An Economy-driven Mapping Heuristic for Hierarchical Master-Slave Applications in Grid Systems

Nadia Ranaldo[1], Eugenio Zimeo[2]

[1]University of Sannio
Department of Engineering
Benevento, 82100, Italy
ranaldo@unisannio.it

[2]University of Sannio
Research Centre on Software Technology
Benevento, 82100, Italy
zimeo@unisannio.it

Abstract

In heterogeneous distributed systems, such as Grids, a resource broker is responsible of automatically selecting resources, and mapping application tasks to them. A crucial aspect of resource broker design, especially in a next commercial exploitation of grid systems, in which economy theories for resource management will be applied, is the support to task mapping based on the fulfilment of Quality of Service (QoS) constraints. The paper presents an economy-driven mapping heuristic, called time minimization, for mapping and scheduling the tasks assigned to the slaves of a master-slave application in a hierarchical and heterogeneous distributed system. The validity and accuracy of such heuristic are tested by implementing it in a resource broker of a hierarchical grid middleware used for running a real world application.

1. Introduction

A main challenge to efficiently execute parallel and distributed applications in a grid system is the effective management of the large amount of available computational, storage and communication resources in order to optimize the mapping on them of application tasks. To this end, a central subsystem of a middleware for grid computing is the *Resource Management System* (RMS). The main goal of the RMS is to enable a transparent access to the pool of resources available in the Grid.

A fundamental component of an RMS is the *resource broker*, which is mainly responsible of accepting execution requests from the user and assigning them to specific resources selected from the pool of available ones [1]. In addition to this task, which is accomplished through an Information System, the broker schedules for the execution the application tasks distributed on the selected resources, which are continuously monitored and managed.

Mapping, scheduling, and execution are performed so that some QoS requirements provided by the requester are satisfied. In particular, the resource broker assigns tasks to resources on the basis of specified policies, called *mapping policies*, used to determine the best schedule for the application on the set of available resources. The mapping task phase, moreover, is one of the most important functions for a resource broker, since it has a direct impact on service response times and, as a consequence, on user satisfaction.

The deployment of grid systems in the next years will foster the adoption of new business models for computing services. Big companies and organizations will prefer to rent computing resources instead of buying them. In such a scenario, the efficiency improvement of resource allocation and scheduling will be possible through the adoption of *economic models* to apply among resource suppliers and requesters. In recent years, in this perspective, some economics theories have been studied for the implementation of resource management in a grid environment [2]. A market, in fact, similarly to a grid system, is decentralized, dynamic and deals with competitive resources. The basic components in a market are producer, consumer and commodities, corresponding respectively to resource owner, resource requester (user) and distributed resources in a grid environment.

An economy-driven resource management system provides a mechanism for regulating the supply-and-demand for resources and allocating them to applications based on user preferences, such as cost and time constraints. It also gives an incentive to resource owners for making resources available for computation and a basis to users for appropriately trading the quality of service guarantees they receive against cost.

1-4244-0054-6/06/$20.00

In grid resource management systems a fine accuracy of mapping decisions requires, in addition to detailed information of the underlying distributed system, the knowledge and model of application features, in terms of computational requirements of each task, data and execution dependencies and communication patterns.

This paper presents the definition of an application-oriented and economy-driven mapping heuristic, called *time minimization*, for mapping distributed tasks on a set of heterogeneous and distributed resources. The proposed heuristic is applied to a specific class of applications which presents the same task dependencies, that is the class of applications based on the master-slave model.

The main reasons of such choice are: (1) the master-slave model is very spread in scientific community and can be adopted to achieve a good performance improvement of a wide class of applications in many fields of science and engineering, such as weather prediction, astrophysics, bioinformatics, earth quake research, ground water pollution, multi particle physics, etc.; (2) applications programmed according to the master-slave model well fit grid systems because of well-defined and limited communication patterns among the distributed computing entities.

In fact, communication takes place only between the master and the slaves and in well-defined times of execution. For these reasons, this model is well suited to be used in Grids where high-latency and shared wide-area networks are a strong limitation for the execution of applications based on tightly-coupled parallelism.

The simplest approach for performing repeatable tests on task mapping, overcoming resource availability problems, is to resort to simulation. Well known projects in the context of grid simulation are GridSim [3] and Simgrid [4]. On the other hand, the simulation approach can be hardly used for studying, in a repeatable and non-intrusive way, the subtle interactions between large-scale shared distributed hardware, application software, and complex resource management policies, which are typical of grid environments. As a consequence, evaluation results produced by simulation tools to study the effectiveness of task mapping algorithms may be inaccurate.

For this reason the accuracy of the proposed heuristic and its usefulness in mapping and scheduling tasks of real world applications with time and cost constrains is studied using such heuristic for implementing the mapping phase of a resource broker of HiMM (Hierarchical Metacomputer Middleware) [5], a grid middleware which delivers basic services of computation, information, communication and resource management for parallel and distributed object-oriented applications. Nevertheless, the solution proposed can be easily applied to other grid middlewares.

The time minimization heuristic is, moreover, tested on an actual test-bed executing applications written for the *TMS* (*Transparent Master Slave*) *Framework* [6], a reflection-based framework for master-slave applications, able to re-use existing code for non-distributed solutions and to automatically parallelize and schedule the tasks among multiple resources, by satisfying user QoS constraints.

The rest of the paper is organized as follows. Section 2 presents related work, Section 3 describes the application and grid model, Section 4 describes the proposed mapping heuristic, Section 5 presents an evaluation of the proposed heuristic performed adopting some experimental results conducted on a real distributed environment. Finally Section 6 summarizes the paper and presents future work.

2. Related Work

Task mapping policies can be distinguished in *system-oriented* policies, which try to optimise the overall resource utilization, and *application-oriented* policies, which try to optimise application execution.

Application-oriented policies are often *QoS-based*, since they permit to satisfy QoS requirements specified by the user. The optimal solution for such kind of policies in a heterogeneous and distributed environment leads to NP-complete problems and different heuristics have been proposed in the literature to reach a near-optimal solution.

Most of the grid systems use application-oriented mapping heuristics which try to optimise only the execution performance, by adopting a mapping that minimizes the overall execution time with respect to the available resources. A simple example is the greedy approach [7], which iteratively allocates each task to the resource that is likely to produce the best performance, without considering the rest of pending tasks. This approach leads usually to sub-optimal solutions, since the mapping decisions are based only on local task information.

