Performance Oriented Development and Tuning of GRID Applications

Emilio Mancini[1], Massimiliano Rak[2], Roberto Torella[2], and Umberto Villano[1]

[1] Universitá del Sannio, Facoltá di Ingegneria
C.so Garibaldi 107, 82100 Benevento, Italy
{epmancini,villano}@unisannio.it
[2] DII, Seconda Universitá di Napoli
via Roma 29, 81031 Aversa(CE), Italy
{massimiliano.rak,r.torella}@unina2.it

Abstract. GRID Application development is a hard task. Good applications should correctly use large distributed systems, whose infrastructure heavily affects the application performance. In this paper we propose a performance oriented approach to GRID application development, founded on the use of a prototype language (MetaPL) for the description of the applications and the use of a heterogeneous system simulation environment (HeSSE) for performance prediction. We developed GRID simulation components for the existing simulation environment (HeSSE) and validated them. After that we extended the MetaPL language in order to explicitly support GRID application features and simulated a simple case study to show how the approach works.

1 Introduction

The presence of distributed software systems is pervasive in current computing applications. In commercial and business environments, the majority of time-critical applications has moved from mainframe platforms to distributed systems. In academic and research fields, the advances in high-speed networks and improved microprocessor performance have made clusters or networks of workstations and Computational GRIDS an appealing vehicle for cost-effective parallel computing. However, the systematic use of distributed programming can be frustrating, especially if the final application performance is more than an issue. Even if great effort has been putting in developing methodologies and tools that could help the final programmer to develop application independently from the underlying architecture, as happen in GRID environment, very few results have been obtained to support prediction and evaluation of prototypal application. In the last few years, our research group has been active in the performance analysis and prediction field, developing HeSSE [6-7], a simulator of distributed applications executed in heterogeneous systems, and MetaPL a prototypal-base language, based on XML, able to support many different programming paradigm. This paper presents a simulation-based methodology, founded on HeSSE and MetaPL, that makes it possible to predict GRID application and system performance, even when the execution environment is not available and the application is not completely developed. This

J. Dongarra, K. Madsen, and J. Waśniewski (Eds.): PARA 2004, LNCS 3732, pp. 509–518, 2006.

methodology can be used as the basis for performance-driven GRID application development or GRID system performance tuning and design. The reminder of the paper is structured as follow: next section will show briefly the related work on the GRID simulation. Section 3.2 will describe our simulation environment and modeling technique and the newly developed the GRID extensions; the section will point out the simulation components developed and their validation. Finally the approach will be applied on a simple case study and then compared to the actual results obtained in the real (i.e., non-simulated) GRID environment, discussing the accuracy of the model used and the effectiveness of the proposed approach. The paper ends with a section on conclusions and future work.

2 Related Work

The performance analysis and tuning of applications, along with the optimization of scheduling and resource allocation algorithms, are widely recognized as particularly hard problems in Grids. Long application running times, non-repeatability of tests, not to mention economic problems, prevent the use of real applications running on real hardware to grade the effectiveness of alternative solutions. Most of the contributions in literature try to predict application running times, network load and end-to-end data transfer times by statistic models of historical data. An alternative, and probably more manageable solution, is to resort to simulation environments that enable reproducible, controlled and systematic evaluation of middleware, applications, and network services for the Grid. However, none of the simulations environments currently available seems able to provide accurate, fast and robust performance evaluations and analysis of full-scale Grid applications and middleware. In particular, the objective of the Bricks project [4] is to simulate alternative scheduling policies for client-server systems that provide remote access to computing services over the Grid. Bricks makes it possible to simulate alternative resource allocation strategies and policies for multiple clients, multiple servers scenarios. However, Bricks follows centralized global scheduling methodology and hence it is not suitable for simulation of environments where there are multiple un-coordinated schedulers. Microgrid [5] is a simulation environment that provides a virtual grid infrastructure for the study of Grid resource management issues. In fact, it is actually an emulator, in that actual application code is executed on a virtual Globus environment. While on the one hand this characteristic leads to high accuracy results, on the other it affects negatively simulation speed. In practice, most applications are executed in Microgrid in a time longer than the one required in the actual environment, thus making the use of the simulator not viable to perform large number of experiments. Simgrid [6] is instead a C language-based toolkit for the simulation of application scheduling. It supports the modeling of time-shared resources, taking into account the load which can be described synthetically or obtained by previously-collected real traces. As Bricks, Simgrid makes it possible to model environments where there is a single scheduling entity, with the further constraint that the systems must be time-shared. Hence it is not directly utilizable for simulating multiple competing users, applications, and schedulers with different policies, as in most Grid environments, not to mention space-shared machines. Gridsim [3] is the most recent proposal, and it extends and enhances the previous

systems, providing modeling of heterogeneous time- and space-shared resources, multiple static or dynamic schedulers, definition of CPU processing power. However, it is fairly limited as far as network and I/O devices simulation is concerned.

