Experiences about Job Migration on a Dynamic Grid Environment

Ruben S. Montero[a*], Eduardo Huedo[b] and Ignacio M. Llorente[ab]

[a]Departamento de Arquitectura de Computadores y Automática,
Universidad Complutense, 28040 Madrid, Spain

[b]Centro de Astrobiología, CSIC-INTA, Associated to NASA Astrobiology Institute,
28850 Torrejón de Ardoz, Spain

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. However, efficient job submission and management continue being far from accessible due to the dynamic and complex nature of the Grid. A Grid environment presents unpredictable changing conditions, such as dynamic resource load, high fault rate, or continuous addition and removal of resources. We have developed an experimental framework that incorporates the runtime mechanisms needed for adaptive execution of applications on a changing Grid environment. In this paper we describe how our submission framework is used to support different migration policies on a research Grid testbed.

1. Introduction

The management of jobs within the same department is addressed by many research and commercial systems [2]: Condor, LSF, SGE, PBS, LoadLeveler... 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 [4] middleware provides the services and libraries needed to enable secure multiple domain operation with different resource management systems and access policies.

However, the user is responsible for manually performing all the submission stages in order to achieve any functionality: system selection, system preparation, submission, monitoring, migration and termination [10]. The development of applications for the Grid continues 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 characteristics, namely: dynamic resource load and cost, dynamic resource availability, and high fault rate.

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 this paper we present a new Globus experimental framework that allows an easier and more efficient execution of jobs on a dynamic Grid environment in a "submit and forget" fashion. Adaptation is achieved by implementing automatic application migration when one of the following circumstances is detected:

- *Grid initiated migration*: A new "better" resource is discovered; the remote resource or its network connection fails; or the submitted job is canceled by the resource administrator.
- *Application initiated migration*: The application detects performance degradation or performance contract violation; or self-migration when the requirements of the application change.

*This research was supported by the National Aeronautics and Space Administration under NASA Contract No. NAS1-97046 while the first and last 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.

The rest of the paper is as follows. The submission framework is described in Section 2. Then the Grid and application initiated migration capabilities of the framework, are demonstrated in Section 3, in the execution of a Computational Fluid Dynamics (CFD) code. Finally, Section 4 highlights related work, and Section 5 includes the main conclusions.

2. Experimental Framework

The GridWay experimental submission framework provides the runtime mechanisms needed for dynamically adapting the application to a changing Grid environment. 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* program and by evaluating its accumulated suspension time. A *Resource Selector* program acts as a personal resource broker to build a 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 its own scheduling.

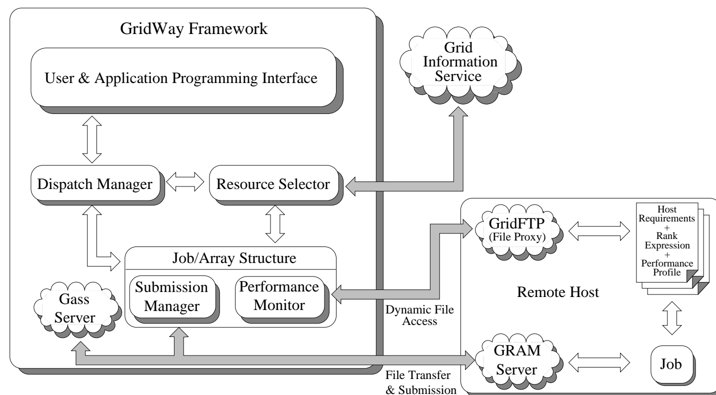

Figure 1. Architecture of the Experimental Framework.

The *Submission Agent* (figure 1) performs all submission stages and watches over the efficient execution of the job. It consists of the following components (see [5] for a detailed description):

- The client application uses a Client API (Application Program Interface)to communicate with the *Request Manager* in order to submit the job along with its `job template`, which contains all the necessary parameters for its execution. The client may also request control operations to the *Request Manager*, such as job *stop/resume*, *kill* or *reschedule*.
- 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 its own 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.
- The *Submission Manager* is responsible for the execution of the job during its lifetime, i.e. until it is *done* or *stopped*. It also probes periodically at each *polling* interval the connection to the jobmanager and Gatekeeper to detect remote failures. 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 the 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.
- *Epiloging*: Preparing the RSL and submitting the *Epilog* executable. The *Epilog* transfers back output files when termination, restart files when 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 flexibility of the framework is guaranteed by a well-defined API for each *Submission Agent* component. The framework has been designed to be modular, 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*, and *Epilog*.

2.1. Resource Selector

Due to the heterogeneous and dynamic nature of the Grid, the end-user must establish the requirements which must be met by the target resources and an expression to assign a rank to each candidate host. Both may combine static machine attributes (operating system, architecture,...) and dynamic status information (disk space, processor load,...). Different strategies for application level scheduling can be implemented, see for example [1,8,6].

