Simulation-based Optimization of Multiple-task GRID Applications

E. P. Mancini, U. Villano,
RCOST and DING, Università del Sannio,
C.so Garibaldi 107, 82100 Benevento, Italy
{epmancini, villano}@unisannio.it

M. Rak, F. Moscato,
DII, Seconda Università di Napoli,
V ia Roma 29, 81031 Aversa(CE), Italy
{massimiliano.rak, francesco.moscato}@unina2.it

March 13, 2012

Abstract

This paper deals with the performance optimization of multiple-task applications in GRID environments. Typically such applications are launched by a Resource Manager, which only takes into account the application's resource requirements and current availability on the GRID. Here a novel approach is presented, that performs resource management in user space, making it possible to exploit application modularity and flexibility and to take into account expected performance figures produced by GRID simulation. The objective is to make optimized choices that can lead to reduced application response times. After an introduction to the GRID simulation environment used, the structure of an application launcher able to optimize a number of application tasks and their mapping on the GRID is sketched, presenting the encouraging performance results obtained.

1 Introduction

Geographically-distributed software systems are widely used in current computing applications. In commercial and business environments, the majority of time-critical applications have moved from mainframe platforms to distributed systems. In academic and research fields, the advances in high-speed networks and the improvement of 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 systems can be frustrating, due to difficulties in resource usage and performance optimizations. In order to

optimize resource usage, administrators of high performance systems such as clusters or NOWs (Network of Workstations) usually adopt resource allocation systems and schedulers (e.g., Nimrod [1], MOAB [2] or MAUI [3]). On the other hand, GRID environments, such as those based on Globus software [4], are composed of several different Virtual Organizations (VO), independently managed by local resource managers. In this context, a high-level resource allocator, such as GRAM [5], works on the top of local schedulers, acquiring and releasing resources throughout the whole distributed system.

Applications act as resource requesters, and GRID environments use suitable languages (e.g., RSL [6]) to describe requests. Typically an application starter script provides the *Resource Manager* with the RSL descriptions; in its turn, the resource manager issues requests to the schedulers (or to the operating systems) and acquires the resources needed for application execution. Only when the resources are available, does the starter script launch the application using requested resources.

On such systems, performance is difficult to predict and resource tuning is a hard task. As a matter of fact, developers have no control over the actual environment used for execution. This makes performance portability [7] and performance engineering [8] a challenge. Another fundamental drawback of the previously examined execution model is that application requests are "static": the total amount of resources needed by an application cannot vary as a function of the current system status. For example, an application has to declare how many nodes or processors it needs, and it has no way to "tune" this choice taking into account the system load: the resource manager simply blocks it until all the requested nodes or processors are available. On the other hand, developers try to design flexible and scalable applications, trying to optimize performance even on systems whose computational power can change during their life cycle (for example after hardware expansion) [9–12].

In other words, resource schedulers may be able to optimize administrator-oriented performance metrics, such as resource utilization, but usually cannot optimize user-oriented performance metrics, such as application response time. Moreover, they are unable to take advantage of application flexibility in order to optimize the system and to improve application performance.

In this paper, we propose an alternative approach, where resource allocation is performed in user space, without resorting to a system high-level resource manager, still hiding optimization mechanisms to users. We have already experimented with a similar approach in different environments (Web services and MPI applications on clusters), presenting the obtained results in [13, 14]. The idea is that if a user-level application launcher is able to access the status of the distributed system, it could tune its resource request choosing a suitable resource allocation scheme that takes into account both system-dependent parameters (i.e., resource status and utilization) and application-dependent parameters (i.e., number of parallel tasks).

In order to show the usefulness of the approach, here the focus will be on a single aspect of the general resource allocation problem, process allocation. In other words, our goal will be to show that it is possible to reduce the response

time of multiple-task applications in a GRID environment making, in user space, the choice of *how many* parallel tasks have to be started on the system and of *where* they have to be allocated. Even if the methodologies presented here are fairly general, throughout the paper the stress will be on cluster and campus GRIDs. Geographically-dispersed GRID environments are currently out of the scope of our research.

The proposed approach relies on a framework made up of:

- a simulator, HeSSE (Heterogeneous System Simulation Environment) [15, 16], which predicts the performance of a given application on a given distributed system cluster in different working conditions;
- a language, MetaPL [17, 18], which describes the behavior of a parallel application and can control the simulation environment;
- an applications launcher, `MHstarter`, which automates the process of performance analysis and resource allocation, making the optimization mechinism transparent to the user.

The simulator is used to predict the performance of an application on the target distributed environment. In [19] we have presented a basic set of HeSSE components able to reproduce an application behavior in a GRID environment in order to analyze its performance. The next section will illustrate the simulation environment and several newly developed components. To describe the application's evolution during its execution and to optimize it, we rely on an extensible description language, MetaPL. Section 3 describes the extensions needed for GRID application description and management, showing how the language can be used throughout the whole application life cycle.

In order to validate the proposed approach, we have developed a demonstration tool on the basis of the experiences described in [13]. Section 4 describes the tool's architecture and how it can be used for performance evaluation and optimization. Then, section 5 proposes a simple case study. Using a parallel application, we will show how the tool can optimize the resource allocation. We then present the obtained results. Finally, section 6 deals with the related work.

