resource failure. The report also includes experimental results of the behavior of our framework on the TRGP testbed.

Key words. Grid technology, adaptive execution, job migration, Globus

Subject classification. Computer Science

1. Introduction. Several research centers share their computing resources in Grids, which offer a dramatic increase in the number of available processing and storing resources that can be delivered to applications. These Grids provide a way to access the resources needed for executing the compute and data intensive applications required in several research and engineering fields. In this work we concentrate on the access to a high number of computing systems to be independently exploited.

In a Grid scenario, a sequential or parallel job is commonly submitted to a given resource by taking the following path [37]:

- *Resource Discovery*: A list of appropriate resources is obtained by accessing to an information service mechanism, the discovery is usually based on a set of job requirements.
- *System Selection*: A single resource or a subset of resources are selected among the candidates discovered in previous step.
- *System Preparation*: The Preparation at least involves setup and input file staging.
- *Submission*: The job is submitted.
- *Monitoring*: The job is monitored over time.
- *Migration*: The user may decide to restart its job on a different resource if performance slowdown is detected or a "better" resource is discovered.

*This research was supported by the National Aeronautics and Space Administration under NASA Contract No. NAS1-97046 while the last two authors were in residence at ICASE, NASA Langley Research Center, Hampton, VA 23681-2199. This research was also supported by the Spanish research grant TIC 2002-00334 and by the Instituto Nacional de Técnica Aeroespacial.

†Centro de Astrobiología, CSIC-INTA, Associated to NASA Astrobiology Institute, 28850 Torrejón de Ardoz, Spain www.cab.inta.es/∼huedoce

‡Departamento de Arquitectura de Computadores y Automática, Universidad Complutense, 28040 Madrid, Spain rubensm@dacya.ucm.es

§Departamento de Arquitectura de Computadores y Automática, Universidad Complutense, 28040 Madrid, Spain & Centro de Astrobiología, CSIC-INTA, Associated to NASA Astrobiology Institute, 28850 Torrejón de Ardoz, Spain www.dacya.ucm.es/nacho

- *Termination*: When the job is finished, its owner is notified and some completion tasks, such as output file staging and cleanup, are performed.

The Globus toolkit [25] has become a *de facto* standard in Grid computing. Globus is a core Grid middleware that provides the following components, which can be used separately or together, to support Grid applications: GRAM (Globus Resource Allocation Manager), GASS (Global Access to Secondary Storage), GSI (Grid Security Infrastructure), MDS (Monitoring and Discovery Service), and Data Grid (GridFTP, Replica Catalog, and Replica Management). These services and libraries allow secure and transparent access to computing and data resources across multiple administrative domains. After installation, Globus components must be adjusted for efficient execution on a given Grid environment or testbed. This procedure is not straightforward, specially when setting up the information system in order to provide the attributes needed for performing an effective and efficient resource discovery and selection.

The MDS 2.1 configuration poses a potential security risk since anonymous searches to the GRIS (Grid Resource Information Service) servers are required to build a hierarchical GIIS (Grid Institution Index Service). The MDS publishes important host information that could be used to gain unauthorized access to grid hosts. This security threat has been eliminated in MDS 2.2 because it provides mutual authentication between GIIS and GRIS servers as well as between GIIS servers in a hierarchy.

The Globus toolkit supports the stages of Grid scheduling by providing resource discovery, resource monitoring, resource allocation, and job control services. However, the user is responsible for manually performing all the submission stages in order to achieve any functionality [37, 38]. In the Resource Discovery stage, the user queries MDS to filter out the resources that do not meet the job requirements. The System Selection is usually performed by ordering the possible resources according to some priority. The Preparation and Termination tasks involve transferring input/output files using GASS or GridFTP. The job is submitted and monitorized using GRAM. Globus does not provide any Migration support.

Application development and execution in the Grid continue requiring a high level of expertise due to its complex nature. Moreover, Grid resources are also difficult to efficiently harness due to their heterogeneous and dynamic nature:

- *Multiple administration domains*: The Grid resource and the client system do not have to belong to the same administrative domain. Once the job is submitted, its owner does not have total control over it. The resource administrator could freely reclaim his resources and decide to suspend or cancel the job.
- *Dynamic resource load*: The idle resources initially available may become saturated. We temporally get access to heterogeneous resources belonging to different administrative domains that are being also exploited by internal users, which usually enjoy higher execution priorities. Moreover, these resources can be also exploited by other Grid users.
- *High fault rate*: In a Grid, resource failure is the rule rather than the exception. Moreover, resources are added and removed continuously.

We believe that end users would improve their computational productivity if they were provided with a friendly, efficient and reliable way to access Grid resources. We have developed a new Globus framework that allows an easier and more efficient execution of jobs on a dynamic Grid environment. This experimental framework has been developed in the context of the GridWay project, whose aim is to develop user-level tools to reduce the gap between Grid middleware and application developers, and so make Grid resources useful and accessible to scientists and engineers. Our personal framework allows users to harness dynamic resources scattered geographically across the Internet at department, organization or global level. Once a

job is submitted, it is initially allocated to a computing resource according to a resource selection policy and may be migrated among the available resources when its requirements change, a "better" resource is discovered, a remote failure is detected or the resource does not provide the expected computing power.

The outstanding feature of our framework is its modular and so extensible approach to deal with changing resource characteristics and job requirements. The aims of this report are to describe its architecture and to present preliminary experiences of its behavior when managing the execution of a computational fluid dynamic code on a research testbed. The architecture of the experimental toolkit is described in Section 2. The following sections motivate and describe its main functionalities: resource discovery and selection in Section 3, file management in Section 4 and job execution in Section 5. The application model is presented in Section 6 and the user interface is described in Section 7. The GridWay framework is compared with other similar approaches in Section 8. Preliminary results of the framework on the TRGP (Tidewater Research Grid Partnership) testbed are presented in Section 9. Finally, Section 10 includes the main conclusions and outlines our future work.

2. Architecture of the Experimental Framework. The GridWay submission framework provides the runtime mechanisms needed for dynamically adapting an application to a changing Grid environment. The core of the framework is a personal *Submission Agent* that performs all submission stages and watches over the efficient execution of the job. Adaptation to changing conditions is achieved by dynamic scheduling. Once the job is initially allocated, it is rescheduled when performance slowdown or remote failure are detected, and periodically at each *discovering* interval. Application performance is evaluated periodically at each *monitoring* interval by executing a *Performance Degradation Evaluator* program and by evaluating its accumulated suspension time. A *Resource Selector* program acts as a personal resource broker to build a priorized list of candidate resources. Since both programs have access to files dynamically generated by the running job, the application has the ability to take decisions about resource selection and to provide its own performance profile. The *Submission Agent* (figure 2.1) consists of the following components:

- *Request Manager*
- *Dispatch Manager*
- *Submission Manager*
- *Performance Monitor*

The flexibility of the framework is guaranteed by a well-defined API (Application Program Interface) for each *Submission Agent* component. Moreover, the framework has been designed to be modular, through scripting, to allow extensibility and improvement of its capabilities. The following modules can be set on a per job basis:

- *Resource Selector*
- *Performance Degradation Evaluator*
- *Prolog*
- *Wrapper*
- *Epilog*

The following actions are performed by the Submission Agent:

- The client application uses a Client API to communicate with the *Request Manager* in order to submit the job along with its configuration file, or `job template`, which contains all the necessary parameters for its execution. Once submitted, the client may also request control operations to the *Request Manager*, such as job *stop/resume*, *kill* or *reschedule*.
- The *Request Manager* checks the `job template`, sets up the data internal structures needed to

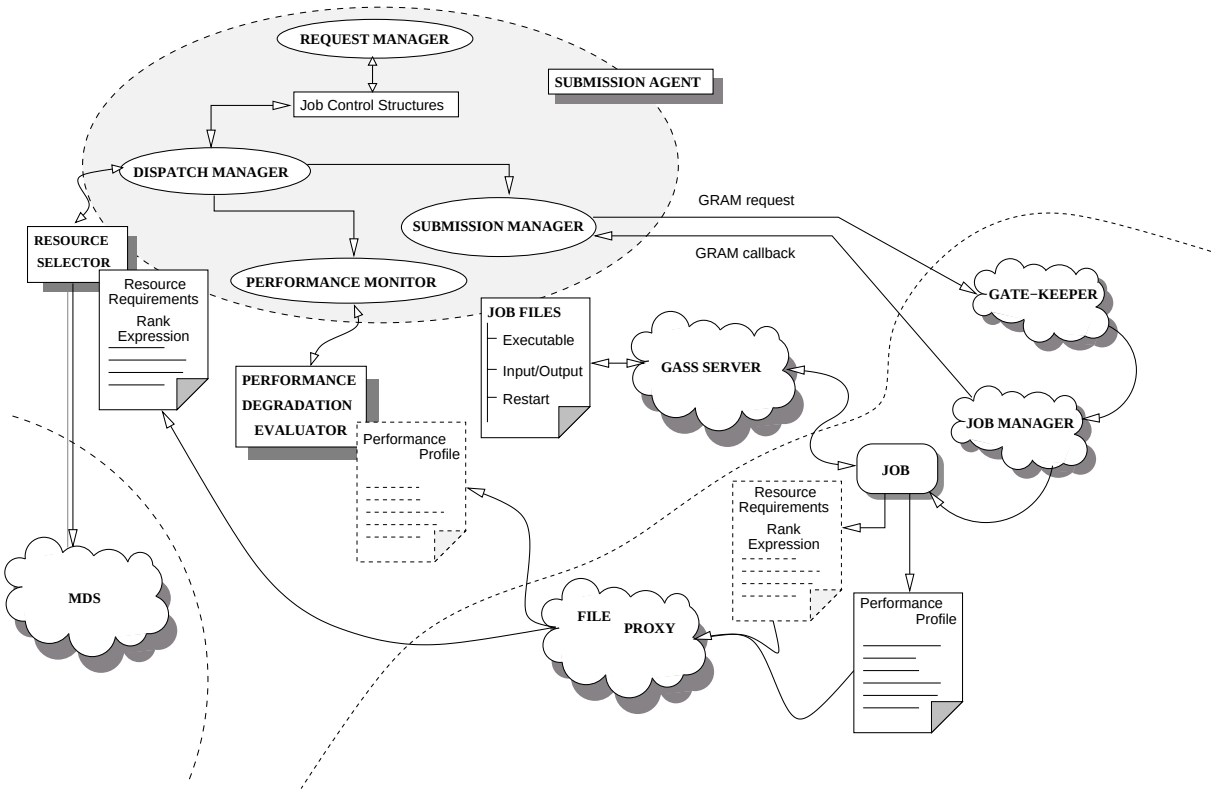

FIG. 2.1. *Architecture of the experimental framework.*

manage the job submission and holds the job in *pending* state. It then waits for client requests.

