InteGrade: a Tool for Executing Parallel Applications on a Grid for Opportunistic Computing[*]

Jose de R. B. Pinheiro Junior, Raphael Y. de Camargo, Andrei Goldchleger, Fabio Kon

[1]Department of Computer Science - University of São Paulo
`http://gsd.ime.usp.br/integrade`

`{jrbraga,rcamargo,andgold,kon}@ime.usp.br`

Abstract. *Grid computing allows the usage of distributed resources possibly belonging to different institutions in a transparent way. This paper presents InteGrade, an object-oriented Grid tool that focuses on leveraging the idle computing power of shared desktop machines. Moreover, it currently allows the execution of sequential, parametric, and BSP parallel applications.*

1. Introduction

In recent years we witnessed a large increase in the computing power of desktop machines. However, these computational resources typically remain idle for most of the time, which incurrs in the waste of resources. Grid computing [Foster and Kesselman, 2003][Foster et al., 2001] allows one to leverage and integrate distributed computing resources belonging to different institutions in order to increase the amount of available computing power.

InteGrade project [Goldchleger et al., 2004a] is a multi-university effort to build a novel Grid Computing middleware infrastructure that leverages the idle computing power of personal workstations. InteGrade features an object-oriented architecture based on the CORBA [Group, 2002] industry standard for distributed objects. InteGrade also strives to ensure that users who share the idle portions of their resources in the Grid shall not perceive any loss in the quality of service provided by their applications. Finally, InteGrade also supports the reliable execution of parallel applications that requires communication among the application processes.

InteGrade currently supports three different classes of applications: sequential, parametric (Parameter Sweep Applications) and BSP [Valiant, 1990]. Sequential applications are composed of a binary that executes on a single machine. Parametric applications are also composed of a single binary, but multiple copies of the same binary are executed on different nodes with different input parameters or data sets. There are numerous areas where parametric aplication are used, such as Biology, Astrophysics, Physics, Bioinformatics, Economics, etc. An example of parametric applications would be a molecular biologist looking for compounds in large chemical data sets that best dock with a particular protein [R. Buyya et al., 2003]. Finally, the BSP applications are developed according to the Bulk Synchronous Parallel computing model, which divides the computation in supersteps, each of them composed of computation, communication, and finished with a mandatory synchronization barrier. There is a considerable amount of BSP applications

[*]This work is supported by a grant from CNPq, Brazil, process #55.2028/02-9.

available[1], several implementations have been developed and can be used by InteGrade users.

This paper is organized as follows. In Section 2 we detail the InteGrade architecture and Section 3 explains the InteGrade application execution protocol. In Section 4 we present the implementation status of InteGrade. Section 5 presents some of the InteGrade tools. Finally, we present our conclusions in Section 6.

2. InteGrade Architecture

The basic architectural unit of an InteGrade Grid is the cluster, a collection of machines typically connected by a local network. Clusters can be organized in a hierarchical inter-cluster architecture, potentially encompassing millions of machines.

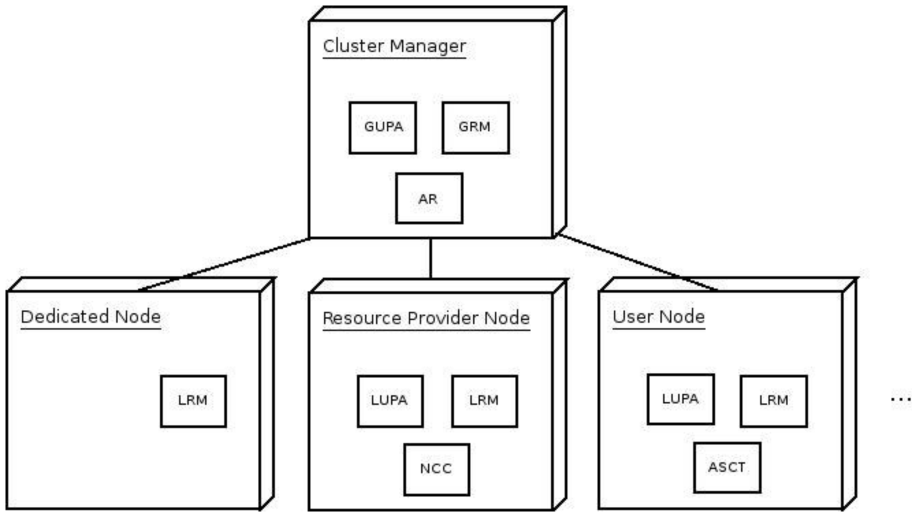


Figure 1. InteGrade Architecture.