2 GRID Simulation with HeSSE

The main goal of this paper is to find a way to delegate resource allocation (in terms of process allocation and processor usage) to final applications. In order to enable applications to make optimal decisions about resource use on GRID nodes, we need to predict the response time and resource utilization of a given system. The prediction obviously depends on application-behaviour and status of the whole system (i.e., number and, possibly, behavior of the processes in execution). Our approach is based on both system and application behavior simulation. The tool used for this has to be able to predict the resource status during the execution and the target application response time. Furthermore, it

has to be "fast enough" to be run concurrently with the application, providing information at rates suitable for adaptive or proactive run-time reconfigurations.

State-of-the-art simulation tools (see Section 6 for a brief survey) are able to predict the overall behavior of a GRID environment, in terms of resource usage. Anyway, predictions are made depending on resource scheduler policies implementations. This approach is not useful when the scheduler policies are not known and when optimization is achieved in user space.

The simulation tool used in our approach, HeSSE, reproduces application behavior in distributed environments by analyzing traces generated from application skeletons. In contrast to other simulation tools, the time needed for the simulation of an application execution time of the order of hundreds of seconds is of the order of hundreds of milliseconds. As we will shown in section 4, this allows on-line simulation for optimization purposes.

Though not strictly necessary in theory, for clarity's sake it is useful to restrict interest to a particular GRID environment to be used as a reference for simulation. In the following, we will choose one of the most famous and stable GRID platforms, the Globus toolkit in its version 2.4. It should be noted that newer versions of Globus, which rely on a web services approach, may be more profitably simulated in HeSSE through alternative approaches, such as the ones we have presented in [14]. In the following, any reference to GRID environments in the text should be interpreted as a reference to a Globus 2.4 system.

2.1 HeSSE overview

HeSSE is a simulation tool based on a compositional modeling paradigm, which 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 conditions.

The compositional modeling approach allows one 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. In more detail, 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 performing a service. 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 easily describing 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.

HeSSE outputs the results of the simulation into an XML file, which contains a complete log of the events simulated. From the XML log, an user or an automatic tool can retrieve information about the application response times (the log reports the simulation time at which each application starts and closes) and the resource status, e.g., the number of processes allocated on each processor.

2.2 GRID Components in HeSSE

The basic components in HeSSE (see [15, 20]) are able to reproduce the main features of common distributed systems, such as operating system scheduling and job management, process synchronization and message passing software layers, Ethernet and Myrinet networks. This is not sufficient for the simulation of a GRID environment, where the application can be allocated across distinct clusters by the Globus resource allocator (GRAM). Furthermore, GRIDs are different from common clusters as far as the messaging layer is concerned. For example, the GRID-enabled version of the MPICH libraries (MPICH-G2) modifies the MPICH lowest level device [21,22], affecting a) application startup synchronizations, integrating into the device an interface for the DUROC Globus component, and b) the implementation of point-to-point primitives. These and other effects have to be carefully considered in the implementation of the simulation components.

Simulation has to reproduce mainly two GRID components: `GRAM` and `GSI`. The first one, `GRAM`, manages the application allocation on the target system. The `GSI` component, instead, manages the GRID environment's security, mainly user authentication. A good description of both can be found in [7].

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

- *GlobusClient*: modeling the user that submits single or batch jobs to the system.
- *GlobusServer*: modeling a gatekeeper behaviour in the Globus environment.
- *GlobusProxy*: modeling that gatekeeper connecting to other gatekeepers in the case when multiple clusters are to be used to complete the job.
- *GPAM (GRID Process Allocation Manager)*: modeling the GRAM of the Globus environment, which manages the allocations of processes on a cluster, their execution, and waits for an external synchronization if multiple clusters are used.

- *Barrier*: modeling the synchronization infrastructure (`DUROC` in Globus) needed to guarantee the concurrent execution of processes in a multiple-cluster execution.

- *SSL*: modeling the overhead due to secure communication in a GRID environment.

Presenting how the the simulation of the whole Globus environment is performed is a complex task, and would be partially out of scope here. The main concern of this paper is the performance optimization of applications made out of multiple tasks and communicating by message-exchange, such as those that can be written using the MPICH-G2 API and run-time libraries. As the messaging layer simulation has been thoroughly dealt with in previous HeSSE papers, we will only discuss here how the initial job submission and the authentication are simulated.

The application simulation starts by GlobusClient sending a job execution request to its default GlobusServer and waiting for the results. The communication between the client and the server is secure, and so it is modeled by the SSL component. When the server receives the request, it verifies if its GPAM is free or busy. Examining the request, the server understands if the job has to be executed on a single or on multiple clusters. If the server's GPAM is free, and the job must be executed on a single cluster, the server sends a request to its GPAM, which allocates the tasks on the machines of its cluster. If the server's GPAM is busy, the server sends the request to its GlobusProxy, asking for the request to be forwarded to another server. If instead the job must be executed on multiple clusters, the server sends the job to its GPAM. The GPAM allocates the tasks and waits for the Barrier synchronization used to guarantee that all the application tasks exist at startup. Then the server sends the job to the proxy, which forwards it to the other servers. When all the tasks have been allocated, the Barrier synchronizations complete, thus unlocking the GPAMs that start the task execution. Upon completion, each GPAM sends to its server the results, which are forwarded back to the first server and then on to the client. It is worth pointing out that in the simulation environment messages are just headers with simulation-oriented information (e.g., message dimension) and do not carry any useful data: the program execution is simulated, not emulated.

The authentication process, needed for systems distributed across many different administration domains, heavily affects the system performance. It is needed each time the client accesses the GRID server, or when the server, through the Globus delegation model [7], accesses another domain to acquire new resources, i.e., when the GRID proxy contacts a GRID server. Figure 1 shows the messages that are exchanged in the startup phase of the application, assuming that two different GRID environments are involved. The HeSSE simulation environment reproduces punctually the complete message exchange. During the implementation of the involved components, a detailed verification process was performed, monitoring both the real environment and the simulated one, to guarantee that exactly the same messages were produced.