More complex mapping algorithms, which try to overcome local optimal solutions, are based on genetic algorithms [8], simulated annealing [9] or branch and bound methods [10]. Other proposals of mapping algorithms are Workerqueue (WQ), Workerqueue with Replication (WQR) and dynamic FPLTF (Fastest Processor to Largest Task First) [11, 12].

Some mapping algorithms specific for master-slave applications have been presented in [13]. These algorithms adopt an accurate grid model as regards computation and communication performance, but do not take into account economic aspects of resources.

On the other hand, in a future commercialisation of the Grid, a resource characterization based only on performance features is not sufficient to properly model

resources. In economy theories, in particular, the price of resource utilization is the only additional parameter necessary to operate, providing so a simplified way to model the resource owner satisfaction and the competition among them. In [2] some economic models, that can be applied for managing grid resources and determining the price of resource utilization, have been studied, for example the commodity market model, the posted price model, the bargaining model, the auction model, the trading model and the monopoly/oligopoly model.

Paper [14] represents one of the first effort to introduce economy-driven mapping algorithms based on deadline and budget constraints. Such algorithms were tested in Nimrod-G [15], a grid resource management system for scheduling parametric applications. The proposed approach, however, does not model task dependencies and do not ensure the assignment and execution of all the tasks, since it considers a shared and dynamic environment in which only a task at a time is assigned until the budget is consumed.

Our idea, instead, is to grant the execution of all the tasks of the application respecting the established terms during the initial negotiation phase between the resource broker and the user. Our approach, in particular, requires to exploit selected resources in an exclusive way, so to maintain the predicted performance by the heuristic in the time necessary to complete the application execution.

A possible approach to grant the complete availability of selected resources to scheduled applications is the use of advance reservation mechanisms, both for communication and computational resources, delivered by the grid middleware [16].

3. Application and Grid Models and Notations

In this section the class of applications and grid systems for which our approach can be applied are described and modelled.

3.1 Application Model

A very simple model of application is the *unstructured application*, that is a distributed application composed of a set of independent and self-contained tasks. This model is considered as reference model in various resource broker systems for the activities of task mapping and scheduling, but can not be considered a realistic model, since distributed applications are typically composed of tasks characterized by data and control dependencies. In particular the computation in a distributed application can be viewed as a precedence graph, that is, a *directed acyclic graph* (DAG). Each node in the DAG corresponds to a computation or communication task. The edges of the DAG express the dependencies between the tasks, that can be of execution, data, or both.

Defining mapping algorithms, able to manage the task dependencies of an application described by a generic DAG, is difficult. In this work we focus on a class of applications described by the same DAG structure, and in particular those which adhere to the master-slave pattern [17]. In this pattern there are two kinds of entities: *master* and *slave*. Generally a master decomposes the problem into smaller tasks, distributes these tasks among a farm of slaves and waits for partial results. Each slave performs its processing on the received task, then returns the result of the processing back to the master, which gathers and assembles the partial results in order to produce the final solution of the computation.

We consider, in particular, the master-slave pattern for coarse-grained parallel applications, which embody a wide spectrum of application domains, from combinatorial optimization problems, to data mining, to processing of large measurement data sets, etc. For such applications the partition of input data is used to induce a partition of the overall workload among slaves (*data partitioning*), and, in particular, the following parallelization technique is adopted: the same sequential algorithm is executed simultaneously by the slaves, processing different input data.

For the master-slave pattern we propose an application model in which the master performs a preliminary task of partitioning of input data, and a task of processing of partial results. The latter is performed when all the partial results are returned by the slaves.

The slaves, moreover, perform the overall workload, which is decomposed into a high number of sub-tasks, called *atomic sub-tasks*, which are parts of the original workload that can not be further decomposed. By definition, the atomic sub-tasks are independent, do not communicate with each other and are considered the smallest tasks which can be mapped onto resources. The application is characterized by the *computation size* (or complexity), called N, that corresponds to the total number of atomic sub-tasks, each of which is characterized by the same complexity in terms of computation, data storage and data transfer aspects (see figure 1).

However, for a large number of distributed resources the centralized control of master can become a bottleneck for applications and a single point of failure.

To overcome such limitations, a hierarchical version of the master-slave pattern is taken into account, by extending the single master to a hierarchy of masters, each of which controls a different group of slaves at different hierarchical levels.

In the *hierarchical master-slave pattern* [18], the master at the top of the hierarchy partitions input data to the underlying masters, and so on, until to reach the

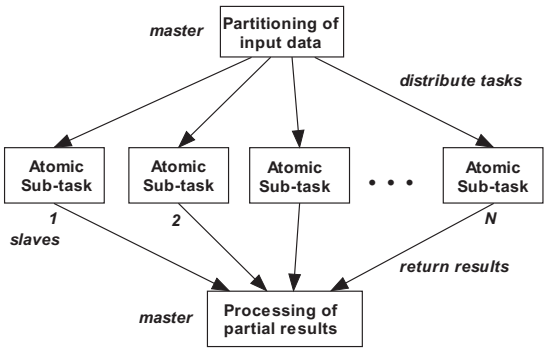

Figure 1. The Master-slave pattern.

slaves, that directly process the received input data (see figure 2).

Hierarchical master-slave applications, executed in a distributed environment, require to host masters and slaves on different computational resources. In this context the DAG structure requires to model, in addition to computation tasks performed by masters and slaves, communication tasks, due to the realistic limited performance of inter-networking resources. Such tasks are related in particular to input data transfers performed by the master towards the sub-master, until to the slaves, and to output data transfers performed by the slaves towards the master hierarchy.

The paper focuses on applications for which partitioning of input data task, processing of partial results task and communication tasks can be disregarded with respect to computation tasks performed by the slaves, whose optimised distribution on distributed heterogeneous resource determines the significant

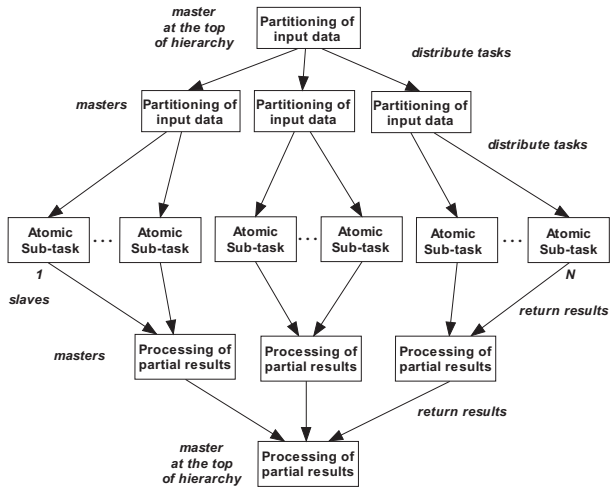

Figure 2. The hierarchical master-slave pattern.

performance improvement of the overall application.