3 GRID Simulation with MetaPl and HeSSE

HeSSE is a simulation tool that, using a compositional modeling paradigm, allows the user to simulate the performance behavior of a wide range of distributed systems for a given application, under different computing and network load condition.

The compositional modeling approach allows to easily describe Distributed Heterogeneous Systems that are modeled by interconnecting simple components. Each component reproduces the performance behavior of a section of the complete system at a given level of detail. A HeSSE component is basically an object, hard-coded with the performance behavior of a section of the whole system. More detailed, each component has to reproduce both the functional and temporal behavior of the subsystem it represents.

In HeSSE, the functional behavior of a component is the service set that it exports to the other components. So connected components can ask other components for services. The temporal behavior of a component describes the time spent servicing. System modeling is performed primarily at the logical architecture level. For example, physical-level performance, such as the one resulting from a given processor architecture, is generally modeled with simple analytical models or by integral, and not punctual behavioral simulation. In other words, the use of a processor to execute instructions is modeled as the total time spent in the processor without considering the per-instruction behavior. Thanks to the chosen approach, HeSSE is capable of describing easily very complex Distributed Heterogeneous Systems at any given level of detail.

HeSSE uses traces to describe applications. A trace is a file that records all the actions of a program relevant for simulation. For example, the trace for an MPI application is a sequence of CPU burst and requests to the run-time environment. Each trace is the representation of a specific execution of the parallel program.

Trace files are simulation-oriented application descriptions, usually obtained through application instrumentation. When the application is in development state, they can be generated using prototypal languages. In the past years we developed an XML-based language for parallel programs description: MetaPL [10]. It is language independent and can easily support many different programming paradigms or communication libraries. It is possible to extend the language, through *Language Extensions* XML DTDS, which introduce new constructs to the language; available examples are PVM, MPI and OpenMP language extensions. Moreover MetaPL is able to generate HeSSE trace files through a filter mechanism.

Detailed description of the MetaPL approach to program description and trace generation is out of this paper scope and can be found in [13,12].

The application analysis process can be represented graphically as in Fig. 1. It is subdivided in three steps: System Description, Simulation and Results Analysis. Analysis can drive to new models and process analysis repetition.

The System Description phase includes:

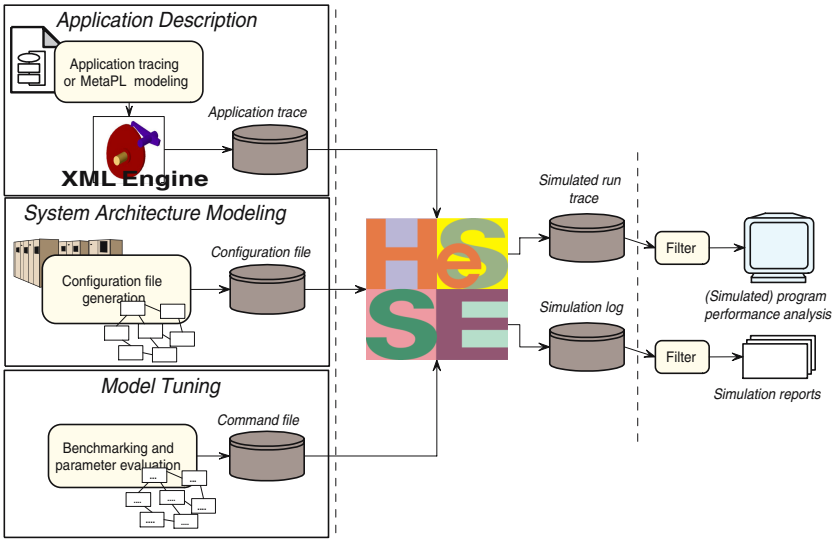

Fig. 1. HeSSE Simulation Session

- MetaPL prototypes development (Application Description);
- system architecture model definition (System Architecture Modelling);
- evaluation of the time parameters (model tuning).

The application description step consists in the MetaPL prototype development. Thanks to the XML prototype, it is possible to generate the trace files needed to drive the simulation execution. Second point consists in choosing (developing, if needed) HeSSE components useful for the simulation, and in composing them through a configuration file; at the end of this step, we are able to reproduce the system evolution. Last step consists in running benchmarks on the target system in order to fill the simulator and the MetaPL description with time information.

