An Economic Model for Grid Scheduling

MASSIMILIANO CARAMIA
Università di Roma "Tor Vergata"
Dipartimento di Ingegneria dell'Impresa
Via del Politecnico, 1 - 00133 Roma
ITALY

STEFANO GIORDANI
Università di Roma "Tor Vergata"
Dipartimento di Ingegneria dell'Impresa
Via del Politecnico, 1 - 00133 Roma
ITALY

Abstract: Grid scheduling, that is, the allocation of distributed computational resources to user applications, is one of the most challenging and complex task in Grid computing. In this paper, we give a quantitative description of a tender/contract-net model. The performance of the proposed market-based approach is experimentally compared with a simple round-robin allocation protocol.

Key–Words: Grid computing, Resource Management, Economic models, Scheduling, Simulation

1 Introduction

Grids are distributed computational systems that allow users to access resources owned by different organizations [3]. One of the most known framework for Grid scheduling is the one introduced by Ranganathan and Foster in [5]. In this architecture, users submit requests for task execution from any one of a number of sites. At each site, besides the local computing system, the system model is composed by three components: an *External Scheduler* (ES) responsible for determining a particular site where a submitted task can be executed; a *Local Scheduler* (LS), responsible for determining the order in which tasks are executed at that particular site; a *Dataset Scheduler* (DS), responsible for determining if and when to replicate data and/or delete local files. On receipt of a task request, the ES interrogates the LSs to ascertain whether the task can be executed on the available resources and meet the user-specified due date. If this is the case, a specific site in which executing that task is chosen. Otherwise, the ES attempts to locate a LS of a site, controlled by another ES, that can meet the task processing requirements, through search mechanisms. If a LS cannot be located within a preset number of search steps, the task request is either rejected or passed to a scheduler that can minimize the due date failure depending on a task request parameter. When a suitable site is located, the task request is passed from the ES to this site and is managed by the associated LS. Within such a framework, most of the related work in Grid computing dedicated to resource management and scheduling adopt a conventional style where a scheduling component decides which jobs are to be executed at which site based on certain cost functions (e.g., AppLeS [6], NetSolve [2]). Such cost functions are often driven by system-centric parameters that enhance system throughput and utilization rather than improving the utility of task processing.

Another important class of models to manage Grid computing environment is that of economic models in which the scheduling decision is not done statically by a single scheduling entity but directed by the end users requirements. Whereas a conventional cost model often deals with software and hardware costs for running applications, the economic model primarily charges the end user for resources that they consume based on the value they derive from it. Pricing based on the demand of users and the supply of resources is the main driver in the competitive, economic market model. Moreover, in economic models, differently from what happens for the external-local scheduler architecture aforementioned, the two main actors driving the Grid marketplace are: Grid Service Providers (GSPs), representing the resource owners (i.e., the producers), playing the same role as the LSs in the Ranganathan and Forster framework, and Grid Resource Brokers (GRBs), representing the users (i.e., the consumers) in the Grid marketplace, whose role is in part encompassed by the ESs in the previous model. In a generic marketplace model framework, consumers interact with their own brokers for managing and scheduling their applications on the Grid. The interaction between GRBs and GSPs during resource trading (service cost establishment) is mediated through a Grid Market Directory (GMD). They use various economic models or interaction protocols for deciding service access price arise from the real world market, e.g., commodity market, tender/contract-net [1, 4].

In this work we apply the *tender/contract-net model* which is one of the most widely used models for service negotiation in a distributed problem solving environment. It is modelled on the contracting mechanism used by businesses to govern the exchange of goods and services. We give a quantitative description of this model, and experimentally evaluate its performance comparing it with a round-robin allocation protocol.

2 Actors and Behavior Patterns

A set of users (clients) submits task requests to the Grid requiring a certain level of service (*los*). This *los* can be represented by means of a due date provided by the user within which he/she desires to retrieve the output/response of its task request and/or by an amount of money (budget) that the user is willing to pay (at most) to have its tasks executed (possibly) within the specified due date. Moreover, a task request can be characterized by many parameters such as the processing requirement or task size (i.e., the number of million instructions (MI) needed for the computation), the task arrival (release) date, the type of task (e.g., a program code, a simulation) that may restrict the possible choices of the computing site (cluster) able to process it. In what follows, let a user submitted task j be characterized by the following parameters: r_j, arrival date; O_j, size (i.e., the processing requirements), in MI; B_j, budget available for task execution, in G$ (Grid $); d_j, due date; w_j, weight, in G$ per time unit.

