Adaptive Grid Scheduling of a High-Throughput Bioinformatics Application*

Eduardo Huedo[1], Rubén S. Montero[2], and Ignacio M. Llorente[2,1]

[1] Laboratorio de Computación Avanzada, Simulación y Aplicaciones Telemáticas
Centro de Astrobiología (CSIC-INTA)
28850 Torrejón de Ardoz, Spain
{huedoce, martinli}@inta.es
[2] Departamento de Arquitectura de Computadores y Automática
Universidad Complutense
28040 Madrid, Spain
{rubensm, llorente}@dacya.ucm.es

Abstract. Grids provide a way to access the resources needed to execute the compute and data intensive applications required in the Bioinformatics field. However, in spite of the great research effort made in the last years, application development and execution in the Grid continue requiring a high level of expertise due to its heterogeneous and dynamic nature. In this paper, we show the procedure to adapt an existing Bioinformatics application to the Grid using the GridWay tool. The GridWay allows the efficient resolution of large computational experiments by reacting automatically to Grid- and application-generated dynamic events.

1 Introduction

The Globus toolkit [1] has become a *de facto* standard in Grid computing. Globus services allow secure and transparent access to resources across multiple administrative domains, and serve as building blocks to implement the stages of Grid scheduling [2]: resource discovery and selection, and job preparation, submission, monitoring, migration and termination. However, the user is responsible for manually performing all the scheduling steps in order to achieve any functionality. Moreover, the Globus toolkit does not provide support for adaptive execution, required in dynamic Grid environments.

Bioinformatics, which relies on the management and analysis of huge amounts of biological data, could enormously benefit from the suitability of the Grid to execute high-throughput applications. Moreover, collections of biological data are growing very fast due to the proliferation of automated high-throughput experimentation processes, and biotechnology organizations. The analysis of the data generated by these high-throughput laboratory techniques will only be possible through high-throughput Grid computing.

* This research was supported by Ministerio de Ciencia y Tecnología through the research grant TIC 2002-00334 and Instituto Nacional de Técnica Aeroespacial.

R. Wyrzykowski et al. (Eds.): PPAM 2003, LNCS 3019, pp. 840–847, 2004.

Probably, one of the most challenging problems that the Grid computing community has to deal with, to efficiently execute applications as the one described above, is the fact that Grids present unpredictable changing conditions, namely:

- *High fault rate*: In a Grid, resource or network failures are the rule rather than the exception.
- *Dynamic resource availability*: Grid resources belong to different administrative domains; so that, once a job is submitted, it can be freely cancelled by the resource owner. Furthermore, the resources shared within a virtual organization can be added or removed continuously.
- *Dynamic resource load*: Grid users access resources that are being exploited by other grid users, as well as by internal users. This fact may cause that initially idle hosts become saturated, and vice versa.
- *Dynamic resource cost*: In an economy driven grid [3], resource prices can vary depending on the time of the day (working/non-working time) or the resource load (peak/off-peak).

Consequently, in order to obtain a reasonable degree of both application performance and fault tolerance, a job must adapt its execution according to grid resource attributes, availability, performance, and cost. Adaptive Grid scheduling has been widely studied in the literature [4,5,6,7,8]; previous works have clearly demonstrated the critical factor of the dynamic information gathered from the grid to generate reliable schedules.

In this paper, we will analyze the Grid execution of an existing Bioinformatics application (section 3) using the Grid*W*ay tool, whose architecture and main functionalities are briefly described in section 2. In section 4 the procedure needed to port the application to the Grid is discussed. Then, in section 5 we will show the benefits of adaptive scheduling in the execution of the Bioinformatics application, to provide both fault tolerance (section 5.1) and performance improvement (section 5.2). The experiments were conducted on the CAB-UCM testbed. The paper ends with some conclusions and hints about future research.

2 The Grid*W*ay Framework

The core of the Grid*W*ay framework [9] is a personal *submission agent* that performs all scheduling stages and watches over the correct and efficient execution of jobs. Adaptation to changing conditions is achieved by dynamic rescheduling of jobs when one of the following circumstances is detected:

1. *Grid-initiated rescheduling:*
   - A new "better" resource is discovered [10].
   - A resource fails or is no longer available.
   - A submitted job is cancelled or suspended.
2. *Application-initiated rescheduling:*
   - Performance degradation or performance contract violation is detected.
   - The resource demands of the application change.

The framework has been designed to be modular. The following modules can be set on a per job basis:

- *resource selector*, which searchs candidate resources meeting the application demands.
- *performance evaluator*, which evaluates the application performance.
- *prolog*, which prepares the remote system and performs input file staging.
- *wrapper*, which executes the actual job and returns its exit code.
- *epilog*, which performs output file staging and cleans up the remote system.