Such simplification will be removed in a future work in which a mapping policy for all the tasks of a hierarchical master-slave computation will be taken into account.

3.2 Grid Model

A grid system is generically composed of a pool of distributed and heterogeneous computational resources, characterized by different computation performance, inter-connected through communication resources and accessible via the Internet. In such distributed systems, some computational resources (especially nodes of clusters and networks of workstations) are often not directly accessible via the Internet and use dedicated, high-performance networks whose protocols may not be IP-compliant. As a consequence, a *hierarchical organization* is a natural topology to represent grid systems and offers many advantages: (1) it can be adopted to recursively divide an application workload into multiple distributed and concurrent tasks running simultaneously on grid resources; (2) each administrative domain can be controlled separately from the others and so remote resource owners can easily enforce their own policies on external users; (3) grid applications can exploit dedicated and high-performance resources (supercomputers, dedicated high-speed networks of workstations, etc.) non-directly accessible through the Internet; (4) it removes the single centralization point for grid management, and so improves scalability.

In a hierarchical topology, the resources are virtually organized into different levels, and only resources belonging to the same level can directly communicate among them. A designed machine of a level is used by the resources to communicate with the level above them or below them. In this organization, grid users access only to resources at the highest level, while the complexity of the underlying hardware and software infrastructure is hidden.

Our grid model considers the grid system as a hierarchy of computational resources distributed on multiple levels. Due to the simplification of the application model, communication resources and their related performance are not modelled: this corresponds to consider a distributed environment characterized by high communication performance.

Each level of the grid system is modelled as a set R of resources, each of which is characterized by computation performance and cost attributes. We indicate a resource with R_i, with, $i:1..M$, where M is the number of available resources (see figure 3). A resource can be *simple*, if it represents a single computer, or *aggregate*, if it represents a pool of computers of a lower level in the hierarchy (such as a cluster or a network of workstations).

The performance of a simple resource R_i is modeled as the total time required for the processing of an atomic sub-task and is indicated with t_i. The cost of a simple resource R_i, called c_i, is modeled as the cost of resource usage for the processing of an atomic sub-task. Called N_i the number of resources of the lower level of R_i, and R_{ij} with $J:1,..N_i$, the lower level resources, the performance of an aggregate resource R_i can be modelled as the sum of the times t_{ij}, which are the times that each resource R_{ij} requires for processing an atomic sub-task, divided by N_i, while the cost can be modelled as the sum of the costs c_{ij} per atomic sub-task of each R_{ij}. A more accurate model of aggregate resources, which takes into account the heterogeneity of resources of the lower level, will be considered in a future work.

4. The Time Minimization Heuristic

The time minimization heuristic is based on the economy model proposed in [2] in which the execution of an application is subject to both time and cost constraints. It regards the problem of finding the "best" resources on which to run the slaves of a master-slave application and the best assignment of distributed tasks to them. It can be adopted into a hierarchical environment, applying periodically the same procedure to each level of the hierarchy and considering both simple and aggregate resources related to that level.

The parameters specified by the user, that determine the objective function, are: (1) the total execution time, which represents the deadline, called D, and (2) the total price for resource usage, which represents the available budget, called B.

The proposed heuristic selects a sub-set of resources from the pool of available ones, so that the aggregate cost for their usage is lower than budget B (but not necessarily the minimum) and that are able to complete the application execution as quickly as possible (time minimization) and within deadline D.

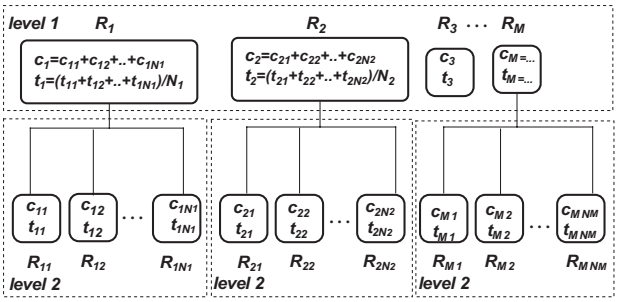

Figure 3. The grid model.

The goal of the proposed heuristic is to minimize the total execution time, so the distribution has to be chosen in order to utilize the best performance resources to minimize execution time on each resource. Since we consider economic aspects of resource usage, the distribution is also performed so as to not exceed the budget.

With respect to the application and grid models described in the previous section, the mapping problem can be formalized as follows. The objective function is:

$$(1) \quad \min(t_i n_i) \quad \forall i : 1..M$$

where n_i denotes the number of atomic sub-tasks assigned to resource R_i and represents the unknown quantity of the algorithm.

Objective function (1) must be reached respecting the following constraints:

$$(2) \quad t_i n_i \le D \quad \forall i : 1..M$$

$$(3) \quad \sum_{i=1}^{M} n_i = N$$

$$(4) \quad \sum_{i=1}^{M} c_i n_i \le B$$

where n_i has to be not negative, (2) ensures that the deadline is not exceeded, (3) is the constraint on the actual execution of N tasks, and (4) ensures that the budget is not exceeded. Since an atomic sub-task can not be further divided, an additional constraint is that n_i must be integer, since it represents the number of atomic sub-tasks assigned to resource R_i.

To minimize the execution time, we assign a non uniform number of atomic sub-tasks to the available slaves such that they will finish at roughly the same time. So (1) is equivalent to the following objective function:

$$(5) \quad t_i n_i \cong t_j n_j \quad \forall i, j : 1..M$$

The approximation in (5) is caused by the heterogeneity of resources, which are characterized by different performance, and by the constraint on n_i to be adjusted to integers.

Indicated with $\bar{t}$ the execution time of an atomic sub-task for the resource with the highest performance, (5) implies:

$$(6) \quad t_i n_i \cong \bar{t}\,\bar{n} \quad \forall i : 1..M$$

where $\bar{n}$ is the number of atomic sub-tasks assigned to the quickest resource. The calculation of $\bar{n}$ permits to evaluate n_i as:

$$(7) \quad n_i \cong \frac{\bar{t}}{t_i}\ \bar{n} \quad \forall i : 1..M$$

that, applied to (3) and (4), leads to:

$$(8) \quad \sum_{i=1}^{M} n_i \cong \sum_{i=1}^{M} \frac{\bar{t}}{t_i}\ \bar{n}\ \cong N$$

$$(9) \quad \sum_{i=1}^{M} n_i c_i \cong \sum_{i=1}^{M} \frac{\bar{t}}{t_i}\ \bar{n} c_i \le B$$

Generally constraints (2), (8) and (9) are not satisfied simultaneously. As a consequence, we propose an iterative algorithm whose steps are described in the following:

(a) Consider set R of all available resources and indicate with R_p the quickest resource. Solve (9) in order to find real value $\bar{n}$ considering to consume the budget, that means to equalize the total cost to B. Then approximate $\bar{n}$ to the nearest smaller integer, so to not exceed B.