We built a simple MetaPL extension that supports GRID specific information, similar to the data that can be retrieved from a globus RSL file, like the number of gatekeeper involved. GRID MetaPL commands will be used by the trace generation mechanism to define high level behavior, like the number of parallel tasks started in the GRID environment.

In order to simulate the GRID infrastructure (i.e. the components of a real GRID environment) we developed a new set of HeSSE components. Following sections will give details about the simulation models adopted.

3.1 GRID Components in HeSSE

Our aims in this paper is to predict the performance of a given application (not completely developed) in a GRID environment in terms of the overall application response time, the time effectively spent in execution or the time needed to be started on the GRID environment.

Basic components in HeSSE (see [8,9]) are able to reproduce main features of common cluster systems, like ethernet and myrinet networks, operating system scheduling and job management, process synchronization and message passing software layers. To reproduce the full GRID environment infrastructure overhead, we need to choice a real implementation to mimic; in the following we will focus on the globus GRID platform solution, mainly on two components: `GRAM` and GIS. The first one, `GRAM`, manages the application allocation on the target system. The `GIS`, component manages the GRID environment security problems, mainly the user authentication. Good description of the cited components can be found in [1].

We developed three libraries containing a number of components needed to a complete simulation. These components are:

- *GlobusClient*: formally the user submitting single or batch jobs to the system.
- *GlobusServer*: this component simulate the gatekeeper in the globus environment.
- *GlobusProxy*: this one simulate the ability of the gatekeeper to connect to other gatekeepers in the case that more, or different, clusters need to be used to complete the job.
- *GPAM (GRID Process Allocation Manager)*: the work carried out by this component is to allocate processes on a cluster, execute them or waiting for an external synchronization if more clusters are used. It formally simulate the GRAM section in the globus environment.
- *Barrier*: it simulate the synchronization infrastructure, `DUROC` in globus, needed to guarantee the contemporaneous execution of all the processes in a multi-cluster application.
- *SSL*: obviously simulate the overhead due to secure communication in a GRID environment.

Now let's have a deeper look to how the simulation is carried out. When the simulation starts, the GlobusClient sends a job execution request to his default GlobusServer and waits for the results. The communication between the client and the server is secure so the SSL component is used. When a server receive a request, verify if he has a GPAM and if it is free or busy. From the request, the server sees if the job must be executed on a single cluster or on multiple clusters. If the server's GPAM is free and the job must be executed on a single cluster, the server sends the request to his GPAM that allocate the tasks on his cluster's machines. If the server GPAM is busy, the server sends the request to his GlobusProxy asking for forwarding to another server. In the case that the job must be executed on multiple clusters, the server sends the job to his GPAM. The GPAM allocates the tasks and waits for the Barrier synchronization. Then the server sends the job to proxy which forward it to the other servers. When all the tasks have been allocated, the Barrier unlock the GPAMs that start the task execution. At the completion, each GPAM sends to his server the results that are forwarded back to the first server and then to the client.

The autenticatio process heavily affects the system performances, iIn order to show the kind of interactions that take place in the startup phase of the application, figure 2 exploits the message exchange in the authentication process, when two different GRID environments are involved. Simulation environment reproduces the complete messag

exchange. A detailed verifying process was carried on, monitoring both real environment and the simulation to grant that exactly the same messages are sent at any layer of the network stack.

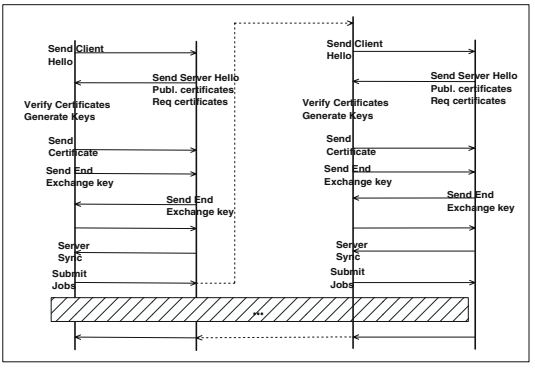

Fig. 2. Message exchange in globus autentication

3.2 Simulation Validation

To validate the correctness of the infrastructure simulation, we tried a simple test application, the unix command *True*, a standard GNU application which simply terminate with error level zero. In that way, the application execution time is almost nought and the measured response time is all due to processes allocation through the clusters. We launched the test application on the target environment (an SMP cluster better described later) in many different configurations. One of the configurations was used to tune the simulation parameters, the others were used to verify the simulation.