The *submission agent* also provides the application with the fault tolerance capabilities needed in such a faulty environment:

- The GRAM *job manager* notifies submission failures as GRAM callbacks. This kind of failures include connection, authentication, authorization, RSL parsing, executable or input staging, credential expiration and other failures.
- The GRAM *job manager* is probed periodically at each *polling* interval. If the *job manager* does not respond, then the GRAM *gatekeeper* is probed. If the *gatekeeper* responds, a new *job manager* is started to resume watching over the job. If the *gatekeeper* fails to respond, a resource or network failure occurred. This is the approach followed in Condor/G [11].
- The job exit code captured can be used to determine whether the job was successfully executed or not. If the job exit code is not set, the job was prematurely terminated, so it failed or was intentionally cancelled.

When an unrecoverable failure is detected, the *submission agent* retries the submission of *prolog*, *wrapper* or *epilog* a number of times specified by the user and, when no more retries are left, it performs an action chosen by the user among two possibilities: stop the job for manually resuming it later, or automatically reschedule it.

We have developed both an API and a command line interface to interact with the *submission agent*. They allow scientists and engineers to express their computational problems in a Grid environment. The capture of the job exit code allow users to define complex jobs, where each depends on the output and exit code from the previous job. They may even involve branching, looping and spawning of subtasks, allowing the exploitation of the parallelism on the work flow of certain type of applications.

Our framework is not bounded to a specific class of applications, does not require new services, and does not necessarily require source code changes. We would like to remark that the GridWay framework does not require new system software to be installed in the Grid resources. The framework is currently 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 very difficult.

3 The Target Application

As target application, we have used a Bioinformatics application aimed at predicting the structure and thermodynamic properties of a target protein from its amino acid sequences. The algorithm [12], tested in the 5th round of Critical Assessment of techniques for protein Structure Prediction (CASP5), aligns with gaps the target sequence with all the 6150 non-redundant structures in the Protein Data Bank (PDB), and evaluates the match between sequence and structure based on a simplified free energy function plus a gap penalty term. The lowest scoring alignment found is regarded as the prediction if it satisfies some quality requirements. For each sequence-structure pair, the search of the optimal alignment is not exhaustive. A large number of alignments are constructed in parallel through a semi-deterministic algorithm, which tries to minimize the scoring function.

We have applied the algorithm to the prediction of thermodynamic properties of families of orthologous proteins, i.e. proteins performing the same function in different organisms. If a representative structure of this set is known, the algorithm predicts it as the correct structure. The test presented in this work is an example of this application, where we have applied the structure prediction algorithm to 88 sequences of the Triose Phosfate Isomerase enzyme present in different organisms. The whole experiment was submitted as an array job, where each sequence was analyzed in a separate task of the array, specifying all the needed information in a `job template` file. The results of the comparative study of this and other proteins are presented elsewhere.

4 Changes in the Application to Be Grid-Aware

Due to the high fault rate and the dynamic rescheduling, the application must generate `restart files` in order to restart the execution from a given point. If these files are not provided, the job is restarted from the beginning. User-level checkpointing managed by the programmer must be implemented because system-level checkpointing is not currently possible among heterogeneous resources. The application has been modified to periodically generate an architecture independent `restart file` that stores the best candidate proteins found to that moment and the next protein in the PDB to analyze.

In order to detect performance slowdown, the application is advised to keep a `performance profile` with its performance activity in terms of application intrinsic metrics. We have modified the application to provide a `performance profile` that stores the time spent on each iteration of the algorithm, where an iteration consists of the analysis of a given number of sequences.

In order to adapt the execution of a job to its dynamic demands, the application must specify its host requirements through a `requirement expression`. The application could define an initial set of requirements and dynamically change them when more, or even less, resources are required. Also, in order to prioritize the resources that fulfil the requirements according to its runtime

Table 1. The CAB-UCM research testbed.

Name	Model	Nodes	Speed	OS	Memory	VO	GRAM
ursa	Sun Blade 100	1	500MHz	Solaris 8	256MB		fork
draco	Sun Ultra 1	1	167MHz	Solaris 8	128MB	DACYA	fork
pegasus	Intel Pentium 4	1	2.4GHz	Linux 2.4	1GB		fork
solea	Sun Enterprise 250	2	296MHz	Solaris 8	256MB	QUIM	fork
babieca	Compaq Alpha DS10	5	466MHz	Linux 2.2	256MB	CAB	PBS

needs, the application must specify its hosts preferences through a `ranking expression`. A compute-intensive application would assign a higher rank to those hosts with faster CPUs and lower load, while a data-intensive application could benefit those hosts closer to the input data.