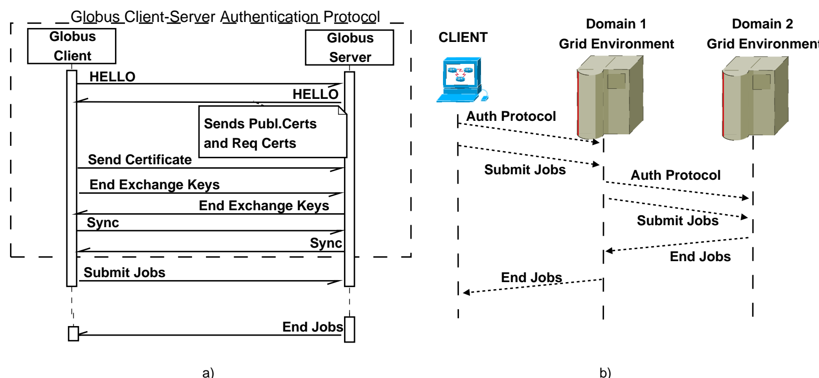

Figure 1: Message exchange for Globus authentication

2.3 Workload Components: the System Status

In order to predict the behavior of an application in a non-dedicated environment, which executes several different applications at a time, some additional support components are needed. These model the presence of many active applications sharing the system resources. Since, as mentioned before, our main concern here is process allocation, the only shared resources taken into account are processors. For simulation purposes, the presence of an additional load of processes ("additional" because they do not belong to the simulated application) is modeled by giving as input to the simulator the number of active processes and their expected execution time. Of course, in HeSSE the full simulation of two (or multiple) applications is possible, but it is worth performing it only if one is interested in the performance behavior of multiple applications submitted concurrently.

2.4 Simulation Validation

To validate the correctness and accuracy of the GRID infrastructure simulation, we performed some experiments with a trivial test application, the Unix command `true`, which simply terminates with error zero.

The test starts up the application on `n` different nodes in parallel, in order to evaluate the simulation of resource acquisition. As the application execution time is negligible, the overall response time is almost integrally due to the GRID overheads for process allocation through the clusters. We launched the test application on a target environment (the SMP cluster described later) in many different configurations. One of the configurations was used to tune the simulation parameters, whereas the others were used to verify the simulation.

To emulate a sufficiently large number of GRID environments using a single cluster, we configured the system in such a way that each node represented a different GRID cluster. Varying the number of tasks allocated to the GRID made it possible to study the peculiarities of task allocation times. In particular,

the experimental results showed that allocation time varies linearly going from two to eight tasks, whereas there is a big gap between the times for one and two tasks. This is due to the job submission to multiple clusters being made in parallel by the first cluster, i.e., by the cluster to which the client submits the job.

Figure 2 shows both the application response times measured on the real system as mentioned above, and the simulated behavior, as a function of the number of GRID nodes involved.

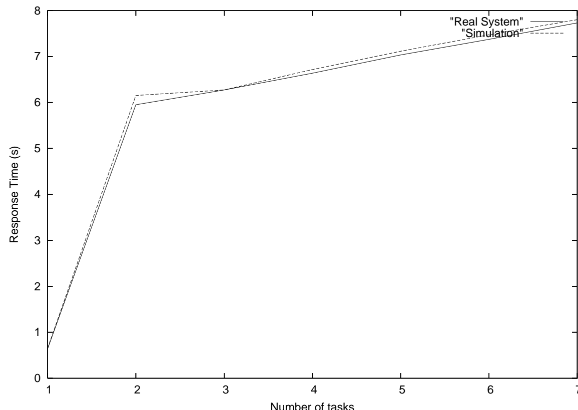

Figure 2: Simulation of the application startup process

3 Specifying GRID Application Behavior: MetaPL

As shown in the previous section, as far as HeSSE simulation is concerned, the application's behavior is specified through trace files, which are simulation-oriented application descriptions representing a given application execution. In the past years, we developed a prototype-oriented language, MetaPL [17], which makes it possible to describe the behavior of a distributed program in a way more close to the programmer's view of the same. MetaPL is an XML-based language, whose elements are able to describe the main constructs of an imperative programming language (e.g., loops and switches), primitives for process management (e.g., spawning of processes). The core of MetaPL can be extended through *Language Extensions* (XML DTDs or schemas), which introduce new constructs for the language. Examples of available extensions are PVM, MPI and OpenMP language extensions. Using language extensions, MetaPL is essentially independent of the programming language used and can support many different programming paradigms or communication libraries.

The MetaPL use is based on the concepts of *views* and *filters*. MetaPL views are synthetic descriptions able to highlight specific aspects of the program, and filters are compositions of XSLT transformations that produce views from MetaPL code. As the set of traces that describe an application execution for

```
<MetaPL>
  <Code>
    <Task name="test">
      <Block>
        <Broadcast dim="1024" />
      </Block>
    </Task>
  <Code>
  <Mapping>
    <NumberOfProcesses value="6" />
  </Mapping>
</MetaPL>
```

Figure 3: A MetaPL description: *broadcast* program

HeSSE is nothing other that a MetaPL view, they can be generated by suitable filters from the MetaPL descriptions, possibly querying the user for execution-related information.