To emulate many GRID environment on a single cluster, we used each node as a different GRID cluster. Varying the number of tasks allocated on the GRID allows to show peculiarity in the allocation time. In particular, has been noted that, from two to eight tasks, the allocation time vary linearly while there is a big gap between one and two. This is due to the fact that all the job submission to multiple clusters are made in parallel by the first cluster.

Figure 3 shows both the real environment and the simulation behavior, varying the number of GRID nodes. Upper side of the figure shows the response time, while lower side contains the percentage simulation error. Note that the behavior showed by simulation correspond to the real system one.

4 An Example: The Gauss-Siedel Application

Aiming at clarify how proposed development approach works, we have applied it on a single case study: development of the Gauss-Siedel method for resolving iteratively linear equations systems. We modeled and simulated the target application. After the analysis, we developed and ran it on the real environment. The execution results were

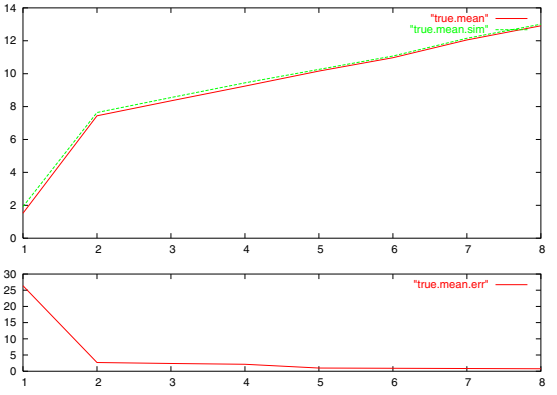

Fig. 3. Gauss-Siedel Simulation

compared with the predicted ones. As previously pointed out the development methodology founds on the following steps:

Application Modeling MetaPL Description of the target application

GRID Environment Modeling Definition of one (or more) HeSSE configuration able to reproduce the target environment(s)

Model Tuning Execution of simple application benchmark able to define the time parameters for the developed models (see [14] for further details)

Application Analysis Application Simulation on the target(s) GRID environments, in order to understand final application performance behavior.

The Gauss-Siedel method for resolving iteratively linear equations systems calculate, at each step, the new unknowns values using those calculated at the previous step. At first step a set of random values is chosen. It works under the assumption that the coefficient matrix is a dominating-diagonal one. Due to the particular method chosen, the parallelization is very simple. In fact each task calculates only a subset of all the unknowns and then gathers with the other tasks the rest of the unknowns vector needed for the next step. Figure 4 shows a partial MetaPL description of the above described application (in order to improve readability, we cut away XML tags and code section not useful for code understanding).

In order to validate the approach, the proposed application was developed and run on a real (even if little) GRID environment: the cluster Cygnus. This cluster is composed of four Pentium III bi-processor nodes with 512MB of memory each and a front-end Pentium IV Xeon with 1GB of main memory. The nodes are connected each other through a switched 100Mbps ethernet LAN while the frontend has two ethernet cards, one private, connected to the switch, and the other public connected to the external world.

Figure 5 shows the results of application execution in the real environment (the cluster) and its simulation; figure upper side contains the response time graph, for both real and simulated executions, while lower side contains relative percentage error between real timings and the simulation prediction. Note that the application trend in simulated and real environment are the same.

```
<MetaPL>
  <Code>
    <Task name="Gauss"> <Block>
      <Command time="t1" name="readdata"/>
      <Broadcast dim="1024" />
      . . .
    </Block> </Task>
  <Code>
  <Mapping>
  <NumberOfProcesses value="6" />
  <NumberOfGatekeeper value="2" />
  <Gatekeeper id="1">
  <Scheduler="fork">
  </Gatekeeper>
  </Mapping>
</MetaPL>
```

Fig. 4. Gauss-Siedel MetaPL description

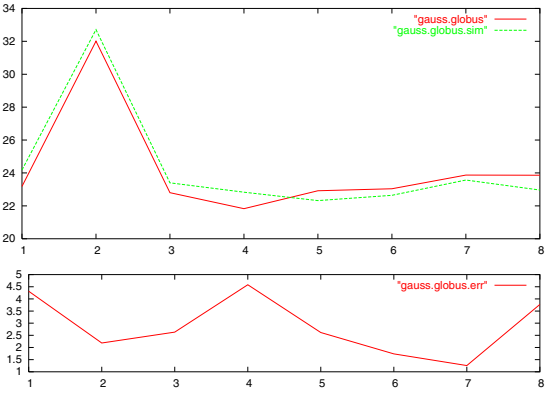

Fig. 5. Gauss-Siedel Simulation

5 Conclusions

This paper shows a performance-oriented approach to GRID application development, founded on the adoption of a simulation environment (HeSSE) and a prototypal language (MetaPL) for performance evaluation of the application at any step of the target software development.