- The *Dispatch Manager* periodically wakes up at each *scheduling* interval, and tries to submit *pending* and *rescheduled* jobs to Grid resources. It invokes the execution of the *Resource Selector* corresponding to each job, which returns a priorized list of candidate hosts. The *Dispatch Manager* submits *pending* jobs by invoking a *Submission Manager*, and also decides if the migration of *rescheduled* jobs is worthwhile or not. If this is the case, the *Dispatch Manager* triggers a *migration* event along with the new selected resource to the job *Submission Manager*, which manages the job migration.
- The *Submission Manager* is responsible for the execution of the job during its lifetime, i.e. until it is *done* or *stopped*. It is initially invoked by the *Dispatch Manager* along with the first selected host, and is also responsible for performing job migration to a new resource. It also probes periodically at each *polling* interval the connection to the GRAM Job Manager and Gatekeeper in order to detect remote failures. Currently, Globus management components are used to support all these actions, although another Grid middleware could be employed. The Submission Manager performs the following tasks:
  - *Prologing*: Preparing the RSL (Resource Specification Language) and submitting the *Prolog* executable. The *Prolog* sets up remote system, transfers executable and input files, and, in the case of restart execution, also transfers restart files.
  - *Submitting*: Preparing the RSL, submitting the *Wrapper* executable, monitoring its correct execution, updating the submission states via Globus callbacks and waiting for *migration*, *stop* or *kill* events from the *Dispatch Manager*. The *Wrapper* wraps the actual job in order to capture its exit code.

- *Canceling*: Canceling the submitted job if a *migration*, *stop* or *kill* event is received by the *Submission Manager*.
- *Epiloging*: Preparing the RSL and submitting the *Epilog* executable. The *Epilog* transfers back output files on termination or restart files on migration, and cleans up remote system.

• The *Performance Monitor* periodically wakes up at each *monitoring* interval. It requests *rescheduling* actions to detect "better" resources when performance slowdown is detected and at each *discovering* interval.

The state of the job is monitored and reported to the user (figure 2.2 shows the job state diagram).

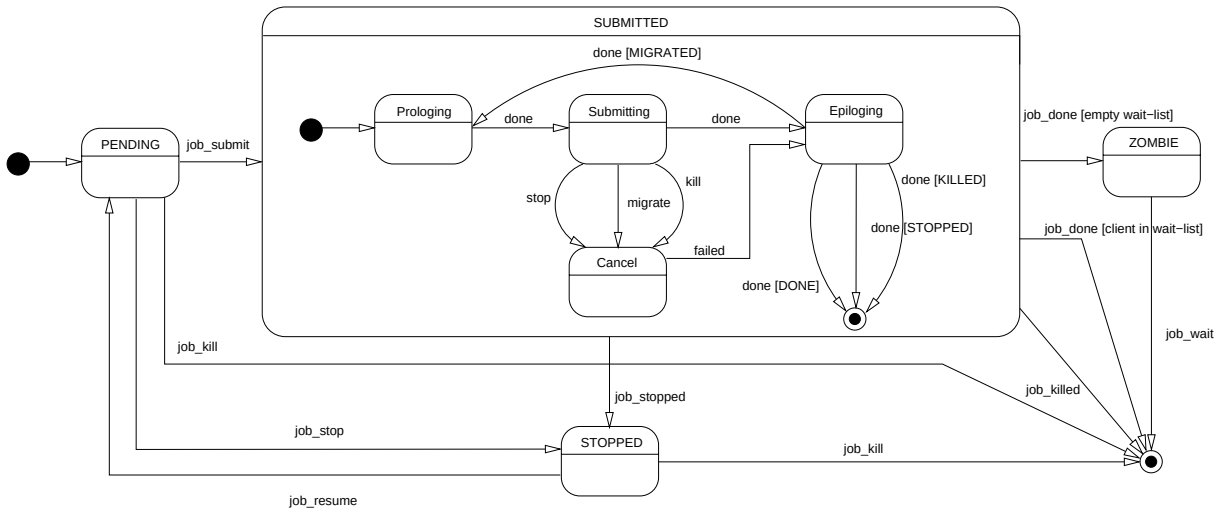

FIG. 2.2. *Job state diagram.*

3. Resource Discovery and Selection. Due to the heterogeneous and dynamic nature of the Grid, the end-user must establish the requirements that must be met by the target resources and a criteria to rank the matched resources. Both may combine static machine attributes (operating system, architecture, software availability...) and dynamic status information (disk space, processor load, free memory,...). In our framework, the *Resource Selector* is responsible for performing resource discovery and selection following the preferences provided by the user.

3.1. The Resource Selector. The *Resource Selector* is executed by the *Dispatch Manager* in order to get a ranked list of candidate hosts when the job is *pending* to be submitted, or a *rescheduling* action has been requested. The *Resource Selector* is a script or a binary executable specified in the `job template`. The template may also include additional parameters needed for resource discovery and selection, such as host requirement files, rank expression files or job limits. Its standard output must show a candidate resource per line in a specific fixed format that includes: data host GRAM Job Manager, execution host GRAM Job Manager, rank, number of slots, and architecture. The job is always submitted to the default queue. However, the *Resource Selector* should verify that the resource keeps a candidate queue according to the job limits.

This modular approach guarantees the extensibility of the resource selection. Different strategies for application level scheduling can be implemented, from the simplest one based on a pre-defined list of hosts to more advanced strategies based on requirement filters, authorization filters and rank expressions in terms of performance models [31, 20]. The static and dynamic attributes needed for resource discovery and selection

must be collected from the information services in the Grid testbed. Examples of such services could be:

- Globus MDS, which provides the configuration, capability and status of the resources.
- NWS (Network Weather Service) [43], which monitors and dynamically forecasts resource and network performance in order to have more accurate load information and also consider network heterogeneity.
- Replica Location Service [19], which provides the physical locations of the logical files involved in the execution in order to also consider file transfer costs.

The decisions taken by the *Resource Selector* are as good as the information provided to it. Some testbeds, like EDG (European Data Grid) [39] and NorduGrid [6], have developed its own MDS schema. There are also projects under development, like GLUE [9] that is available within MDS 2.2, to provide unified and standardized schemas for the testbeds to interoperate.

The *Resource Selector* could also access to other Grid services to negotiate a performance contract with, for example, a service-oriented Grid architecture for distributed computational economies [35]. A contract negotiation process could be included within the resource selection functionality and its specifications could be continually monitored by the *Performance Monitor* component.

The application could be required to be aware of the Grid environment and take dynamically decisions about its own resource selection. For example, its requirement profile could change when more resources (memory, disk...) or different resources (software or license availability) are required, or when a lack of capacity (disk space for example) is detected. A mechanism to deal with changing requirement profiles and rank expressions is provided since the files processed by the *Resource Selector* can be dynamically generated by the running job.

3.2. Scheduling Policy. The goal of the *Resource Selector* is to find a host that minimizes total response time (file transfer and job execution). Consequently, our application level scheduler promotes the performance of each individual application [26] without considering the rest of *pending*, *rescheduled* or *submitted* applications. This greedy approach is similar to the one provided by most of the local distributed resource management tools [22] and Grid projects [31, 20, 11]. The remote resources are "flooded" with requests and subsequent monitoring of performance degradation allows a better balance by migration.

It is well known that this is not the best approach to improve the productivity of the Grid in terms of the number of jobs executed per time unit because it does not balance the interests of different applications [40]. Efficient application performance and efficient system performance are not necessarily the same. For example, when competing applications are executing, the scheduler should give priority to short or new jobs by temporally stopping longer jobs.

Although currently not provided, the *Dispatch Manager* could make decisions taking into account all *pending*, *rescheduled* and *submitted* jobs with the aim of making an intelligent collective scheduling of them, that is, a user level scheduling approach. Collective scheduling becomes highly important when dealing with parametric jobs. The *Resource Selector* could also communicate with higher-level schedulers or metaschedulers to take into account system level considerations [40].

4. Data Management. The files which do not have to be accessible to the local host (submission client) during job execution on a remote host are referred as *static files*. Examples of these files are input, output and restart files. On the other hand, the files which, being generated on the remote host by the running job, have to be accessible to the local host during job execution are referred as *dynamic files*. Examples of these files are host requirement, rank expression or performance profile files which are needed to support dynamic resource selection and performance monitoring.

