Available online at www.sciencedirect.com

ScienceDirect

Computer Communications 30 (2007) 499–515

computer communications

www.elsevier.com/locate/comcom

Adjusted fair scheduling and non-linear workload prediction for QoS guarantees in grid computing

Nikolaos Doulamis [a,*], Anastasios Doulamis [a], Antonios Litke [a], Athanasios Panagakis [a], Theodora Varvarigou [a], Emmanuel Varvarigos [b]

[a] *National Technical University of Athens, Department of Electrical and Computer Engineering, 9 Heroon Polytechniou str., Zografou 157 73, Greece*
[b] *Department of Computer Engineering and Informatics, University of Patras, Computer Networks Lab, 26500 Patras, Greece*

Available online 9 January 2006

Abstract

In this paper, we propose an efficient non-linear task workload prediction mechanism incorporated with a fair scheduling algorithm for task allocation and resource management in Grid computing. Workload prediction is accomplished in a Grid middleware approach using a non-linear model expressed as a series of finite known functional components using concepts of functional analysis. The coefficient of functional components are obtained using a training set of appropriate samples, the pairs of which are estimated based on a *runtime estimation model* relied on a least squares approximation scheme. The advantages of the proposed non-linear task workload prediction scheme is that (i) it is not constrained by analysis of source code (*analytical methods*), which is practically impossible to be implemented in complicated real-life applications or (ii) it does not exploit the variations of the workload statistics as the *statistical approaches* does. The predicted task workload is then exploited by a novel scheduling algorithm, enabling a fair Quality of Service oriented resource management so that some tasks are not favored against others. The algorithm is based on estimating the *adjusted fair completion times* of the tasks for task order selection and on an earliest completion time strategy for the grid resource assignment. Experimental results and comparisons with traditional scheduling approaches as implemented in the framework of European Union funded research projects GRIA and GRIDLAB grid infrastructures have revealed the outperformance of the proposed method.
© 2005 Elsevier B.V. All rights reserved.

Keywords: Grid-enabled networking middleware; Workload prediction; Non-linear workload modeling; Fair grid scheduling; QoS requirements

1. Introduction

Today we experience an increasingly need for collaboration, data sharing, and other modes of interaction that involve multiple heterogeneous and geographically distributed resources, including supercomputers, PC's, PDA's, workstations, storage systems, databases, and special purpose applications with various requirements (CPU, I/O, or memory) [1]. For this reason, new abstractions and concepts should be introduced at *network architecture* and *middleware* level to allow applications to access and share resources or services in an efficient manner. *Computational Grid* is a new computing paradigm which supports the sharing, interconnection, and use of diverse resources in dynamic computing systems to deliver services under a transparent, integrated, and distributed computing infrastructure [1,2].

It is envisioned that Next Generation Networks (NGN) will be enhanced with features that would advance the networks operators' perspectives for a richer set of capabilities regarding the management, *Quality of Service* (QoS) control and exploitation of their resources. This will allow the provision of attractive stringent services reaching the highest user expectations. Consequently, this trend aims to design a network architecture that will be flexible, scalable, robust, and optimized, and will be able to support

* Corresponding author.
E-mail addresses: ndoulam@cs.ntua.gr (N. Doulamis), dora@telecom.ntua.gr (T. Varvarigou).

0140-3664/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.comcom.2005.11.013

also appropriate interfaces to the emerging Grid environments in an efficient, integrated, and transparent manner. Up to now, the Grid application requirements are implemented solely in the *Grid middleware* while the network plays the role of a transport layer for enabling communication between two endpoints. Grid infrastructures can provide services in many areas and in a way that will utilize features and services of the network middleware layer. Thus, the intersection point of the Next Generation Networks and Grids is expected to be located in the middleware since it will establish the communication between these two, unrelated so far, layers.

The ability to provide an agreed upon QoS is important for the success of the Grid, since, without it, users may be reluctant to pay for services or contribute resources to Grids. There are generally two types of QoS attributes: the *quantitative* (such as network latency, CPU performance, and storage capacity), and the *qualitative* (i.e., service reliability and user satisfaction) for a Grid infrastructure. The qualitative characteristics are important but it is difficult to measure them objectively. In Grids the common QoS aspects include exclusive access to resources, number of CPUs utilized, reliability, ease of access, fairness, etc. However, the most important attribute is the satisfaction of a task's *deadline* set by the user, since it is the criterion of satisfying business contracts with hard time limits [3]. To provide, however, guarantees of the negotiated QoS requirements, the Grid infrastructure should be enhanced with appropriate and efficient scheduling and resource management mechanisms, specifying how Grid middleware can provide resource coordination for client applications transparently. This need has been confirmed by the Global Grid Forum (GGF) in the special working group dealing with the area of scheduling and resource management for Grid computing [4].

In this paper, we present an approach for workload prediction based on extracting appropriate descriptors (features) able to characterize a task's computational complexity. Therefore, our scheme is valid for *specific applications* or for a *class of applications* of similar properties, such as the 3D image rendering algorithms or the finite elements structuring methods which have been studied under the framework of the GRIA project. Task workload prediction is performed using a generalized non-linear model between the extracted descriptor and the respective task computational complexity. The use of simple linear or non-linear models of specific type – exponential, logarithmic – is not adequate to provide a sufficient estimate of the workload of real-life applications since there is no simple linear relationship, which associates the application descriptors (extracted for representing the computational complexity) to the actual task computational load. Although the use of linear approaches can accurately predict the task workload for some specific applications [5], these methods cannot be applied for the applications considered in our work since they involve among others parameters of pre-measured executions on idle machines. For this reason, we use the non-linear relationship among the descriptors extracted to describe the computational complexity of the task and the respective task workload. Using concepts derived from functional analysis, the generalized non-linear model is expanded as a series of known finite functional components [6,7]. Then, the coefficients of each component are estimated based on a training set of representative samples, so that an efficient approximate of the non-linear relationship between the extracted descriptors and the respective task workload is obtained.

For the scheduler implementation, a fair policy is adopted, using a Max–Min fair sharing algorithm [8]. The scheduling scheme consists of two different phases; the first refers to the order that the tasks are assigned for execution (*task queuing order phase*), while the second determines the processor on which the tasks are scheduled (*processor assignment phase*). In the proposed fair scheduling algorithm, which is called Adjusted Fair Task Order (AFTO), the task queuing order is selected with respect to the task *adjusted fair completion times*. The most appropriate processor that the tasks are assigned for execution is selected using the Earliest Completion Time (ECT) strategy, modified so that processor capacity gaps are taken into account.

This rest of the paper is organized as follows: Section 2 provides an overview of related works for the various topics that are addressed in the scope of this paper. Section 3 presents the problem formulation and the notation of the paper. Section 4 discusses the non-linear model used for task workload prediction, the method used for training the model, as well as the way of constructing the training set. In this section, we also present how the task workload is estimated from the actually measured task runtime. The Grid fair scheduling algorithms adopted in the proposed Grid architecture are reported in Section 5, while Section 6 illustrates the experimental results. Finally, Section 7 concludes the paper.

2. Related work

Several toolkits and Grid middleware approaches have been designed and developed addressing the need to reflect and consider the QoS requirements of users that submit their tasks in a Grid infrastructure. The most well known toolkit for Grid computing is Globus [9]. Globus addresses a wide range of metacomputing issues including heterogeneous environments and includes software services and libraries for resource monitoring, discovery, and management. Development, implementation, and evaluation of mechanisms that support high throughput computing (HTC) on a large collections of distributed computing resources is also addressed in the framework of Condor project [10]. The Grid version of Condor, called Condor-G, uses Globus toolkit to manage Grid jobs [11]. While Condor has been designed to run jobs within a single administrative domain Globus toolkit has been designed to run jobs across many administrative domains. Condor-G combines the strengths of both. Condor-G introduces grid scheduler and manager to allow full-featured queuing services, credential manager, and fault-tolerance issues. The Grid Application Development Software (GrADS) tool aims at simplifying distributed heterogeneous computing in the same way that the World Wide Web simplifies information sharing over the Internet [12]. In the application-level scheduling is performed by minimizing the application execution time of a set of potentially shared resources. Instead, in the meta-scheduling level, many applications are considered at once to improve the overall system performance.

Apart from the aforementioned Grid infrastructures, several works have been reported in the literature dealing with scheduling and resource allocation management in Grid computing. The main goal of these approaches is to enhance the Grid infrastructure with mechanisms able to guarantee the negotiated QoS parameters. In [12] extension of the scheduling algorithm of GrADS tool is discussed by (i) introducing more sophisticating clustering and data mining schemes, (ii) reducing the computational complexity, and (iii) providing a single-site scheduling in case of invalidation of multi-site recourse selection. The scheduling objective of [13] is to minimize the total completion time of the tasks. Since minimization of task completion time in a multiprocessor scheduling environment is a NP-hard problem, scheduling heuristics are discussed and compared with each other in this work. Genetic algorithms methods are presented in [14,15] for minimizing the total task completion time. The algorithms model the scheduling process as a *genetic evolution* and estimate at which Grid resource a task should be assigned for execution so that the completion time is minimized. The scheme is evaluated with heuristic scheduling strategies. Evaluation of different scheduling mechanisms for Grid computing is also presented in [16], such as the First Come First Serve (FCFS), the Largest Time First (LTF), the Largest Cost First (LCF), the Largest Job First (LJF), the Largest Machine First (LMF), the Smallest Machine First (SMF), and Minimum Effective Execution Time (MEET). A very interesting work on Grid scheduling implementation steps is presented in [17].

Task *workload prediction* is performed using either a *statistical* or an *analytical* model. In the statistical approach, the application is analyzed and the application parameters that affect the execution time are determined. In each resource, a number of scenarios using different parameter values are executed by the application and the results are classified into groups which are depended on the input parameters, and the technology and configuration of the resources [18]. When a scenario of the specific application is submitted for execution, the system classifies the task according to the value range of the input application specific parameters. According to this classification and the particular resource, the system returns an estimate of the task runtime. The main difficulty of these approaches is that they statistically describe the computational complexity of the submitted tasks not the complexity of an individual task requesting for service.

The analytical method for runtime estimation can be applied with much better results than the statistical one, especially in heterogeneous computer systems, but it is required to have analysis of the application code. The basic procedures that affect the execution time of the application are chosen and the execution time is measured for each different platform. Since the execution time of individual processes is known in a given platform, then the estimation of the total execution time can be directly calculated for the specific application scenario [19,20]. However, the main drawback of the analytical approaches is that they refer to a *known source code*.

A drawback of the previously mentioned approaches is that (i) scheduling is performed without taken into account *fair considerations* and (ii) they use simple workload prediction algorithms. For example, the works of [13–15] try to minimize the total completion time by dropping over-demanded tasks (e.g., tasks of high workload and short deadlines), which is not a fair policy. In addition, statistical estimation of the task workload may result in a significant deviation of the actual task workload to the predicted one. On the other hand, the analytical approaches assume a known source code and they cannot provide satisfactory results for complex real-life applications, where the generated source code is difficulty analyzed. These two aspects are confronted in this paper by introducing (a) a fair sharing scheduling policy and (b) a highly non-linear model for workload prediction using concepts derived form functional analysis.