(b) Use $\bar{n}$ to calculate n_i of all the other resources, solving (7) for all $i:1,..M$ except p.

(c) Use constraint (8), considering the equality to N, in order to find the factor x to normalize the calculated n_i values so that their sum is N:

$$\sum_{i=1}^{M} (n_i x) = N$$

i. If $x>1$, that means that (8) can not be satisfied using the current makespan, continue to step (e);

ii. If $x<=1$, that means that (8) can be satisfied using the current makespan, recalculate n_i multiplying them by the factor x.

(d) Verify constraint (2).

i. If (2) is violated the algorithm ends without a solution;

ii. If (2) is not violated then the algorithm ends with a solution given by n_i calculated at step (c).ii.

(e) Sort list R_o of resources by increasing order of cost per atomic sub-task. If two or more resources have the same cost, order them for decreasing performance, so to prefer, among resources with the same cost, the quickest ones:

$$R_o = (R_1,.., R_j,.., R_M)$$

$$with \quad c_j < c_{j+1}\ or\ c_j = c_{j+1}\ and\ t_j < t_{j+1} \quad \forall j = 1..M$$

In this order the most expensive resource is R_M.

(f) Because of the required exceeding budget, it is necessary to decrease the number of atomic sub-tasks assigned to the most expensive resource. As a consequence, a part of atomic sub-tasks assigned to R_M, called ε, is distributed among the *(M-1)* resources in a proportional manner to their performance. To this aim, the following system has to be solved:

$$(10) \quad \begin{cases} \sum_{i=1}^{M-1} \frac{\bar{t}}{t_i}(\bar{n}+\varepsilon)c_i + \frac{\bar{t}}{t_M}(\bar{n}-\varepsilon)c_M = B \\ \sum_{i=1}^{M-1} \frac{\bar{t}}{t_i}(\bar{n}+\varepsilon)_i + \frac{\bar{t}}{t_M}(\bar{n}-\varepsilon) = N \end{cases}$$

i. If there is a solution with $(\bar{n}-\varepsilon) > 0$ that verifies constraint (2) on deadline, then the solution of the time minimization heuristic is given by n_i calculated as:

$$(11) \quad \begin{cases} n_i = \frac{\bar{t}}{t_i}(\bar{n}+\varepsilon) \quad \forall i = 1..M-1 \\ n_M = \frac{\bar{t}}{t_M}(\bar{n}-\varepsilon) \end{cases}$$

rounding the obtained values to the nearest integers.

ii. If there is not a solution with $(\bar{n}-\varepsilon) > 0$ that satisfies the constraint on deadline, then the procedure is iterated from step (a), eliminating the resource R_M from the set of available resources. If steps from (a) to (e) are iterated for M times terminating without a solution, it is possible to conclude that there is not a sub-set of the available resources which permits to satisfy simultaneously the constraints on deadline and budget.

Starting from objective function (5) and constraints (2), (3) and (4), an optimal solution could be reached with real values of n_i. On the other hand, a good rounding to the nearest integers permits to remain very close to the optimal solution.

In order to obtain the solution that as best as possible approximates the optimal one, we propose a generic procedure for the rounding of n_i values to integers, respecting the established deadline and budget constraints and maintaining N as total summation of all n_i. In this procedure an n_i value is considered *integer* when the absolute value of the difference between it and the nearest integer is less than a specified little value $\delta>0$.

Consider list R_o of resources ordered as indicated at step (e) of the procedure.

- If n_1 is not an integer, if deadline is not exceeded for resource R_1, n_1 is approximated to the nearest greater

integer, indicated with g_1, else it is approximated to the smaller one, indicated with s_1.

In the case of approximation to the nearest greater integer, the difference between g_1 and n_1 is subtracted to n_i of the necessary resources in the list, starting from R_2, until such values become the respective nearest smaller integers (eventually except for the last one).

In the case of approximation to the nearest smaller integer:

- the difference between n_1 and s_1 is added to the resources in list R_o, starting from resource R_2, until their values become the respective nearest greater integers (eventually except for the last one, called R_L).
- (9) constraint is checked. If (9) is violated, repeat the previous step without considering R_L and the following ones, and so on. If no adjustment permits to satisfy (9), the procedure ends without a solution.
- (2) constraint, for resources whose n_i was changed, is checked. If (2) is violated for one or more resources, the total quantity of n_i values which cause the violation is distributed among the previous resources and the previous step is repeated.

- Repeat starting from the following resource in list R_o, (R_2, R_3, until to R_M) and until all n_i are rounded to integer.

Such rounding procedure determines the relative error of the heuristic result from the optimal one, that can be considered very trivial in the following case:

$$t_i n_i >> t_j \quad \forall i, j : 1..M$$

which is verified in the case of high values of n_i.

5. Experimental Results

The time minimization heuristic was tested on applications written for the TMS Framework, an object-oriented framework for transparent and hierarchical master-slave applications, which is able to re-use existing code for non-distributed solutions and to automatically parallelize and distribute the tasks generated by the execution among multiple distributed resources.

5.1 The TMS Framework

Thee TMS Framework manages the functionalities of resource management, scheduling and communication required to deploy the application and is able to leverage already existing services delivered by the middleware used as underlying layer.

Such framework was implemented through reflection mechanisms provided by the run-time proxy-based Meta-Object Protocol (MOP) provided by the ProActive library for distributed programming [19]. By using MOP, the TMS Framework permits to use every existing class to transparently instantiate the set of active objects that correspond to masters and slaves instances in the hierarchical topology, maintaining the application similar to that used for a sequential computation. Therefore, the hierarchical master-slave pattern is dynamically implemented and every active object can be turned in a master able to transparently split the service task into sub-tasks and in a slave able to perform the assigned part of the overall task.