4.1. Static File Management. Data transfer of static files is performed by two executables, *Prolog* and *Epilog*, specified by the user in the `job template` file:

- The *Prolog* executable is responsible for creating the remote experiment directory and transferring the executable and all the files needed for remote execution, such as input or restart files corresponding to the execution architecture.
- The *Epilog* executable is responsible for transferring back output or restart files, and cleaning up the remote experiment directory.

These file transfers are performed through a reverse-server model. The file server (GASS or GridFTP) is started on the local system, and the transfer is initiated on the remote system. Executable and input files are assumed to be stored in an experiment directory. The user provides an executable file per each architecture. Input, output and restart files are by default assumed to be architecture independent (ASCII or HDF [5]). However, the tool is also able to manage architecture dependent formats at an efficiency cost because the future candidate resources may be significantly reduced.

As static file transferring only depends on the *Prolog* and *Epilog* modules, an experienced user could implement its own strategy or adapt a existing one to its particular environment:

- A "simple" file staging strategy in which files go from client to remote host before execution and back to client after execution.
- A more complex file staging strategy in which caches and third-party data transfers are used to minimize data thrashing between client and remote resources.
- A database strategy in which data are extracted from a remote database (for example, a protein data bank [17]).

Prolog and *Epilog* are always submitted to the fork GRAM Job Manager. The *Wrapper*, and therfore the actual job, is submitted to the local queue system GRAM Job Manager. In this way, our tool is well-suited for closed systems such as clusters where only the front-end node is connected to the Internet and the computing nodes are connected to a system area network, so they are not accessible from the client. Other submission toolkits (Nimrod/G [16] or EDG JSS [3]) only use one executable, or job wrapper, to set up the remote system, transfer files, run the executable and retrieve results. A comparison between both alternatives can be found in [1].

4.2. Dynamic File Management. Dynamic file transferring is not possible through a reverse-server model. Closed systems prevent jobs running on "private" computational nodes from updating files on the "public" client host. This problem has been solved by using a *file proxy* (i.e. GASS or GridFTP server) on the front-end node of the remote system. In this way, the running job updates its dynamic files locally within the cluster, via for example NFS, and they are accessible to the client host through the remote file proxy (figure 4.1).

5. Job Execution. A Grid environment presents unpredictable changing conditions, such as dynamic resource load, high fault rate, or continuous addition and removal of resources. Migration is the key issue for adaptive execution of jobs on dynamic Grid environments. Much higher efficiency can be achieved if an application is able to migrate among the Grid resources, adapting itself according to its dynamic requirements, the availability of the resources and the current performance provided by them. In our work, we have considered the following circumstances under which a migration could be initiated:

1. *Grid initiated migration*
    - A new "better" resource is discovered
    - The remote resource or its network connection fails

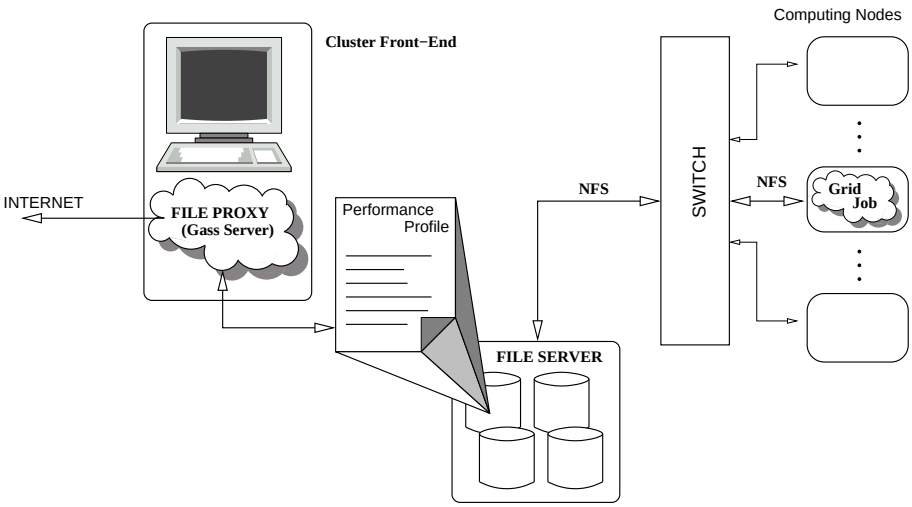

FIG. 4.1. *Access to dynamic files on closed systems through a file proxy.*

- The submitted job is canceled by the resource administrator

2. *Application initiated migration*
   - The application detects performance degradation or performance contract violation
   - Self-migration when the resource requirements of the application change
3. *User initiated migration*
   - The user explicitly requests a job migration

5.1. Migration Policies. Our framework currently considers the following reasons for rescheduling, i.e. situations under which a *migration* event could be triggered:

1. The *Request Manager* receives a *rescheduling* request from the user.
2. The *Performance Monitor* detects a "better" resource in a given *discovering* interval.
3. The *Performance Monitor* detects performance slowdown in a given *monitoring* interval:
   (a) A *rescheduling* action is requested if the *Performance Degradation Evaluator* returns a true value.
   (b) The *Submission Manager* takes count of the accumulated suspension time spent by the job on *pending* and *suspended* Globus states. A *rescheduling* action is requested if the *suspension time* exceeds a given *maximum suspension time* threshold.
4. The *Submission Manager* detects a failure:
   (a) Job cancellation or premature termination are assumed if a non-cero exit code is captured by the *Wrapper*.
   (b) *Wrapper* cancellation or premature termination are assumed if the job exit code is not received by the *Submission Manager*.
   (c) The *Submission Manager* probes periodically at each *polling* interval the connection to the GRAM Job Manager and Gatekeeper to detect remote host, network or GRAM Job Manager crashes

The reason for rescheduling is evaluated to decide if the migration is feasible and worthwhile. Some reasons, like job cancellation or failure, make the *Dispatch Manager* immediately trigger a *migration* event with a new selected host, even if the new host presents lower rank than the current one. Other reasons, like

new resource discovery, make the *Dispatch Manager* trigger a *migration* event only if the new selected host presents a higher rank. Other conditions, apart from the reason for rescheduling and the rank of the new selected host, could be also evaluated: time to finalize, restart file size...

5.2. Performance and Job Monitoring. A *Performance Degradation Evaluator* is periodically executed at each *monitoring* interval by the *Performance Monitor* to evaluate a rescheduling condition. The *Performance Degradation Evaluator* is a script or a binary executable specified in the `job template`, which can also include additional parameters needed for the performance evaluation. This modular approach guarantees the extensibility of the performance and job monitoring. Different strategies could be implemented, from the simplest one based on querying the Grid information system about workload parameters to more advanced strategies based on detection of performance contract violations [41].

The application could be required to have the ability to provide its own `performance profile`. These profiles keep a performance measure which describes the performance activity of the running job. A mechanism to deal with application own metrics is provided since the files processed by the *Performance Degradation Evaluator* could be dynamic and therefore generated by the running job. For example, a `performance profile` file could maintain the time consumed by the code in the execution of a set of given fragments, in each cycle of an iterative method or in a set of given input/output operations. The rescheduling condition verified by the *Performance Degradation Evaluator* could be based on the performance history using advanced methods like fuzzy logic or comparing the performance with the initial performance attained or a base performance. This technique can be also applied for self-migration when, for example, a job changes its dynamic `requirement file`.

A running job could be temporally suspended by the administrator or by the scheduler of the remote resource manager. The *Submission Manager* takes count of the overall suspension time of its job and requests a *rescheduling* action if a given threshold is exceeded. Notice that the *maximum suspension time* threshold is only effective on queue-based resource managers.

In order to detect remote failure, we have followed an approach similar to that provided by Condor/G [27]. The GRAM Job Manager is probed by the *Performance Monitor* periodically at each *polling* interval. If the GRAM Job Manager does not respond, the GateKeeper is probed. If the GateKeeper responds, a new GRAM Job Manager is started to resume watching over the job. If the GateKeeper fails to respond, a *rescheduling* action is immediately requested.

5.3. Execution Management. When a migration order is received by the *Submission Manager*, it cancels the job (if it is still running), invokes the *Epilog* on the current host (if the files are accessible) and the *Prolog* on the new remote host. The local host always keeps the last checkpoint files in case the connection with the remote host fails. Due to the size of the checkpoint files, migration may be an expensive operation that is not suitable for given applications or situations.

The *Submission Manager* uses a wrapper to submit the job on a remote host. The *Wrapper* executes the submitted job and writes its exit code to standard output, so the *Submission Manager* can read it via GASS. Three situations can occur:

- The exit code is set to a zero value: the job is considered to be *done* with a *success* status.
- The exit code is set to a non-zero value: the job is considered to be *done* with a *failed* status.
- The exit code is not set: the job has been *canceled* and, consequently, a job *rescheduling* action is requested.

We would like to remark that the capture of the remote execution exit code allows users to define complex jobs, where each depends on the output and exit code from the previous job. They may even

involve branching and loops.

6. Application Model. We next describe the required modifications in an application source code for its execution through the experimental framework. Figure 6.1 shows a standard application model. The application processes input files in order to generate results written in output files. Standard input, standard output and standard error are by default redirected to the user terminal. Additionally restart files can be generated to provide user level checkpointing. Checkpoint data may be periodically recorded in long-running applications to provide basic fault tolerance. This technique is a common practice in scientific codes involving iterative methods.

