# An implementation of the GRAND Hierarchical Application Management Model using the ISAM/EXEHDA system

**Patrícia Kayser Vargas**[2,3] **, Lucas Alberto Souza Santos**[1] **, Cláudio F. R. Geyer**[1] **, Inês de Castro Dutra**[3]

[1]Instituto de Informática, UFRGS – Porto Alegre, RS, Brasil

[2]UniLaSalle – Canoas, RS, Brasil

[3]COPPE/Sistemas, UFRJ – Rio de Janeiro, RJ, Brasil

kayser@cos.ufrj.br, {lassantos,geyer}@inf.ufrgs.br, ines@cos.ufrj.br

***Abstract.*** *Several works on grid computing have been proposed in the last years. However, most of them, including available software, can not deal properly with some issues related to control of applications that spread a very large number of tasks across the grid network. The GRAND (Grid Robust Application Deployment) model is a step toward dealing with these kinds of applications. GRAND is an architectural model based on partitioning and hierarchical submission and control of such applications. It can control the execution of a huge number of distributed tasks preserving data locality while reducing the load of the submit machines. GRAND also offers a simple, but yet powerful, application description language, GRID-ADL (Grid Application Description Language). This paper presents a prototype implemented using the ISAM/EXEHDA system, which implements some of the main functionalities of the GRAND model.*

## 1. Introduction

Many applications have a high demand for computational resources such as CPU cycles and/or data storage. For instance, research in bioinformatics [Kurata 2003] and high energy physics (HEP) [Bunn and Newman 2003] usually requires processing of large amounts of data using processing intensive algorithms. Usually, these applications are composed of tasks and most systems deal with each individual task as if they are standalone applications. Very often, the application is composed of hierarchical tasks that need to be dealt with altogether, because either they need some feedback from the user or they need to communicate. These applications can also present a large-scale nature and spread a very large number of tasks requiring the execution of thousands or hundreds of thousands of experiments.

These high demand applications can profit of a grid environment. Several works on grid computing have been proposed in the last years [Foster and Kesselman 1998, Roure 2003, Berman, Fox and Hey 2003]. However, scheduling and execution management of applications that spread a very large number of tasks is still a technical and scientific challenge mainly due to two aspects. First, the large number of tasks can stall the submission machine when there is a centralized component to dispatch the application. Second, execution monitoring and handling errors manually is not feasible since

such applications (a) use a great amount of resources, and (b) can take several days or weeks to successfully terminate. The management and monitoring of the many tasks that compose these kinds of applications can become cumbersome and very often the user is forced to write code in some script language in order to keep track of the huge number of experiments. Another issue is related to the fact that these tasks can present dependencies through file sharing, which adds another degree of complexity to the problem.

The GRAND (**G***rid* **R***obust* **A***pplicatio***n D***eployment*) model provides management and monitoring of applications that spread a very large number of tasks in grid environments [Vargas, Dutra and Geyer 2003], [Vargas, Dutra and Geyer 2004] . It provides an architectural model to be implemented at a middleware level. It has a simple application description language as input. The dependencies between tasks are programmed using data files. User describes only the task characteristics and the dependencies between tasks are inferred automatically through a data flow dependency analysis. The GRAND model handles three important issues in the context of applications that spread a huge number of tasks: (1) partitioning applications such that dependent tasks will be placed in the same grid node to avoid unnecessary migration of intermediate and/or transient data files, (2) partitioning applications such that tasks are allocated close to their required input data, and (3) distributing the submission process such that the submit machine does not get overloaded.

To evaluate the GRAND model, the application submission and execution control are being implemented using the Java language and the EXEHDA system [Yamin 2003]. In this text we present some details of this implementation. The remaining of this text is organized as follows. In Section 2 we outline the GRAND model. We present and discuss our prototype in Section 3. Finally, we conclude this text with our final remarks and future work in Section 4 .