Figure 3 shows a simple MetaPL description describing a program that performs an MPI broadcast. The application description is subdivided into two elements: `Code`, which contains the task descriptions, and `Mapping`, which contains informations about the execution context of the program. The language provides an additional element, `System`, used to specify the target execution environment. A detailed description of the MetaPL language and of its use for program description and trace generation can be found in [16–18].

3.1 MetaPL and Optimization

The flexibility of MetaPL not only makes it possible to extend the core language in order to cope with other languages/environments, but also to annotate the application description with additional information. In the context of application performance optimization, it is of paramount importance to point out explicitly the application and environmental parameters that an optimization may tune in order to optimize the application response time and/or resource usage.

In particular, in two previous papers dealing with performance optimization [14, 23] we have proposed the use of two new elements, to be inserted in the Mapping section: `Parameter` and `Target`. These are used to provide an optimization tool with the parameters that affect the application performance and map the abstract process structure onto the computing system, respectively. For example, the code in Figure 4 can be added to the code in previous example, to specify that we wish to tune the number of processes in order to reduce the application response time. The parameter element has attributes that enable the user to define the range of variation of the parameter.

```
<Mapping>
  <Parameter value="NumberOfProcesses" />
  <Target value="ApplicationTime" />
</Mapping>
```

Figure 4: Mapping section for the *broadcast* program

3.2 MetaPL and GRID

The core MetaPL notation includes `System` tags, which enable the developer to outline the application execution context, i.e., to describe the environment in which the application will be executed. This is of fundamental importance for GRID application optimization, due to the intrinsically redundant, distributed and heterogeneous characteristics of a typical GRID environment. In order to support the optimization framework, we wish to obtain from the MetaPL GRID description (by means of suitable filters):

1. the RSL file for executing the application in the real system;
2. the configuration file needed to simulate the application in HeSSE.

For simplicity's sake, we will assume here that the GRID system is a collection of independently-managed clusters (Virtual Organization, VO). To build the RSL file, it is necessary to know which resources are exported by each VO in the system. To build the simulation configuration file, it is necessary to know how each component cluster is structured, and how the clusters are interconnected.

The MetaPL elements used to provide this information are:

`Cluster` representing a single virtual organization. It provides three attributes: `name`, which is a fully qualified name that identifies uniquely the element; `HeSSEconfig`, which contains the name of an HeSSE configuration file for this cluster and contains a list of `Node` elements.

`Node` representing a node of the cluster. It is described by three attributes: `name`, which contains the node hostname, `CPUs`, which contains the number of available CPUs and `rsl`, which contains the name of a file with the rsl description of the node.

`Network` describing the network connection between two or more clusters. It contains a list of element `ClusterRef` (see below), and an element `kind`, which briefly describes the type of connection adopted (Ethernet, WAN, ...).

`ClusterRef` being just a reference to a cluster. It contains a `target` element pointing to a cluster name.

```
<System>
  <Cluster name="Orion"
           HeSSEconfig="orion.HeSSE">
    <Node name="C0-0" rsl="C0-0" />
    <Node name="C0-1" rsl="C0-1"/>
  </Cluster >
  <Cluster name="Fab4"
           HeSSEconfig="fab4.HeSSE" />
    <Node name="C0-0" rsl="C0-0"/>
    <Node name="C0-1" rsl="C0-1"/>
  <Cluster >
  <Network kind="Ethernet" >
    <ClusterRef name="orion" />
    <ClusterRef name="fab4" />
 </Network >
</System>
```

Figure 5: A MetaPL GRID description

`Client` referring to the application client terminal (see section 2). It provides `name` and `HeSSEconfig` attributes.

We have implemented the filters `RSL.xsl` and `HeSSEconf.xsl`, which process the above information and generate the RSL and configuration files. Figure 5 shows the description of a sample GRID, made of two clusters connected by an Ethernet network.

4 The Optimizing Application Starter: MHStarter

As mentioned in the introduction, we propose to perform resource allocation choices directly in user space, without higher-level intermediaries, in order to exploit application flexibility (essentially, the choice of a carefully-chosen number of processes). The idea is to exploit the ability of MetaPL and HeSSE to predict the performance of an application parameterized in the number of component tasks on a given system configuration, i.e., on a given subset of the whole GRID environment, and to perform automatic optimization at application startup. It should be noted that the availability of simulated performance figures for alternative execution scenarios can support a wide number of resource optimization policies, both at application startup and at run-time (e.g., involving process migration). However, in this paper we will consider only optimizations at application startup, and in particular, among all possible optimizations, the choice of the optimal number of processors for a given problem dimension and their allocation to best subset of available processors in the GRID environment. By “optimal number of processors” and “best subset of processors” we mean the

combination that leads to the *expected* shortest application response time *in the actual load conditions* of the target machine.

The implementation devised, which led to the results presented in the next section, relies on the use of an intelligent "starter" process, which, instead of launching the application blindly on a random or predefined set of nodes, evaluates possible configuration scenarios using current load measurements and querying the simulation framework for expected response times. In our test implementation, the optimizing starter `MHstarter` (from *MetaPL-HeSSE starter*) maintains MetaPL descriptions and HeSSE configuration files, queries the GRID environment for information about the system load status using MDS, chooses alternative configurations, performs simulations and finally starts up the application on the chosen subset of GRID nodes.

From a user perspective, `MHstarter` is just an application launcher, just like `mpirun` or `globus-run-job`, even if it needs to be suitably configured before the application is started. In practice, the user has to provide the tool with the application executable and its MetaPL description. Before starting up the application, the tool has to be informed of the problem dimension (the problem dimension is necessary to perform the simulations, and entails constraints on the possible number of application tasks). Every time that the user asks the starter to launch the application, the tool collects GRID status information, chooses a set of possible configurations (number of tasks, nodes to be used) and tests their response time using the simulator. Then it chooses the configuration leading to the shortest expected response time, and starts it on the chosen GRID subset.