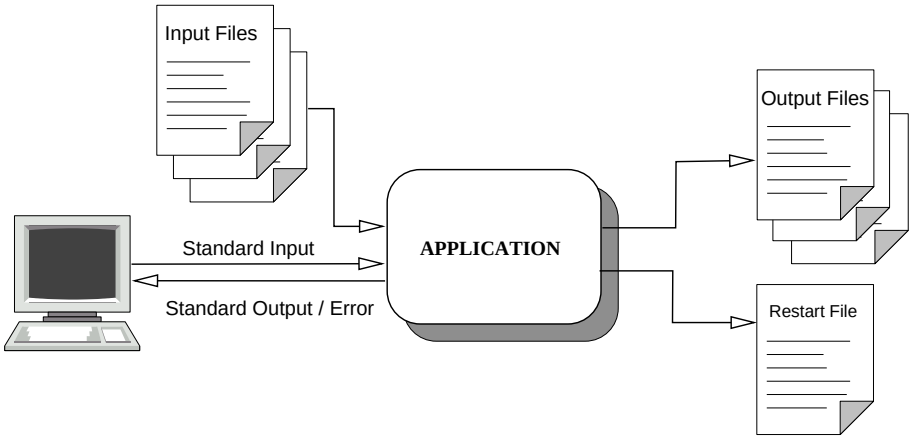

FIG. 6.1. *Standard application model.*

The execution of an application in a Grid environment requires modifications in the previous model. For example, standard input, standard output and standard error can no longer be redirected to user terminal and have to be redirected to files. Moreover, restart files are highly advisable if dynamic scheduling is performed. Our migration framework assumes user-level checkpointing managed by the programmer. Migration is implemented by restarting the job on the new candidate host. Therefore, the job should generate restart files at regular intervals in order to restart execution from a given point. If these files are not provided the job is restarted from the beginning. The restart files have to be architectural independent (ASCII or HDF [5]). In the current version of our tool, users must explicitly manage their own checkpoint data. We expect in future versions to incorporate the Grid Checkpoint Recovery Application Programming Interface under specification by the Grid Checkpoint Recovery Working Group of the Global Grid Forum [4].

The application source code does not have to be modified if the application is not required to be aware of the Grid environment. That is, if the resource selection is based on static requirements and rank expressions, and performance slowdown is detected by performance profiles not generated by the running job. However, our infrastructure requires changing the source code or inserting instrumentation instructions in compiled code when the application takes decisions about resource selection and provides its own `performance profile`. Both the *Resource Selector* and the *Performance Degradation Evaluator* are always executed on the local host but they may process dynamic files generated by the running job on the remote host. Hence, the application model to fully exploit the capabilities of our submission framework is shown in figure 6.2.

7. User Interface. The current version of the experimental framework is a submission command for a single job. The user must perform the following operations:

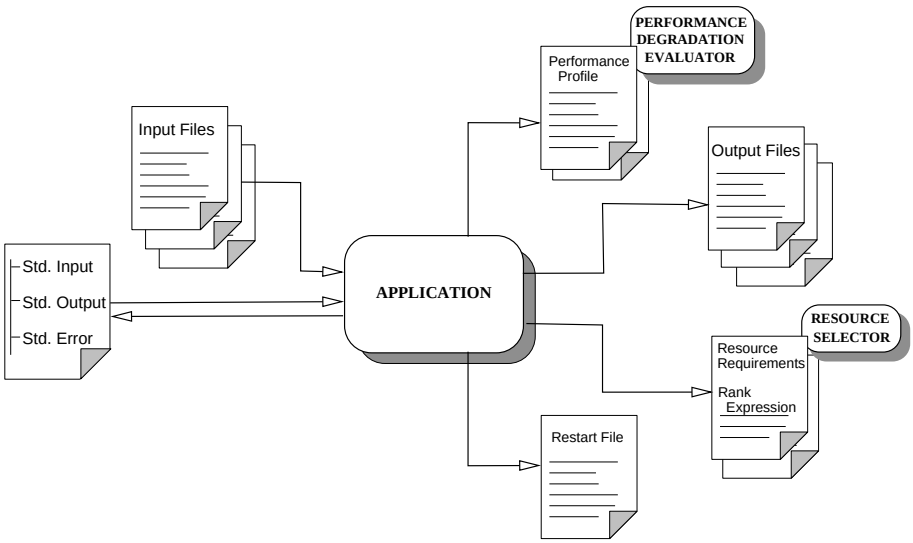

FIG. 6.2. *Application model for the submission framework*

- Build executables for target platforms and place them in the experiment directory. The command is based on Globus and so it is not able to do interactive input/output. The standard input/output can be redirected to files.
- Place input files in the experiment directory.
- Create the job configuration file that specifies the following settings:
  - Modules: *Resource Selector, Performance Degradation Evaluator, Prolog, and Epilog*
  - *Resource Selector* and *Performance Degradation Evaluator* parameters: `host requirement` files, `rank expression` files, `performance profile` files...
  - Static files: input files, executable file, output files, and restart files.
  - Submission agent tuning: *monitoring*, *scheduling*, *polling* and *discovering* intervals.
  - Job execution parameters: application arguments, standard input file, standard output file, standard error file, execution environment, job limits...
- Run the `grid-proxy-init` to set up globus credentials.

The command watches over the execution of the job on the Grid. A log file is updated with changes in the job state, execution host or error condition. The installation of the prototype can be performed by the end user in his home directory without administrator intervention. Although users could write their own modules, the best approach is to provide users with a resource selector, performance degradation evaluation, prolog and epilog module library. In this way, the end user does not have to know about the deployment details of a given Grid. Moreover if a module requires the job to generate dynamic files, a programming API should be also provided to be directly applicable to code developers. They could include its functions to access the adaptive functionality offered by the module.

8. Related Work. The management of jobs within the same department is addressed by many research and commercial systems [22]: Condor, Load Sharing Facility, Sun Grid Engine, Portable Batch System, LoadLeveler... Some of these tools, such as Sun Grid Engine Enterprise Edition [10], also allow the interconnection of multiple departments within the same administrative domain, which is called enterprise interconnection, as long as they run the same distributed resource management software. Other tools, such as Condor Flocking [24], even allow the interconnection of multiple domains, which is called worldwide in-

terconnection. However, they are unsuitable in computational Grids where resources are scattered across several administrative domains, each with its own security policies and distributed resource management systems.

The Globus middleware [25] provides the services needed to enable secure multiple domain operation with different resource management systems and access policies. There are different projects underway, like Condor/G and EveryWare, which are developing user-oriented submission tools over the Globus middleware to simplify the efficient exploitation of a computational Grid.

The Condor/G [27] system combines the advances of the Globus toolkit, security and resource access in multi-domain environments, with the features of the Condor system, management of computation and harnessing of resources within a single administrative domain. The focus of the Condor/G project is to simplify the exploitation of Grid resources by job monitoring, logging, notification, policy enforcement, fault tolerance and credential management.

The EveryWare [42] toolbox is a set of processes and libraries designed to allow an application to harness heterogeneous Grid infrastructures. A single application can simultaneously combine the useful features offered by Globus, Legion, Condor, Netsolve, Java, Windows NT or Unix to form a temporary virtual computational Grid for its own use. EveryWare consists of services implemented over whatever infrastructure is present: communication and process control, performance sensing and forecasting, scheduling agents and persistent state management.

Other submission tools, like AppLeS, Nimrod/G, and ILab, provide extra functionalities like management of array jobs for parametric studies or definition of dependencies between jobs for complex tasks. Since our tool currently does not provide this feature they cannot be compared in this context. Future versions will include support for parametric studies and complex tasks.

AppLeS (Application Level Scheduling) [18] is a job submission tool that predicts the state of a system when the application will be executing by using NWS, user preferences and application performance models. The focus of the AppLeS project is on scheduling, it builds scheduling agents for each application class. For example, the AppLeS Parameter-Sweep Template (APST) targets Parameter Sweep Applications (PSAs) and is composed by a Controller (similar to our *Request Manager*), a Scheduler, similar to our *Dispatch Manager*, an Actuator, similar to our *Submission Manager*, and a Meta-Data Bookkeeper to provide access to several information services.

Nimrod/G [15], Grid-enabled version for Nimrod, creates parameter studies, performs resource discovery and scheduling, and manages job submission and remote storage. The focus of this tool is on scheduling schemes based on computational economy. Individual resources are agglomerated to satisfy deadline or budget user requirements. The ILab [21] project is focused on the development of an advanced graphical user tool for parameter study creation that includes a Globus job model.

All aforementioned application schedulers share many features, with differences in the way they are implemented [29]. We believe that there is no better tool, each focuses on a specific issue and contributes significant improvements in the field. The outstanding feature of our framework is its modular and so extensible approach for adaptive execution of applications on dynamic Grid environments. In fact, to the best of the authors' knowledge, none of previous tools addresses adaptation to changing environments.

The need of a nomadic migration approach for job execution on a Grid environment has been previously discussed in [30]. The prototype of a migration framework, called the "Worm", was executed in the Egrid testbed [23]. The "Worm" was implemented within the Cactus programming environment. Cactus is an open source problem solving environment for the construction of parallel solvers for partial differential equations

that enables collaborative code development between different groups [13]. The extension of the "Worm" migration framework to make use of Grid services is described in [30].