The *Resource Selector* used in the experiments 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 filter, which is used by the *Resource Selector* to query Globus 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.

- A rank expression based on workload parameters 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:

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

  where $FLOPS$ is the peak performance achievable by the host CPU, and CPU_{15} is the average load in the last 15 minutes.

2.2. Prolog and Epilog

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. The executable (one per each architecture) and input files are assumed to be stored in an experiment directory. In the experiments, 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.

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* (host requirement, rank expression and performance profile files). Dynamic file transferring is not possible through a reverse-server model in closed systems, such as Beowulf clusters. This problem has been overcome by using a *file proxy* on the front-end node of the remote system.

2.3. Performance and Job Monitoring

The GridWay framework provides two mechanisms to detect performance slowdown:

- A *Performance Degradation Evaluator (PDE)* is periodically executed at each *monitoring* interval by the *Performance Monitor* to evaluate a rescheduling condition. In our experiments, the solver of the CFD code is an iterative multigrid method. The time consumed in each iteration is appended by the running job to a dynamic `performance profile` file. The *PDE* verifies at each *monitoring* interval if the time consumed in each iteration is higher than a given threshold. This performance contract and contract monitor are similar to those used in [3].

- A running job could be temporally suspended by the resource administrator or by the local queue scheduler on the remote resource. The *Submission Manager* takes count of the overall *suspension time* of its job and requests a *rescheduling* action if it exceeds a given threshold.

3. Experiences

We next demonstrate the migration capabilities of the experimental framework in the execution of a CFD code. The target application is an iterative robust multigrid algorithm that solves the 3D incompressible Navier-Stokes equations [9]. Application-level checkpoint files are generated at each multigrid iteration. In all the experiments, *monitoring*, *polling* and *scheduling* intervals were set to 10 seconds. The following experiments where conducted in the TRGP research testbed, whose main characteristics are described in table 1.

Table 1
TRGP (Tidewater Research Grid Partnership) resource characteristics

host	Model	Nodes	OS	Memory	Peak Performance	VO
coral	Intel Pentium II, III, 4	104	Linux 2.4	56GB	89 Gflops	ICASE
whale	Sun UltraSparc II	2	Solaris 7	4GB	1.8 Gflops	ICASE
urchin	Sun UltraSparc I	2	Solaris 7	1GB	672 Mflops	ICASE
carp, tetra	Sun UltraSparc IIi	1	Solaris 7	256MB	900 Mflops	ICASE
sciclone	Sun UltraSparc II, IIi	160	Solaris 7	54GB	115 Gflops	W&M

3.1. Periodic rescheduling to detect new resources

In this case, the *discovering* interval has been deliberately set to a small value (60 seconds) in order to quickly reevaluate the performance of the resources. The execution profile of the application is presented in figure 2 (left-hand chart). 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 2 (left-hand chart) 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.

3.2. 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. The execution profile of the application is presented in figure 2 (right-hand chart).

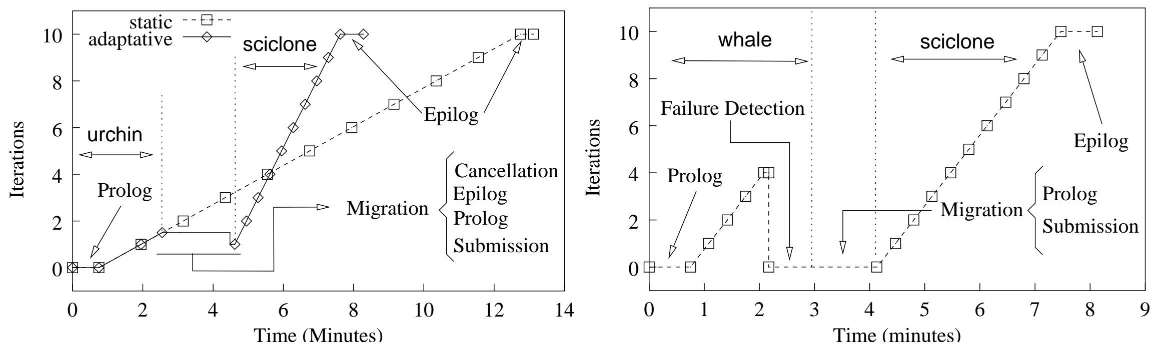

Figure 2. Execution profile of the application when a new "better" resource is detected (left-hand chart). Execution profile of the application when the remote resource fails (right-hand chart).