## 2. The GRAND model

We designed an architectural model considering that the resources and tasks are modeled as graphs. At the resource side, each node corresponds to a grid site, and therefore has information about individual resources in the site, including access restrictions, computational power, costs etc. The edges of the resources graph correspond to the connections available between the grid nodes. At the application side, each node corresponds to an application task, and each edge represents a task dependence that indicates a precedence order. The GRAND model relies on already available software components to solve issues such as allocation of tasks within a grid node, or authorization control.

**Describing the application** The management and monitoring of applications in grid computing environments is an important problem to be considered, because the user needs to keep track of all experiments, process results, and deal with fault experiments. A typical application that runs on grid environments is usually composed by many tasks. These tasks can have dependencies through data files. In order to be able to keep track of these tasks, users usually resort to script languages that help them to maintain control over the tasks being executed.

As most users are acquainted with this kind of routine, we opted to maintain this

classical approach, and provide to the user a simple description language that can quickly represent the user applications and needs. We propose a description language simple yet powerful.

Few works in the literature have approached this problem and provide a description language by which the user can describe the nature of the application. For example, VDL [Foster 2002], in the context of the GriPhyN project[Avery and Foster 2001], is being used by physicists, with help from the computer scientists, to describe task dependencies that are converted to a graph represented in another language used by Condor DAGMan [Thain, Tannenbaum and Livny 2003]. Condor DAGMan manages task dispatching based on this graph.

Our description language called GRID-ADL (Grid Application Description Language) [Vargas, Dutra and Geyer 2004] combines features of the VDL and Condor DAGMan languages, and extends their functionalities. GRID-ADL has the legibility and simplicity of shell scripts and DAGMan [Thain, Tannenbaum and Livny 2003] language while presents data relations that allow to infer automatically the directed acyclic graph (DAG) in the same way Chimera [Foster 2002] does. The user specifies a file describing its tasks, indicating input and output files, and the application graph is inferred by our system. We also added to our language some shell script like constructions.

**Submitting the Application** The GRAND proposal for application submission is a management mechanism that can control the execution of a huge number of distributed tasks. In our design we control the application through a hierarchical organization of controllers that is transparent to the user. We believe a distributed hierarchical control organization meets the applications needs, is scalable [Krayter, Buyya and Maheswaran 2002] and can alleviate the load of the submit machine avoiding stalling the user working machine.

The application submission and control is done through the following hierarchy of managers: (level 0) the user submits an application in a submission machine through the `Application Submit`; (level 1) the `Application Submit` partition the application in subgraphs and send to some `Submission Managers` the task descriptions; (level 2) a `Submission Manager` instantiates on demand the `Task Managers` that will control the task submission to the local resource management systems (RMSs) on the grid nodes; (level 3) RMSs on grid nodes receive requests from our `Task Managers` and schedule the tasks to be executed locally.

The higher level of the application control must infer the DAG and make the partitioning. The user submits his/her application through a *submit machine* that is a machine that has the `Application Submit` component installed, which is our higher level controller. The `Application Submit` is in charge of: (a) processing the user submit file describing the tasks to be executed; (b) partitioning the tasks into subgraphs; (c) communicating to the `Submission Managers`; and (d) showing, in a user friendly way, the status and monitoring information about the application.

Subgraphs defined by the partitioning algorithm are assigned to the second level controllers, called `Submission Managers`, which will instantiate the `Task Manager` processes to deal with the actual submission of the tasks to the nodes of the grid. This

third level is necessary to isolate implementation details related to specific local resource managers. The `Submission Manager` main functions are :

- when some task have dependencies with other subgraphs assigned to another `Submission Manager`, these `Submission Managers` will communicate to synchronize.
- to create daemons called `Task Manager` to control the actual task execution. Each daemon keeps control of a subgraph of tasks defined by the partitioning;
- to keep information about computational resources; and
- to supply monitoring and status information useful to the user. It stores in log files the information in a synthetic way. This information is sent to the `Application Submit` that has the responsibility to present it to the user. This periodic information flow is also used to detect failures.