We assume that the task due date can be exceeded, but this implies a certain loss of the *los*. The amount of this loss constitutes a *penalty* cost for the user, who has submitted the task, which is assumed to be proportional to the application tardiness or delay (i.e., the positive difference between task completion time and its due date). This cost is summed up with the computational cost that the user has to pay to the owner of the resources required for task execution, and the specified budget is the maximum amount that the user is willing to pay to cover both these two costs. The amount of penalty cost per time unit is specified by the weight w_j of task j.

Tasks are processed by machine clusters (servers) and pre-emption is not possible, that is, a task cannot migrate to another cluster once its execution is started. Moreover, we assume that tasks are malleable, that is, the number of resources of a cluster assigned to a task may change during its execution, and a task can be executed on several machines (of the same cluster) in parallel and spread over an arbitrarily large fraction of the available computational resources of a cluster.

The computational Grid responsible of task execution is composed by a number of computing sites or machine clusters (servers), each one controlled by a local scheduler. Let us consider a cluster m being characterized by the following parameters: P_m, number of machines (i.e., PCs, workstations, processors) of the cluster; $R^{\infty}_{m,i}$, peak performance (computation capacity), in million instructions per time unit (e.g., second) (MIPS), of the i-th machine of the cluster; p^i_m, computation price of machine i of cluster m, in G$ per time unit. For simplicity, we assume that the machines of cluster m are homogeneous; hence, $R^{\infty}_{m,i} = R^{\infty}_m$. Moreover, we assume that the computational resource of each machine of a cluster can be split and allocated to different tasks.

Task requests generated by the users are analyzed by a set of external schedulers; we assume that each external scheduler is associated with a subset of users, and is responsible only for their task submissions. For instance, we can assume that user requests are clustered, based on their nature, into topics, leading to a sort of thematic submissions, i.e., there are users that are interested in economics, others that are interested in sciences, and so on. In this context, each external scheduler is employed to work for a particular kind of task submissions. The role of an external scheduler is twofold: on the one hand, it acts in the Grid on the behalf of the user looking at feasibility of the *los* required; on the other hand, it has to interact with the resource owners in the Grid to allow a matching between tasks and resources for their execution.

This kind of mechanism requires also a sort of cooperation between these two layers, i.e., the external scheduler and the local schedulers. When a user request is submitted, it is routed to an external scheduler for evaluation. Together with the request, the external scheduler receives from the user the *los* desired. Since the users is not aware of the status of the Grid and thus he/she does not know if enough resources are available to allow the immediate processing of the task, it can happen that the desired *los* is not obtainable. Thus, the external scheduler has to decide whether to accept or not that task request and in the former case guaranteeing the associated *los*. To do so, it has to interact with the local schedulers controlling machine clusters where resources reside to take a decision about the task request acceptance. With this setting, the performance of the system can be evaluated in terms of both number of successfully processed tasks, i.e., the number of tasks finished respecting their due dates, violations of the due dates, and the number of tasks rejected, i.e., those not accepted by the external scheduler.

3 Local scheduling policy

We assume that the computation cost that a user has to pay to the computational resource owner for executing its task is driven by a sort of supply-and-demand model. That is, the more a cluster is loaded and the greater is the price per MI (million instruction) that the a new user has to to pay for scheduling its task on that cluster.

In order to represent the fact that the cluster price per MI is an increasing function of the cluster utilization, we assume that the machines of cluster m are indexed in non-decreasing computation price order, and we assume that the local scheduler of cluster m allocates part of the computation capacity of machine i in time period $[t, t+dt)$ only if machine $i-1$ is already fully allocated (busy) in that period. Accordingly, we model the computation price of machine i of cluster m per time unit as $p_m^i = p_m^{\max} - \frac{P_m - i}{P_m - 1}(p_m^{\max} - p_m^{\min})$, with $p_m^{\max}$ and $p_m^{\min}$ being the maximum and minimum price per time unit of a machine of cluster m, respectively. For example, if at a given unit time period (the first) k machines are fully allocated, the computation cost for executing one additional MI of application a in that unit time period is equal to c_m^{k+1}/R_m^{∞}.