We developed GRID simulation components for the existing simulation environment, able to reproduce the main globus components performance behavior. The proposed components were validated on a test-bed, built upon an SMP cluster.

Then a simple application was developed in the prototypal language (MetaPL), adopting the newly developed GRID extensions, in order to drive the simulations and its performances were predicted.

To validate the approach, we developed the application and compared the results with the predicted ones, showing that the simulated behavior correspond to real system evolution.

Acknowledgment

This work was partially supported by "Centro di Competenze" regione Campania. We want to thank Raffaele Vitolo for his technical support and Rocco Aversa for his contribution.

References

1. I. Foster, C. Kesselman, J. Nick, and S. Tuecke: “The physiology of the grid: An open grid services architecture for distributed systems integration". Technical report, Open Grid Service Infrastructure WG, Global Grid Forum, June 2002.
2. I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. International J. Supercomputer Applications, 15(3), 2001.
3. Buyya, R. , Murshed, M.: “GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing", The Journal of Concurrency and Computation: Practice and Experience (CCPE), Wiley Press, May 2002.
4. Takefusa Bricks: A Performance Evaluation System for Scheduling Algorithms on the Grids. JSPS Workshop on Applied Information Technology for Science (JWAITS 2001). 2001.01.
5. Huaxia Xia, Holly Dail, Henri Casanova and Andrew Chien, The MicroGrid: Using Emulation to Predict Application Performance in Diverse Grid Network Environments, In Proceedings of the Workshop on Challenges of Large Applications in Distributed Environments (CLADE ’04), held in conjunction with the Thirteenth IEEE International Symposium on High-Performance Distributed Computing, Honolulu, Hawaii, June 2004 .
6. Henri Casanova and Arnaud Legrand and Loris Marchal, Scheduling Distributed Applications: the SimGrid Simulation Framework,Proceedings of the third IEEE International Symposium on Cluster Computing and the Grid (CCGrid’03)
7. Aversa, R., Mazzeo, A., Mazzocca, N., Villano, U.: Developing Applications for Heterogeneous Computing Environments using Simulation: a Case Study. Parallel Computing 24 (1998) 741-761
8. Mazzocca N., Rak M., Villano U. 2000. “The Transition from a PVM Program Simulator to a Heterogeneous System Simulator: The HeSSE Project". Recent Advances in Parallel Virtual Machine and Message Passing Interface, in J. Dongarra et al. (eds.) Lecture Notes in Computer Science, Vol. 1908, Springer-Verlag, Berlin 2000, (pp. 266-273).
9. N. Mazzocca, M.Rak, R. Torella, E. Mancini and U. Villano, The HeSSE simulation environment. Proc. ESMc’2003, 27-29 Oct. 2003, Naples, Italy, pp. 270-274.
10. Mazzocca N., Rak M., Villano U., 2001. “MetaPL: a Notation System for Parallel Program Description and Performance Analysis" Parallel Computing Technologies, in Malyshkin. V. (ed.), Lecture Notes in Computer Science, Vol. 2127, Springer-Verlag, Berlin 2001,(pp. 80-93)
11. Labarta, J., Girona, S., Pillet, V., Cortes T., Gregoris, L.: DiP: a Parallel Program Development Environment. Proc. Euro-Par ’96, Lyon, France (Aug. 1996) Vol. II 665- 674
12. E. Mancini, N. Mazzocca, M. Rak, and U. Villano, Integrated Tools for Performance-Oriented Distributed Software Development. Proc. SERP’03 Conference, Las Vegas (NE), USA, June 23-26, 2003, vol. I, pp. 88-94

13. N. Mazzocca, M. Rak, U. Villano, The MetaPL approach to the performance analysis of distributed software systems. Proc. 3rd International Workshop on Software and Performance (WOSP02), IEEE Press (2002) 142-149
14. E. Mancini, M. Rak, R. Torella, "U. Villano, Off-line Performance Prediction of Message-Passing Applications on Cluster Systems, Lecture Notes in Computer Science, Vol. 2840, Springer-Verlag, Berlin 2003, (pp. 45-54).