Figure 1 depicts the most important components in an InteGrade cluster. The Cluster Manager node represents one or more nodes that are responsible for managing the cluster and interacting with managers in other clusters. A Grid User Node is one belonging to a user who submits applications to the Grid. A Resource Provider Node, typically a PC or a workstation in a shared laboratory, exports the idle portion of its computing resources, making them available to Grid users. A Dedicated Node is reserved for usage by the Grid. This kind of node is shown to emphasize that, if desired, InteGrade can also encompass dedicated resources. Note that these categories may overlap: for example, a node can be both a Grid User Node and a Resource Provider Node.

Local Resource Manager (LRM) and Global Resource Manager (GRM) cooperatively handle intracluster resource management. LRM is executed in each resource provider node, collecting information about the node status, such as memory, CPU, disk, and network utilization. LRMs send this information periodically to GRM, which uses it for scheduling executions within the cluster. This process is called the Information Update Protocol.

[1]A list of papers and technical reports on algorithms in the BSP model is available at `http://www.scs.carleton.ca/~bsp/papers.html`

Similarly to LRM/GRM cooperation, Local Usage Pattern Analyzer (LUPA) and Global Usage Pattern Analyzer (GUPA) handle intracluster usage pattern collection and analysis. LUPA executes in each resource provider node and collects data about its usage patterns. Based on long series of data, it derives usage patterns for that node throughout the week. This information is made available to GRM through GUPA, which allows better scheduling decisions due to the possibility of predicting a node's idle periods.

Node Control Center (NCC) allows resource providers to set the conditions for resource sharing, such as the amount of shared resources and the periods where they can be shared.

Application Submission and Control Tool (ASCT) allows InteGrade users to submit Grid applications for execution, monitoring each execution status and collecting execution results. Application Repository (AR) provides a centralized storage for application binaries that will be submitted for execution.

Finally, the support for executing BSP parallel applications [Goldchleger et al., 2004b] is provided by a library which implements the Oxford BSPlib API [Hill et al., 1998]. Consequently, the task of converting an existing BSPlib application to execute over InteGrade consists only in including a different header file, recompiling and relinking the application with the appropriate InteGrade libraries.

3. Execution Protocol

The Figure 2 depicts the InteGrade Application Execution Protocol [Goldchleger, 2004]. After uploading the application to the Application Repository (not illustred in the picture), the user requests their execution by using the ASCT (1). The GRM then selects a candidate node for execution (2), based on resource availability and application requirements. the GRM then forwards the request to an LRM (3), which verifies if the required resources are available at that moment. If they are not available, the GRM selects another candidate node and this process is repeated until an LRM accepts the execution request. If the resources are available, then the LRM requests the application to the AR, and the input files to the ASCT (4 and 5). The application is then launched on the selected node (6) and an execution notification is sent to the ASCT (7).

4. Implementation

When implementing InteGrade modules, we had to consider the requirement that the owners of the resource providing machines should not perceive a loss of quality of service when using their resources. Consequently, the modules executing on shared machines should have a small memory and CPU usage footprints. In the case of cluster management and user nodes this is not a strict requirement.