The principal drawback of this mode of operation is the growth of the application startup latency, essentially due to the time required for simulations before choosing the optimal configuration. In order to reduce this time, a large set of information is collected off-line (i.e., during the set-up of the application). This makes it possible to reduce the number of simulations to be performed when the application starts. The idea is that when the user prepares a new application for optimized startup, a preliminary set of simulations takes place, to reduce the number of alternative configurations to be simulated at application launch.

This paper aims to be a proof-of-concept paper, which tries to explore the validity of the approach in a GRID environment. In this context, the number of available configurations may be very high, and some heuristics are needed to reduce the number of tests to be performed and to find, possibly, a sub-optimal solution. In the following, we will adopt very simple policies for choosing the tests to be performed off-line and at startup. Moreover, we will assume that the final user explicitly specifies the minimum and maximum number of tasks the application may be composed of. In [13], we proposed a way to reduce the tests in a cluster. More complex configuration pruning strategies are indeed necessary in a GRID environment, and will be the object of our future research.

5 Approach Validation

In order to assess the validity of the approach, we set up a testbed to compare the performance results with different execution scenarios. As execution environment we adopted Orion, a cluster of the PARSEC laboratory at the 2nd University of Naples. Orion is an IBM Blade Center cluster with an X345 Front-End and seven SMP dual-Pentium nodes, with Rocks software. We configured the cluster nodes as GRID gatekeepers, in such a way that each node may be seen as an access point to a GRID environment. This let us emulate a cluster GRID, i.e., a GRID of clusters in a geographically-limited area interconnected by Ethernet. In this way, each GRID node has two processors, and the front-end acts as a GRID client. In the following, each internal Orion node will be denoted by the name $Node_i$, where $i \in \{1, \cdots, 7\}$. Figure 6 shows the MetaPL description of application and testbed. The `Code` section reports the application skeleton, the `Mapping` section the parameters used for trace generation and the `System` element outlines the GRID environment. The `MHstarter` tool dynamically generates new MetaPL descriptions assigning values to the `NumberofProcesses` element and changing the nodes of the GRID environment in the `System` description.

When the target application starts, the high-level schedulers try to acquire the needed resources, locking the application until all the processors are available, and then starting the application. They usually grant that the application has exclusive use of the resources, but cannot guarantee how long they should wait until the resource can be acquired. Note that when resources are only partially available, the most common solution is to acquire all the available, thus reducing resource utilization. Deadlock is prevented because all applications are queued on the same high-level schedulers.

In our proposal, the application shares the resources, if it thinks that this can improve the application performance (response time). The drawback is that application startup time increases, due to the time required for simulation. However, this overhead can be reduced, restricting the number of solutions to be compared; furthermore, it is predictable, unlike the idle time due to resource acquisition. Moreover, the time spent in simulation is much less than the time needed by Globus GRAM to distribute the tasks when the resources are already available. Figure 7 shows the overhead introduced by Globus (the time spent to start an MPICH-G2 application) as the number of adopted nodes grows. Simulation time is between 0.5 and 1 second, so `MHstarter` affects the application startup with an overhead of less than 10%, which actually depends on the number of simulations to be performed.

As the `MHstarter` tool has no great negative impact on the startup performance, it is important to show that sharing the resources in many cases can improve the performance of the target application. Moreover, we aim to show that the application parameters (i.e., the number of parallel tasks) chosen "off-line" can be varied at application startup in order to increase application performance and resource optimization.

We built the following test: we started a heavy workload on the GRID

```
<MetaPL>
  <Code>
    <Task name="Jacobi"><Block>
      <CodeBlock coderegion="MPI_Init" time="10" />
      <CodeBlock coderegion="Init" time="10" />
      <Loop iteration="10" variable="nstep" >
        <Block>
          <CodeBlock coderegion="Calc" time="10" />
          <MPIAllGather dim="6720" />
        </Block>
      </Loop>
      ...
    <Print message="ApplicationTime" />
    </Block></Task>
  </Code>
  <System>
    <Cluster name="Node1" HeSSEconfig="n1.HeSSE" />
    <Cluster name="Node2" HeSSEconfig="n2.HeSSE" />
    <Cluster name="Node3" HeSSEconfig="n3.HeSSE" />
    <Cluster name="Node4" HeSSEconfig="n4.HeSSE" />
    <Cluster name="Node5" HeSSEconfig="n5.HeSSE" />
    <Cluster name="Node6" HeSSEconfig="n6.HeSSE" />
    <Cluster name="Node7" HeSSEconfig="n7.HeSSE" />
    <Network kind="Ethernet">
      <ClusterRef name="Node1" />
      <ClusterRef name="Node2" />
      <ClusterRef name="Node3" />
      <ClusterRef name="Node4" />
      <ClusterRef name="Node5" />
      <ClusterRef name="Node6" />
      <ClusterRef name="Node7" />
    </Network kind="Ethernet">
    <Client name="orion"
              HeSSEconfig="frontend.HeSSE" />
  </System>
  <Mapping>
    <NumberOfProcesses value="Parameter" />
    <Parameter name="NumberOfProcesses" />
    <Target name="ApplicationTime" />
  </Mapping>
</MetaPL>
```