Adaptive execution is also being explored in the context of the Grid Application Development Software (GrADS) project [14]. The aim of the GrADS projects is to simplify distributed heterogeneous computing in the same way that the World Wide Web simplified information sharing over the Internet. GrADS provides new external services to be accessed by Grid users and, mainly, by application developers in order to develop grid-aware applications. Its execution framework [28] is based on three components: Configurable Object Program, which contains application code and strategies for application mapping, Resource Selection Model, which provides estimation of the application performance on specific resources, and Contract Monitor, which performs job interrupting and remapping when performance degradation is detected. GrADS is an ambitious project that involves several outstanding research groups in Grid technology. The GrADS project is under development. However, two preliminary versions of this execution model have been prototyped in two well-known applications: Cactus and ScaLAPACK.

Cactus incorporates adaptive resource selection mechanisms (Resource Locator Service) that allow automatic application migration to "better" resources (Migrator Service) to deal with changing resource characteristics and adaptive resource selection mechanisms [12]. The adaptation to dynamic Grid environments has been studied by job migration to "faster/cheaper" systems, considered when better systems are discovered, when requirements change or when job characteristics change.

In the context of the GrADS project, it has been demonstrated the usefulness of a Grid to solve large numerical problems by integrating numerical libraries like ScaLAPACK into the GrADS system [33]. The Resource Selector component accesses MDS and NWS to provide the information needed by the Performance Modeler to apply an application specific execution model and so obtain a list of final candidate hosts. The list is passed to the Contract Developer that approves the contract for the Application Launcher. The submitted application is monitored by the Contract Monitor through the Autopilot manager [36] that can detect contract violations by contacting the sensors and determining if the application behaves as predicted by the model. In the future, a Scheduler could receive this information and migrate the application if the contract is violated.

The aim of the GridWay project is similar to that of the GrADS project: simplify distributed heterogeneous computing. However, its scope is different. Our experimental framework provides a submission agent that incorporates the runtime mechanisms needed for transparently executing jobs on a Grid. In fact, our framework could be used as a building block for much more complex service-oriented Grid scenarios like GrADS. Other projects have also addressed resource selection, data management, and execution adaptation. We do not claim innovation in these areas, but remark the advantages of our modular architecture for job adaptation to a dynamic environment:

- It is not bounded to a specific class of application generated by a given programming environment, which extends its application range.
- It does not require new services, which considerably simplify its deployment.
- It does not necessarily require code changes, which allows reusing of existing software.
- It is extensible, which allows its communication with the Grid services available in a given testbed.

We would like to remark that the experimental framework does not require new system software to be installed in the Grid resources. The framework is functional on any Grid testbed based on Globus. We believe that is an important advantage because of socio-political issues; cooperation between different research centers, administrators and users can be difficult.

9. Experiences. We next demonstrate the capabilities of the previously described experimental framework. The behavior of its adaptation functionality is demonstrated in the execution of a computational fluid dynamics (CFD) code. The target application solves the 3D incompressible Navier-Stokes equations in the simulation of a boundary layer over a flat plate. The numerical core consists in an iterative robust multigrid algorithm characterized by a compute-intensive execution profile [32, 34]. The application generates checkpoint files at each multigrid iteration. A part of the `job template` is shown in figure 9.1.

```
#------------------------------
# Job Template, CFD simulation
#------------------------------

 EXECUTABLE_FILE=NS3D

 EXECUTABLE_ARGUMENTS=input

 STDOUT_FILE=ns3d.out

 STDIN_FILE=/dev/null

 STDERR_FILE=ns3d.err

 INPUT_FILES="input grid"

 OUTPUT_FILES="residual profile"
```

FIG. 9.1. *Job template for the target application.*

The experiments have been carried out on the TRGP (Tidewater Research Grid Partnership) Grid. The TRGP Grid is made up of two virtual organizations (VO), ICASE and The College of William and Mary computational facilities, that provide an overall performance of 209 Gflops, 116 GB of RAM memory and 3.4 TB of disk. This testbed is a highly heterogeneous environment consisting of different processor architectures (UltraSPARC and Intel), operating systems (Solaris and Linux), local resource managers (PBS and fork) and interconnection networks (Fast Ethernet, Gigabit Ethernet, Myrinet and Giganet). See appendix A for a detailed description of TRGP.

9.1. Description of the Experiment. The experiments have been performed with the following settings:

1. **Resource Selector.** Resource selection consists in a shell script that queries MDS for potential execution hosts, attending the following criteria:
   - Host requirements are specified in a `host requirement` file, which can be dynamically generated by the running job. The host requirement setting is a LDAP (Lightweight Directory Access Protocol) filter, which is used by the *Resource Selector* to query MDS and so obtain a preliminary list of potential hosts. In the experiments below, we will impose two constraints, an SPARC architecture and a minimum main memory of 512MB, enough to accommodate the CFD simulation. The `host requirement` file will be of the form:

     ```
     (& (Mds-Computer-isa=sparc)
     (Mds-Memory-Ram-freeMB>=512))
     ```

     The *Resource Selector* also performs an user authorization filter (via GRAM dry-run) on those hosts.
   - A rank is assigned to each potential host. Since our target application is a computing intensive simulation, the rank expression benefits those hosts with less workload and so better

performance. The following expression was considered:

$$rank = \begin{cases} FLOPS & \text{if} \quad CPU_{15} \geq 1; \\ FLOPS \cdot CPU_{15} & \text{if} \quad CPU_{15} < 1. \end{cases} \tag{9.1}$$

where $FLOPS$ is the peak performance achievable by the host cpu, and CPU_{15} is the total free CPU time in the last 15 minutes, as provided by the MDS default scheme. It is interesting to remark that in the case of heterogeneous clusters $FLOPS$ is the average performance of all computing nodes. However other alternatives have been proposed in the literature. For example, in the NorduGrid project [6] a more conservative approach is taken, equaling $FLOPS$ to the performance of the slowest node.

2. **Performance Degradation Evaluator.** The performance of the target application is based on the time spent in each multigrid iteration. The time consumed in each iteration is appended by the running job to a `performance profile` file specified as dynamic in the `job template`. The *Performance Degradation Evaluator* verifies at each *monitoring* interval if the time consumed in each iteration is higher than a given threshold. This performance contract (iteration threshold) and contract monitor (*Performance Degradation Evaluator*) are similar to those used in [12].
3. **File Staging.** The *Prolog* and *Epilog* modules were implemented with a shell script that uses Globus transfer tools (i.e. `globus-url-copy`) to move files to/from the remote host. These files include the executable itself and a computational mesh (needed for the CFD simulation) in the stage-in step (*Prolog*), and the output solution or restart file in the stage-out step (*Epilog*). Transfer sizes and *Prolog/Epilog* execution times between TRGP VO's are shown in table 9.1, for both first and restart executions.

TABLE 9.1
Prolog and Epilog execution times between TRGP virtual organizations

Stage Step		Size (Kbytes)	Transfer Time (sec.)	
			ICASE ↔ WMCOMPSCI	ICASE ↔ ICASE
First Execution	Prolog	3080	57	34
	Epilog	13	17	11
Restart Execution	Prolog	4037	75	45
	Epilog	970	29	19

4. **Intervals.** *Monitoring*, *polling* and *scheduling* intervals were set to 10 seconds.

9.2. Results. The scenarios described below were artificially created. However, they resembles real situations that may happen in a Grid.

1. **Periodic rescheduling to detect new resources.**
   In this case, the *discovering* interval has been deliberately set to 60 seconds in order to quickly re-evaluate the performance of the resources. The execution profile of the application is presented in figure 9.2. Initially, only ICASE hosts are available for job submission, since SciClone has been shutdown for maintenance. The *Resource Selector* chooses urchin to execute the job, and the files are transferred (prolog and submission in time steps 0s-34s). The job starts executing at time step 34s. A *discovering* period expires at time step 120s and the *Resource Selector* finds SciClone to present higher rank than the original host (time steps 120s-142s). The migration process is then initiated

(cancellation, epilog, prolog and submission in time steps 142s-236s). Finally the job completes its execution on SciClone.

Figure 9.2 shows how the overall execution time is 42% lower when the job is migrated. This speedup could be even greater for larger execution times. Note that migration time, 95 seconds, is about 20% of the overall execution time. We would like to remark that the framework also allows the user to force a job reschedule to detect new resources.