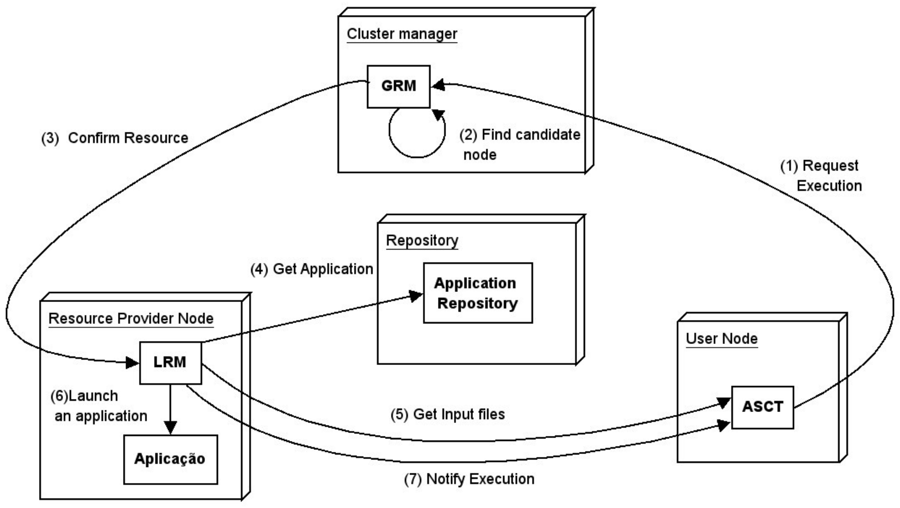


Figure 2. Application Execution Protocol.

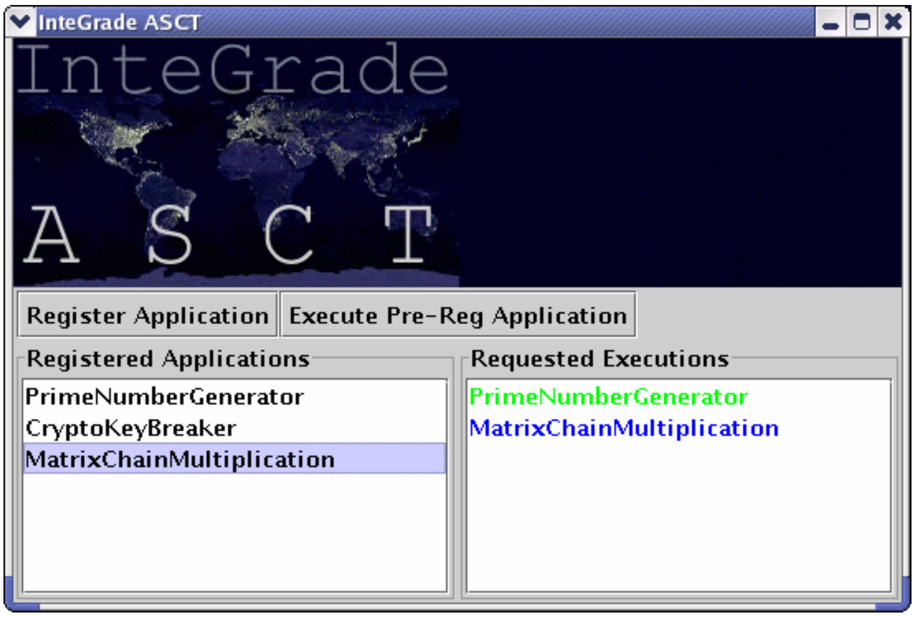
The four required modules for the InteGrade execution protocol, namely the GRM, LRM, ASCT and Application Repository are already implemented. Respecting the QoS requirements of the shared machines, LRM uses O2 [2], a very small memory footprint CORBA compatible ORB developed by InteGrade collaborators at PUC-Rio. O2 is written in Lua [Ierusalimschy et al., 1996] and can be accessed via a C API. In this new middleware platform, we have implemented the intracluster information protocol that allows LRMs to send node status to GRMs. We also implemented the intra-cluster execution protocol, which allows applications to be remotely executed in an InteGrade cluster. LRM is currently implemented in C++ using O2. The GRM, which runs on a server node, is implemented in Java on top of JacORB [3]. The GRM uses the JacORB Trader to store the information it receives from LRMs.

The current GRM implementation contains a very simple scheduler that receives an execution request and selects suitable nodes for the execution of the application. The GRM also holds a list of the currently active LRMs, as well as the available resources on each LRM. When an LRM receives an execution request from the GRM, it launches the application process. The LRM also monitors the execution of these processes and return the application results to the ASCT. It also monitors the machine resources. The current LRM version uses the Linux `/proc` pseudo-filesystem in order to obtain the resource availability. The ASCT is implemented in Java, and provides a user-friendly graphical interface, the ASCTGui. From this graphical interface the user can upload application binaries, specify execution parameters and input files, as well as collect the execution results. The Application Repository currently available is very simple, being able to store and retrieve application binaries stored in a flat namespace. An updated version containing a hierarchical directory structure is under development and will be incorporated soon to InteGrade.