Fair scheduling schemes have been extensively studied in the networking area, especially for scheduling packet flows over Internet routers. Towards this direction, the Generalized Processor Sharing scheme (GPS) has been proposed for fair scheduling over packet switched networks [21]. The GPS algorithm can provably provide guarantees on the delay and bandwidth of a session in a network of switches, but cannot be implemented in practice. For this reason, it is emulated by the Weighted Fair Queuing (WFQ) algorithm [22], which exploits concepts of the Max–Min Fair sharing scheme [8].

3. Problem formulation and notation

3.1. Resource parameters

Let us assume that a Grid infrastructure consists of M *resources*. The performance of a Grid resource is expressed with the *resource parameter vector* $\mathbf{r}_j$, $j = 1, 2, \ldots, M$

$$\mathbf{r}_j = [r_{j,1}, r_{j,2}, \ldots, r_{j,m}]^{\mathrm{T}} \tag{1}$$

the elements of which $r_{j,k}$, $k = 1, 2, \ldots, m$, indicate independent internal processes of the jth resource that affect its performance. Usually, three parameters ($m = 3$) are sufficient for characterizing the performance of a Grid resource; *the CPU speed*, *the average memory I/O* and the *average disk I/O rate*. These parameters usually refer as computational *parameters* of the *j*th Grid resource [23]. Extension to additional resource parameters, such as the communication (e.g., the *Send* and the *Receive Communication Bandwidth* both measured in Kbytes/s) and availability parameters (e.g., the *Minimum Free Memory* (in MB) and the *Minimum Free Disk Space* (in KB)), can straightforwardly be included in the following analysis.

For the evaluation of the CPU speed, the Million Floating Point Operations per second (MFLOPS/s) are used since it is a performance measure independent of the CPU architecture, thus allowing comparisons among different CPUs [23]. On the other hand, the memory performance is evaluated through the average rates of MB/s requiring for read and/or write on the resource memory [24,25]. Similarly, the hard disk performance is measured as the average read/write disk Input/Output (I/O) bandwidth (defined as the average I/O Kbytes/s) [24].

The resource vector $\mathbf{r}_j$ is estimated through a benchmarking process that can be applied in the same way for every heterogeneous resource platform comprising the Grid, and provides thus a way to compare the performance of heterogeneous resources. By using the same parameters for every heterogeneous resource platform we can incorporate different platforms on the system and we can define a cost of use per performance unit.

The benchmarking process is repeated at constant short time intervals, say τ, to handle the cases of time-varying resource parameters that are dynamically change through time. In this case, the resource parameter vector is expressed with a time variable τ, as $\mathbf{r}_j(\tau)$, to indicate the time varying behavior of the resource parameter vector. In the following analysis, we assume a given time τ to simplify the notation of $\mathbf{r}_j(\tau)$ as $\mathbf{r}_j$.

3.2. Workload parameters

Let us now assume a task, say T_i, allocated for execution in a Grid infrastructure and let us denote as $\mathbf{x}_i$ a vector which describes the workload of the task T_i. Vector $\mathbf{x}_i$ is independent from the resource on which the task T_i is assigned for execution. It only depends on the application type that this task belongs to and the respective task complexity. In other words and using physics terminology, the parameters $\mathbf{x}_i$ express the "work" that task T_i requires for. Therefore, the $\mathbf{x}_i$ is independent from the capacity of a machine that is used for producing the "work" $\mathbf{x}_i$.

The workload parameters, expressed by vector $\mathbf{x}_i$, are defined in association with the resource parameters of Eq. (1). Since in this paper we use three computational parameters for evaluating the performance of a Grid resource, $\mathbf{x}_i$ is a 3-element vector expressed as

$$\mathbf{x}_i = [x_{i,1}, x_{i,2}, x_{i,3}]^{\mathrm{T}}. \tag{2}$$

The elements $x_{i,k}, k = 1,2,3$ correspond to the CPU instructions per task (in Million Floating Point Operations-MFLOP), the average memory I/O amount per task (in MB) and the average disk I/O amount (in KB) per task T_i similarly to the elements of the resource vector $\mathbf{r}_j$.

3.3. Task workload description parameters

Vector $\mathbf{x}_i$ is estimated by a workload predictor module using appropriate *descriptors* extracted from the task T_i. Let us denote as $\mathbf{s}_i$ the vector containing the *descriptor parameters* of task T_i:

$$\mathbf{s}_i = [s_{i,1}, s_{i,2}, \ldots, s_{i,n}]^{\mathrm{T}}. \tag{3}$$

The elements of $s_{i,j}$ $j = 1, \ldots, n$ of (3) describe particular features of the task that affect its computational load. Variable n refers to the number of the descriptors used. For example, in case we refer to tasks derived from 3D rendering applications, the descriptors $s_{i,j}$ could be the image resolution, number of polygons, number of light sources and so on, as it is described in [26]. It is clear that tasks of different applications require different descriptors to characterize their workload [26]. The proposed scheme is applied in cases where we have tasks that incorporate descriptors, which means tasks comprising of input files and user specified parameters which influence the computational workload of the task itself when it is to be executed. Although it is not required to know the source code of the application, an analysis, for engineering purposes, is mandatory for the input files and parameters whenever a new application is going to be integrated in the Grid. This approach is more suitable than the case of having known source code, since it preserves on one hand the confidentiality of a company's source code, and on the other hand it can be based only on specific patterns and specifications of input files while using the user's input in a transparent way.

3.4. Task runtime

Let us now denote as $t_{i,j}^{(\mathrm{run})}$ the runtime of task T_i on the *j*th Grid resource. Let us also assume, for simplicity, that there is no overlap between the various internal processes (i.e., the CPU, the memory, and the hard disk) that affect the task execution. A similar approach of independent computational parameters is also adopted in [23]. Then, the total execution time of a task equals the sum of individual execution times of each of the specific internal process. Since in our case, we have taken into account three different processes for evaluating the performance of a Grid resource, i.e., the CPU, the memory and the hard disk speed, the total runtime for executing task T_i on the *j*th resource is given by

$$t_{i,j}^{(\mathrm{run})} = \sum_{p=1}^{m} t_{i,j}^{(p)}, \tag{4}$$

where $t_{i,j}^{(p)}, p = 1,2,3$ refers to the individual execution times of the three different internal processes of the task.

The individual execution times $t_{i,j}^{(p)}$ are estimated through the resource parameters $r_{j,p}$ of the j resource along with the respective workload of the task $x_{i,p}$. In particular, we have that

$$t_{i,j}^{(P)} = \frac{x_{i,p}}{r_{j,p}}. \qquad (5)$$

A block diagram of the above mentioned architecture comprising the Grid middleware approach is presented in Fig. 1. As is observed, the architecture consists of the five modules. The T*ask Load Description*, which is responsible for estimating the descriptor parameters $\mathbf{s}_i$, the *Grid Resource Description*, which is used for benchmarking the Grid resources to obtain their capacities, the *Run-Time Estimation* responsible for estimating the task run time on a resource, the *Workload Prediction* used for forecasting task workload using the descriptor vector $\mathbf{s}_i$ and the *Grid Scheduler*, which is used for selecting the task order and the respective processor that a task is assigned for execution.

4. Task workload prediction

To estimate the task workload $\mathbf{x}_i$, a predictor is required, able to forecast the task actual load based on a set of several descriptors, appropriate extracted to represent the task computational complexity. This approach assumes that the *type of the application* is known instead of the *task code itself* as it happens in the analytical schemes. Let us recall that for each task T_i several descriptors are extracted to represent the task computational complexity and included in a feature vector, say $\mathbf{s}_i$. An example of the way that the descriptors are extracted in case of 3D image rendering tasks is presented in Section 6. The exact type and number of the elements of vector $\mathbf{s}_i$ depends on the application that the tasks derive from. In the most general case, the extracted descriptors are related to the actual task workload through a non-linear relationship as

$$\mathbf{x}_i = [x_{i,1}x_{i,2}x_{i,3}]^{\mathrm{T}} = [g_1(\mathbf{s}_i)g_2(\mathbf{s}_i)g_3(\mathbf{s}_i)]^{\mathrm{T}} = \mathbf{g}(\mathbf{s}_i), \qquad (6)$$

where $\mathbf{g}(\cdot) = [g_1(\cdot)\ g_2(\cdot)\ g_3(\cdot)]^{\mathrm{T}}$ is a vector-valued function, the elements of which $g_i(\cdot)$, $i = 1, 2, 3$ express the non-linear relationships between the input descriptors and the respective three computational parameters that assemble the task workload [see Eq. (2)].

The main difficulty in implementing Eq. (6) is that $\mathbf{g}(\cdot)$ is actually unknown. Using concepts derived from functional analysis, we can model the unknown function $\mathbf{g}(\cdot)$ as a parametric relationship of finite known functional components, within any degree of accuracy [6], that is

$$g_m(\mathbf{s}_i) \approx \sum_{l=1}^{L} v_l^{(m)} \cdot \Phi_l^{(m)}\left(\sum_{k=1}^{P} w_{k,l}^{(m)}(s_{i,k})\right), \quad \forall m = 1, 2, 3, \qquad (7)$$

where $v_l^{(m)}$ and $w_{k,l}^{(m)}$ refer to the model parameters of the unknown functional elements, $g_m(\cdot)$, $m = 1, 2, 3$, while the $\Phi_l^m(\cdot)$ to the functional components. Eq. (7) indicates that the functions $g_m(\cdot)$ is expanded as a linear relation of the non-linear functional components $\Phi_l^{(m)}(\cdot)$, whose inputs are linear relations of descriptor parameters $\mathbf{s}_i$. In Eq. (7), variable L expresses the approximation order of functions $g_m(\cdot)$. It is clear that the approximation precision increases, as the order L increases, at the expense of an increase in the number of parameters [6].

The most familiar class of functional components $\Phi_l^{(m)}(\cdot)$ is the sigmoid functions, which are equal to

$$\Phi_l^{(m)}(x) = 1/(1 - \exp(-a \cdot x)), \qquad (8)$$

where a is a constant which regulates the curve steepness. Thus, $P \times L$ parameters are required to approximate functions $g_m(\cdot)$.