Figure 6: Jacobi MetaPL description

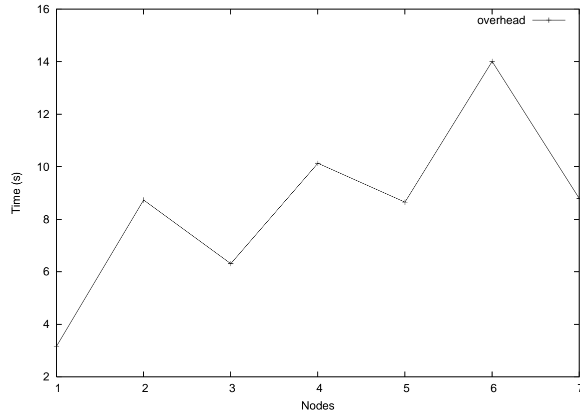

Figure 7: Time spent by Globus (MPICH-G2) to allocate the tasks

Number	Description	HeSSE Best Allocation
1	System Initial Load: 4 tasks on node	Application spawned in 6 tasks Each task is allocated on a free node
2	System Initial Load: 8 tasks equally distributed on two different nodes	Application spawned in 5 tasks Each task allocated on a free node
3	System Initial Load: 16 tasks equally distributed on four different nodes	Application spawned in 4 tasks Each task allocated on a free node
4	System Initial Load: 24 tasks equally distributed on six different nodes	Application spawned in 4 tasks One task is allocated onto the free node and the other three tasks allocated on nodes with initial load
5	System Initial Load: 8 tasks equally distributed on 4 nodes (2 on each node); 2 tasks equally distributed on 2 other nodes (1 on each node)	Application spawned in 3 tasks Two tasks allocated on the free node and one task allocated on a node with one initial task
6	System Inital Load: 12 tasks equally distributed on 6 nodes (2 on each node); 1 tasks on another node	Application spawned in 10 tasks 2 tasks allocated on the node with one initial task; 8 tasks on nodes with two initial tasks

Table 1: The Proposed Tests: Workload Conditions and Predicted Best Configurations

clusters (the Orion nodes). The workload is generated by the same application we adopted as testbed, but with a very high number of equations and iterations. We chose a set of starting workloads to help us evaluate the behavior of the `MHstarter` tool.

Table 1 summarizes the tests we used to validate our approach. The second column in the table describes the type of initial workload, while the tool predictions are in the third column. It should be noted that the prediction shows that the more irregular the initial workload, the more difficult it is to predict *off-line* which configuration assures the best performance on the system.

In order to validate these results, we have built a testbed where the initial workloads for each test in Tab.1 are submitted to the real system. Then we start up the test application, choosing from 1 to 7 (for the first four tests) and to 14 (for the last two tests) the number of tasks to be spawned in the GRID environment. For the first four tests, each spawned task is allocated on a different node. The nodes with initial workload are allocated first. So if 7 tasks are spawned for the first test, the first one is allocated on the node with

initial workload and the others on the remaining free nodes. All the performance results presented in the following have been statistically validated repeating the tests a suitable number of times.

Figure 8 shows the resulting response time for the first four tests. Even if the application scales well for less than 3 tasks (when the number of equations or iterations increases, this value grows), its performance grows up to 7 tasks. Moreover, sometimes the use of 4 tasks improves the performance when the system is subject to load. As it is possible to note from the figure, the predictions are validated. Starting the application without any global scheduler, 3 tasks will be a good choice.

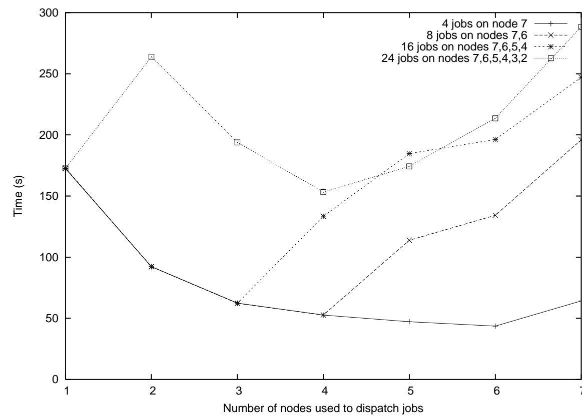

Figure 8: Tests 1-4

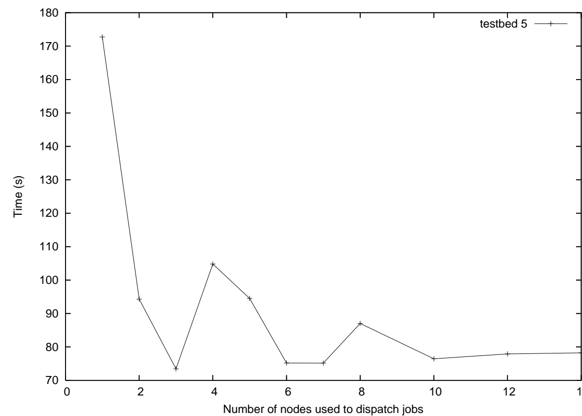

Figure 9: Test 5

Figures 9 and 10 report the response time for applications when Testbed 5 and 6 are reproduced, respectively. In these cases, applications are spawned from 1 to 14 tasks and two of them are allocated on each node in the same way