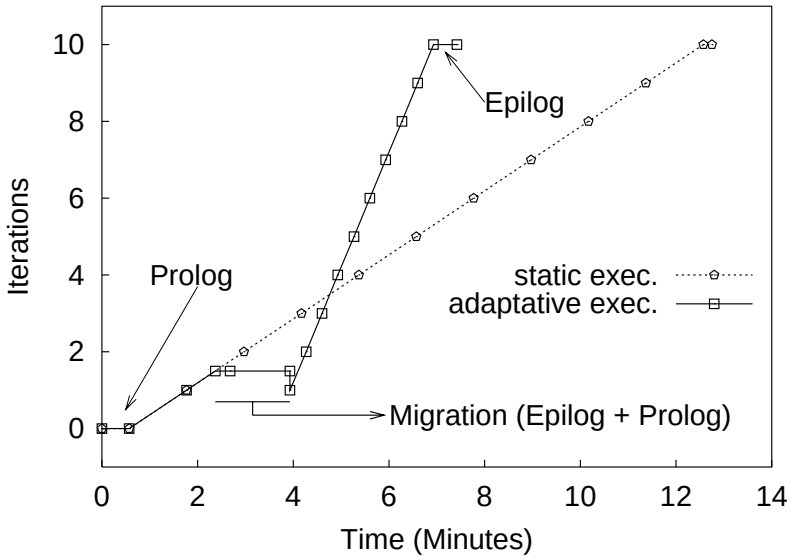

FIG. 9.2. *Execution profile of the CFD code when a new "better" resource is detected.*

2. **Performance degradation detected using a `performance profile` dynamic file.** The *Resource Selector* finds whale to be the best resource, and job is submitted (prolog and submission in time step 0s-34s). However, whale is overloaded with a compute-intensive workload at time step 34s. As a result, a performance degradation is detected when the iteration time exceeds the iteration time threshold (40 seconds) at time step 209s. The job is then migrated to SciClone (cancellation, epilog, prolog and submission in time steps 209s-304s), where it continues executing from the last checkpoint context. The execution profile for this situation is presented in figure 9.3. In this case the overall execution time is 35% lower when the job is migrated. The cost of migration, 95 seconds, is about 21% of the execution time.
3. **Self-migration when job requirements change.** The multigrid method in the CFD code is a full multigrid algorithm (FMG). The computations are initiated in the coarsest grid, once the discrete system is solved the solution is transferred to the next finer level. The prolongated solution is then used as initial guess of the multigrid method in such level, this procedure is repeated until the finest grid is reached. The running job requests a higher amount of RAM memory, from 512 MB to 2 GB, before processing the finest grid in order to assure that it will fit in main memory. The job changes the `host requirement` dynamic file into:

```
(& (Mds-Computer-isa=sparc)
(Mds-Memory-Ram-freeMB>=2040))
```

and request a rescheduling by writing a high iteration time in the `performance profile` dynamic file. As result of this update the job is migrated from urchin, where was initially submitted, to whale.

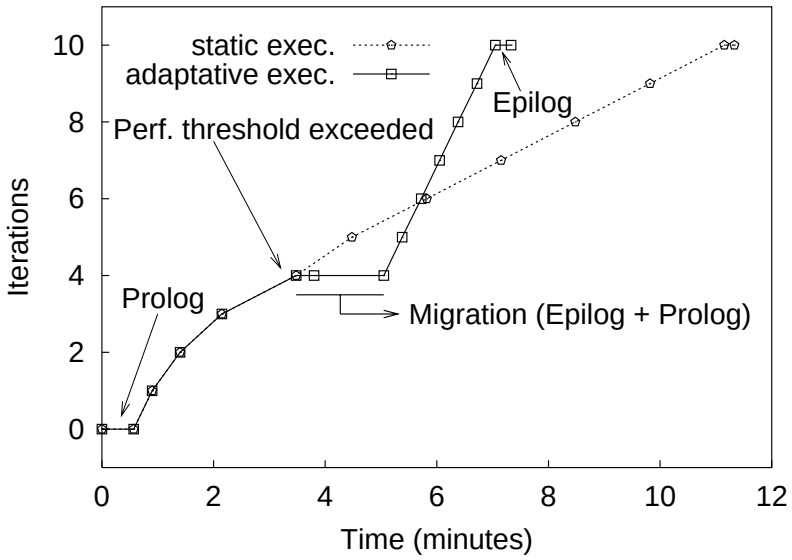

FIG. 9.3. *Execution profile of the application when a workload is executed.*

4. **The remote resource or its network connection fails.** The *Resource Selector* finds whale to be the best resource, and the files are transferred (*Prolog* and submission in time steps 0s-45s). Initially, the job runs normally. At time step 125s, whale is disconnected. After 50 seconds, a connection failure is detected and the job is migrated to SciClone (*Prolog* and submission in time steps 175s-250s). The application is executed again from the beginning because the local host does not have access to the checkpoint files generated on whale. Finally the job completes its execution on SciClone.

10. Conclusions and Future Work. The exploitation in an aggregated form of geographically distributed resources is far from being straightforward. In fact, the Grid will be for many years a challenging area due to its dynamic and complex nature. We believe that the GridWay submission framework is a step forward to insulate the application developer from the complexity of the Grid submission.

The experimental framework provides the runtime mechanisms needed for dynamically adapting an application to a changing Grid environments. The core of the framework is a personal *Submission Agent* that transparently performs all submission stages and watches over the efficient execution of the job. Adaptation to the dynamic nature of the Grid is achieved by implementing automatic application migration following performance degradation, "better" resource discovery, requirement change, owner decision or remote resource failure. The application has the ability to take decisions about resource selection and to self-migrate to a new resource. Its design is based on a modular architecture to allow extensibility of its capabilities. An experienced user can incorporate different resource selection and performance degradation strategies to perform job adaptation.

The experimental results are promising because they show how application adaptation achieves enhanced performance. The response time of the target application is reduced when it is submitted through the experimental framework. Simultaneous submission of several applications could be performed in order to harness the highly distributed computing resources provided by a Grid. Our framework is able to efficiently manage applications suitable to be executed on dynamic conditions. Mainly, those that can migrate their

state efficiently.

The described submission toolbox addresses important issues: performance and job monitoring, migration by job restarting, data movement, resource selection and remote connection failure. However, the toolbox should address other desirable issues like: credential expiry, improved checkpoint management, client failure tolerance, improved API and command line interface, graphical user interface and array jobs.

Acknowledgments. This work has been performed in part using computational facilities at The College of William and Mary, which were enabled by grants from Sun Microsystems, the National Science Foundation, and Virginia's Commonwealth Technology Research Fund. We would like to thank Thomas Eidson and Tom Crockett for their help with TRGP.

REFERENCES

[1] *Comparison of HTB and Nimrod/G: Job Dispatch.* Available at http://www-unix.mcs.anl.gov/~giddy/comp.html.

[2] *Coral Cluster Project.* http://www.icase.edu/CoralProject.html.

[3] *European DataGrid Workload Management Work Package.* http://server11.infn.it/workload-grid/.

[4] *Global Grid Forum Grid Checkpoint Recovery Working Group.* http://gridcpr.psc.edu/GGF.

[5] *Hierarchical Data Format Version 5.* http://hdf.ncsa.uiuc.edu/HDF5.

[6] *The Nordugrid Project.* http://www.nordugrid.org/.

[7] *SciClone Cluster Project.* http://www.compsci.wm.edu/SciClone/.

[8] *Tidewater Research Grid Partnership.* http://www.tidewaterrgp.org.

[9] *Glue Schema.* Available at http://www.hicb.org/glue/glue.htm, 2002.

[10] *How Sun Grid Engine, Enterprise Edition 5.3 Works*, Tech. Rep. Sun Microsystems White Paper, Sun Microsystems, 2002. Available at http://wwws.sun.com/software/gridware/sgeee53/wp-sgeee/index.html.

[11] *WP1 Workload Management Software.* Available at http://server11.infn.it/workload-grid/documents.html, 2002.

[12] G. Allen, D. Angulo, I. Foster, G. Lanfermann, C. Liu, T. Radke, E. Seidel, and J. Shalf, *The Cactus Worm: Experiments with Dynamic Resource Discovery and Allocation in a Grid Environment*, in Proceedings of European Conference on Parallel Computing (EuroPar) 2001, Lecture Notes in Computer Science, August 2001. Manchester, UK.

[13] G. Allen, T. Goodale, G. Lanfermann, T. Radke, and E. Seidel, *The Cactus Code: A Problem Solving Environment for the Grid*, in Proceedings of the 9th IEEE International Symposium on High Performance Distributed Computing (HPDC9), May 2001. Pittsburg Pennsylvania, USA.

[14] F. Berman, A. Chien, K. Cooper, J. Dongarra, I. Foster, D. Gannon, L. Johnsson, K. Kennedy, C. Kesselman, J. Mellor-Crummey, D. Reed, L. Torczon, and R. Wolski, *The GrADS Project: Software Support for High-Level Grid Application Development*, International Journal of High Performance Computing Applications, 15 (2001), pp. 327–334.

[15] R. Buyya, D. Abramson, and J. Giddy, *Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computation Grid*, in Proceedings of the 4th IEEE International Conference on High Performance Computing in Asia-Pacific Region (HPC Asia), 2000. Beijing, China.

[16] R. Buyya, D. Abramson, and J. Giddy, *A Computational Economy for Grid Computing and its Implementation in the Nimrod-G Resource Broker*, Future Generation Computer Systems, (2002). Elsevier Science (to appear).

[17] R. Buyya, K. Branson, J. Giddy, and D. Abramson, *The Virtual Laboratory: A Toolset for Utilising the World-Wide Grid to Design Drugs*, in Proceedings of the 2nd IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2002), May 2002. Berlin, Germany.

[18] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman, *Heuristics for Scheduling Parameter Sweep Applications in Grid Environments*, in Proceedings of the 9th Heterogeneous Computing Workshop (HCW2000), May 2000. Cancun, Mexico.

[19] A. Chervenak, E. Deelman, I. Foster, L. Guy, W. Hoschek, A. Iamnitchi, C. Kesselman, P. Kunst, M. Ripenu, B. Schwartzkopf, H. Stockinger, K. Stockinger, and B. Tierney, *Giggle: A Framework for Constructing Scalable Replica Location Services*, in Proceedings of Supercomputing 2002 (SC2002), 2002.

[20] H. Dail, H. Casanova, and F. Berman, *A Modular Scheduling Approach for Grid Application Development Environments*, Tech. Rep. CS2002-0708, UCSD CSE, 2002. Submitted to Journal of Parallel and Distributed Processing.

[21] A. DeVivo, M. Yarrow, and K. M. McCann, *A Comparison of Parameter Study Creation and Job Submission Tools*, Tech. Rep. NAS-01-002, NASA, 2001. Available at http://www.nas.nasa.gov/Research/Reports/Techreports/2001/.