Note that according to the supply-and-demand model, the use of a cluster when its load is high is discouraged, while it is encouraged the opposite situation, aiming in this way to a certain load balancing. In fact, we assume that the local scheduling policy aims to minimizing the maximum peak of total cluster utilization (load).

We assume that when an external scheduler ask the local scheduler of cluster m to schedule on that cluster a task j, beside the task size O_j, it also specifies a required completion time C_j for task j. The local scheduler finds the best resource allocation to task j according to the request (O_j, C_j), trying to balance as much as possible the total cluster utilization during the time interval when the task should be executed. Let $\overline{R}_m^i(t) \leq R_m^{\infty}$ be the amount of the computational resource of machine i of cluster m available in time period $[t, t+dt)$. The local scheduler of cluster m computes the amount of computational resources $\rho_{j,m}^i(t)$ (with $0 < \rho_{j,m}^i(t) \leq \overline{R}_m^i(t)$) of machine i to be allocated to j, for each time period $[t, t+dt)$ contained in time window $[s_j, C_j)$ with $s_j \geq r_j$, such that $\int_{s_j}^{C_j} \sum_{i=1}^{P_m} \rho_{j,m}^i(t)dt = O_j$, and the maximum total load $\max_{t\in[s_j,C_j)} \left\{\sum_{i=1}^{P_m}(R_m^{\infty} - \overline{R}_m^i(t) + \rho_{j,m}^i(t))\right\}$ of cluster m in that time window is minimized.

The computation cost that the user should pay to the resource owner for executing task j on cluster m with completion time C_j is therefore $c_{j,m}(C_j) = \int_{s_j}^{C_j} \sum_{i=1}^{P_m} \frac{p_m^i}{R_m^{\infty}} \rho_{j,m}^i(t)\, dt$.

Note that if we suppose that the total amount $R_m^{load}(t) = \sum_{i=1}^{P_m}(R_m^{\infty} - \overline{R}_m^i(t))$ of allocated resources (resource loading profile) of cluster m (before scheduling task j) is a non-increasing function in the time interval $[r_j, +\infty)$, the optimal allocation of required resources for the execution of all the O_j operations in the interval $[s_j, C_j)$ can be obtained by guaranteeing that after scheduling j the total load of m is constant in that interval.

Moreover, this (let us say *perfect*) load balance in time interval $[s_j, C_j)$ also guarantees that the total amount of allocated resources is still non-increasing in the interval $[r_j, +\infty)$ after scheduling task j. Therefore, w.l.o.g., when a new task j' is submitted to the Grid at time $r_{j'}$, we assume that the resource loading profile $R_m^{load}(t)$ of cluster m is a non-increasing function for $t \geq r_{j'}$.

4 Market-based resource management: a tender/contract-net model

In this section we describe in detail the application of an economic model, based on the *tender/contract-net* protocol, for allocating Grid resources to user applications. A user/resource broker asking for a task to be solved is called the *manager*, and a cluster that might be able to execute the task is called the potential *contractor*.

In the tender/contract-net protocol GRBs (managers) announce their task requirements and invite bids from GSPs (contractors). Interested GSPs evaluate the requirements and submit their bids. Each GRB awards the contract to the most appropriate GSP (maximizing its utility function). In details, the steps performed when a new application is submitted to the Grid are reported in Table 1. The tender model allows directed contracts to be issued without negotiation. The selected GSP responds with an *acceptance* or *refusal* of award. In particular, Step 4 is done if the award is accepted by the GSP, otherwise GRB awards the contract to the second best GSP.

When selecting the GSP to which award the contract, on the behalf of the user, the GRB maximizes its utility function. The utility of GRB (i.e., the user utility) for executing and complete task j at time C_j on cluster m is $U_{j,m}(C_j) = B_j - c_{j,m}(C_j) - w_j \max(0, C_j - d_j)$, where we recall that B_j is the budget the user is willing to pay for executing task j, whose cost is the computational (resource) cost $c_{j,m}(C_j)$ plus the penalty cost $w_j \max(0, C_j - d_j)$ for the task tardiness, if task will be completed at time C_j. In particular, the computational cost $c_{j,m}(C_j)$ is

Step 1:	The user submits task j to a GRB.
Step 2.1:	GRB announces resource requirements to GSPs (through the GMD) forexecuting task j of size O_j in time interval $[r_j, C_j)$, and invites offer bidsfrom GSPs.
Step 2.2:	Interested GSPs evaluate the announcement, and respond by submitting their bids to GMD.
Step 3.1:	GRB evaluates the bids submitted by GSPs.
Step 3.2:	GRB identifies the GSP responding with the best bid (the one maximizing GRB utility) among all the offers.
Step 3.3:	If the identified GSP guarantees to GRB a (maximum) utility not less than zero task j is accepted, and GRB awards the contract to that GSP for executing the task; otherwise, it is rejected.
Step 4:	GRB uses the machine cluster of the selected GSP to execute task j, and proceed to the payment of the resource fees to GSP.
Step 5:	The user pays the GRB for executing its task.

Table 1: Steps performed at an user task submission

the bid of the GSP of cluster m, when answering to the GRB request announcement represented by (O_j, C_j).

Recall that, w.l.o.g., we assume that the resource loading profile (i.e., the total allocated resource) $R_m^{load}(t) = \sum_{i=1}^{P_m}(R_m^{\infty} - \overline{R}_m^i(t))$ of cluster m at time t is a non-increasing function, for $t \geq r_j$. Also, recall that the P_m machines of cluster m are indexed in non-decreasing cost order, and that some resources of machine $i+1$ are allocated to some scheduled applications in time period $[t, t+dt)$ only if machine i is fully loaded in that time period.

Next, we show that, for $C_j \geq r_j$, the utility function $U_{j,m}(C_j)$ is piece-wise linear, and even if it is not concave, in general, finding its maximum value can be computed very quickly. This follows from the fact that the resource cost $c_{j,m}(C_j)$ is a piece-wise linear and non-increasing function of C_j. In fact, since in the expression of $U_{j,m}(C_j)$ the budget B_j is constant and the penalty cost $w_j \max(0, C_j - d_j)$ is equal to zero for $C_j < d_j$ and linear for $C_a \geq d_j$, we may restrict the analysis of the resource cost $c_{j,m}(C_j)$ for $C_j \geq r_j$. As a consequence, the maximum value of $U_{j,m}(C_j)$ can be searched only among C_j values where $U_{j,m}(C_j)$ changes its slope: that is, for $C_j = d_j$, and for the times when the slope of the resource cost $c_{j,m}(C_j)$ changes.

W.l.o.g., we assume $c_{j,m}(C_j) = +\infty$ if there is no sufficient amount of resources of cluster m for executing j in time interval $[r_j, C_j)$, and we say that the completion time C_j is infeasible. Therefore, from now we consider only feasible completion times for j.

Proposition 1 *The resource cost $c_{j,m}(t)$ is a non-increasing function, for feasible completion times $t \geq r_j$.*

Let τ_m^h (with $\tau_m^h \geq r_j$) be the h-th time when the load $R_m^{load}(t)$ of cluster m changes (decreases). Note that the number of such times is at most equal to the number of tasks previously scheduled on cluster m, which should be completed after time r_j. Let us denote with $\mathcal{T}_m$ the subset of feasible completion times among times τ_m^h. Each $\tau_m^h \in \mathcal{T}_m$ corresponds to the maximum feasible completion time for task j, when j is restricted to use only resources that are available at time $t < \tau_m^h$.

Let $\theta_{j,m}^i$ (with $\theta_{j,m}^i \geq r_j$) be the minimum feasible completion time of task j, when j is restricted to use only resources belonging to the first (cheapest) i machines (i.e., machines $1, \ldots, i$) among the P_m machines of cluster m. Let us denote with $\Theta_{j,m}$ the set of times $\theta_{j,m}^i$. Note that $\theta_{j,m}^1 \geq \theta_{j,m}^2 \geq \ldots \geq \theta_{j,m}^{P_m}$.

Let $T_{j,m} = (t_{j,m}^1, \ldots, t_{j,m}^{q_{j,m}})$ be the non-decreasing ordered list of feasible completion times of task j, with $T_{j,m} = \Theta_{j,m} \cup \mathcal{T}_m \backslash \{\tau_m^h \in \mathcal{T}_m : \tau_m^h \geq \theta_{j,m}^1\}$. In particular, $t_{j,m}^1 = \theta_{j,m}^{P_m}$, and $t_{j,m}^{q_{j,m}} = \theta_{j,m}^1$.