The programming model of the TMS Framework requires to provide the name of the configuration file used to dynamically configure the deployment of active objects. It is an XML-based file, called *Job Description Format* (JDF), which follows a well-defined format. In particular, a part depends on the underlying middleware adopted for active object management and communication, while the other one is common and is used for the reflection mechanisms. The common part, in particular, contains the following information: the method whose invocations have to be distributed over master and slaves objects; the input parameters that have to be partitioned and the policy to partition each of them; the assembly policy of the output parameter.

Thanks to the ProActive-HiMM middleware developed by the authors in a previous research work [20], the TMS Framework can be easily adopted to leverage HiMM, a hierarchical component and Java-based middleware for grid computing, which delivers grid services of information, communication and resource management. HiMM, in particular, provides a resource broker which allows users to submit applications specifying application code, input data, features and QoS requirements described using XML-based descriptor files [21].

The time minimization heuristic, implemented for the task mapping phase of the HiMM resource broker, adopts information on the application structure and economic-based requirements contained in the XML-based files specified by the user and the performance and cost information about the available resources delivered by the *Resource Managers*, HiMM components whose task is to publish information about available resources in the grid system. The performance information is necessary in order to predict the execution time of the assigned number of atomic sub-tasks to each resource.

In a heterogeneous environment, accurate performance prediction of resources is a fundamental aspect in order to obtain an efficient mapping and so the real fulfillment of user QoS requirements.

The prediction performance represents an important research area in which many efforts have been made. Several solutions have been proposed and adopted to evaluate the resource performance. These can be based on

benchmarking, analytical modelling, and monitoring.

In benchmarking methodologies well-defined applications are executed on systems to measure the performance in term of CPU utilization, RAM occupation, etc., which can be also used as a basis for comparisons with other systems. Analytical modelling methodologies require the construction of a mathematical or logical model that represents the behaviour of the system and that has to be analytically evaluated (an example is the LogP model [22]). Monitoring approach is used to continuously measure and analyse the performance of systems (such as NWS [23]).

In the current approach, because of the assumption of using resources in a exclusive way by adopting reservation mechanisms, a simple benchmarking-based performance estimation is adopted to predict resource performance in terms of execution times. Since the adoption of a generic benchmark can not be used to accurately evaluate the execution time of a specific application, our solution is to evaluate resource performance with respect to the specific atomic sub-task of an application. As a consequence we suppose to use a Resource Manager which publishes information related to the time required to execute an atomic sub-task of an application and the cost per task expressed in dollars. Atomic sub-tasks are identified by symbolic names, execution times are expressed in seconds, and have to refer to the computational size of the application, if they depend on it.

5.2 A Case-study

The time minimization heuristic was tested on various applications written for the TMS Framework, among which matrix multiplication, finite integral calculation, Mandelbrot set computation, image rendering, etc. For each of these applications, application complexity, atomic sub-task, partition and assembly policies were individuated. For example for the multiplication of two square matrices of n^2 dimension, the application complexity corresponds to the number of rows, that is n, an atomic sub-task is the computation of a row of the result matrix, which can be calculated using as input data the right matrix and the corresponding row of the left matrix. The partition policy requires to distribute to all the slaves the right matrix and to the *j-th* slave n_j rows of the left matrix whose indices correspond to those of the result matrix rows which the slave has to compute.

An intense experimentation on a real test-bed, moreover, was conducted on the scientific application called *On-line Power System Security analysis* (OPSSA) which regards the security analysis of electrical networks useful to control system operation when power system operators increase the infrastructure exploitation to maximize their profits [24].

The OPSSA application deals with the on-line assessment of the electrical system's capacity to maintain the flow of electricity from generators to customers, under unforeseen phenomena, called *contingencies*, that could compromise the correct system behaviour. A contingency can be, for example, an unexpected modification in the power system structure or a sudden change of the operational conditions.

The OPSSA works on real-time distributed data measurements, used to get the most recent estimation of the system state variables. Moreover, since such application requires stringent time constraints, in order to output data to be usefully leveraged by the system operators, it is particularly apt for testing the behavior of the proposed mapping heuristic.

The OPSSA consists of three main steps:

1. *the screening phase*: screening of the most "credible" contingencies, that are the most probable and the most dangerous ones;
2. *the predicting phase*: the prediction of the impact of each credible contingency on the entire system operation through the system behavior simulation, called *Power Flow problem*, in both static and dynamic regime (it is the most compute-intensive task of the OPSSA);
3. *the controlling phase*: the identification of preventive and corrective proper control actions able to reduce the risk of system malfunctioning.

The master-slave solution is based on the distribution to the slaves of the set of contingencies, each of which can be analysed independently from each other (see figure 4).

In particular the master has the following main tasks:

- to collect static and dynamic data of the electric network necessary to perform the Power Flow problem;
- to opportunely partition the set of credible contingencies;
- to distribute the set of contingencies among the slaves;
- to collect the partial results replied by the slaves;
- for each contingency, to perform the check of constraints and to generate an alarm in the case of constraint violation.

The slaves, instead, have the following main tasks:

- to receive from the master the most recent state estimation of the electrical grid and the sub-set of contingencies to analyze;
- to perform the system simulation, through the Power Flow problem solution, considering each assigned contingency;
- to reply to the master the system behavior evaluation for each assigned contingency.

Experiments on the OPSSA application execution were performed on the IEEE 118-nodes test network. Such network is composed of 118 nodes and the experiments refer to potential 186 contingencies related to the breaking of one of the 186 transmission lines.

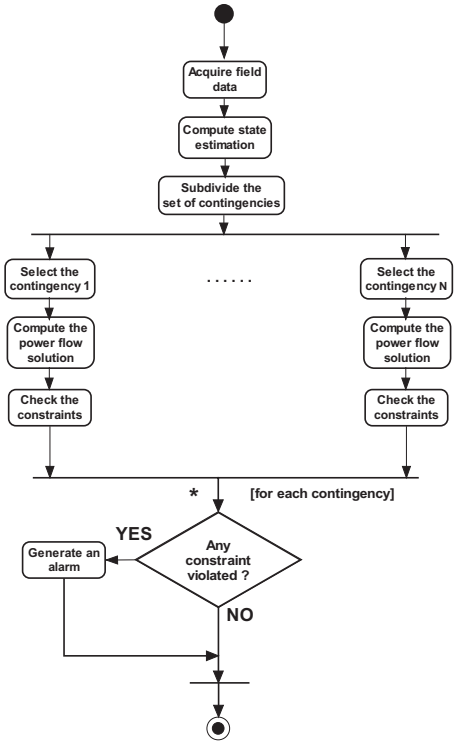

Figure 4. Activity diagram of the OPSSA application.