3.3. Performance degradation detected by the maximum suspension time

Sciclone is selected to run on initially, and the files are transferred (*Prolog* and submission in time steps 0s-51s). The job is explicitly held just after Prologing by executing the `qhold` PBS command on the remote system. The job is rescheduled as soon as the maximum suspension time is exceeded (40 seconds). The *Resource Selector* selects whale as next resource, and the migration process is then initiated (Cancellation, *Prolog* and submission in time steps 102s-165s). Finally the job completes its execution on whale. The execution profile of the application is presented in figure 3 (left-hand chart).

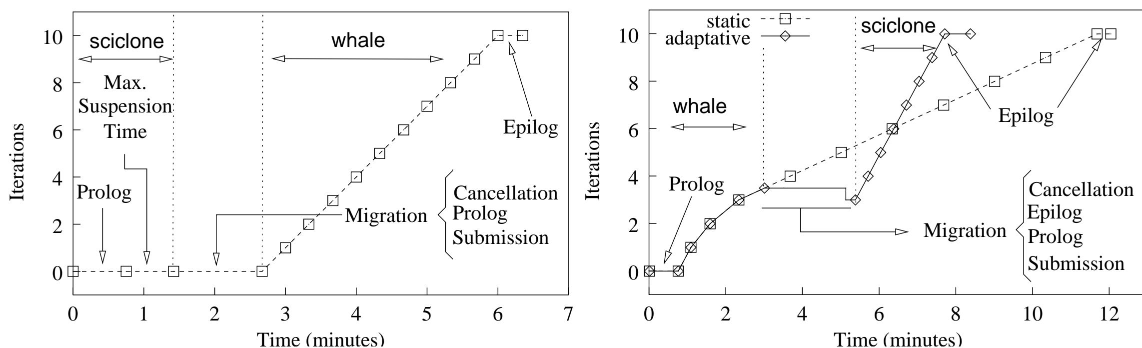

Figure 3. Execution profile of the application when the maximum suspension time is exceeded (left-hand chart). Execution profile of the application when a workload is executed (right-hand chart).

3.4. Performance degradation detected by 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 3 (right-hand chart). 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.

4. Related Work

The need of a nomadic migration approach for job execution on a Grid environment has been previously discussed in [7]. Also, the prototype of a migration framework called the "Worm", was implemented within the Cactus programming environment [3]. In the context of the GrADS project, a migration framework that takes into account both the system load and application characteristics is described in [11].

The aim of the GridWay project is similar to the aim of the GrADS project, simplify distributed heterogeneous computing. However, its scope is different. Our framework provides a submission agent that incorporates the runtime mechanisms needed for transparently executing jobs in a Grid; it is not bounded to a specific class of applications; it does not require new services; and it does not necessarily require source code changes. In fact, our framework could be used as a building block for much more complex service-oriented Grid scenarios like GrADS.

5. Conclusions

We have demonstrated the migration capabilities of the GridWay experimental framework for executing jobs on Globus-based Grid environments. 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.

Acknowledgments

This work has been performed 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] H. Dail, H.Casanova, and F.Berman. A Modular Scheduling Approach for Grid Application Development Environments. Technical Report CS2002-0708, UCSD CSE, 2002.

[2] T. El-Ghazawi, K. Gaj, N. Alexandinis, and B. Schott. Conceptual Comparative Study of Job Management Systems. Technical report, George Mason University, February 2001.

[3] G. Allen et al. The Cactus Worm: Experiments with Dynamic Resource Discovery and Allocation in a Grid Environment. *Intl. J. of High-Performance Computing Applications*, 15(4), 2001.

[4] I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *Intl. J. of Supercomputer Applications*, 11(2):115–128, 1997.

[5] E. Huedo, R. S. Montero, and I. M. Llorente. An Experimental Framework for Executing Applications in Dynamic Grid Environments. Technical Report 2002-43, ICASE-NASA Langley.

[6] E. Huedo, R. S. Montero, and I. M. Llorente. Experiences on Grid Resource Selection Considering Resource Proximity. In *Proc. of 1st European Across Grids Conf.*, February 2003.

[7] G. Lanfermann et al. Nomadic Migration: A New Tool for Dynamic Grid Computing. In *Proc. of the 10th Symp. on High Performance Distributed Computing (HPDC10)*, August 2001.

[8] R. S. Montero, E. Huedo, and I. M. Llorente. Grid Resource Selection for Opportunistic Job Migration. In *Proc. Intl. Conf. on Parallel and Distributed Computing (EuroPar)*, August 2003.

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

[10] J. M. Schopf. Ten Actions when Superscheduling. Technical Report WD8.5, The Global Grid Forum, 2001. Scheduling Working Group.

[11] S. Vadhiyar and J. Dongarra. A Performance Oriented Migration Framework for the Grid. In *Proceedings of the 3rd Int'l Symposium on Cluster Computing and the Grid (CCGrid)*, 2003.