The BSP programming library implementation uses CORBA for inter-task com-

[2]`http://luaforge.net/projects/o-two/`
[3]`http://www.jacorb.org`

munication. CORBA gives us the advantages of an easier and cleaner communication environment, shortening development and maintenance time. One could argue that CORBA´s IIOP is far from being the ideal communication protocol for a parallel programming library. However, we remind that InteGrade benefits from otherwise wasted computing resources, and applications are executed on a highly dynamic environment, so raw performance is not one of our major objectives at the moment. Additionally, some experiments [Román et al., 2001] with compact ORBs show a slowdown in communications of only 15% when comparing CORBA to raw sockets. This means that using CORBA does not necessarily imply in poor communication performance.



PrimeNumberGenerator
CryptoKeyBreaker
MatrixChainMultiplication

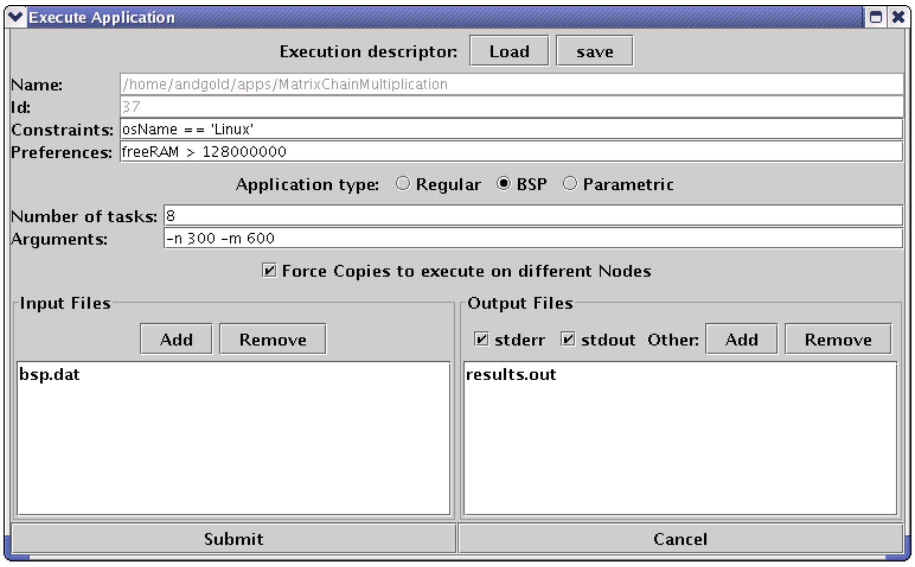
PrimeNumberGenerator
MatrixChainMultiplication

Figure 3. AsctGUI.

5. Executing Applications on InteGrade

The submission of application for executing on InteGrade is performed using AsctGui tool, show in Figure 3, a graphic application that allows the submission and monitoring of Grid applications. The left panel shows the applications already registered on the Application Repository, and the right one shows the execution status of the submitted applications. The execution status is depicted by a color code applied to each of the applications name: green indicates an application that is still executing, while blue denotes an application that is still running.

Figure 4 shows the screen from where a BSP application can be configured for execution. In this example, constraints and preferences parameters are used to determine the application's required operating system and desired amount of free memory. The application MatrixChainMultiplication displayed on this example, calculates the systolic multiplication of two matrixes [Alves et al., 2003]. It is configured to execute on eight different nodes and its arguments, -n 300 -m 600, are adjusted correctly.



Name: /home/andgold/apps/MatrixChainMultiplication
Id: 37
Constraints: osName == 'Linux'
Preferences: freeRAM > 128000000
Application type: ○ Regular [✓ BSP] ○ Parametric
Number of tasks: 8
Arguments: -n 300 -m 600
[✓] Force Copies to execute on different Nodes

bsp.dat

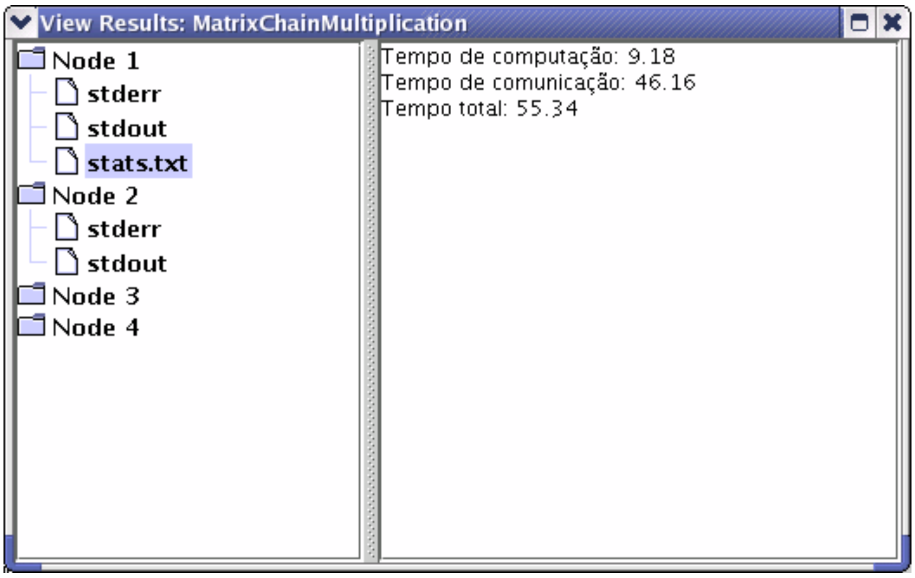
[✓] stderr [✓] stdout Other:

results.out

Figure 4. Configuration of a BSP application execution.

Still in Figure 4, the `bsp.dat` and `results.out` files are used for the storage of the computation. The input file `bsp.dat` is uploaded to each of the nodes where the application will be executed. Finally, at the end of the execution, the standard output, standard error (checked on figure), and the file `results.out` is uploaded back to the AsctGui.

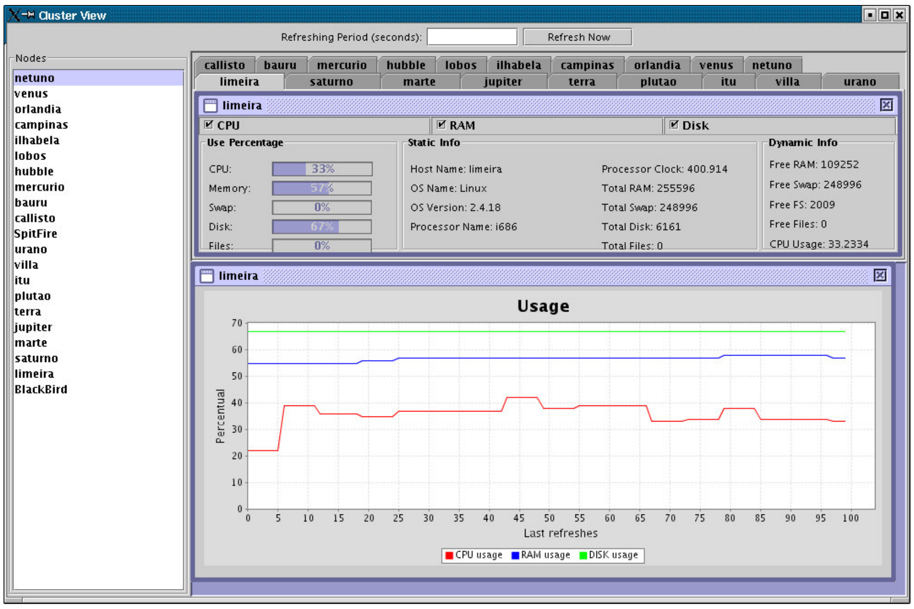
AsctGui also allows the visualization of the application output files. The window in Figure 5 shows a number of folders, each of them representing the output of one of the application nodes. For example, regarding Node 1 the files `stderr`, `stdout` and `stats.txt` are available for visualization. `stderr` and `stdout` are the standard error and standard output respectively as defined in ANSI C. `stats.txt` is a simple output file generated by the application.



```
Tempo de computação: 9.18
Tempo de comunicação: 46.16
Tempo total: 55.34
```

Figure 5. Output files visualization.