In the experiments described in the next section the application does not impose any requirement to the resources. The `ranking expression` uses a performance model to estimate the job turnaround time as the sum of execution and transfer time, derived from the performance and proximity of the candidate resources [13]. The application doesn't dynamically change its resource demands.

The `requirement expression` and `ranking expression` files are used by the *resource selector* to build a list of potential execution hosts. Initially, available compute resources are discovered by accessing the GIIS server and those resources that do not meet the user-provided requirements are filtered out. At this step, an authorization test is performed to guarantee user access. Then, the resource is monitored by accessing its local GRIS server. The information gathered is used to assign a rank to each candidate resource based on the user-provided preferences. Finally, the resultant prioritized list of candidate resources is used to dispatch the jobs.

In order to reduce the information retrieval overhead, the GIIS and GRIS information is locally cached at the client host and updated independently in order to separately determine how often the testbed is searched for new resources and the frequency of resource monitoring. In the following experiments we set the GIIS cache timeout to 5 minutes and the GRIS cache timeout to 1 minute.

5 Experiences

We have performed the experiments in the CAB-UCM research testbed, depicted in table 1. The testbed is highly heterogeneous and dynamic, and consists of three *virtual organizations* (VO). QUIM and DACYA VOs are connected through a metropolitan area network belonging to the UCM, and both are connected to the CAB VO through a wide area network belonging to the RedIRIS spanish academic network.

The experiment files consists of: the executable (0.5MB) provided for all the resource architectures in the testbed, the PDB files shared and compressed

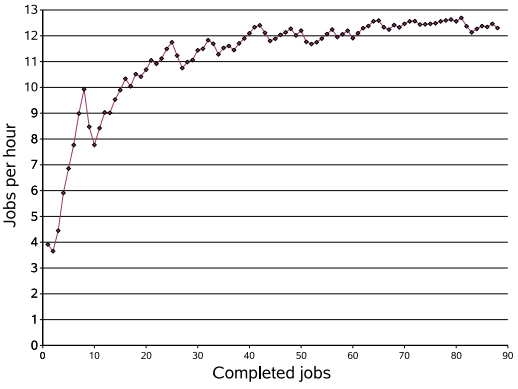

Fig. 1. Throughput when all machines in the testbed were up.

(12.2MB) to reduce the transfer time, some parameter files (1KB), and the file with the sequence to be analyzed (1KB). The final file name of the executable is obtained by resolving the variable `GW_ARCH` at runtime for the selected host, and the final name of the file with the sequence to be analyzed, with the variable `GW_TASK_ID`. Input files can be local or remote (specified as a GASS or GridFTP URL), and both can be compressed (to be uncompressed on the selected host) and declared as shared (to be stored in the GASS cache and shared by all the jobs submitted to the same resource).

Figure 1 shows the behaviour when all machines in the testbed were up. Total experiment time was 7.15 hours, and the mean throughput was 12.30 jobs/hour, which supposes a mean job turnaround time of 4.88 minutes.

5.1 Adaptive Scheduling to Provide Fault Tolerance

Figure 2 shows the attained throughput when babieca was temporarily shutdown for maintenance. Total experiment time was 7.31 hours (only 9.6 minutes more than the previous experiment), and the mean throughput was 12.04 jobs/hour, which supposes a mean job turnaround time of 4.98 minutes. The mean throughput dropped from 12.71 to 10.61 jobs/hour when babieca was down, but when it went up, it begun to grow to 12.04 jobs/hour.

5.2 Adaptive Scheduling to Provide Performance Improvement

Figure 3 shows the attained throughput when pegasus was discovered in the middle of the experiment, because it was turned on in that moment. Total experiment time was 8.65 hours, and the mean throughput was 10.17 jobs/hour, which supposes a mean job turnaround time of 5.9 minutes. Before discovering pegasus, the mean throughput was only 8.31 jobs/hour, and after that, it increased to 10.17 jobs/hour.

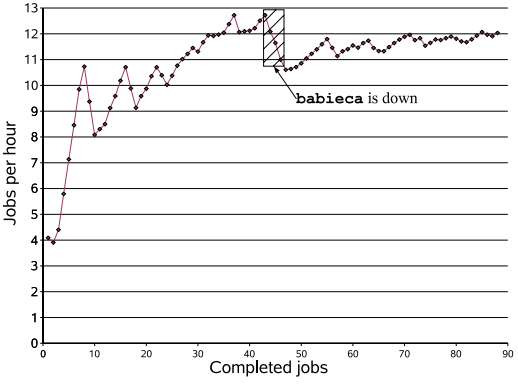

Fig. 2. Throughput when babieca was temporarily down.