4.1. Training set construction

Estimation of the model parameters $v_l^{(m)}$ and $w_{k,l}^{(m)}$ can be obtained using a training set of representative samples. In particular, let us denote as S a set, which contains selected representative samples that are used to model the non-linear input-output relationship $\mathbf{g}(\cdot)$. The set S has the form of

$$S = \{\ldots, (\mathbf{s}_i, \mathbf{x}_i), \ldots\}, \qquad (9)$$

where the pairs $(\mathbf{s}_i, \mathbf{x}_i)$ contain the description of a task complexity along with the respective workload for the ith task. To provide a robust estimation of the model

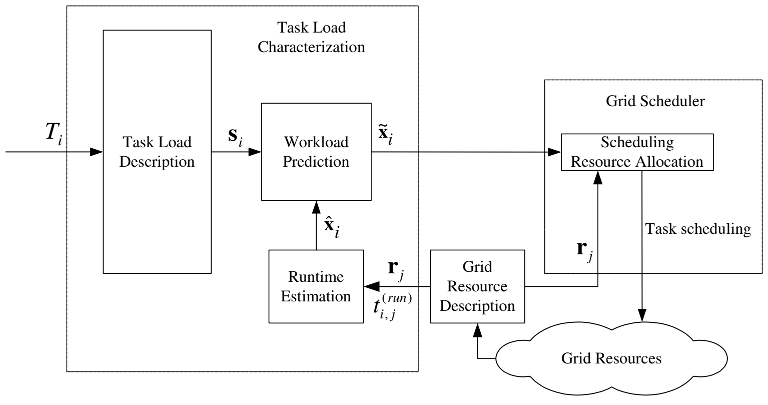


Fig. 1. The proposed middleware approach for enhancing Grid enabled architectures.

parameters $v_l^{(m)}$ and $w_{k,l}^{(m)}$, several representative samples are selected and included in the training set S.

The main difficulty in constructing the training set S, i.e., the samples $(\mathbf{s}_i, \mathbf{x}_i)$, is the reliable estimation of the actual task workload as expressed by the vector $\mathbf{x}_i$. This is due to the fact that what we can measure is the actual task execution time on a particular resource, i.e., $t_{i,j}^{(\mathrm{run})}$, along with the specific resource capacity, $\mathbf{r}_j$ instead of vector $\mathbf{x}_i$. Thus, vector $\mathbf{x}_i$ can be estimated using the resource vectors $\mathbf{r}_j$ and the task runtime $t_{i,j}^{(\mathrm{run})}$ on several resources as described in following.

4.2. Workload estimation

Using Eqs. (4) and (5), we can express the total execution time $t_{i,j}^{(\mathrm{run})}$ of the task T_i on the jth resource as

$$t_{i,j}^{(\mathrm{run})} = \sum_{p=1}^{3} \frac{x_{i,p}}{r_{j,p}} = \mathbf{x}_i./\mathbf{r}_j, \tag{10}$$

where the symbol ./ is the division of the elements of vector $\mathbf{x}_i = [x_{i,1}\ x_{i,2}\ x_{i,3}]^{\mathrm{T}}$ with the respective elements of the vector $\mathbf{r}_j = [r_{j,1}\ r_{j,2}\ r_{j,3}]^{\mathrm{T}}$ (i.e., element by element division).

To provide a reliable estimation of vector $\mathbf{x}_i$, we execute a given task T_i over all M available Grid resources and calculate the respective task runtimes, i.e., the $t_{i,j}^{(\mathrm{run})}$, for all $j = 1, 2, \ldots, M$. Since, vector $\mathbf{x}_i$ depends only on the task T_i and not on the resource on which the task is assigned for execution, the optimal estimate of vector $\mathbf{x}_i$, say $\hat{\mathbf{x}}_i$, is given by minimizing the following error (least squares)

$$\hat{\mathbf{x}}_i = \arg\min_{\mathbf{x}_i \in R^3} E \tag{11}$$

$$\text{with } E = \sum_{j=1}^{M} (\mathbf{x}_i - t_{i,j}^{(\mathrm{run})} \cdot \mathbf{r}_j)^{\mathrm{T}} \cdot (\mathbf{x}_i - t_{i,j}^{(\mathrm{run})} \cdot \mathbf{r}_j).$$

The optimal estimate $\hat{\mathbf{x}}_i$ which minimizes Eq. (11) is given by differentiating Eq. (11) with respect to vector $\mathbf{x}_i$ and then setting the derivatives equal to zero, i.e., $\frac{\mathrm{d}E}{\mathrm{d}\mathbf{x}_i} = 0$. In this case, the optimal estimate $\hat{\mathbf{x}}_i$ is given as

$$\hat{\mathbf{x}}_i = \frac{1}{M} \cdot \sum_{j=1}^{M} t_{i,j}^{(\mathrm{run})} \cdot \mathbf{r}_j. \tag{12a}$$

From (12a), we can measure the estimated task runtime as

$$\hat{t}_{i,j}^{(\mathrm{run})} = \hat{\mathbf{x}}_i./\mathbf{r}_j. \tag{12b}$$

Having estimated in an optimal way the task workload, we can construct the samples of the training set used to approximate the parametric weights (i.e., the coefficients) of Eq. (7). Thus, the actual set S is constructed by pairs of the form $(\mathbf{s}_i, \hat{\mathbf{x}}_i)$, where the estimated task workload $\hat{\mathbf{x}}_i$ has been optimally obtained using Eq. (12a).

4.3. Non-linear model training

Using the training set S, the parameters $v_l^{(m)}$ and $w_{k,l}^{(m)}$ of Eq. (7) are estimated. In our case, a second order method has been used, for training based on the Marquardt–Levenberg algorithm [27]. This method has been selected due to its efficiency and fast convergence, since it was designed to approach second order training speed without having to compute the Hessian matrix. Other types of training algorithms can be adopted for estimating the parameters $v_l^{(m)}$ and $w_{k,l}^{(m)}$, such as the ones presented in [28].

To increase the generalization performance of the workload predictor, i.e., its prediction performance for data outside the training set, the cross validation method is adopted during the training. According to this technique, the training data are divided into two subsets; the one used for training and the one used for evaluating the training performance (validation set). The error on the validation set is monitored during the training phase. Normally, this error decreases as the number of training iterations increases, as the error in the training set does. However, when data overfitting is noticed, the error on the validation set begins to increase and the training phase is terminated. This point is called *early stopping* and defines the most appropriate time instance that the training phase should be completed [29].

4.4. Task runtime prediction

The aforementioned non-linear model is used to predict the workload (and thus the runtime) of a task that is requesting for service to a processor. Then, the predicted information is used by the Grid scheduler so as to assign the tasks to the available processors.

For this reason, initially description values are extracted from a task T_i and included in the vector $\mathbf{s}_i$. Then, using the model presented in Eq. (7), (the model parameters have been already optimally estimated using the samples of the training set S and the training algorithm), a prediction of the task workload is obtained as

$$\tilde{\mathbf{x}}_i = \sum_{l=1}^{L} v_l^{(m)} \cdot \Phi_l^{(m)} \left(\sum_{k=1}^{P} w_{k,l}^{(m)} (s_{i,k}) \right), \tag{13}$$

where as $\tilde{\mathbf{x}}_i$, we denote the predicted workload of the task T_i.

Then, the respective task runtime is provided by the following equation:

$$\tilde{t}_{i,j}^{(\mathrm{run})} = \tilde{\mathbf{x}}_i./\mathbf{r}_j, \tag{14}$$

where $\tilde{t}_{i,j}^{(\mathrm{run})}$ corresponds to the predicted runtime for the task T_i on the jth Grid resource.

5. Fair grid scheduling algorithms

Let us assume that at a given time, N tasks are to be scheduled with a predicted workload $\tilde{\mathbf{x}}_i$, $i = 1, 2, \ldots, N$. Let us also assume that each task is characterized by a deadline D_i, which is the desirable time that the task should complete its execution. Let us also denote as $\delta_{i,j}$ the earliest time at which it is feasible for the task T_i to start execution on processor j. The $\delta_{i,j}$ is calculated from the communication delays between the user and processor j (e.g., the time that elapses between a decision is made by the resource manager to assign task T_i to processor j, and the arrival of all files necessary to run task T_i to processor j) and from the time that processor j is ready for executing the task T_i (e.g., the processor may execute other tasks). Then, the *demanded computation rate* X_i of the task T_i is defined as follows:

$$X_i = \frac{\|\tilde{\mathbf{x}}_i\|}{D_i - \delta_i}, \tag{15}$$

where δ_i is the weighted average of $\delta_{i,j}$ over all processors

$$\delta_i = \frac{\sum_{j=1}^{M} \delta_{ij} \cdot \|\mathbf{r}_j\|}{\sum_{j=1}^{M} \|\mathbf{r}_j\|}. \tag{16}$$

In the following analysis, we assume that the tasks are *non-preemptable* and *non-interruptible*, i.e., when they start execution on a machine they run continuously on that machine until completion. We also assume that time-sharing is not available and a task served on a processor occupies 100% of the processor capacity. The tasks do not have any inter-dependencies and so they can not block the execution of each other. For the purpose of our scheduling algorithms two different phases are considered; In the first phase, we determine the order in which the tasks are assigned for execution (the "queuing order" phase) while in the second phase, we determine the processor on which each task is scheduled (the "processor assignment" phase).

5.1. Task queuing order selection

The most widely used algorithm for task order selection is the Earliest Deadline First (EDF) method, also known as the deadline driven rule [30]. This method dictates that at any point the system must assign the highest priority to the task with the most imminent deadline. The concept behind the EDF scheme is that it is preferable to serve first the most urgent tasks (i.e., the task with the earliest deadline) and then serve the remaining tasks according to their urgency. However, this queuing order selection algorithm does not make any attempt to handle the tasks requesting for service in *a fair way*. For example, tasks with relative urgency may be favored against the remaining tasks, regardless of the respective workload. In addition, using an EDF scheduling scheme, there is no motivation for a user to specify a honest deadline, since tasks of late deadlines are given low priority. To overcome the aforementioned difficulties, an alternative approach is presented in this section, by handling tasks requesting for service with respect to *their fair completion times*.

5.1.1. Fair rates estimation

Based on the demanded rates of the tasks, the task fair rates are estimated using the Max–Min fair sharing method [8]. Intuitively, in the non-weighted max–min fair sharing scheme, all users are given an equal share of the total resources, unless some of them do not need their whole share, in which case unused resources are divided equally among the remaining "bigger" users in a recursive way. The fair sharing is accomplished by the total offered processor capacity as measured by

$$C = \sum_{j=1}^{M} \|\mathbf{r}_j\|, \tag{17}$$

where we recall that $\mathbf{r}_j$ is the resource vector of the jth Grid resource.