The `Task Manager` is responsible for communicating with remote machines and launching tasks to remote nodes. It works similarly to a wrapper being able to communicate with a specific local resource manager.

Figure 1 illustrates the three main components of our model and their relationship.

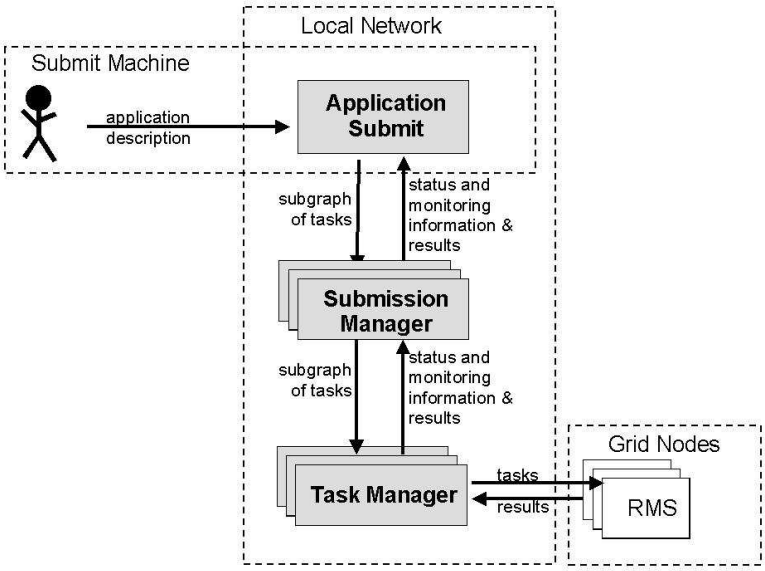


**Figure 1: Hierarchical task management main components**

When the user submits his/her application in the Submit Machine, the `Application Submit` can already be active or can be started due to the current request. When an application submission request arrives to the active `Application Submit`, the application is partitioned in subgraphs. The `Application Submit` uses its local information and choose one or more `Submission Managers` to accomplish the required tasks. The chosen `Submission Managers` will receive subgraphs in an internal representation. At this moment, there is no transfer of executables or input data files. Then, periodically the `Submission Managers` will communicate to the `Application Submit` to report about the execution progress. This communication provides online monitoring information to the user and also fault detection. Each `Submission Manager` must find the most suitable resources to run its subgraphs.

The `Task Manager` is responsible for translating the internal subgraph description to the appropriate format for tasks submission. For example, if it communicates with a Condor pool, it must prepare a Condor submit file and send the command to start tasks.

## 3. AppMan: a GRAND prototype

In this section we give some details about AppMan, a prototype of GRAND implemented over the EXEHDA system. The user provides an application description as input to our prototype. The user must use our description language GRID-ADL. Using GRID-ADL, only the task characteristics are specified. Task dependencies are inferred automatically through a data (file) dependency analysis. A parser which reads the input file and infers the directed acyclic graph (DAG) was implemented using the JavaCC [Java Compiler Compiler] tool. After getting the DAG, the graph is partitioned. Then, the prototype can execute the tasks according to the precedence order detected.

Application submission and execution control are implemented using the Java language and the EXEHDA system. EXEHDA (**Ex***ecution* **E***nvironment for* **H***igh* **D***istributed* **A***pplications*) [Yamin 2003] has facilities to allow the implementation and execution of distributed applications.

The code is organized in three packages: grand.parsing, grand.clustering and grand.appman. The main class of the grand.parsing package is the SimpleParser. This class was generated with javacc and implements the GRID-ADL grammar. The class ApplicationDescription stores information parsed. After all file is parsed, the DAG is inferred and is stored in DAG class of grand.clustering package. The grand.appman package implements the submission and control of the DAG partitioned. It also has some classes to implement a simplified monitoring graphical user interface.

The graphical interface was built to provide user with execution feedback. The application task graph is presented with four possible colors for each node which represents a task: (a) red: it depends on a data not yet available; (b) blue: task ready to execute, waiting for a free resource; (c) yellow: represents a running task; (d) green: a finished task (it was already executed).