[22] T. El-Ghazawi, K. Gaj, N. Alexandinis, and B. Schott, *Conceptual Comparative Study of Job Management Systems*, tech. rep., George Mason University, February 2001. Available at http://ece.gmu.edu/lucite/reports/.

[23] G. A. et al, *Early Experiences with the Egrid Testbed*, in Proceedings of the IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2001), May 2001. Brisbane, Australia.

[24] X. Evers, J. F. C. M. de Jongh, R. Boontje, D. H. J. Epema, and R. van Dantzig, *Condor Flocking: Load Sharing Between Pools of Workstations*, Tech. Rep. DUT-TWI-93-104, Delft, The Netherlands, 1993.

[25] I. Foster and C. Kesselman, *Globus: A Metacomputing Infrastructure Toolkit*, Intl. J. Supercomputer Applications, 11 (1997), pp. 115–128.

[26] ——, *The Grid: Blueprint for a New Computing Infrastructure*, Morgan-Kaufman, 1999.

[27] J. Frey, T. Tannenbaum, I. Foster, M. Livny, and S. Tuecke, *Condor/G: A Computation Management Agent for Multi-Institutional Grids*, in Proceedings of the 10th IEEE Symposium on High Performance Distributed Computing (HPDC10), August 2001. San Francisco California, USA.

[28] K. Kennedy, M. Mazina, J. Mellor-Crummey, K. Cooper, L. Torezon, F. Berman, A. Chien, H. Dail, O. Sievert, D. Angulo, I. Foster, D. Gannon, L. Johnsson, C-Kesselman, R. Aydt, D. A. Reed, J. Dongarra, S. Vadhiyar, and R. Wolski, *Toward a Framework for Preparing and Execution Adaptive Grid Applications*, in Proceedings of NSF Next Generation Systems Program Workshop, International Parallel and Distributed Processing Symposium, April 2002. Fort Lauderdale California, USA.

[29] K. Krauter, R. Buya, and M. Maheswaran, *A Taxonomy and Survey of Grid Resource Management Systems for Distributed Computing*, International Journal of Software: Practice and Experience (SPE), 32 (2002), pp. 135–164.

[30] G. Lanfermann, G. Allen, T. Radke, and E. Seidel, *Nomadic Migration: A New Tool for*

Dynamic Grid Computing, in Proceedings of the 10th Symposium on High Performance Distributed Computing (HPDC10), August 2001. San Francisco California, USA.

[31] C. Liu, L. Yang, I. Foster, and D. Angulo, *Design and Evaluation of a Resource Selection Framework for Grid Applications*, in Proceedings of the 11th IEEE Symposium on High-Performance Distributed Computing, 2002.

[32] R. S. Montero, I. M. Llorente, and M. D. Salas, *Robust Multigrid Algorithms for the Navier-Stokes Equations*, Journal of Computational Physics, 173 (2001), pp. 412–432.

[33] A. Petitet, S. Balckford, J. Dongarra, B. Ellis, G. Fagg, K. Roche, and S. Vadhiyar, *Numerical Libraries and the Grid: The GrADS Experiments with ScaLAPack*, International Journal of High Performance Computing Applications, 15 (2001), pp. 359–374.

[34] M. Prieto, R. Santiago, I. M. Llorente, and F. Tirado, *A Multigrid Solver for the Incompressible Navier-Stokes Equations on a Beowulf-class System*, in Proceedings of the 30th International Conference on Parallel Processing (ICPP01), 2001. Valencia, Spain.

[35] J. G. R. Buyya and D. Abramson, *A Case for Economy Grid Architecture for Service-Oriented Grid Computing*, in Proceedings of the 10th IEEE International Heterogeneous Computing Workshop (HCW 2001), April 2001. San Francisco California, USA.

[36] R. L. Ribler, J. S. Vetter, H. Simitci, and D. A. Reed, *Autopilot: Adaptive Control of Distributed Applications*, in Proceedings of the 7th IEEE International Symposium on High Performance Distributed Computing (HPDC7), July 1998. International Parallel and Distributed Processing Symposium 2002.

[37] J. M. Schopf, *A General Architecture for Scheduling on the Grid*. Available at http://www-unix.mcs.anl.gov/∼schopf, 2002. Submitted to special issue of Journal of Parallel and Distributed Computing on Grid Computing.

[38] J. M. Schopf and B. Nitzberg, *Grids: The Top Ten Questions*. Available at http://www-unix.mcs.anl.gov/∼schopf, 2002. To appear in Scientific Programming, special issue on Grid Computing.

[39] M. Sgaravatto, *WP1 Inputs to the Datagrid Grid Information Service Schema Specification*. Available at http://server11.infn.it/workload-grid/documents.html, 2001.

[40] S. S. Vanhiyar and J. J. Dongarra, *A Metascheduler for The Grid*, in Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing (HPDC'02), July 2002. Edinburgh, UK.

[41] F. Vraalsen, R. A. Aydt, C. L. Mendes, and D. A. Reed, *Performance Contracts: Predicting and Monitoring Grid Application Behavior*, in Proceedings of the 2nd International Workshop on Grid Computing, November 2001. Denver Colorado, USA.

[42] R. Wolski, J. Brevik, C. Krintz, G. Obertelli, N. Spring, and A. Su, *Running Everyware on the Computational Grid*, in Proceedings of the SuperComputing (SC98), High Performance Computing Challenge, November 1998. Orlando Florida, USA.

[43] R. Wolski, N. Spring, and J. Hayes, *The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing*, Journal of Future Generation Computing Systems, 15 (1999), pp. 757–768.

Appendix A. The Tidewater Research Grid.[1]

The Tidewater Research Grid Partnership (TRGP) [8] was started in summer 2001 to foster the development and use of Grid computing for a variety of applications in the computational sciences. TRGP members at the end of August 2002, when the experimental results were taken, include ICASE and the College of William & Mary (see figure A.1 for a schematic representation).

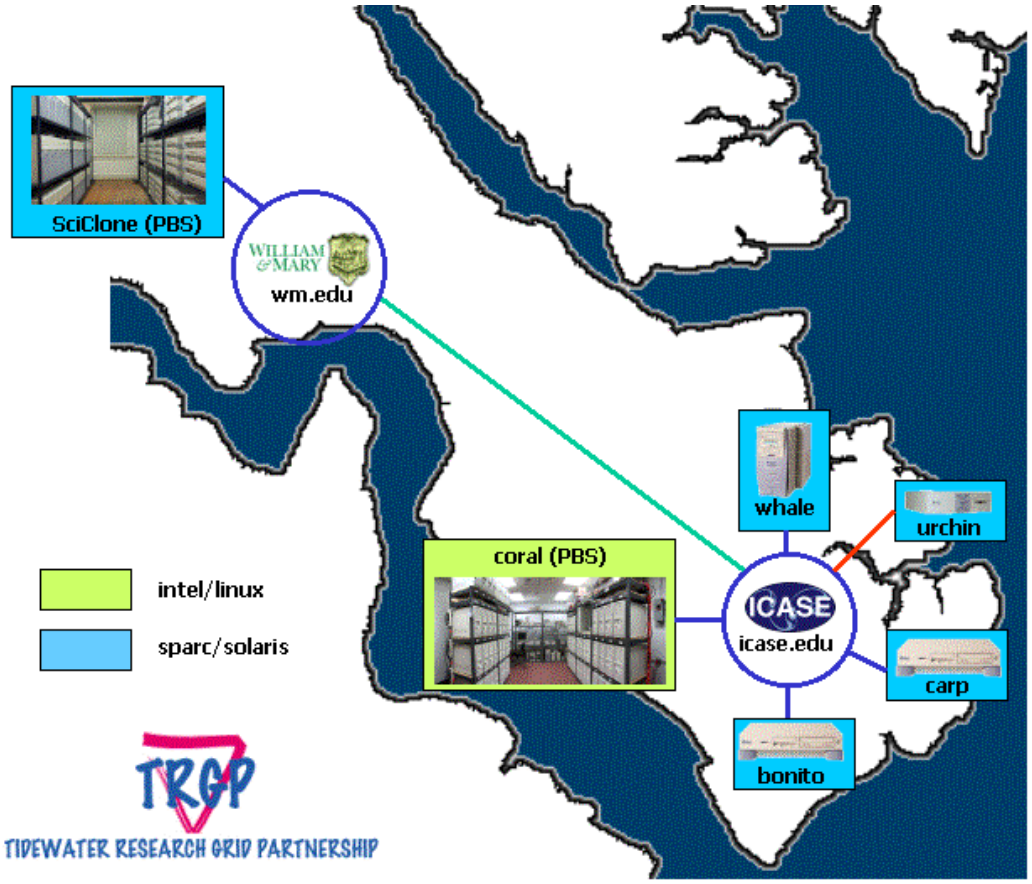

FIG. A.1. *Schematic representation of the TRGP Grid.*

The TRGP Grid is highly heterogeneous. It consists of systems with different processor architectures (UltraSPARC and Intel), operating systems (Solaris and Linux), local resource managers (PBS and fork) and interconnecting networks (Fast Ethernet, Gigabit Ethernet, Myrinet and Giganet). The workload is dynamic since resources are mainly exploited by internal users. Each resource provides Globus 2.0 GSI, GRAM, GridFTP, GASS and MDS components. The hierarchical TRGP information service is shown in figure A.2.