Figure 6 shows ClusterView, a tool that allows the visualization of all the nodes in an InteGrade cluster, including their characteristics, such as CPU speed, amount of memory, and operating system, as well as the amount of available resources at that moment. This tool can also display dynamic information in a periodically updating chart, allowing the usage history visualization.



Use Percentage

CPU: 33%
Memory: 57%
Swap: 0%
Disk: 67%
Files: 0%

Static Info

Host Name: limeira
OS Name: Linux
OS Version: 2.4.18
Processor Name: i686
Processor Clock: 400.914
Total RAM: 255596
Total Swap: 248996
Total Disk: 6161
Total Files: 0

Dynamic Info

Free RAM: 109252
Free Swap: 248996
Free FS: 2009
Free Files: 0
CPU Usage: 33.2334

Figure 6. ClusterView.

6. Conclusion and Future Work

In this paper, we described InteGrade, a tool for executing parallel applications on an opportunistic computing Grid. The current implementation provides support for the execution of sequential, parametric, and BSP parallel applications. InteGrade allows the effective use of almost all available computing power, since it uses computing resources that would otherwise remain idle are used to execute Grid applications.

However, some work is required to improve this system. LUPA, GUPA, and NCC modules are still being implemented. We also need to develop a good distributed scheduling policy. Finally, we are developing a security architecture to protect resource providers from malicious users and applications.

InteGrade is available as free software and can be obtained from the InteGrade project main site `http://gsd.ime.usp.br/integrade`. This site provides all information about the project, source-code, list of members and publications.

References

Alves, C. E. R., Cáceres, E. N., Dehne, F., and Song, S. W. (2003). A Parallel Wavefront Algorithm for Efficient Biological Sequence Comparison. In *The 2003 International Conference on Computational Science and its Applications - ICCSA 2003.Lecture Notes in Computer Science*, pages 126–133, Venice, Italy.

Foster, I. and Kesselman, C. (2003). *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers Inc.

Foster, I., Kesselman, C., and Tuecke, S. (2001). The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *The International Journal of Supercomputer Applications*, 15(3):200–222.

Goldchleger, A. (2004). Integrade: Um sistema de middleware para computação em grade oportunista. Master thesis, IME/USP.

Goldchleger, A., Kon, F., Goldman, A., Finger, M., and Bezerra, G. C. (2004a). InteGrade: object-oriented Grid middleware leveraging the idle computing power of desktop machines. *Concurrency and Computation: Practice and Experience*, 16(5):449–459.

Goldchleger, A., Queiroz, C. A., Kon, F., and Goldman, A. (2004b). Running Highly-Coupled Parallel Applications in a Computational Grid. In *Proceedings of the 22th Brazilian Symposium on Computer Networks (SBRC' 2004)*, Gramado-RS, Brazil. Short paper.

Group, O. M. (2002). *CORBA v3.0 Specification*. Needham, MA. OMG Document 02-06-33.

Hill, J. M. D., McColl, B., Stefanescu, D. C., Goudreau, M. W., Lang, K., Rao, S. B., Suel, T., Tsantilas, T., and Bisseling, R. H. (1998). BSPlib: The BSP programming library. *Parallel Computing*, 24(14):1947–1980.

Ierusalimschy, R., de Figueiredo, L. H., and Filho., W. C. (1996). Lua an extensible extension language. *Software Practice and Experience*, 26(6):635–652.

R. Buyya, K. B., Giddy, J., and Abramson, D. (2003). The Virtual Laboratory: Enabling Molecular Modelling for Drug Design on the World Wide Grid. *Journal of Concurrency and Computations: Practice and Experience*, 15(6):1–25.

Román, M., Kon, F., and Campbell, R. (2001). Reflective Middleware: From Your Desk to Your Hand. *IEEE Distributed Systems Online*, 2(5). Available at `http://dsonline.computer.org/0105/features/rom0105_print.htm`.

Valiant, L. G. (1990). A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111.