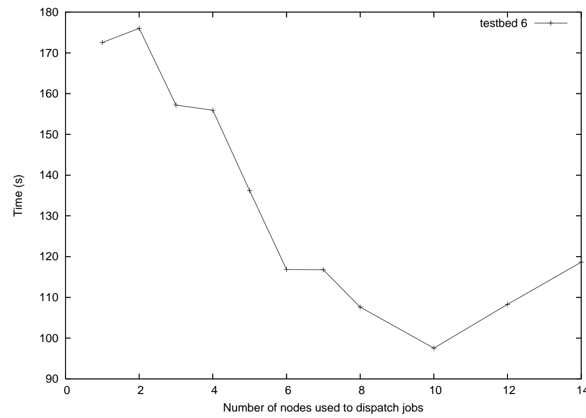

Figure 10: Test 6

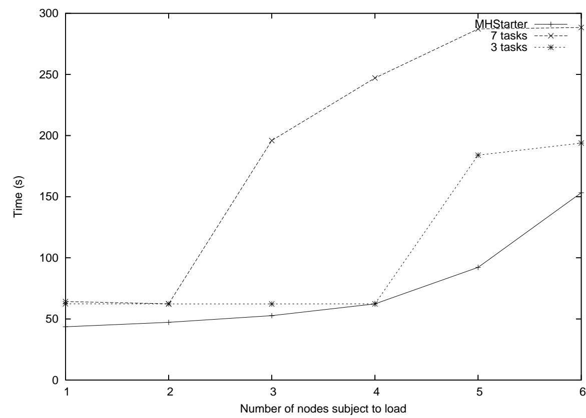

Figure 11: Jacobi application with increasing load

as in the previous tests. In both cases, HeSSE predictions are verified, but it is interesting to note that the more complex the initial workload distribution on nodes, the more irregular the application behavior when spawning it in different number of tasks.

Finally, Figure 11 compares the static choices of 3 parallel tasks, 7 parallel tasks, and the tasks chosen by the MHstarter. The x-axis corresponds to the number of nodes subject to load, and the y-axis shows the application response time. The static choice of 3 tasks (as already shown in the previous figures) leads to a shorter response times than the static choice of 7 tasks. MHstarter chooses always the best configuration, independently of the load conditions, and leads always to the best performance, i.e., to the shortest response time.

6 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 the 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 a 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 [24]. However, none of the simulation environments currently available seem 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 [25] 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 a centralized global scheduling methodology and hence it is not suitable for simulation of environments where there are multiple uncoordinated schedulers. Microgrid [26] 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 in a virtual Globus environment. While on the one hand this characteristic leads to high accuracy results, on the other it negatively affects simulation speed. In practice, most applications are executed in Microgrid in a time longer than the one required in the actual environment, thus using the simulator is not viable when performing a large number of experiments. Simgrid [27] 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. However, the model of resource sharing used for communication is not suitable for short-lived TCP flows. In fact, in conditions such as those analyzed in this paper, Simgrid would require the use of an external packet-level simulator. Gridsim [28] 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 and definition of CPU processing power. However, it is fairly limited as far as network and I/O devices simulation is concerned.

7 Conclusions

In this paper, we have proposed an approach for resource allocation that does not resort to a system high-level resource manager. The idea is that a user-level application launcher, accessing the distributed system status and exploiting per-

formance predictions obtained by simulation, may tune the application resource requests choosing a suitable resource allocation scheme that takes into account both system-dependent parameters (i.e., resource status and utilization) and application-dependent parameters (i.e., number of parallel tasks). The objective is to make optimized choices that can lead to reduced application response time.

In order to show the usefulness of the approach, here we have considered a single aspect of the general resource allocation problem, process allocation. Moreover, we considered only optimizations at application startup, and in particular, among all possible optimizations, the choice of the optimal number of processors for a given problem dimension and their allocation to best subset of available processors in the GRID environment. By "optimal number of processors" and "best subset of processors" we mean the combination that leads to the *expected* shortest application response time *in the actual load conditions* of the target machine.

In order to validate the proposed approach, we have developed an optimizing application launcher, `MHstarter`, and tested it on a single cluster suitably configured to emulate the behavior of a cluster GRID in different working conditions. We showed that startup choices affect the resulting performance of the target applications and that our approach may help to optimize the response time.

On the basis of these results, our future work will try to exploit the full potential of the approach. We will explore the possibilities to reduce the number of configurations to be compared by simulation, defining a flexible way to describe the policies using the same language (MetaPL) adopted to describe the applications. Moreover, we will widen the kind of resources we are able to manage, extending the simulator capabilities. Finally, we intend also to explore in more detail what happens in heterogeneous systems, building up a larger testbed enabling us to emulate more complex architectures.

Acknowledgements

We would like to thank the anonymous referees for their precise and helpful suggestions that allowed us to improve the previous version of this paper.

References

[1] R. Buyya, D. Abramson, J. Giddy, Nimrod/G: An architecture of a resource management and scheduling system in a global computational grid, in: Proc. of HPC Asia 2000, Beijing, China, 2000, pp. 283–289.

[2] Cluster Resources Inc., Moab Grid Scheduler (Silver) Administrator's Guide, `http://www.clusterresources.com/products/mgs/docs/index.shtml` (2005).

[3] D. B. Jackson, H. L. Jackson, Q. Snell, Simulation based hpc workload analysis, in: IPDPS '01: Proceedings of the 15th International Parallel & Distributed Processing Symposium, IEEE Computer Society, Washington, DC, USA, 2001, p. 47.

[4] I. Foster, C. Kesselman, S. Tuecke, The anatomy of the Grid: Enabling scalable virtual organization, in: The International Journal of High Performance Computing Applications, Vol. 15, num. 3, Sage Publications, USA, 2001, pp. 200–222, http://www.globus.org/research/papers/anatomy.pdf.