In the weighted modification, a weight, say w_i, is assigned for each task, which defines the priority of the ith task. More specifically, the fair computational task rates, say f_i, are estimated using the following equations:

$$f_i(n) = \begin{cases} X_i & \text{if } X_i < w_i \cdot \sum_{k=0}^{n} O(k), \\ w_i \cdot \sum_{k=0}^{n} O(k) & \text{if } X_i \geqslant w_i \cdot \sum_{k=0}^{n} O(k), \end{cases} \quad n \geqslant 0 \tag{18}$$

where

$$O(n) = \frac{C - \sum_{i=1}^{N} f_i(n-1)}{\tilde{N}(n)}, \quad n \geqslant 1 \tag{19}$$

and

$$O(0) = C/\tilde{N}, \quad \text{with } \tilde{N} = \sum_{i=1}^{N} w_i, \tag{20}$$

where $f_i(n)$ is the fair rates of the ith task at the nth iteration of the algorithm, while $\tilde{N}(n)$ is the sum of the weights of the tasks whose assigned fair rates are smaller than their demanded computation rates at the beginning of the nth iteration of the algorithm, that is:

$$\tilde{N}(n) = \sum_i w_i : \text{ for all } i : X_i > f_i(n-1) \text{ and } \tilde{N}(0) = \tilde{N}. \tag{21}$$

The process is terminated at an iteration n_o at which either $O(n_o) = 0$ or *card* $\{\tilde{N}(n_o)\} = 0$. Therefore, the estimated task fair rates are given by

$$f_i = f_i(\eta_0). \tag{22}$$

Example: To clarify the way of estimating the fair rates of the tasks, a simple example is presented in the following. In particular, let us assume four tasks of demanded rates 10, 8, 5, and 15, respectively. The weights of the first and third tasks equal 1, while the weights of the second and fourth tasks equal 2. The overall processor capacity in this example is assumed to be 30. Then, initially the total processor capacity is divided by 6 (i.e., the sum of the task weights) and thus a share unit equal to 5 (30/6) is assigned. For the first and second tasks, one share unit (30/6) is assigned (since their weights are 1) while for the second and four two share units (60/6) are assigned (since their weights are 2). In the initial assignment, the second and third task are totally satisfied with a remaining processor rate of 2, which is equally shared to the first and fourth tasks whose the demanded rates are not satisfied at the initial stage of the algorithm. In particular, the remaining capacity (i.e., the 2) is divided by 3 (the sum of the weights of first and fourth tasks) and a share unit, (equals 2/3) is calculated, which is assigned to the tasks (1 share unit to the first task and 2 share units to the fourth task). Table 1 presents the estimated fair rates of the tasks along with the respective demanded rates and the weights.

5.1.2. Adjusted fair rates estimation

When certain tasks become inactive (e.g., because they complete execution), more capacity becomes available to be shared among the active tasks, and therefore the fair rates of the active tasks should increase. In addition, when new tasks become active (e.g., because of new arrivals), the fair rates of existing tasks should decrease. In other words, the fair computational rate of a task is not really a constant f_i, as assumed before, but it is actually a function of time, which increases when tasks complete execution, and decreases when new tasks arrive, resulting in the *adjusted fair rates*.

The adjusted fair rates are estimated by re-calculating the task rates each time a task become active/inactive. This is performed using the max-min fair sharing scheme *only for the active tasks*. Thus, the rate of the *i*th task is a function of time, i.e., $f_i^a(t)$. We call this rate, *adjusted fair rate*. This approach results in a scheduling scheme that is fairer than the other approaches.

5.1.3. Fair completion times estimation

Let us denote as t_i^a the adjusted fair completion times, which are estimated from the adjusted fair task rates. The adjusted fair completion times are estimated with the following algorithm.

Initially, we assume that the rates $f_i^a(t)$ of all tasks have been normalized so that the minimum task rate equals 1. Then, we introduce a variable called the *round number*, which defines the number of rounds of service that has been completed (e.g., the workload) at a give time. A non-integer round number represents a partial round of service. The round number depends on the rates of the active tasks at a given time. In particular, the round number increases with a rate equal to the sum of the rates over all the active tasks, i.e., with a slope equal to $1/\sum_i f_i^a(t)$. Thus, the rate that the round number increases, changes each time the state of the active tasks changes, i.e., when a new arrival or a task completion takes place. Therefore, each time the number of active tasks change, a recalculation of the round number has to be performed.

Based on the round number, we define the *finish number* $F_i(t)$ of task T_i at time t as

$$F_i(t) = R(\tau) + \tilde{x}_i/f_i^a(t), \tag{23}$$

where τ is the last time a change in the number of active tasks occurred (and therefore the last time that the round number was re-calculated), and $R(\tau)$ is the round number at time τ. $F_i(t)$ should be re calculated each time new arrivals or task completions take place. Note that $F_i(t)$ is *not* the time that task T_i will complete its execution. It is only a service tag that we will use to select the order in which the tasks will be assigned to processors. The adjusted fair completion times t_i^a can be computed as the time at which the round number reaches the estimated finish number of the respective task. Thus,

$$t_i^a : R(t_i^a) = F_i(t_i^a). \tag{24}$$

The task adjusted fair completion times determine the order in which the tasks are considered for execution to the Grid resources: the task with the earliest adjusted fair completion time is assigned first, followed by the second earliest, and so on.

Example: Let us concentrate on the four tasks of Table 1 and let us assume that the task workload are 10, 8, 5, and 20, respectively. The initial task fair rates are the ones presented in Table 1 (i.e., 5.66, 8.5, and 10.66). At time $t = 1$, the tasks 2 and 3 complete their execution. Thus, they become inactive. For this reason, a new fair allocation of the total processor capacity takes place for the tasks, 1 and 4. In particular, the total processor capacity of 30 units is shared into 3 parts (one for the task 1 of weight 1 and 2 for the task 4 of weight 2). Thus, the adjusted fair rates of the tasks 1 and 4 are estimated which equal to the demanded rates (10 and 15, respectively). At time $t = 1.43$, the task 1 finishes its execution since the task workload equals 10. Finally, at $t = 1.62$ the task 4 finishes its execution. The adjusted fair rates of the four tasks versus time are presented in Fig. 2 along with their respective adjusted completion times. Therefore, the task queuing order is selected as, task 3 first, following by task 2, following by task 1 and then by task 4.

5.2. Processor assignment selection

The processor at which a given task is assigned for execution is estimated through the Earliest Completion Time (ECT) scheme as described in the following.

Table 1
An example of the Max–Min fair sharing algorithm if the overall processor capacity is 30

Demanded rates	Workload	Weights	First weighted sharing rates	Fair rates	Adjusted completion times	Task order
10	10	1	5	5.66	1.43	3
8	8	2	10	8	1	2
5	5	1	5	5	1	1
15	20	2	10	10.66	1.62	4

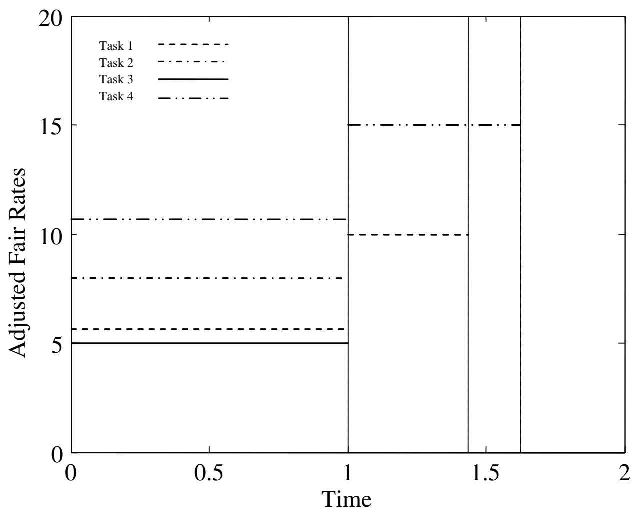


Fig. 2. The adjusted fair for the tasks of Table 1.

Let us assume that $s_{i,j}$ is the time that task T_i starts its execution on processor j. Then, the estimated completion time of the ith task is given as $s_{i,j} + \tilde{t}^{(\mathrm{run})}_{i,j}$, where we recall that $\tilde{t}^{(\mathrm{run})}_{i,j}$ is the predicted runtime for the task T_i on the jth Grid resource. The runtime $\tilde{t}^{(\mathrm{run})}_{i,j}$ is calculated through the Eq. (14). Then, the ECT method selects, among the M available processors, the one that yields the minimum estimated completion time, i.e., minimizes the following quantity,

$$\hat{j} = \arg\min_{j\in\{1,\ldots,M\}}\{s_{i,j} + \tilde{t}^{(\mathrm{run})}_{i,j}\}. \tag{25}$$

The starting time $s_{i,j}$ has to be larger than the ready time $\delta_{i,j}$ of the task T_i on processor j (i.e., $s_{i,j} \geqslant \delta_{i,j}$), which in turn depends on the ready time at which the respective processor becomes available for executing the task and the communication delay of the task.

One approach is to estimate $s_{i,j}$ as the *processor release time*, that is, the time at which all tasks that have already been scheduled on this processor finish their execution. Fig. 3 illustrates a scheduling scenario in which (a) the *task queuing order* is selected using the EDF algorithm (b) the *processor assignment* is selected using the ECT approach and (c) the task starting time $s_{i,j}$ is chosen as the processor release time.

In this figure, we assume that all tasks T_i, $i = 1,2,\ldots,6$ request service at time $t = 0$, the processors have equal computational capacity, and both processors are initially idle. We also assume that $D_1 < D_2 < \cdots < D_6$ and $\delta_{i1} = \delta_{i2}$. Initially, the task T_1 of the earliest deadline D_1 is assigned for execution, and processor 1 is chosen (randomly in this case, since there is tie). Then, task T_2 is assigned for execution on processor 2, since this yields the earliest completion time. In a similar way, we assign the remaining tasks.

Setting the task starting time $s_{i,j}$ as the processor release time makes $s_{i,j}$ easy to record and independent of the task that is about to be scheduled, but it has the drawback that gaps in the utilization of a processor may be created (see

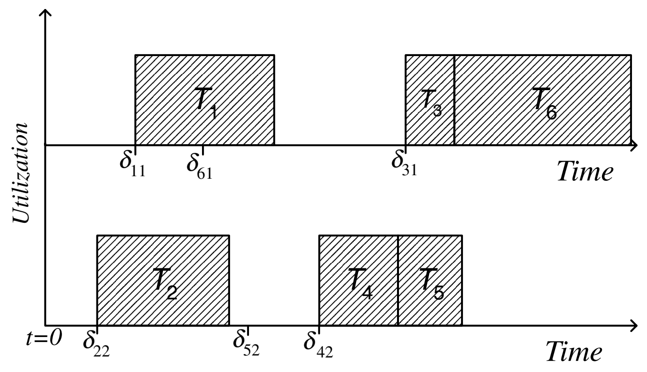


Fig. 3. An example of the Earliest Completion Time (ECT) algorithm for processor selection in case that the task starting time is estimated as the processor released time.

the gap between the tasks T_1 and T_3 in Fig. 3), resulting in a waste of the processor capacity and thus in a deterioration of the scheduling performance. To overcome this problem, an alternative approach is adopted, which estimates the processor available time more precisely. Particularly, the capacity gaps are examined and in case that a selected task can be served within a capacity gap it is assigned to this time interval. Among all candidates time intervals the one, which provides the earliest completion time, is selected. Fig. 4 presents how the scheduling of Fig. 3 is improved by exploiting the capacity gaps. In this case, the completion time of tasks T_5 and T_6 is shorter than that obtained in Fig. 3. In particular, the capacity of processor 2 is better exploited, since the capacity gaps are significantly reduced. Also, although the capacity gap of processor 1 is not reduced, its processing availability increases, leaving more capacity for scheduling future tasks.

6. Experimental results

6.1. Grid infrastructure

The aforementioned described architecture has been implemented in the framework of the GRIA ("Grid

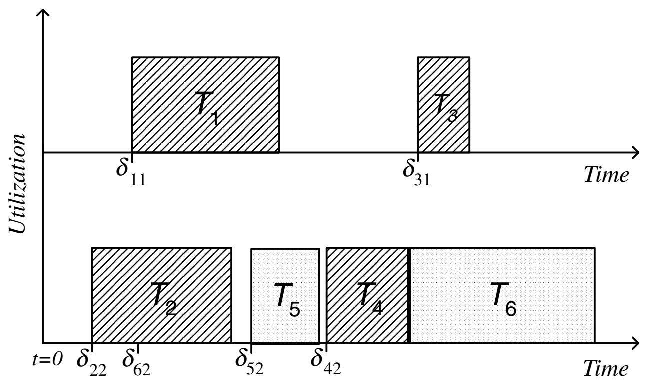


Fig. 4. An example of the ECT algorithm for processor selection by exploiting processor capacity gaps.

Resources for Industrial Applications") and GRIDLAB ("A Grid Application Toolkit and Testbed") European Union funded projects [31,32]. Fig. 5 presents the architecture implemented in the framework of these projects. The architecture is discriminated into two main parts; the client side architecture (Fig. 5(a)) and the server side architecture (Fig. 5(b)). The main module of the client side is the *task load characterization*, the sub-modules of which are presented in Fig. 1. On the other hand, the Grid scheduler constitutes the heart of the server side architecture.

The main parts of the adopted Grid architecture at the client side are summarized as follows.

6.1.1. Grid application

This module provides an interface required for interacting the user with the Grid infrastructure. The interface is designed to control a collection of Grid services for the user desktop, i.e., the deadlines of the submitted tasks, the task priori-ties and so on.

6.1.2. Workflow enactor

This is an intermediate module with interacts with all modules at the client side. The workflow enactor is responsible for activating each time an appropriate module at the client side.

6.1.3. Task load characterization

This module is responsible for modeling and predicting the task workload characteristics. This information is then provided to the architecture of the server side along with the associated task deadlines so that an appropriate scheduling scheme is accomplished. The main sub-modules of task load characterization are presented in Fig. 1 (the task load description, workload prediction, and runtime estimation).

6.1.4. Grid access authorization

The authorization module checks whether the user is authorized to access the Grid resources and on which terms.

6.1.5. Grid service proxy

This module instantiated by the workflow enactor to handle invocation of remote Grid servers, either in the application or in the negotiation steps.

On the contrary, the main parts of the Grid infrastructure at the server side are the following.

6.1.6. Grid scheduler

The scheduler is the heart of the server architecture and determines when (task queuing order) and at which processor (processor assignment) the submitted tasks should be executed so that the demanded QoS parameters are satisfied as much as possible. The scheduler uses information obtained by the task load characterization module and the current resource availability.

6.1.7. Negotiation service

In case that the demanded QoS parameters of the submitted tasks can not be satisfied (i.e., the task deadlines are violated), the negotiation service is activated to inform the users for the violation and ask them whether they are willing to submit the task with the supported by the Grid infrastructure QoS parameters.

6.1.8. Resource manager

This module is responsible for sending the submitted tasks for execution in the Grid resources.

The workload predictor module has been integrated in the client side of the architecture and has been directly interfaced with the client GUI. The user of the application, after initiating an authorization process based on Public Key Infrastructure (PKI) technology, selects the files that comprise the job and that will be submitted for execution. The job will be assigned on a single processor for execution, based on the service profile specified by the user. The service profile refers to the specification of the quality of the execution which reflects the deadline of the results submission and the cost that will be allocated for the given job. While the job is executed, the user can suspend the session and resume it later on. When the job has terminated its execution, the results will be prompted to the user as a link

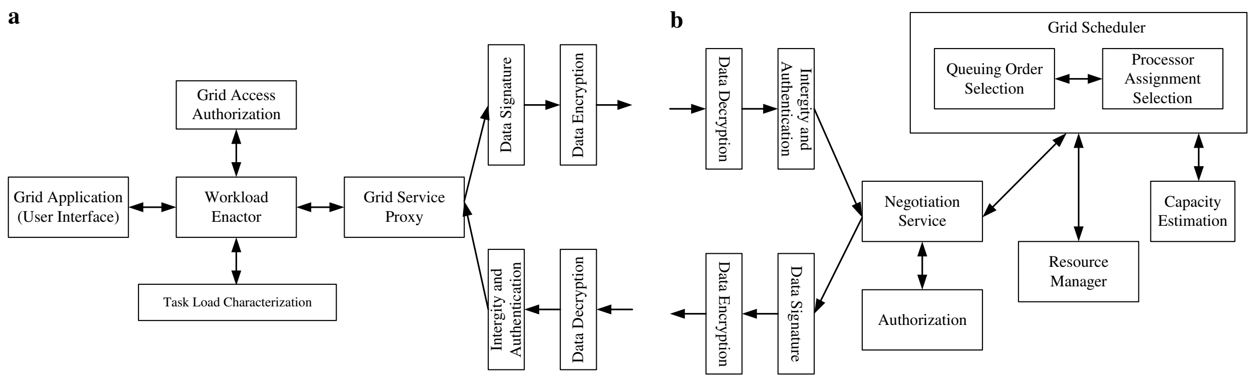


Fig. 5. The adopted Grid infrastructure. (a) The client side. (b) The server side.

where he can download them from, making use of his X.509 certificate and username/password he has been assigned explicitly for this purpose. The communication between the various endpoints is made under the framework of Web services through the Simple Object Access Protocol (SOAP). The Application Programming Interface (API) for the respective functionality of workload prediction and estimation of the resources capacity and the runtime of a given task on a selected resource is presented in Table 2.

6.2. Grid resource description

In order to estimate the task workload $\mathbf{x}_i$, the resource vector $\mathbf{r}_j$ should be available. Vector $\mathbf{r}_j$ is estimated through a benchmarking process, which is applied to any resource of the Grid. The benchmarking process is conducted to be platform and application independent.

In our experiments, three different C++ programs were developed and integrated in the server side part for providing a reliable estimation of the resource vector. The measured quantities of the resource vector reflect the performance capability of a Grid resource. As *CPU speed measurement*, we use the MFLOPs which are the amount of floating point division operations that the CPU can process per second. As *I/O memory bandwidth* measurements we use the average of input and output MB/s ratios with 50 packets of 1024 KB each, for write and read processes, respectively. Finally, as *I/O disk storage bandwidth* measurement, we take the average of input and output KB/s ratios with 50 packets of 512 KB each, for write and read processes, respectively. The design of the three C++ programs was performed so that they can be compiled for both Linux and Microsoft Windows platforms, enabling the incorporation of heterogeneous operational systems in the proposed Grid architecture.

In the benchmarking method, adopted in this study, agent technology has been included. In particular, an agent is assigned for each resource in the Grid which triggers the benchmarking procedure on a regular basis by executing a script containing the three C++ classes. Then, the outcome comprising the three resource parameters values is stored in an ASCII file of pre-determined format. This file is parsed by a Java class deployed as Web Service on the server side and the values as passed to the scheduler for the estimation of the runtime in each resource, respectively.

Table 3 presents the benchmarking results for 10 selected representative resources of the proposed Grid infrastructure.

Table 3
Resource parameters values for the 10 representative resources comprising the examined Grid infrastructure

Grid resource id	Resource parameters		
	MFLOPs ($r_{j,1}$)	Mbytes/s ($r_{j,2}$)	Kbytes/s ($r_{j,3}$)
1	941	445	10780
2	2103	1827	23119
3	400	116	5286
4	1982	1014	21443
5	2341	1974	27964
6	806	920	13135
7	466	122	9251
8	987	457	15463
9	627	339	11268
10	1612	468	9984

6.3. Workload prediction results

In order to evaluate the performance of the proposed workload prediction model, tasks derived from a real-life commercial application are considered. In our experiments the tasks stem from 3D image rendering, which is an interesting commercial application with a wide impact in many fields ranging from simulation and design to education, entertainment and advertisement. 3D image rendering refers as a process of creating computer generated photo-realistic imaging of complex scenes from 3D synthetic geometrical primitives [33]. A fundamental difficulty in achieving total visual realism of synthetic images is the complexity of the real world (such as reflections, refractions, surface texture, shadows, light conditions, and object irregularities) which results in high demands of processing power and memory size. For this reason, 3D rendering can be solved more feasibly and in a reasonably time and cost using a Grid infrastructure. In our case, the 3D image rendering tasks have been developed using the Blue Moon Rendering Tool (BMRT) version 2.6, which is a set of rendering programs and libraries adhering to the RenderMan(TM) standard as set forth by Pixar.

To implement, however, the workload prediction model, a training set of representative 3D rendering tasks should be constructed. The set contains pairs of appropriate features extracted from each task to describe the computational complexity along with the respective task workload. The task workload is estimated through the runtime estimation model as described in the following section. In addition, in Section 6.3.2 we present the features extracted for each task to describe the task computational complexity.

Table 2
The API in Java for the workload prediction module and the resource capacity and runtime estimators

```
double[] inputFileParse (String filepath, double[] inparams)
double[] restoreWorkloadPredictor(String workpredID, double[] inputArray)
double[] capacityEstimator()
double runtimeEstimator (double[] workLoad , double[] resourceCapacity)
```

6.3.1. Runtime estimation model results

Initially, the grid resources are divided into two independent parts. The *training resource set*, which is responsible for the estimation of the task workload and the *evaluation resource set*, which is used for evaluating the task workload estimation performance.

A set of different 3D image rendering tasks has been constructed and assigned for execution in all resources of the training resource set. Then, the actual runtime $t_{i,j}^{(\mathrm{run})}$, for all the examined tasks and for all training Grid resources is estimated. Using the actual runtime $t_{i,j}^{(\mathrm{run})}$ and the resource vectors r_j, as they are obtained through the benchmarking process, the estimates $\hat{x}_i$ of all tasks are computed [see Eq. (12a)].

To test the estimation accuracy of the task workload, the Grid resources of the evaluation set are used. More specifically, the examined 3D rendering tasks are assigned for execution in all resources of the evaluation set and then the actual runtime $t_{i,j}^{(\mathrm{run})}$ of these tasks is measured. The actual runtime of the tasks is compared with the estimated one, $\hat{t}_{i,j}^{(\mathrm{run})}$ as obtained using Eq. (12b), i.e., the estimated task workload and the respective Grid resource vector.

Fig. 6 illustrates the workload estimation accuracy. In particular, the solid line of Fig. 6 refers to the actual runtime (i.e., the $t_{i,j}^{(\mathrm{run})}$) while the dotted line refers to the estimated one (i.e., the $\hat{t}_{i,j}^{(\mathrm{run})}$) as have been measured on two Grid resources of the evaluation set. The results have been derived from 32 representative 3D rendering tasks among 200 examined ones. In Fig. 6, the *y*-axis is presented in logarithmic form. As can be seen, the runtime estimation model has produced runtime values very close to the actual ones for all the examined tasks, regardless of the actual time required for their execution. This means that the model behaves well both for tasks of low and high computational complexity and thus it provides an efficient estimation of the task workload. The square error defined as

$$E_r = \frac{\left| t_{i,j}^{(\mathrm{run})} - \hat{t}_{i,j}^{(\mathrm{run})} \right|}{t_{i,j}^{(\mathrm{run})}} \tag{26}$$

over the two examined Grid resources of the evaluation set are shown in Fig. 7. The two selected resources are the first two of Table 3, and have been selected so as to show different potentialities in terms of MFLOPS/s, disk I/O bandwidth and memory I/O bandwidth. What we can see from these figures is that the square error for the workload prediction of the chosen tasks, can be different when executing the same tasks on different resource, but the nonlinear prediction module once trained, provides accurate results. It should be also mentioned that in this figure, the error has been plotted versus the task runtime and not the task number. As can be seen, in the worst case the error is lower than 14%, while the majority of the runtime estimation error ranges between 4% and 12%, with an average square error of about 7.5%. Fig. 8 presents the Q–Q (Quantiles–Quantiles) plots of the actual and the predicted task runtimes over the two examined Grid resources. In this figure, the solid line corresponds to perfect fit. The advantage of the Q–Q plots is that they show all prediction differences with the same accuracy, regardless of the value of actual task runtime. It can be seen that the plotted data are close to the line of perfect fit, meaning that the proposed runtime model is accurate.

6.3.2. Feature extraction results

In our case, the features used to describe the computational complexity of a 3D rendering task are provided by parsing a RenderMan Interface Bytestream (RIB) formatted file, which provides a general structure for describing a synthetic world. RIB format includes information about the object geometric primitives (such as cylinder, cone, and sphere), object transformation, object material and texture, number of light sources, rendering algorithm

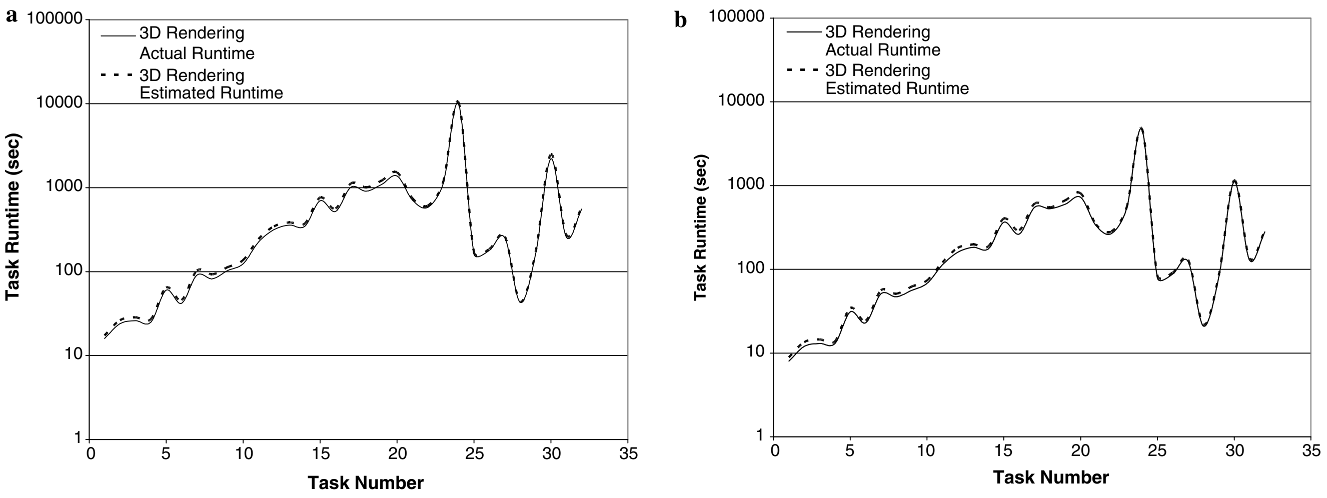


Fig. 6. The actual ($t_{i,j}^{(\mathrm{run})}$) and the estimated ($\hat{t}_{i,j}^{(\mathrm{run})}$) runtimes for 32 3D image rendering tasks into two Grid resources as obtained using the least square method of Section 3.2. (a) The first resource. (b) The second resource.

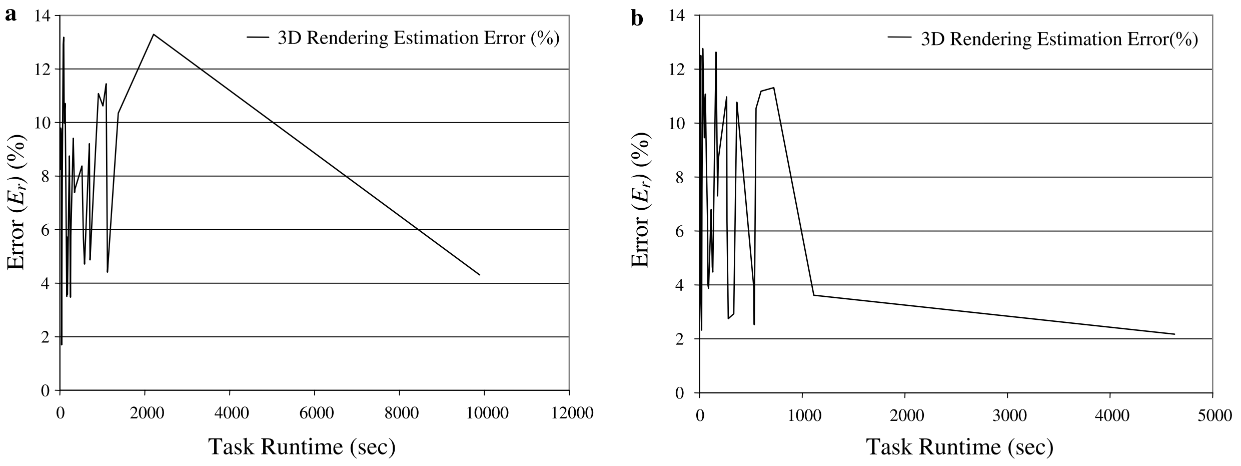


Fig. 7. The error E_r between the actual and the estimated runtime for the 32 3D rendering tasks of Fig. 6. (a) For first resource, (b) for the second resource.

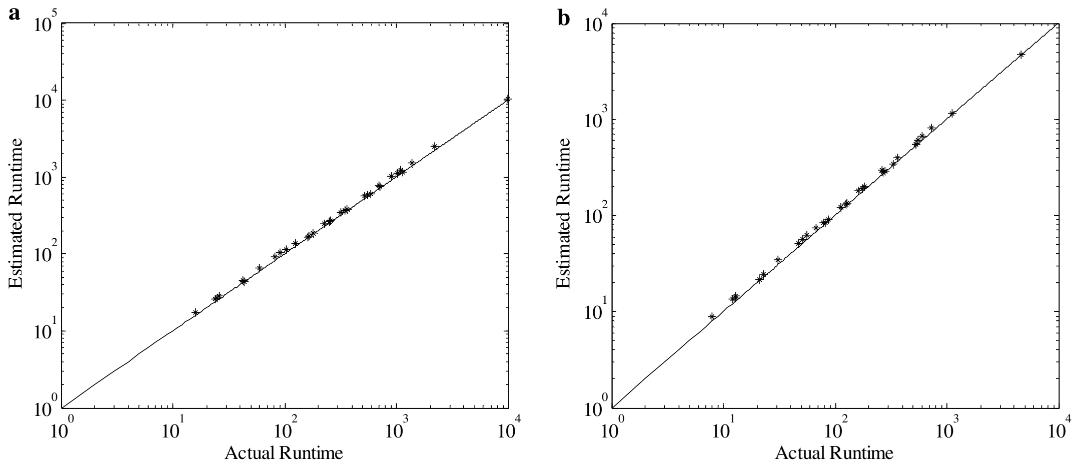


Fig. 8. The Q–Q plots of the actual and the estimated task runtime. (a) For first resource, (b) for the second resource.

parameters and in general any detail used for creating the rendered images.

A RIB file example is presented in Table 4, in which the synthetic world consists of a cylinder. The cylinder surface is "shiny", characterized by diffuse reflection coefficient of 0.2. The statement "WorldBegin" defines the begin of the synthetic world, while the statement "WorldEnd" the end of it. In this particular example, the ray tracing algorithm has been used for 3D rendering with maximum level of ray-tree to equal four (4) as indicated by the command line "option "render" integer max raylevel [4]". Perspective projection is adopted, while image resolution is of 200 × 150 pixels as results from "format" statement. Table 5 presents the extracted descriptors used to characterize the computational complexity of a 3D image rendering task in case that the Ray Tracing algorithm is used.

Table 4
An example of a RIB file format

```
Projection "perspective" "fov" 40
Format 200 150 1
Option "render" "integer max raylevel" [4]
WorldBegin
  Cylinder 1 -1 1 360
  Surface "shiny" "Kd" [0.2]
WorldEnd
```

An important issue, which affects the rendering workload, is the algorithm used to render a synthetic scene. Several rendering algorithms have been proposed in the literature, each affecting the computational load with a different way. Among the most popular and most commonly used is the Ray Tracing algorithm, which is adopted in the following. For each type of algorithm, different descriptors are extracted and then used for the workload prediction. A Java class has been designed and developed to parse the RIB file that comprises the task and to extract the values of the respective descriptor parameters so as to pass them in the Workload Predictor for further processing.

6.3.3. Workload prediction performance

To train the non-linear model, which is used to predict the task workload, we initially create a measurement set consisting of 200 pairs of rendering descriptors along with the respective workload. The measurement set is randomly

Table 5
The extracted descriptors s_i used to characterize the workload of 3D rendering Ray tracing algorithms

Descriptor	RIB statement
Ray tracing descriptors	
Maximum number of recursive rays	Option "render" "integer max raylevel" [4]
Minimum Shadow distance	Option "render" "float minshadowbias" [0.01]
Surface shadow property	Attribute "render" "string casts shadows" ["Os"]
Other factors	
Image resolution (width-height-pixel aspect ratio)	Format 200 150 1
Displacement type	Attribute "render" "integer truedisplacement" [0]
Bounding box coordinates	Attribute "displacementbound" "string coordinatesystem" ["current"] "float sphere" [0]
Patch multiplier	Attribute "render" "float patch multiplier" [1.0]
Minimum/Maximum level of subdivision	Attribute "render" "float patch maxlevel" [256] Attribute "render" "float patch minlevel" [1]
Surface complexity	For example: PatchMesh "bicubic" 13 "nonperiodic" 10 "nonperiodic" "P"
Surface material type	Surface "shiny" "Kd" [.1] "Kr" [0.5] "Ka" [0.1]
Light source type	For example: LightSource "pointlight" 1 "from" [0100]

The "[ ]" refers to the default value of parameter.

partitioned into three disjoint sets; the training set, the validation set and the test set. The training and the validation set is used to estimate the model parameters using the least squares Marquardt–Levenberg algorithm [27]. The test set is responsible for evaluating the performance of the non-linear model. The 60% of data of the measurement set comprises the training set, while the rest 40% are equally divided for the validation and the test set.

Fig. 9 illustrates the relative squared error of the actual workload (as provided using the method of Section 4.2) and the predicted one (as provided by the non-linear model), for the 50 samples of the test set. The error is expressed as

$$E_p = \frac{\|\hat{\mathbf{x}} - \tilde{\mathbf{x}}\|}{\|\hat{\mathbf{x}}\|} * 100\%, \tag{27}$$

where $\|\cdot\|$ denotes the L_2-norm. As can be seen, the error is less than 20% in the worst case, with an average value being equal to 14%. These error values point out the significance and validity of the non-linear workload prediction model that has been designed and proposed in this paper, since it gives a well-bound prediction of a rendering task workload.

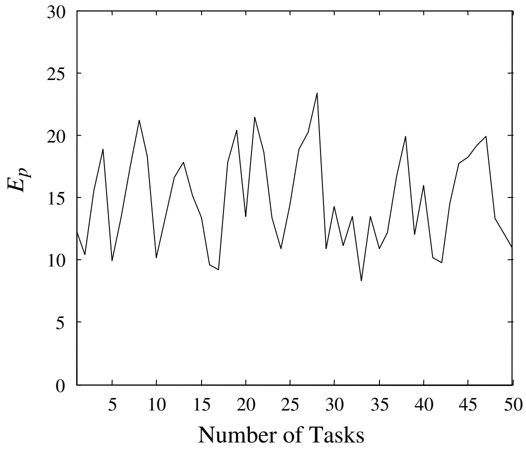


Fig. 9. The prediction error as obtained from the proposed non-linear model for 50 3D rendering tasks outside the training set.

6.4. Scheduling

For the scheduling results, initially we define the *normalized load* of the Grid infrastructure as follows

$$\rho = \frac{\sum_i X_i}{C}, \tag{28}$$

where we recall that X_i is the demanded task rate and C the total processor capacity.

Fig. 10(a) presents the scheduling efficiency as obtained using three different scheduling algorithms; the proposed fair scheduling scheme (AFTOS), the Earliest Deadline First (EDF), and the First Come First Serve (FCFS) policy. The results have been obtained by submitting 40 real file 3D rending tasks to the Grid infrastructure presented in Section 6.1. For each 3D rendering task, the workload prediction module is activated to provide an approximate of the actual task complexity, which is then used for the scheduling policies.

In this figure the scheduling efficiency is measured as the relative error between the demanded task rates and the actual schedulable rates,

$$E_X = \sum_i \frac{|X_i - X_i^c|}{X_i}, \tag{29}$$

where X_i^c is the actual rate of the ith task. The actual task rates equals $X_i^c = \{X_i, 0\}$ for the FCFS and EDF scheduling scheme (depending whether the task are assigned for execution or not) and $X_i^c = r_i$ for the AFTOS policy. In this figure the error E_1 has been plotted versus the normalized load ρ. As is observed, the proposed Grid fair scheduling algorithm outperforms the traditional scheduling policies for all values of normalized load (smaller values of error E_1). The fluctuation in the values of error E_1 are due to the fact that the results have been obtained using real life 3D rendering tasks.

Fig. 10(b) presents the scheduling efficiency as measured by the average relative deviation of the demanded task deadlines to the actual task completion times,

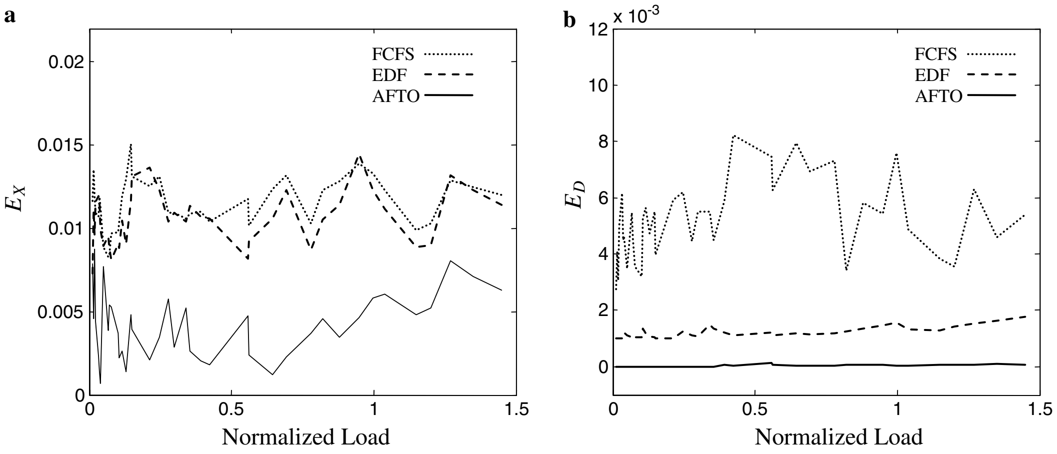


Fig. 10. The evaluation performance of three scheduling schemes, the First Come First Serve (FCFS), the Earliest Deadline First (EDF) and the Adjusted Fair Task Order (AFTO). (a) The error E_X and (b) the error E_D.

$$E_D = \frac{1}{N}\sum_i \frac{\left|D_i - \max(D_i^c, D_i)\right|}{D_i}, \tag{30}$$

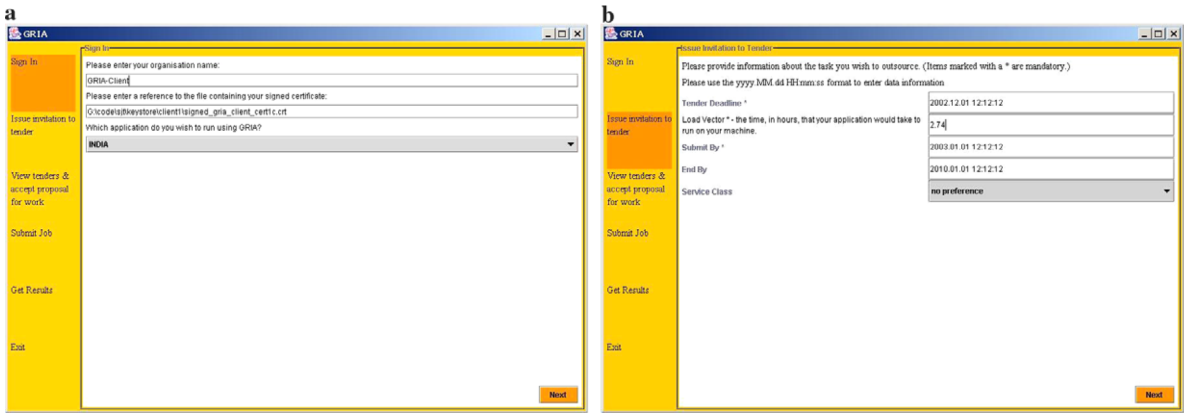
where D_i is the requested deadline and D_i^c is the actual completion time of the ith task. Note that tasks whose actual completion time are smaller than their respective deadlines do not contribute to E_D. Again, we observe that the proposed fair Grid scheduling algorithm yields much better efficiency compared to the FCFS and EDF policies.

Fig. 11 present an example of the Graphical User Interface (GUI) initially implemented in the framework of the GRIA project. In particular, Fig. 11(a) presents the initial GUI, where the user selects the type of application (i.e., rendering, finite element method), while Fig. 11(b) presents the GUI where the task information are defined by the user (i.e., task deadline or the task workload as predicted by the load characterization/prediction module, location of the input files, selection of services class, etc.).

7. Conclusions and future work

In this paper, we present a fair Grid scheduling scheme incorporated with an efficient non-linear task workload prediction mechanism for enhancing the capabilities of Grid computing in satisfying the negotiated QoS requirements set by the users. The task workload prediction is achieved by modeling the task workload through a non-linear function, the parameters of which are estimated using concepts derived from functional analysis. More specifically, initially appropriate descriptors are extracted from each task to describe its computational complexity. Then, the non-linear relationship of the extracted descriptors and the task workload is modeled as a finite number of known functional components. The contribution of each functional component to the non-linear relation (i.e., the model coefficients) is estimated by a second order training algorithm, such as the Marquardt–Levenberg approach using a training set of appropriate samples.

The pairs of the training set are the task load descriptors and the respective task workload. However, the desired task workload cannot be measured, in contrast to the task runtime on a particular resource. For this reason an optimal estimate of the task workload is accomplished by applying a least squares algorithm, which exploits information of the actual task run times on several resources and simultaneously takes into account the resource capacity.


a
b
GRIA
Sign In
Issue invitation to tender
View tenders & accept proposal for work
Submit Job
Get Results
Exit
Next

Sign In

Please enter your organisation name:

GRIA-Client

Please enter a reference to the file containing your signed certificate:

G:\code\sjt\keystore\client1\signed_gria_client_cert1c.crt

Which application do you wish to run using GRIA?

INDIA

Issue Invitation to Tender

Please provide information about the task you wish to outsource. (Items marked with a * are mandatory.)

Please use the yyyy.MM.dd HH:mm:ss format to enter data information

Tender Deadline * 2002.12.01 12:12:12

Load Vector * - the time, in hours, that your application would take to run on your machine. 2.74

Submit By * 2003.01.01 12:12:12

End By 2010.01.01 12:12:12

Service Class no preference

Fig. 11. Examples of the Graphical User Interface (GUI) used in the experiments in the adopted Grid infrastructure. (a) The sign in GUI. (b) The task parameters GUI.

A fair scheduling scheme is implemented for assigning the tasks to the most appropriate processors for execution. In our case, the task queuing order is selected with respect to the adjusted task fair completion times, which are calculated from the task fair rates, taking into account new arrivals and tasks completions. On the contrary, processor assignment is performed using the Earliest Completion Time (ECT) modified so that processors gaps are examined.

The proposed architecture has been implemented in a Grid infrastructure in the framework of GRIA and GRIDLAB European Union funded Research projects. Our study has been shown that the demanded QoS users' parameters are satisfied in a better and a fairer way. This is due to the fact that in the proposed architecture, we achieve very good performance in task workload prediction and therefore in task runtime estimation, which significantly enhances the scheduling performance. On the other hand, the proposed fair scheduling scheme outperforms the traditional scheduling approaches in the sense that it handles the demanded for execution tasks in a fair way.

One issue for further investigation is to introduce an adaptable non-linear workload prediction mechanism so as to increase the prediction accuracy in complicated real-life applications. The proposed workload prediction scheme assumes that tasks derive from a specific class of applications and therefore the non-linear relation between the workload descriptors and the task actual load (non-linear workload model) is constant. To overcome this constraint, adaptable non-linear modeling is required able to update the non-linear relation according to the current task characteristics.

Acknowledgements

The authors thank Mr. Dionysios Karaoglanoglou for his assistant in implementing in Java environment the Grid fair scheduling algorithms used in the proposed architecture. The authors also thank the European Union for the financial support in implementing the Grid architecture in the framework of IST GRIA and GRIDLAB project of this work.