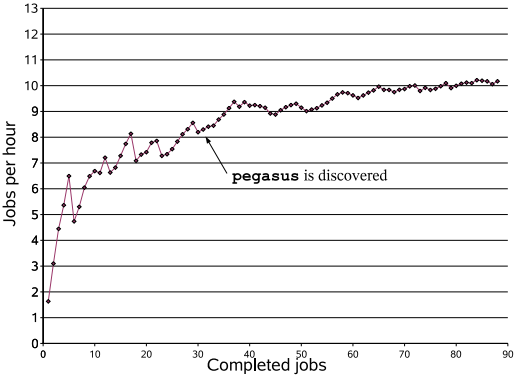

Fig. 3. Throughput when pegasus was discovered in the middle of the experiment.

6 Conclusions

We have tested the Grid*W*ay tool in our research testbed with a high-throughput application. We have seen the benefits of adaptive scheduling to provide both fault tolerance and performance improvement. In a future work we will study the adaptation of the application execution when there are more nodes than tasks to be executed in the Grid. In this case, tasks allocated to slow nodes, would be migrated to the available fast nodes when the *performance evaluator* detects a performance slowdown or the *resource selector* finds them as better resources, thus providing *preemptive scheduling* that will promote the application performance.

This promising application shows the potentiality of the Grid to the study of large numbers of protein structures, and suggests the possible application of this methods to the whole set of proteins in a complete microbial genome.

Acknowledgments. We would like to thank Ugo Bastolla, staff scientist at Centro de Astrobiología and developer of the Bioinformatics application utilized in the experiments, for his support on understanding and modifying the application.

References

1. Foster, I., Kesselman, C.: Globus: A Metacomputing Infrastructure Toolkit. Intl. J. of Supercomputer Applications **11** (1997) 115–128
2. Schopf, J.M.: Ten Actions when Superscheduling. Technical Report WD8.5, Global Grid Forum (2001) Scheduling Working Group.
3. Buyya, R., D.Abramson, Giddy, J.: A Computational Economy for Grid Computing and its Implementation in the Nimrod-G Resource Broker. Future Generation Computer Systems (2002) Elsevier Science (to appear).
4. Berman, F., et al.: Adaptive Computing on the Grid Using AppLeS. IEEE Transactions on Parallel and Distributed Systems **14** (2003) 369–382
5. Buyya, R., Abramson, D., Giddy, J.: Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computation Grid. In: Proceedings of the 4th IEEE Intl. Conference on High Performance Computing in Asia-Pacific Region (HPC Asia). (2000) Beijing, China.
6. Wolski, R., Shao, G., Berman, F.: Predicting the Cost of Redistribution in Scheduling. In: Proceedings of the 8th SIAM Conference on Parallel Processing for Scientific Applications. (1997)
7. Allen, G., et al.: The Cactus Worm: Experiments with Dynamic Resource Discovery and Allocation in a Grid Environment. Intl. Journal of High-Performance Computing Applications **15** (2001)
8. Vadhiyar, S., Dongarra, J.: A Performance Oriented Migration Framework for the Grid. In: Proceedings of the 3rd IEEE/ACM Intl. Symposium on Cluster Computing and the Grid (CCGrid). (2003)
9. Huedo, E., Montero, R.S., Llorente, I.M.: An Experimental Framework for Executing Applications in Dynamic Grid Environments. Technical Report 2002-43, ICASE – NASA Langley (2002) To appear in Intl. J. of Software – Practice and Experience.
10. Montero, R.S., Huedo, E., Llorente, I.M.: Grid Resource Selection for Opportunistic Job Migration. In: Proceedings of Intl. Conf. on Parallel and Distributed Computing (EuroPar 2003). LNCS, Springer-Verlag (2003)
11. Frey, J., et al.: Condor/G: A Computation Management Agent for Multi-Institutional Grids. In: Proceedings of the 10th IEEE Symposium on High Performance Distributed Computing (HPDC10). (2001)
12. Bastolla, U.: Sequence-Structure Alignments With the Protfinder Algorithm. In: Abstracts of Fifth Community Wide Experiment on the Critical Assessment of Techniques for Protein Structure Prediction. (2002) Available at http://predictioncenter.llnl.gov/casp5.
13. Huedo, E., Montero, R.S., Llorente, I.M.: Experiences on Grid Resource Selection Considering Resource Proximity. In: Proceedings of 1st European Across Grids Conference. (2003)