[5] Globus Alliance, WS GRAM: Developer's Guide, http://www-unix.globus.org/toolkit/docs/3.2/gram/ws/developer (2005).

[6] The Globus Alliance, The Globus Resource Specification Language RSL, http://www-fp.globus.org/gram/rsl_spec1.html (2005).

[7] I. Foster, C. Kesselman, J. M. Nick, S. Tuecke, The physiology of the Grid: An open grid services architecture for distributed systems integration., 2002, http://www.globus.org/research/papers/ogsa.pdf.

[8] C. Smith, L. Williams, Software performance engineering for object-oriented systems, in: Proc. CMG, Orlando, USA, 1997.

[9] K. Kleese van Dam, S. Sufi, G. Drinkwater, L. Blanshard, A. Manandhar, R. Tyer, R. Allan, K. O'Neill, M. Doherty, M. Williams, A. Woolf, L. Sastry, An integrated e-science environment for environmental science, in: Proc. of Tenth ECMWF Workshop, Reading, England, 2002, pp. 175–188.

[10] K. Keahey, T. Fredian, Q. Peng, D. P. Schissel, M. Thompson, I. Foster, M. Greenwald, D. McCune, Computational grids in action: the national fusion collaboratory, FGCS Future Generation Computer Systems 18 (8) (2002) 1005–1015.

[11] G. Folino, G. Spezzano, An autonomic tool for building self-organizing Grid-enabled applications, FGCS Future Generation Computer Systems 23 (5) (2007) 671–679.

[12] M. Parashar, H. Klie, U. Catalyurek, T. Kurc, W. Bangerth, V. Matossian, J. Saltz, M. F. Wheeler, Application of grid-enabled technologies for solving optimization problems in data-driven reservoir studies, FGCS Future Generation Computer Systems 21 (1) (2005) 19–26.

[13] E. Mancini, M. Rak, R. Torella, U. Villano, Self-optimizing mpi applications: A simulation-based approach, in: L. T. Yang, O. F. Rana, B. D. Martino, J. Dongarra (Eds.), High Performance Computing and Communications: First Int. Conf., LNCS, Vol. 3726, Sorrento, Italy, 2005, pp. 143–155.

[14] E. Mancini, M. Rak, R. Torella, U. Villano, Predictive autonomicity of web services in the mawes framework, To appear in Journal of Computer Science.

[15] N. Mazzocca, M. Rak, U. Villano, The transition from a PVM program simulator to a heterogeneous system simulator: The HeSSE project, in: J. Dongarra et al. (Ed.), Recent Advances in Parallel Virtual Machine and Message Passing Interface, Lecture Notes in Computer Science, Vol. 1908, Springer-Verlag, Berlin (DE), 2000, pp. 266–273.

[16] N. Mazzocca, M. Rak, U. Villano, The MetaPL approach to the performance analysis of distributed software systems., in: Proc. of 3rd International Workshop on Software and Performance (WOSP02), IEEE Press, 2002, pp. 142–149.

[17] N. Mazzocca, M. Rak, U. Villano, MetaPL a notation system for parallel program description and performance analysis parallel computing technologies, in: V. Malyshkin (Ed.), Parallel Computing Technologies, Lecture Notes in Computer Science, v Edition, Vol. 2127, Springer-Verlag, Berlin (DE), 2001, pp. 80–93.

[18] E. Mancini, N. Mazzocca, M. Rak, U. Villano, Integrated tools for performance-oriented distributed software development, in: Proc. SERP'03 Conference, Vol. 1, Las Vegas (NE), USA, 2003, pp. 88–94.

[19] E. Mancini, M. Rak, R. Torella, U. Villano, Performance oriented development and tuning of grid applications., in: Applied Parallel Computing, State of the Art in Scientific Computing, Lecture Notes in Computer Science, Vol. 3732 of Lecture Notes in Computer Science, Springer, 2006, pp. 509–518.

[20] N. Mazzocca, M. Rak, R. Torella, E. Mancini, U. Villano, The HeSSE simulation environment, in: Proc. ESMc'2003, 2003, pp. 27–29.

[21] J. Dongarra et al., MPI 1.1 Standard, `http://www-unix.mcs.anl.gov/mpi/mpi-standard/mpi-report-1.1/mpi-report.htm` (June 1995).

[22] MPICH-G2 Standard, `http://www-unix.mcs.anl.gov/mpi/mpich1` (June 2005).

[23] E. Mancini, M. Rak, R. Torella, U. Villano, A simulation-based framework for autonomic web services, in: Proc. of the Eleventh Int. Conf. on Parallel and Distributed Systems, Fukuoka, Japan, 2005, pp. 433–437.

[24] I. Dimov, J. Dongarra, K. Madseni, J. Wasniewski, Z. Zlatev, Application of Distributed and Grid Computing, FGCS Future Generation Computer Systems, Elsevier, 2007.

[25] K. Aida, A. Takefusa, H. Nakada, S. Matsuoka, U. Nagashima, A performance evaluation model for effective job scheduling in global computing systems., in: Proc. of HPDC, 1998, pp. 352–353.

[26] H. Xia, H. Dail, H. Casanova, A. A. Chien, The microgrid: Using online simulation to predict application performance in diverse grid network environments., in: Proc. of CLADE, 2004, p. 52.

[27] A. Legrand, L. Marchal, H. Casanova, Scheduling distributed applications: the simgrid simulation framework., in: CCGRID, 2003, pp. 138–145.

[28] R. Buyya, M. M. Murshed, Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing, Concurrency and Computation:practice and Experience 14 (2002) 1175–1220.