## 4. Conclusion

This paper presented the GRAND framework for application management in grid environments, whose central idea is to have a hierarchical organization of controllers, where load of the user machine (submit) is shared with other machines. Our proposal wants to take advantage of hierarchical structures, because this seems to be the most appropriate organization for grid environments. We also presented a prototype that is being developed using the EXEHDA system. Some main contributions of our prototype implementation are the possibility of verification of (a) the soundness of the GRAND model, and (b) the potential of the EXEHDA programming environment.

## References

AVERY, P.; FOSTER, I. *The GriPhyN Project: Towards Petascale Virtual-Data Grids*. [S.l.], 2001. 26 p. Available at `http://www.griphyn.org/documents/`

document_server/show_docs.php?series=i%vdgl&category=tech&id=501.

BERMAN, F.; FOX, G.; HEY, T. *Grid Computing: Making the Global Infrastructure a Reality*. 1 ed. [S.l.]: John Wiley & Sons, 2003. 1060 p. ISBN 0-470-85319-0.

BUNN, J. J.; NEWMAN, H. B. Data intensive grids for high energy physics. In: BERMAN, F.; FOX, G.; HEY, T. (Ed.). *Grid Computing: Making the Global Infrastructure a Reality*. [S.l.]: John Wiley & Sons, 2003. p. 859–906.

FOSTER, I.; KESSELMAN, C. *The Grid: Blueprint for a New Computing Infrastructure*. 1 ed. San Francisco, California, USA: Morgan Kaufmann, 1998. 701 p. ISBN 1-55860-475-8.

FOSTER, I. et al. Chimera: A virtual data system for representing, querying and automating data derivation. In: *Proceedings of the 14th Conference on Scientific and Statistical Database Management*. Edinburgh, Scotland: [s.n.], 2002. p. 10p.

JAVA Compiler Compiler. https://javacc.dev.java.net/.

KRAYTER, K.; BUYYA, R.; MAHESWARAN, M. A taxonomy and survey of grid resource management systems for distributed computing. *Software – Practice and Experience*, v. 32, n. 2, p. 135–164, 2002.

KURATA, K.-i. et al. Evaluation of unique sequences on the European data grid. In: *Proceedings of the First Asia-Pacific conference on Bioinformatics 2003*. [S.l.]: Australian Computer Society, Inc., 2003. p. 43–52. ISBN 0-909-92597-6.

ROURE, D. D. et al. The evolution of the Grid. In: BERMAN, F.; FOX, G.; HEY, T. (Ed.). *Grid Computing: Making the Global Infrastructure a Reality*. [S.l.]: John Wiley & Sons, 2003. p. 65–100.

THAIN, D.; TANNENBAUM, T.; LIVNY, M. Condor and the Grid. In: BERMAN, F.; FOX, G.; HEY, T. (Ed.). *Grid Computing: Making The Global Infrastructure a Reality*. [S.l.]: John Wiley & Sons, 2003.

VARGAS, P. K.; DUTRA, I. C.; GEYER, C. F. *Hierarchical Resource Management and Application Control in Grid Environments*. COPPE/Sistemas - UFRJ, 2003. 8 p. Technical Report ES-608/03.

VARGAS, P. K.; DUTRA, I. C.; GEYER, C. F. *Application Partitioning and Hierarchical Application Management in Grid Environments*. COPPE/Sistemas - UFRJ, 2004. 42 p. Technical Report ES-661/04.

VARGAS, P. K.; DUTRA, I. C.; GEYER, C. F. Application partitioning and hierarchical management in grid environments. In: *1st International Middleware Doctoral Symposium 2004*. Toronto - Canadá: [s.n.], 2004. p. 314–318.

YAMIN, A. et al. Towards merging context-aware, mobile and grid computing. *International Journal of High Performance Computing Applications*, Sage Publications, London, v. 17, n. 2, p. 191–203, June 2003.