The execution time of the OPSSA would not exceed few minutes in order to predict in a useful time the impact of each credible contingency on the correct system operation. For this reason in the experimentation we chose as deadlines 15 minutes (900 seconds) and 20 minutes (1200 seconds).

5.3 The test-bed

The OPSSA application was implemented for the TMS Framework and was deployed on a real test-bed with a two-level topology composed of eight homogeneous resources of a cluster, accessed through a front-end, and three directly accessible heterogeneous computers (see figure 5). The configuration of the cluster resources and of the front-end is indicated with R1 and those of the three single computers are indicated with R2, R3 and R4 (see table 1). The client was executed on a notebook with Intel Pentium III 650 MHz, 256 MB of RAM and running Windows 2000 operating system.

The software platform to manage the distributed architecture was implemented by using Java provided by the JDK 1.4.1, HiMM version 1.1 and ProActive version 2.2.

For the OPSSA application, an atomic sub-task (further called task for brevity) is the analysis of a contingency, which strongly depends on the characteristics of the electrical network. For this reason some preliminary experiments were conducted to evaluate the mean execution time for the analysis of a contingency for the 118-nodes test network. The mean execution time was evaluated for each machine, but, because the measured values were very similar for resources with the same configuration, we associated them to the resource configuration.

The estimated execution times of a contingency

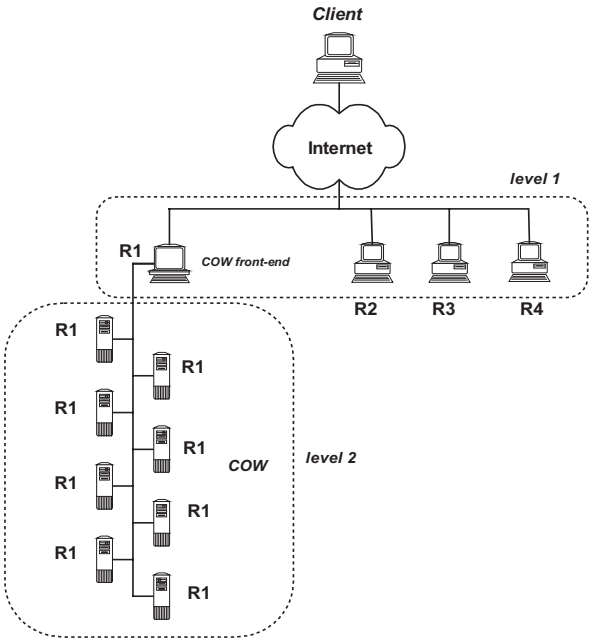

Figure 5. The test-bed.

Resource Configuration	CPU	Clock (MHz)	RAM (MB)	OS
R1	Intel Pentium II (dual processor)	350	128	Windows 2000
R2	Intel Pentium III	350	256	Windows 98
R3	Intel Pentium III	350	256	Windows 2000
R4	Intel Pentium III	500	64	Windows 98

Table 1. Resource configuration.

Resource Configuration	OPSSA Atomic Sub-task Execution Time	OPSSA Atomic Sub-task Cost
R1	76.5 s	1.25 $
R2	75.0 s	4 $
R3	77.0 s	5 $
R4	70.0 s	5 $

Table 2. OPSSA atomic sub-task evaluation.

analysis and related costs, (chosen with a nearly-random criterion) are summarized in table 2.

Because the proposed heuristic requires to characterize a resource only with a mean execution time of the task and the related cost (which are data collected by the resource broker from the Resource Managers), the cluster has to be considered as an aggregate resource, with a performance given by the mean execution time of the nodes divided by the number of nodes. As a consequence the cluster will be characterized by the following mean execution time:

$$t_{COW} = \frac{t_{R1}}{8} = 9.56 \ s$$

Moreover the cost per task of the cluster is modelled as the cost of a cluster node multiplied by the number of nodes, that is 10 $.

5.4 Experimental Results

Table 3 reports the execution times and expended budgets predicted by the time minimization heuristic in relation to a deadline of 900 seconds and a budget of 1200 $.

The estimated execution time for the overall computation is evaluated as the maximum value among the execution times of each resource. As it is possible to observe, using this test-bed and budged, it is not possible to execute 130 tasks within the deadline (the estimated execution time is 910 seconds), that means that it is possible to complete the analysis of only a part of all the possible contingencies.

Figure 6 shows the number of tasks assigned to each resource varying the total amount of tasks to perform. Because of similar capabilities of resources R2, R3 and R4, the number of assigned tasks to each of them is very similar. On the contrary, because the cluster (indicated with COW) has the best performance with respect to the others, it receives a greater amount of tasks, which in particular increases with the total number of tasks to perform.

Figure 7 is related to the same conditions of the previous figure and shows the estimated execution time for each resource. In this scenario the budget is sufficient to permit the assignment of all the tasks minimizing the execution time of all the resources, without requiring adjustments. Even if the proposed algorithm requires an approximation of the calculated optimal real numbers of tasks, in order to assign a whole number of tasks to each resource, this figure shows as the algorithm is able to obtain quite similar execution times for each resource.

The shift around the mean execution time is due to the rounding applied by the heuristic. As much different as the performance of resources are, as more its value is greater. In particular, in this case, the shift is about a minute, which is the difference of execution times among the resource COW and the other resources.

Figure 8 shows the distribution of the contingencies among the resources, under maximum exploitation of deadline and budget, considering 900 seconds as deadline and different budget values.

Number of tasks	30	40	50	60	70	80	90	100	110	120	130
Execution time (s)	231	277	375	450	525	564	631	698	770	841	910
Expended budget ($)	249	337	420	503	586	675	758	841	935	1014	1096

Table 3. Estimated execution times and expended budgets (D=900s, B=1200$).

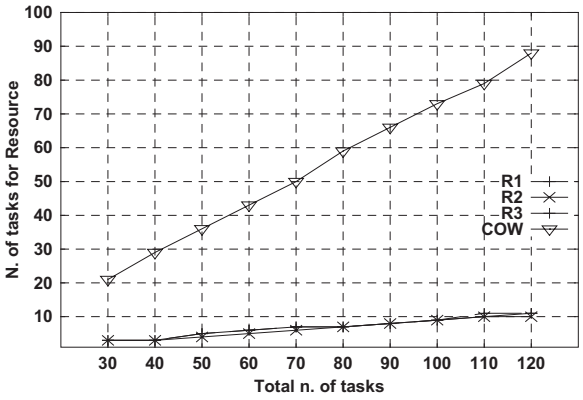

Figure 6. Tasks assigned to each resource varying the total amount of tasks (D=900s, B=1200$).