The TRGP Grid consists of the following computing resources:

- Coral [2] is a 68-node Linux Intel Pentium cluster with a total of 103 processors, 55GB of RAM memory, 1.9TB of raw disk and 89 Gflops peak floating-point performance. Table A.1 summarizes the main features of its architecture.
- SciClone [7] is a 108-node Solaris Sun UltraSPARC cluster with a total of 160 processors, 54 GB of

[1] This section presents the TRGP configuration at the end of August 2002, when the experimental results were taken.

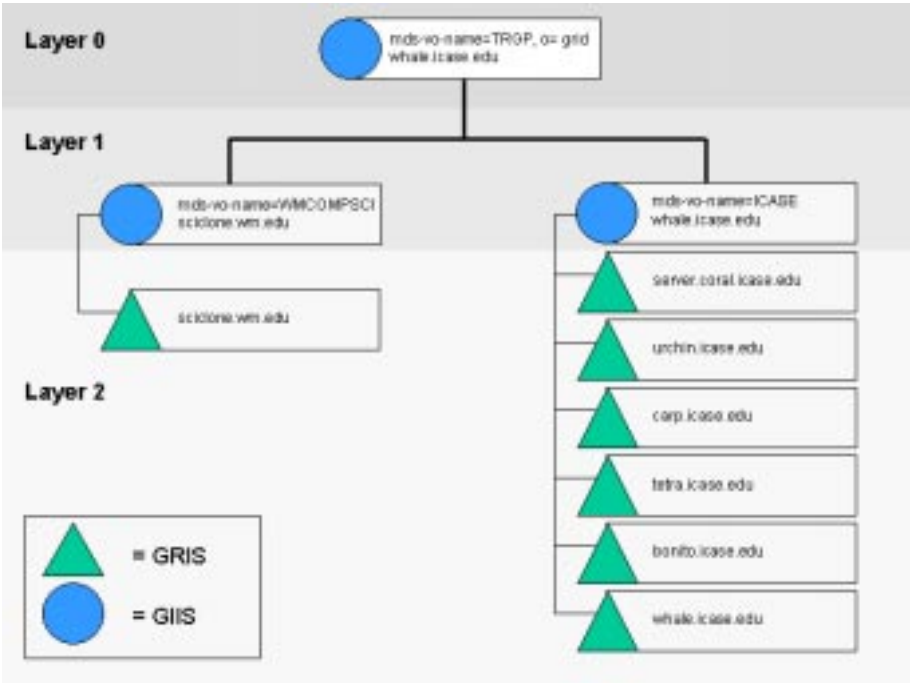

FIG. A.2. *TRGP GIIS hierarchy configuration.*

RAM memory, 1.6 TB of disk capacity and 115 Gflops peak floating-point performance. Table A.2 summarizes the main features of its architecture.

- 5 Sun Solaris workstations (whale, urchin, carp, tetra and bonito) with a total of 7 processors, 5.7 GB of RAM memory, 0.2 TB of disk and 4.9 Gflops peak floating-point performance.

The complete Grid brings together 181 nodes with 271 processors, 116 GB of RAM memory, 3.4 TB of disk and 209 Gflops of peak performance. The interconnection between both sites is performed by a "public" non-dedicated network. The ICASE workstations and servers are interconnected by a Fast Ethernet switched network.

TABLE A.1
Coral *cluster*

Subcluster	**Nodes**	**CPU Type**	**Storage (per node)**		**Network**
			RAM	**HD**	
Phase I	8	Single 400MHz/0.5MB PII	384MB	6.5GB (EIDE)	Fast Ether.
Phase II	16	Dual 500MHz/0.5MB PIII	512MB	14.4GB (EIDE)	Fast Ether. Giganet
Phase III	16	Dual 800MHz/0.5MB PIII	1GB	30GB (EIDE)	Fast Ether. Giganet
Phase IV	24	Single 1.7GHz/0.25MB P4	1GB or 2GB	6.5GB or 20GB (EIDE)	Fast Ether.
Front-end	1	Dual 400MHz/0.5MB PII	512MB	76GB* (EIDE)	Gigabit Ether.
File Server	2	Dual 500MHz/0.5MB PIII	512MB	108GB* (SCSI)	Gigabit Ether.
	1	Dual 400MHz/0.5MB PII	384MB	640GB* (EIDE) 18GB* (SCSI)	
Subcluster interconnection	• Phase II and Phase III nodes share a 32-port Fast Ethernet and a 32-port Giganet switches • Phase I and Phase IV nodes share a 32-port Fast Ethernet switch • There is a dual Gigabit Ethernet link between both (Phase II/III and Phase I/IV) Fast Ethernet switches • A Gigabit Ethernet switch interconnects both (Phase II/III and Phase I/IV) Fast Ethernet switches, the front-end server, and the file server				
O. S.	Linux Red Hat 7.2				
GRAM service	Fork Job Manager PBS Job Manager				

* Unformatted capacity

TABLE A.2
Sciclone cluster

Subcluster	**Nodes**	**CPU Type**	**Storage (per node)**		**Network**
			RAM	**HD**	
Typhoon	64	Single 333MHz/2MB UltraSPARC IIi (Sun Ultra 5)	256MB	9.1GB (EIDE)	Fast Ether.
Tornado	32	Dual 360MHz/4MB UltraSPARC II (Sun Ultra 60)	512MB	18.2GB (SCSI)	Fast Ether. Myrinet
Hurricane	4	Quad 450MHz/4MB UltraSPARC II (Sun Enterprise 420R)	4GB	18.2GB (SCSI)	Gigabit Myrinet
	1	Dual 360MHz/4MB UltraSPARC II (Sun Ultra 60)	512MB	18.2GB (SCSI)	
Front-end	1	Dual 400MHz/4MB UltraSPARC II (Sun Ultra 60)	1GB	91GB (SCSI)	Gigabit Myrinet
File Server (Gulfstream)	6	Dual 360MHz/4MB UltraSPARC II (Sun Ultra 60)	512MB	45.5GB or 63.7GB (SCSI)	Gigabit Myrinet
Subcluster interconnection	• Typhoon nodes are interconnected by two 36-port Fast Ethernet switches • Tornado nodes are interconnected by one 36-port Fast Ethernet switch • Gulfstream nodes are interconnected by one 12-port Gigabit Ethernet switch • Hurricane nodes are interconnected by one 12-port Gigabit Ethernet switch • A 12-port Gigabit Ethernet switch interconnects Typhoon and Tornado 36-port Fast Ethernet switches, and Gulfstream 12-port Gigabit Ethernet switch • Three Gigabit links interconnects Gulfstream and Hurricane 12-port Gigabit Ethernet switches • A 64-port Myrinet switch interconnects Tornado, Gulfstream, and Hurricane nodes				
O. S.	Sun Solaris 7				
GRAM service	Fork Job Manager PBS Job Manager				

TABLE A.3
ICASE workstations

Workstation	**O.S.**	**GRAM**	**CPUs**	**Storage (per node)**	
				RAM	**HD**
Whale	Sun Solaris 7	Fork	Dual 450MHz/4MB UltraSPARC II (Sun Ultra 80)	4GB	40GB (SCSI)
Urchin	Sun Solaris 7	Fork	Dual 168MHz/0.5MB UltraSPARC I (Sun Ultra 2)	1GB	150GB (SCSI)
Carp	Sun Solaris 7	Fork	Single 450MHz/0.2MB UltraSPARC IIi (Sun Ultra 5)	256MB	8.1GB (EIDE)
Tetra	Sun Solaris 7	Fork	Single 400MHz/0.2MB UltraSPARC IIi (Sun Ultra 5)	256MB	19GB (EIDE)
Bonito	Sun Solaris 7	Fork	Single 360MHz/0.2MB UltraSPARC IIi (Sun Ultra 5)	256MB	8.1GB (EIDE)

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	November 2002	Contractor Report

4. TITLE AND SUBTITLE
AN EXPERIMENTAL FRAMEWORK FOR EXECUTING APPLICATIONS IN DYNAMIC GRID ENVIRONMENTS

5. FUNDING NUMBERS
C NAS1-97046
WU 505-90-52-01

6. AUTHOR(S)
Eduardo Huedo, Ruben S. Montero, and Ignacio M. Llorente

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
ICASE
Mail Stop 132C
NASA Langley Research Center
Hampton, VA 23681-2199

8. PERFORMING ORGANIZATION REPORT NUMBER
ICASE Report No. 2002-43

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
Langley Research Center
Hampton, VA 23681-2199

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
NASA/CR-2002-211960
ICASE Report No. 2002-43

11. SUPPLEMENTARY NOTES
Langley Technical Monitor: Dennis M. Bushnell
Final Report
Submitted to the International Parallel and Distributed Processing Symposium (IPDPS '03).

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified–Unlimited
Subject Category 60, 61
Distribution: Nonstandard
Availability: NASA-CASI (301) 621-0390

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The Grid opens up opportunities for resource-starved scientists and engineers to harness highly distributed computing resources. A number of Grid middleware projects are currently available to support the simultaneous exploitation of heterogeneous resources distributed in different administrative domains. However, efficient job submission and management continue being far from accessible to ordinary scientists and engineers due to the dynamic and complex nature of the Grid. This report describes a new Globus framework that allows an easier and more efficient execution of jobs in a "submit and forget" fashion. Adaptation to dynamic Grid conditions is achieved by supporting automatic application migration following performance degradation, "better" resource discovery, requirement change, owner decision or remote resource failure. The report also includes experimental results of the behavior of our framework on the TRGP testbed.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Grid technology, adaptive execution, job migration, Globus	30
	16. PRICE CODE: A03

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified		