An Advanced Architecture for a Commercial Grid Infrastructure

Antonios Litke[1], Athanasios Panagakis[1], Anastasios Doulamis[1], Nikolaos Doulamis[1], Theodora Varvarigou[1], and Emmanuel Varvarigos[2]

[1] Electrical and Computer Engineering Dept.
National Technical University of Athens
ali@telecom.ntua.gr
[2] Computer Engineering and Informatics Dept., University of Patras

Abstract. Grid Infrastructures have been used to solve large scale scientific problems that do not have special requirements on QoS. However, the introduction and success of the Grids in commercial applications as well, entails the provision of QoS mechanisms which will allow for meeting the special requirements of the users-customers. In this paper we present an advanced Grid Architecture which incorporates appropriate mechanisms so as to allow guarantees of the diverse and contradictory users' QoS requirements. We present a runtime estimation model, which is the heart of any scheduling and resource allocation algorithm, and we propose a scheme able to predict the runtime of submitted jobs for any given application on any computer by introducing a general prediction model. Experimental results are presented which indicate the robustness and reliability of the proposed architecture. The scheme has been implemented in the framework of GRIA IST project (Grid Resources for Industrial Applications).

1 Introduction

Grid computing is distinguished from conventional distributed computing by its focus on large-scale resource sharing, innovative applications, and, in some cases, high-performance orientation. It supports the sharing, interconnection and use of diverse resources in dynamic computing systems that can be sufficiently integrated to deliver computational power to applications that need it in a transparent way [1], [2].

However, until now grid infrastructure has been used to solve *large-scale scientific* problems that are of known or open source code and *do not* have specific *Quality of Service* (QoS) requirements [1], [3]. For example, in the current Grid architecture, there is no guarantee that particular users' demands, such as the deadlines of the submitted tasks, are always satisfied. This means that the current Grid architecture can not provide an agreed upon QoS, which is important for the success of the Grid, especially in commercial applications. Users of the Grid are not willing to pay for Grid services or contribute resources to Grids, if there are not appropriate mechanisms able to guarantee the negotiated QoS

M. Dikaiakos (Ed.): AxGrids 2004, LNCS 3165, pp. 32–41, 2004.

users' requirements. This need has been confirmed by the Global Grid Forum (GGF) in the special working group dealing with "scheduling and resource management" for Grid computing [4].

Scheduling and resource allocation is of vital important in the commercialization of Grid, since it allows management of the contradictory and diverse QoS requirements. Furthermore, scheduling and resource allocation is strongly related with the adopted charging policy. More resources of the Grid should be given to users that are willing to pay more. However, efficient scheduling and resource allocation requires estimation of the *runtime of each task requesting* for service in the Grid, which in the sequel requires *prediction of the task workload as well as task modeling.* Different applications are characterized by different properties and thus require different modeling and prediction schemes.

In this paper, we enhance the current Grid architecture by incorporating all the aforementioned described mechanisms so as to allow guarantees of the diverse and contradictory users' QoS requirements. Our focus is oriented on developing a proper runtime estimation model, which is the heart of any scheduling and resource allocation algorithm. In particular, we propose a scheme able to predict the runtime of submitted jobs for any given application on any given computer by introducing a general prediction model. The model is applied to any application using features derived from the task modeling module. To achieve this goal, a set of *common parameters* is defined, which affect the runtime and are the same for any application.

The proposed runtime estimation model is separated in two parts. The part of the consumer's (client's) side, which is responsible for *workload estimation* and the supplier's part, which evaluates the *resource performance.* The resource performance parameters are designed so that they can be applied to heterogeneous platforms, while the workload performance parameters are designed to be the same for every application.

The workload parameters are classified into a) computation, b) communication and c) availability parameters. Computation parameters are associated with the task execution. These are: the *number of float point operations per task*, *the number of exchanged memory I/O messages per task* and *the number of exchanged disk I/O messages per task.* The communication parameters are separated in the two parts; the *send* and the *receive* part. In this analysis we assume that the amount of bytes which is *sent and received* are used as communication parameters. Finally, the availability parameters are *the minimum free memory* (i.e., the sum of available minimum free memory, which is allocated by the system during processing), *the minimum disk space* (i.e., the sum of storage space, which is allocated by the resource during processing), and *the queue time interval* (i.e., the total waiting time in the queue for a newly arrived job).

As far as the resource performance parameters are concerned, *the CPU speed* (expressed as the MFLOPs rate), *the average memory I/O bandwidth* (in Mbytes/sec) and *the average disk I/O bandwidth* (in Kbytes/sec) are selected. The choice of these parameters is due to the fact that they are measurable in any heterogeneous platform and characterize the performance of a system [5].

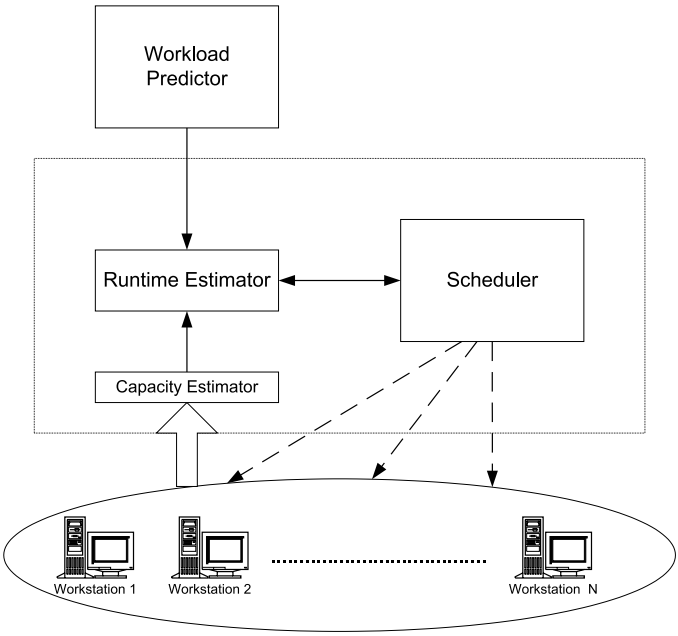


Fig. 1. The proposed architecture adopted for the commercialization of the Grid

The MFLOPs rate is a performance measure independent of the CPU architecture, thus allowing comparing different CPUs. Moreover, most of the MFLOPs benchmarks take into account the bottleneck due to L1 and L2 cache, eliminating the need of benchmarking L1 and L2 cache performance. Besides L2 cache, the main bottleneck in I/O communication is the RAM bandwidth. Since every application accesses RAM in a different way, in order to cover any option we take an average measure of memory I/O. For applications that need a huge amount of disk I/O we consider the disk performance bottleneck, defined as the average I/O bandwidth in Kbytes/sec [5].

2 The Proposed Commercial Grid Architecture

In this section we present the proposed architecture for a commercial Grid infrastructure, which extends the Globus architecture [3] by contributing QoS aspects in the resource management model. Even that the Globus layered architecture can be enhanced with such mechanisms, QoS in Grid computing has not been addressed currently. In the sequel, we present the necessary architectural components that implement the QoS mechanism proposed. The architecture has been implemented in the framework of GRIA project [6]. Figure 1 presents the proposed adopted architecture.

a. Workload Predictor. It is the part that evaluates the application specific input parameters that affect the runtime. These parameters are then used to

predict the workload performance parameters that are needed for the runtime estimation. It estimates a set of workload parameters for a given set of application specific input parameters. These are passed to the Runtime Estimator in order to be used for further processing so as to estimate the execution time of the job. The Workload Predictors are each one dedicated for each different application that is incorporated on the system.

b. Capacity Estimator. It calculates the resource performance parameters for each different resource of the supplier. The Capacity Estimator should calculate the resource performance parameters of the runtime estimation model, through a benchmarking process that is the same for every heterogeneous resource platform, thus providing a way to compare heterogeneous resource performance. 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.

c. Runtime Estimator. It uses a mathematical model to combine the workload parameter set from the Workload Predictor, with the resource parameter set from the Capacity Estimator to give estimation about the execution time of the specific job on the specific resource.

d. Scheduler. It is the main module that applies the scheduling policy and procedure based on the Runtime Estimator, and according to the deadlines of the jobs as they are given by the customer.

3 Runtime Estimation Model

3.1 Resource Performance Parameters for Heterogeneous Platforms

A generic PC architecture consists of a CPU, L1, L2 cache and RAM (memory hierarchy) and the hard disk. There are several different architectures for each of the above components and different operating systems. For the CPU performance the most suitable generic measurement is the MFLOPS benchmark [5]. Since the majority of the MFLOPS benchmarks take into account the L1 and L2 cache rates, we can assume that the only suitable measurement for the memory performance is the average RAM I/O bandwidth. Also for the hard disk performance we consider as suitable performance measurement the average read/write disk I/O bandwidth.

We now see that the achieved application performance can be expressed in conjunction with these three values. Therefore, we denote as $\mathbf{r}$, the *resource parameter vector*, the elements of which r_i, i=1,2,3 correspond to the CPU speed (in MFLOPS/sec), average memory I/O (in MB/sec) and average disk I/O (in KB/sec).

$$\mathbf{r} = [r_1, r_2, r_3]^T \tag{1}$$

The resource parameter vector can be calculated for each resource through a benchmarking process. In this study we assume that the resources that are taken into consideration are limited to standalone PC platforms and not clusters or

batch systems. Thus we consider a Grid infrastructure with single node tasks on single PCs. The important thing about the runtime estimation model is that on every resource that we want to have runtime estimation we have to use the same benchmarks. Since the benchmarks can be compiled for different operating systems and platforms, we can use the same **r** vector for any heterogeneous platform, thus having the capability of incorporating any heterogeneous resource on the Grid infrastructure using the same resource performance description.

Equation (1) refers to the computational resource parameters. Concerning the communication parameters, we use the *Send Communication Bandwidth* and the *Receive Communication Bandwidth* both measured in Kbytes/sec. For the availability resource parameters we use the *Minimum Free Memory* (in MB) and *Minimum Free Disk Space* (in KB) of the resource where the task will be allocated. However these additional parameters are not being taken into consideration within the scope of this paper, since the work presented in this paper is focused on the Runtime Estimation model. The overall system that has been designed and implemented within the framework of GRIA project [6] uses the aforementioned parameters to calculate a Remote Runtime that consists additionally of the communication time (send and receive time intervals of the application data) and the queue time interval that is the waiting time for the application to start execution on the selected resource. However the scope of this paper is to propose and validate a new model of Grid architecture incorporating QoS aspects, and thus it is focused on proving the validity of the proposed Runtime Estimation model, which is used to calculate the execution time only.

The proposed scheme is not used for application tasks that run in parallel. Thus the latency factor has not been taken into consideration because there are no continuous transaction between the individual resources during the execution phase of a job.

3.2 Definition of Application Workload Parameters

The workload parameters must be defined in conjunction with the resource parameters, in order to use the same runtime estimation model for every application. In this paper we have used for the resource parameters the MFLOPS/sec, the average memory I/O and the average disk I/O (see section 3.1), and thus the workload parameters are defined only by the *computational parameters*. Extension of our study for including the affect of the other workload parameters can be performed in a similar way.

To estimate the workload of a task we need a) to extract features which describe the specific application from which the task derive and b) to define a set of parameters which are in conjunction with the three resource performance values [see equation (1)].

Let us first denote as **x** a vector which describes the "computational load" of a task derived from an application. We call vector **x** *workload parameter vector*. In our paper and without loss of generality we assume that vector **x** consists of three elements

$$\mathbf{x} = [x_1, x_2, x_3]^T , \tag{2}$$

where the elements x_i, i=1,2,3 correspond to the CPU instructions per task (in MFLO), the average memory I/O amount per task (in MB) and the average disk I/O amount (in KB).

To estimate vector **x,** we need to extract for each specific application those features which affect the respective workload. Let us denote as **s** the *descriptor parameters vector*

$$\mathbf{s} = [s_1, s_2, ..., s_n]^T , \tag{3}$$

the elements of which s_i, i=1,..,n correspond to the individual application *descriptors*. The descriptors s_i are independent of the execution environment. For example, in case we refer to 3D rendering applications the descriptors s_i are the image resolution, number of polygons, number of light sources and so on. It is clear that for different applications different descriptors are required [7], [8], [9]. So, for each application incorporated to the system we must construct a different predictor for estimating vector **x**.

3.3 Workload Prediction

Vector **s** is used as input to the Workload Predictor module, which is responsible for estimating vector **x** from **s,** through a non-linear model $\mathbf{x} = g(\mathbf{s})$. Generally, the function $g(\mathbf{s})$ is unknown and thus it can not be estimated in a straightforward way. For this reason, modeling of function $g(\cdot)$ is required for predicting vector **x** from vector **s**. Usually, linear models cannot effectively estimate the application workload. This is caused since usually there does not exist a simple linear relation, which maps the specific input parameters (e.g., vector **s**) with the corresponding workload parameters (e.g. vector **x**). Alternatively, modeling can be performed using simplified non-linear mathematical models (such as exponential and/or logarithmic functions) and applying estimation techniques [10] for predicting the vector **x**. However, these approaches present satisfactory results only in case of data that follow the adopted pre-determined function type, which is not be extended to any type of application.

In order to have a *generic* workload prediction module, which can be applied to any type of application, modeling of the unknown function$g(\cdot)$ is performed through a neural network architecture. This is due to the fact that neural networks are capable of estimating any continuous non-linear function with any degree of accuracy [11]. In particular, neural networks provide an approximation of function $g(\cdot)$, say $\hat{g}(\cdot)$, through a training set of samples consisting of appropriate selected vectors $\mathbf{x}_i$ and the respective vectors $\mathbf{s}_i$. Training is performed based on a Least Squares algorithms, such as the Marquardt-Levenberg algorithm [11].

3.4 The Runtime Estimation Model

As already mentioned, the amount of workload that is served per second is given by the resource capability. We recall that x_i is the workload of the i-th task being executed on a resource characterized by a resource parameter r_i. We denote as

t_i the time interval needed to accomplish the execution of x_i on a resource with related resource parameter r_i. The t_i is related with x_i and r_i as follows

$$t_i = \frac{x_i}{r_i}. \tag{4}$$

To estimate the total run time of a task, we assume that total execution time equals the sum of individual execution times. Therefore, we have that

$$t_{run} = \sum_{i=1}^{n} t_i, \quad or \quad t_{run} = \sum_{i=1}^{n} x_i \cdot r_i^{-1} \tag{5}$$

However, estimation of the total run time based on the previous equation does not result in reliable results since only one measure is taken into consideration. To have a more reliable estimate of the total run time, several measurements are taken into account and a linear system is constructed for estimating t_{run}. In particular, the total run time t_{run} is provided by minimizing the following equation

$$\hat{t}_{run} = \arg\min E, \tag{6}$$

where

$$E = \sum_{j=1}^{N} \{\hat{t}_{run} - \sum_{i=1}^{n} x_{i,j} \cdot r_{i,j}^{-1}\}^2 \tag{7}$$

In previous equation $x_{i,j}$, $r_{i,j}$ is the j-th sample of the x_i and r_i respectively. Minimization of (7) is accomplished through the Least Square method.

4 Scheduling

The purpose of a scheduling algorithm is to determine the "queuing order" and the "processor assignment" for a given task so that the demanded QoS parameters, i.e., the task deadlines, are satisfied as much as possible. The "queuing order" refers to the order in which tasks are considered for assignment to the processors. The "processor assignment" refers to the selection of the particular processor on which the task should be scheduled.

In the proposed Grid architecture, two approaches for queuing order selection have been adopted, which are described shortly in the following. The first algorithm exploits the urgency of the task deadlines, while the second is based on a fair policy. The most widely used urgency-based scheduling scheme is the Earliest Deadline First (EDF) method, also known as the deadline driven rule [12],[13]. 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. The above mentioned queuing order selection algorithm does not make any attempt to handle the tasks requesting for service in a fair way.

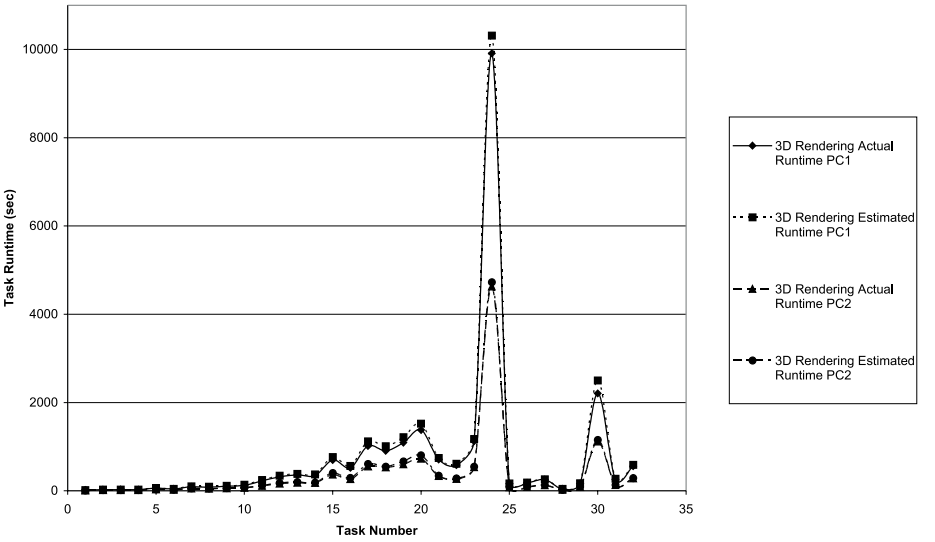


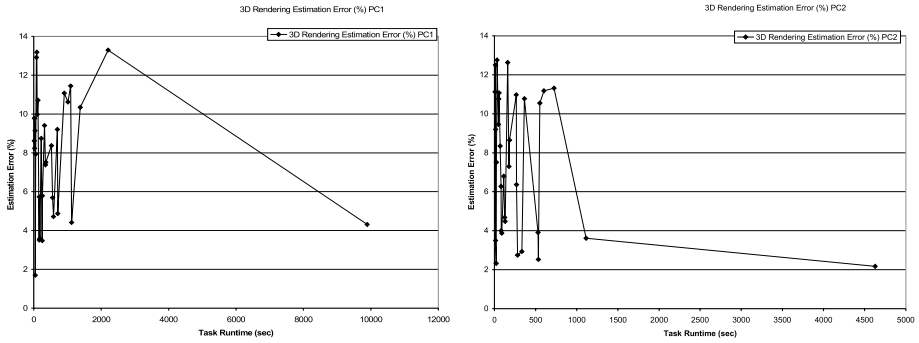
Fig. 2. The *actual* and the *estimated* total run time for the two *PCs* that have been used for the 32 different 3D image rendering tasks

The second algorithm uses a Max-Min fair sharing approach for providing fair access to Grid resources to all users. When there is no shortage of resources, the algorithm assigns to each task enough computational power for it to finish within its deadline. In case of congestion the CPU rates assigned to the tasks are reduced fairly, so that the share of the resources that each user gets is proportional to the users' contribution to the Grid infrastructure or alternatively to the price he is willing to pay. As an example, we can assume three tasks whose fair completion times are 8, 6 and 12 respectively. As a result, the second, first and finally the third task is assigned for execution.

5 Experimental Results

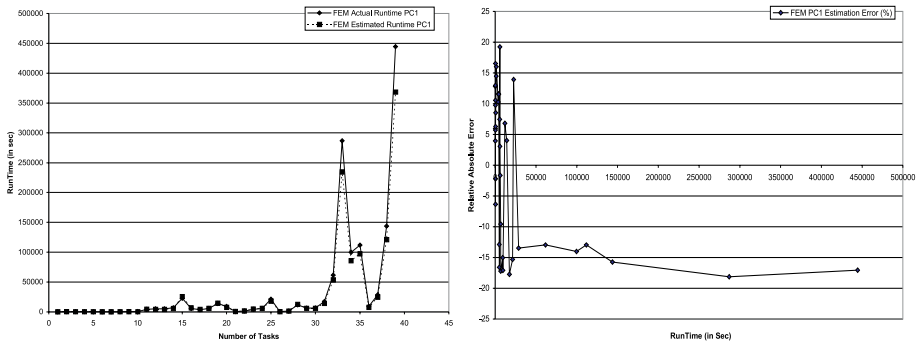
The Grid architecture of the GRIA Project has been tested for two different applications, 3D image rendering with BMRT2.6 and the Finite Element Method (FEM) with INDIA, used in construction engineering. The results indicate the validity of the Runtime Estimation model that has been described in this paper. Ten computers of the Grid infrastructure have been used in this study as resources and they were benchmarked with SiSoftware Sandra Standard Unicode (32-bit x-86) 2003.1.9.31 under Microsoft Windows 2000.

For the 3D rendering application the runtime of 32 different tasks has been measured over the 10 different computers. For each one of the 8 PCs out of the 10 we formed the equation (5). We solved this over-determined system of the 8 equations (N=8), so as to calculate the $\mathbf{x}$ vector in (2). The actual values of the execution on the remaining 2 PCs are compared against the estimated



(a) (b)

Fig. 3. 3D image rendering case - The error for the runtime estimation is calculated as the *relative absolute error* (in percentage) for a) the first of the 2 PCs and b) for the second PC



(a) (b)

Fig. 4. (a) The *actual* and *estimated* total *runtime* for the FEM case; (b) The *error* for the run time estimation (in percentage) for the FEM case

time which is calculated using this **x** vector. Figure 2 presents the actual and predicted runtime of the 3D rendering application for the two PCs used for testing the results, while Fig. 3 presents the error for the two PCs. We can see that the error of the runtime estimation model is less than 13%.

For the FEM application we measure the runtime of 39 different tasks over 6 different computers. The over-determined system of 5 equations (N=5) has been solved to estimate the run time model, while the remaining 1 PC has been used to compare the estimated time with the actual one. Again, we can see that the error does not exceed 19%. Figure 4(a) presents the actual and the predicted run time, while Fig. 4(b) the error for the 1PC.

6 Conclusions

In order to commercialize the Grid infrastructure, we need to satisfy the QoS requirements imposed by the users who are willing to use the Grid infrastructure

for fulfilling their commercial needs. To accomplish such Grid commercialization we need a modification of the existing architectures, so that the QoS requirements are satisfied as much as possible. This is proposed in this paper by introducing a Workload Predictor, a Capacity Estimator, a Runtime Estimator and a Scheduler. We also propose an accurate Runtime Estimation model. This model has been implemented and evaluated in the framework of the GRIA EU funded project. The experimental results illustrate accurate runtime prediction of the model in all the examined cases. The results have been obtained using 2 different commercial applications, the 3D image rendering and the Finite Element Method used in construction engineering.

References

1. Foster, I., Kesselman, C., Tuecke, S.: The Anatomy of the Grid: Enabling Scalable Virtual Organizations. Inter. Journal Supercomputer Applications. **15** (2001).
2. Leinberger, W., Kumar, V.: Information Power Grid: The new frontier in parallel computing? IEEE Concur., **7** (1999), 75-84.
3. Foster, I., Kesselman, C., Nick, J., Tuecke, S.: The Physiology of the Grid, An Open Grid Services Architecture for Distributed Systems Integration. www.globus.org (The Globus Project), 6/22/2002.
4. Scheduling Working Group of the Grid Forum, Document: 10.5, September 2001.
5. Vraalsen, F., Aydt, R., Mendes, C., Reed, D.: Performance Contracts: Predicting and Monitoring Grid Application Behavior. Proceedings of the 2nd International Workshop on Grid Computing/LNCS, **2242** (2001), 154-165.
6. IST-2001-33240: Grid Resources for Industrial Applications (GRIA). European Union program of Information Societies Technology.
7. Doulamis, N., Doulamis, A., Panagakis, A., Dolkas, K., Varvarigou, T., Varvarigos, E.: A Combined Fuzzy-Neural Network Model for Non-Linear Prediction of 3D Rendering Workload in Grid Computing. IEEE Trans. on Systems, Man and Cybernetics -Part B (to be published in 2004).
8. Doulamis, N., Doulamis, A., Panagakis, A., Dolkas, K., Varvarigou T., Varvarigos, E.: Workload Prediction of Rendering Algorithms in GRID Computing. European Multigrid Conference, (2002), 7-12.
9. Doulamis, N., Doulamis, A., Dolkas, K., Panagakis, A., Varvarigou T., Varvarigos, E.: Non-linear Prediction of Rendering Workload for Grid Infrastructure. International Conference on Computer Vision and Graphics, Poland Oct. 25-28, 2002.
10. Kobayashi, H.: Modeling and Analysis. Addison-Wesley 1981.
11. Haykin, S.: Neural Networks: A Comprehensive Foundation. New York: Macmillan.
12. Peha, J.M., Tobagi, F.A.: Evaluating scheduling algorithms for traffic with heterogeneous performance objectives. IEEE Global Telecom. Conf., **1**, (1990) 21-27.
13. Ku, T.W., Yang, W.R., Lin, K.J.: A class of rate-based real-time scheduling algorithms. IEEE Trans. on Computers, **51** (2002),708-720.