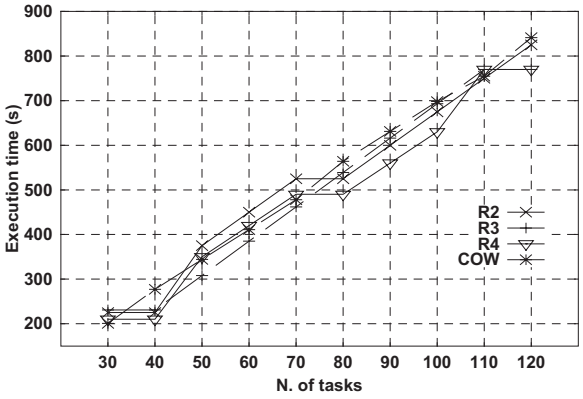

Figure 7. Estimated execution times for each resource varying the total amount of tasks (D=900s, B=1200$).

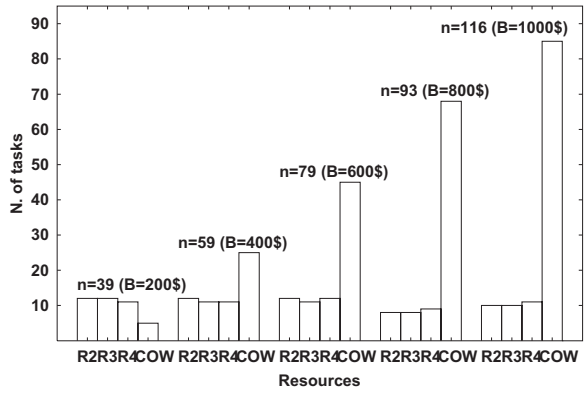

Figure 8. Maximum number of tasks assigned to each resource with different budgets (D=900s).

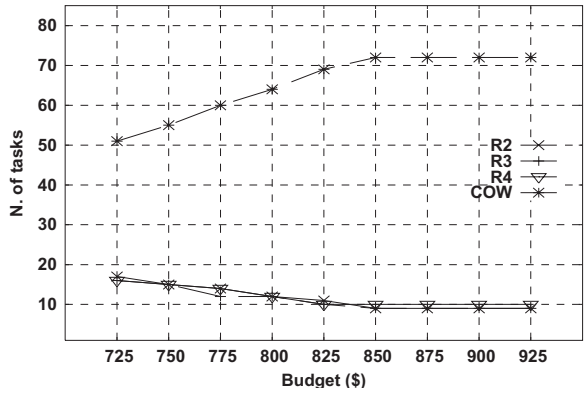

Figure 9. Number of tasks assigned to each resource varying the budget (D=1200s, total number of tasks=100).

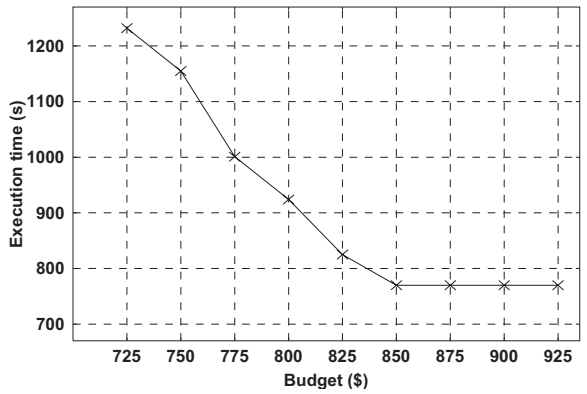

Figure 10. Estimated execution time varying the cost budget (D=1200s, total number of tasks=100).

The figure shows as, by increasing the budget, the more expensive (but even faster) resource, that in this case is the cluster, can be used to map an increasing number of contingencies.

The time minimization heuristic was further tested considering a budget not sufficient to minimize the execution time of all the resources and some adjustments, using the proposed iterative algorithm, were performed in order to exploit more economic ones.

In particular figure 9 shows the number of tasks assigned to each resource for deadline of 1200 seconds, total number of tasks of 100, and a budget varying from 725 $ to 925 $. Until to about a budget of 850 $, it is possible to minimize the execution time until to 707 seconds (see figure 10), which is obtained assigning 72 tasks to the cluster, 9 to R2, 9 to R3 and 10 to R4. For values from 825 $ to 750 $ the limited budget requires to decrease the number of tasks assigned to the most expensive resource, that is the cluster, in particular from 72 until 55 tasks. Finally, with 725 $ no adjustment permits to remain within the deadline (the minimum execution time is 1232 seconds).

Finally figure 11 shows the execution times that were actually measured and the estimated execution times produced by the proposed heuristic, varying the problem computation size (number of tasks) under fixed deadline and budget. Since HiMM does not implement reservation mechanisms, the experiments were conducted considering a single user and avoiding the execution of external jobs, so to not affect execution times and to not introduce a degree of unpredictable variability in performance evaluations.

The execution times have a nearly linear trend with respect to the number of tasks to execute, condition which proves a good scalability of the overall system, which did not show problems of performance reduction increasing the number of tasks to execute.

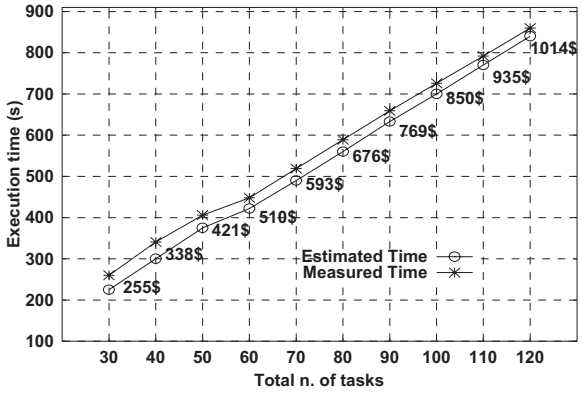

Figure 11. Measured and predicted execution times varying the total number of tasks (D=900s, B=1200$).

The experimental analysis shows, moreover, as the estimated execution times closely match the measured ones, so proving the good level of accuracy of the proposed heuristic.

The little difference between the estimated time and the measured one, that was observed during the experimentation, is mainly due to the communication tasks that, even if the distributed application is executed in a computing environment characterized by a high inter-networking capability, require interval times which are not taken into account by the proposed mapping heuristic.

6. Conclusions

This paper proposed an economy-driven mapping heuristic, the time minimization heuristic, able to map and schedule the tasks assigned to the slaves of a hierarchical master-slave application and that we feel can be usefully adopted for scheduling a large class of parallel and distributed applications.