References

[1] I. Foster, C. Kesselman, S. Tuecke, The anatomy of the grid: enabling scalable virtual organizations, International Journal Supercomputer Applications 15 (3) (2001).
[2] W. Leinberger, V. Kumar, Information power grid: the new frontier in parallel computing? IEEE Concurrency 7 (4) (1999) 75–84.
[3] R. Al-Ali, K. Amin, G. von Laszewski, O. Rana, D. Walker, M. Hategan, N. Zaluzec, Analysis and provision of QoS for distributed grid applications, Journal of Grid Computing 2 (2) (2004) 163–182.
[4] Scheduling Working Group of the Grid Forum, Document: 10.5, September 2001.
[5] P. Dinda, Online prediction of the running time of tasks, Cluster Computing 5 (3) (2002).
[6] E. Kreyszig, Introductory Functional Analysis with Applications, Wiley, New York, 1989.
[7] S. Haykin, Adaptive Filter Theory, Prentice-Hall, Englewood Cliffs NJ, 1996.
[8] D. Bertsekas, R. Gallager, Data Networks, Prentice-Hall, Englewood Cliffs NJ, 1992, The section on max-min fairness starts on p. 524.
[9] I. Foster, C. Kesselman, Globus: a metacomputing infrastructure toolkit, International Journal of Supercomputer Applications 11 (2) (1997) 115–128.
[10] J. Basney, M. Livny, T. Tannenbaum, High throughput computing with condor, HPCU news 1 (2) (1997).
[11] D. Thain, T. Tannenbaum, M. Livny, Condor and the grid, in: F. Berman, A.J.G. Hey, G. Fox (Eds.), Grid Computing: Making The Global Infrastructure a Reality, Wiley, New York, 2003, ISBN: 0-470-85319-0.
[12] F. Berman, A. Chien, K. Cooper, J. Dongarra, I. Foster, D. Gannon, L. Johnsson, K. Kennedy, C. Kesselman, J. Mellor-Crummey, D. Reed, L. Torczon, R. Wolski, The GrADS project: software support for high-level grid application development, International Journal of High Performance Computing Applications 15 (4) (2001) 327–344.
[13] S. Zhuk, A. Chernykh, A. Avetisyan, S. Gaissaryan, D. Grushin, N. Kuzjurin, A. Pospelov, A. Shokurov, Comparison of scheduling heuristics for grid resource broker, in: IEEE Fifth Mexican International Conference in Computer Science, Colima, México, 2004, pp. 388–392.
[14] D.P. Spooner, S.A. Jarvis, J. Cao, S. Saini, G.R. Nudd, Local grid scheduling techniques using performance prediction, IEE Proceedings in Computers and Digital Techniques 150 (2) (2003) 87–96.
[15] S. Kim, J.B. Weissman, A genetic algorithm based approach for scheduling decomposable data grid applications, in: International Conference on Parallel Processing (ICPP'04), Montreal, Quebec, Canada, 2004, pp. 406–413.
[16] Keqin Li, Experimental performance evaluation of job scheduling and processor allocation algorithms for grid computing on metacomputers, in: Proceeding of the IEEE 18th International Parallel and Distributed processing Symposium (IPDPS), Santa Fe, New Mexico, 2004, pp. 170–177.
[17] J.M. Schopf, Ten actions when grid scheduling, in: J. Nabrzyski, J.M. Schopf, J. Weglarz (Eds.), Grid Resource Management: State of the Art and Future Trends, Kluwer, Dordrecht, 2003 (Chapter 2).
[18] M.A. Iverson, F. Ozguner, L. Potter, Statistical prediction of task execution times through analytic benchmarking for scheduling in a heterogeneous environment, IEEE Transactions on Computers 48 (12) (1999) 1374–1379.
[19] D. Pease, A. Ghafoor, I. Ahmad, D.L. Andrews, K. Foudil-Bey, T.E. Karpinski, M.A. Mikki, M. Zerrouki, PAWS: a performance evaluation tool for parallel computing systems, Computers 24 (1) (1991) 18–29.
[20] J. Yang, A. Khokhar, S. Sheikh, and A. Ghafoor, Estimating Execution Time for Parallel Tasks in Heterogeneous Processing (HP) Environment, in: Proceedings of the Heterogeneous Computing Workshop, 1994, pp. 23–28.
[21] A.K. Parekh, R.G. Gallager, A generalized processor sharing approach to flow control in integrated services networks: the single-node case, IEEE/ACM Transactions on Networking 1 (3) (1993) 344–357.
[22] A. Demers, S. Keshav, S. Shenker, Design and analysis of a fair queuing algorithm, in: Proceedings of the ACMSIGCOMM, Austin, 1989.
[23] F. Vraalsen, R.A. Aydt, C.L. Mendes, D.A. Reed, Performance contracts: predicting and monitoring grid application behavior, in: Proceedings of the 2nd International Workshop on Grid Computing (GRID), vol. 2242, Denver, Colorado, Nov. 2001, pp. 154–165.
[24] J.L. Hennessy, D.A. Patterson, D. Goldberg, Computer Architecture: A Quantitative Approach, third ed., Morgan Kaufmann, San Mateo, CA, 2002.
[25] T. Stricker, T. Cross, Global Address Space, Non-Uniform Bandwidth: A Memory System Performance Characterization of Parallel

Systems. in: Proceedings of IEEE Symposium on High-Performance Computer Architecture (HPCA '97), February 1997.

[26] N. Doulamis, A. Doulamis, A. Panagakis, K. Dolkas, T. Varvarigou, E. Varvarigos, A combined fuzzy-neural network model for non-linear prediction of 3D rendering workload in grid computing, IEEE Transactions on Systems Man and Cybernetics, Part-B 34 (2) (2004) 1235–1247.

[27] D.W. Marquardt, An algorithm for least-squares estimation of nonlinear parameters, Journal of the Society for Industrial and Applied Mathematics 11 (2) (1963) 431–441.

[28] S. Haykin, Neural Networks: A Comprehensive Foundation, Macmillan, New York, 1994.

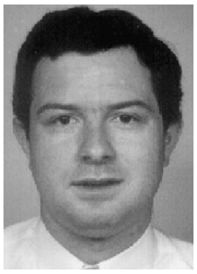
[29] E.B. Baum, D. Haussler, What size net gives valid generalization? Neural Computation 1 (1) (1989) 151–160.

[30] M.S. Fineberg, O. Serlin, Multiprogramming for Hybrid Computation, in: Proceedings of IFIPS Fall Joint Computer Conference, Thompson, Washington, DC, 1967.

[31] G. Allen, K. Davis, K. Dolkas, N. Doulamis, et al., Enabling applications on the grid: a gridlab overview, International Journal of High Performance Computing Applications 17 (4) (2003) 449–466.

[32] IST-2001-33240, Grid Resources for Industrial Applications (GRIA), European Union program of Information Societies Technology.

[33] A. Watt, M. Watt, Advanced Animation and Rendering Techniques: Theory and Practice, Addison-Welsey, New York, 1992.

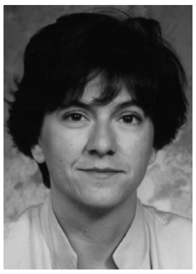


Nikolaos Doulamis received the Diploma degree in Electrical and Computer Engineering from the National Technical University of Athens (NTUA) in 1995 with the highest honor and the Ph.D. degree in electrical and computer engineering from NTUA in 2000. He joined the Image, Video, and Multimedia Lab of NTUA in 1996 as research assistant. His Ph.D. thesis was supported by the Bodosakis Foundation Scholarship. From 2002, he is a senior researcher in the NTUA. In 2000, he was served as Chairman of technical program committee of the VLBV'01 workshop, while he has also served as program committee in several international conferences and workshops. In 2000, he was given the Thomaidion Foundation best journal paper award in conjunction with A. Doulamis. He is reviewer of IEEE journals and conferences as well as and other leading international journals. His research interest includes video transmission, content-based image retrieval, summarization of video sequences and intelligent techniques for video processing.

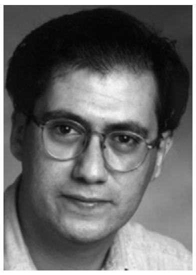
Anastasios Doulamis received the Diploma degree in Electrical and Computer Engineering from the National Technical University of Athens (NTUA) in 1995 with the highest honor. In 2000, he has received the Ph.D. degree in electrical and computer engineering from the NTUA. From 1996–2000, he was with the Image, Video, and Multimedia Lab of the NTUA as research assistant while in 2002, he joined the NTUA as senior researcher. He is now Assistant Professor at Technical University of Crete. In 2001, he served as technical program chairman of the VLBV'01. He has also served as program committee in several international conferences and workshops. He is reviewer of IEEE journals and conferences as well as and other leading international journals. He is author of more than 100 papers in the above areas, in leading international journals and conferences. His research interests include, non-linear analysis, neural networks, multimedia content description, intelligent techniques for video processing.

Antonios Litke received the diploma from the Computer Engineering and Informatics Department, University of Patras, Greece in 1999. After that he has worked in the industrial sector as a Telecom Software Engineer. Currently, he is pursuing his Ph.D. in the Telecommunication Laboratory of Electrical and Computer Engineering, Department of National Technical University of Athens and works as research associate in the Institute of Communication and Computer Systems participating in numerous EU and National funded projects. His research interests include Grid computing, resource management in heterogeneous systems, Web services, and information engineering.

Athanasios Panagakis received the B.Tech. degree from the National Technical University of Athens, Greece, in 2000 and the Ph.D. degree from the National Technical University of Athens, in 2003. Since 2000, he is a research associate in the Communications, Electronics, and Information Engineering Division of the Electrical and Computer Engineering, Department of NTUA. He has been involved in many national and EU research projects (e.g., GRIA, GridLab) and his current research interests include workload prediction and grid accounting mechanisms.



Theodora Varvarigou received the B. Tech degree from the National Technical University of Athens, Athens, Greece in 1988, the MS degrees in Electrical Engineering (1989) and in Computer Science (1991) from Stanford University, Stanford, California in 1989 and the Ph.D. degree from Stanford University as well in 1991. She worked at AT&T Bell Labs, Holmdel, New Jersey between 1991 and 1995. Between 1995 and 1997, she worked as an Assistant Professor at the Technical University of Crete, Chania, Greece. Since 1997, she is working as an Associate Professor at the National Technical University of Athens. Her research interests include Grid Technologies parallel algorithms and architectures, fault-tolerant computation, optimisation algorithms, and content management.



Emmanouel (Manos) Varvarigos received a Diploma in Electrical and Computer Engineering from the National Technical University of Athens in 1988, and the M.S. and Ph.D. degrees in Electrical Engineering and Computer Science from the Massachusetts Institute of Technology, Cambridge, MA, in 1990 and 1992, respectively. In 1990, he worked as a researcher at Bell Communications Research, Morristown, NJ. From 1992 to 1998, he was an Assistant and later an Associate Professor at the department of Electrical and Computer Engineering at the University of California, Santa Barbara. In 1998–1999, he was an Associate Professor at the Electrical Engineering department at Delft University of Technology, the Netherlands. In 1999, he became a Professor at the department of Computer Engineering and Informatics at the University of Patras, where he is the director of the Communication Networks Laboratory. He is also the Director of Network Technologies Sector of the Research Academic Computer Technology Institute (RA-CTI). He was the organizer of the 1998 Workshop on Communication networks and was in the program committee of several international conferences. His research activities are in the areas of protocols and algorithms for high-speed networks, all-optical networks, high-performance switch architectures, grid computing, parallel architectures, performance evaluation, and ad hoc networks.