The paper presented, moreover, an evaluation of the proposed mapping heuristic performed through an experimental analysis conducted in a hierarchical and heterogeneous environment on a real world engineering application, which proved the good level of accuracy of estimated execution times and the scalability of the overall system.

Improvements to the heuristic will be made by including master and communication tasks in the application model, and the inter-networking resource performance and usage cost in the grid model. Moreover, we will implement the heuristic in a framework that aims to simplify design, implementation, deployment and execution of applications whose complexity can be reduced by adopting the "divide and conquer" approach.

References

[1] K. Krauter, R. Buyya, M. Maheswaran. A Taxonomy and Survey of Grid Resource Management Systems for Distributed Computing, *International Journal of Software: Practice and Experience*, *Wiley Press*, USA, 32(2), 2002.

[2] R. Buyya, D. Abramson, J. Giddy, H. Stockinger. Economic Models for Resource Management and Scheduling in Grid Computing. *Special Issue on Grid Computing Environments, The Journal of Concurrency and Computation: Practice and Experience, Wiley Press*, USA, May 2002.

[3] R. Buyya, 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 (CCPE) Journal*, *Wiley Press*, volume 14, issue 13-15, pages 1175-1220, USA, November - December 2002.

[4] H. Casanova. Simgrid: a Toolkit for the Simulation of Application Scheduling. In *IEEE/ACM International Symposium on Cluster Computing and the Grid*, Brisbane, Australia, May 2001.

[5] M. Di Santo, F. Frattolillo, W. Russo, E. Zimeo. A Component-based Approach to Build a Portable and Flexible Middleware for Metacomputing. *Parallel Computing*, *Elsevier*, 28(12) pages 1789-1810, 2002.

[6] N. Ranaldo. A Transparent Framework for Hierarchical Master-Salve Grid Computing based on an Economy-driven Broker. *PhD Thesis*, University of Sannio, Italy, May 2005.

[7] E. Huedo, R. S. Montero, I. M. Llorente. An Experimental Framework for Executing Applications in Dynamic Grid Environments. *ICASE Technical Report*, 2002.

[8] V. Martino, M. Mililotti. Scheduling in a Grid Computing Environment using Genetic Algorithms. In *International Parallel and Distributed Processing Symposium*, 2002.

[9] A. YarKhan, J. Dongarra. Experiments with Scheduling Using Simulated Annealing in a Grid Environment. *Workshop on Grid Computing*, pp. 232-242, Baltimore, USA, November 2002.

[10] M. O. Neary, P. Cappello. Advanced Eager Scheduling for Java-Based Adaptively Parallel Computing. In *2002 Joint ACM-ISCOPE Conference on Java Grande*, 2002.

[11] D. Paranhos, W. Cirne, F. Brasileiro. Trading Cycles for Information: Using Replication to Schedule Bag-of-Tasks Applications on Computational Grids. In *Euro-Par 2003*, August 2003.

[12] W. Cirne, D. Paranhos, L. Costa, E. Santos-Neto, F. Brasileiro, J. Sauve. Running Bag-of-Tasks Applications on Computational Grids: The MyGrid Approach. In *International Conference on Parallel Processing*, pages 407-416, October 2003.

[13] F. Berman et al.. Adaptive Computing on the Grid Using AppLeS. *IEEE Transactions on Parallel and Distributed Systems*, 14(4) pages 369-382, 2003.

[14] R. Buyya, M. Murshed, D. Abramson. A Deadline and Budget Constrained Cost-Time Optimization Algorithm for Scheduling Task Farming Applications on Global Grids. In *International Conference on Parallel and Distributed Processing Techniques and Applications*, Las Vegas, USA, June 2002.

[15] R. Buyya, D. Abramson, J. Giddy. Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid. In *4th International Conference and Exhibition on High Performance Computing in Asia-Pacific Region*, Beijing, China, May 2000.

[16] I. Foster, C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt, A. Roy. A Distributed Resource Management Architecture that Supports Advance Reservations and Co-Allocation. In *International Workshop on Quality of Service*, 1999.

[17] F. Bushmann et al., Pattern-Oriented Software Architecture: A System of Patterns. *J. Wiley and Sons*, 1996.

[18] M. Di Santo, N. Ranaldo, A. Vaccaro, E. Zimeo. Java-based Distributed Architectures for Intensive Computations in Electrical Grids. In *6th International Workshop on Java for Parallel and Distributed Computing*, Santa Fe, New Mexico, April 2004.

[19] D. Caromel, W. Klauser, J. Vayssiere. Towards Seamless Computing and Metacomputing in Java. *Concurrency:*

Practice and Experience, 10(11-13) pages 1043-1061, September-November 1998.

[20] M. Di Santo, F. Frattolillo, N. Ranaldo, W. Russo, E. Zimeo. Programming Metasystems with Active Objects. In *5th International Workshop on Java for Parallel and Distributed Computing*, Nice, France, April 2003.

[21] M. Di Santo, N. Ranaldo, E. Zimeo. A Broker Architecture for Object-Oriented Master/Slave computing in a Hierarchical Grid System. *Parallel Computing*, *Elsevier*, volume 13, pages 609-616, Dresda, Germany, September 2003.

[22] D. Culler et al.. LogP: Towards a Realistic Model of Parallel Computation. In *ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 1-12, 1993.

[23] R. Wolski, T. Spring Neil, J. Hayes. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. *Journal of Future Generation Computing Systems*, October 1999.

[24] M. Di Santo, A. Vaccaro, D. Villacci, E. Zimeo. A Distributed Architecture for on-line Power Systems Security Analysis. *IEEE Transactions on Industrial Electronics*, 51(6), December 2004.

Biographies

Nadia Ranaldo is a Research Assistant at the Department of Engineering, University of Sannio. She received the PhD degree in Computer Science from University of Sannio in Benevento, Italy, in 2005. Her main research interests are resource management systems, frameworks for distributed systems, parallel computing, wireless and sensor networks and grid and service computing.

Eugenio Zimeo is an Assistant Professor of computer science in the School of Engineering and member of the Research Centre on Software Technology (RCOST) at the University of Sannio. His primary research interests are software architectures and frameworks for distributed systems, high performance middleware, agent-oriented programming, mobile and ubiquitous computing, wireless and sensor networks, parallel and distributed systems, and grid and service computing. Zimeo has a PhD in Computer Science from the University of Naples, Italy. He is a member of the IEEE Computer Society.