Proposition 2 *The resource cost $c_{j,m}(t)$ is a linear function of t, for $t \in (t_{j,m}^s, t_{j,m}^{s+1})$, with $s = 1, \ldots, q_{j,m} - 1$.*

Note that, by definition, task j cannot be completed in a feasible way before time $t_{j,m}^1$, hence, we assume that $c_{j,m}(C_j) = +\infty$ for $C_j < t_{j,m}^1$; moreover, according to Proposition 1, the resource cost values $c_{j,m}(t_{j,m}^s)$ does not increase for increasing index s, and $c_{j,m}(C_j)$ is constant for $C_j \geq t_{j,m}^{q_{j,m}}$ since in this case task j will use only resources of machine 1 of cluster m. Moreover, Proposition 2 states that in any time interval $(t_{j,m}^s, t_{j,m}^{s+1})$, with $s = 1, \ldots, q_{j,m} - 1$, the resource cost $c_{j,m}(t)$ varies linearly. Hence, this proves that

Theorem 3 *The resource cost $c_{j,m}(C_j)$ is a non-increasing piece-wise linear function, for feasible completion times $C_j \geq r_j$, and $T_{j,m} = \{t_{j,m}^1, \ldots, t_{j,m}^{q_{j,m}}\}$ is the set of times where $c_{j,m}(C_j)$ changes the slope.*

According to the definition of the utility function $U_{j,m}(C_j)$, and since the resource cost $c_{j,m}(C_j)$ is non-increasing, we have that $U_{j,m}(C_j)$ is a non-decreasing function in the interval $[r_j, d_j)$. Therefore, there is no utility for the GRB to demand computational resources allowing task j to be completed before its due date d_j. Hence, w.l.o.g., in order to find the maximum value of the utility function $U_{j,m}(C_j)$, we may restrict the analysis of $U_{j,m}(C_j)$ for feasible completion times $C_j \geq d_j$. By Theorem 3, $U_{j,m}(C_j)$ is also piece-wise linear, and the maximum value $U_{j,m}^*$ is therefore reached for $C_j^* \in \{d_j\} \cup \{t_{j,m}^s \in T_{j,m} : t_{j,m}^s > d_j\}$.

Since $T_{j,m} = (\Theta_{j,m} \cup \mathcal{T}_m \backslash \{\tau_m^h \in \mathcal{T}_m : \tau_m^h \geq \theta_{j,m}^1\})$, the optimal completion time C_j^* can be determined in linear time with respect to the number of machines P_m plus the number of times in $\mathcal{T}_m$, that is, the number of tasks currently in execution on cluster m at time r_j. In particular, by definition, $T_{j,m}$ is the union of a subset of the times τ_m^h when the resource loading profile $R_m^{load}(t)$ of cluster m changes, and the set $\Theta_{j,m}$ of P_m times. Therefore, assuming that at the (current) time r_j when task j is submitted, the information about the resources of cluster m (i.e., the number P_m of machines of m, and their peak performance R_m^{∞}) is supplied by GSP of that cluster and stored in the GMD, and also the resource loading profile $R_m^{load}(t)$ of m (i.e., the set $\mathcal{T}_m$ of times τ_m^h when $R_m^{load}(t)$ changes, along with the values of $R_m^{load}(\tau_m^h)$) is known and supplied to the GMD, there is no need in Step 2.1 for the GRB to make an announcement (O_j, C_j) to the GPS of cluster m, for every $C_j \geq r_j$, but only for $C_j \in T_m$. Note that the set $T_{j,m}$ can be easily determined by GRB by interrogating the GMD where the information about the current status of the Grid resources is available: the GRB interrogates the GMD to obtain the set $\mathcal{T}_m$ of times when the resource loading profile data of cluster m changes, and to determine the set $\Theta_{j,m}$ of times $\theta_{j,m}^i$ on the basis of the task size O_j, the cluster resources, and the resource loading profile $R_m^{load}(t)$ of m stored in the GMD.

5 A simulation study

We experimentally evaluate the performance of the proposed economic model comparing it with the round-robin protocol. We consider two different scenarios for the Grid system: Scenario 1, that considers the case where tasks are mono-thematic applications and their requests are submitted to the same External Scheduler (GRB) that interacts with the Local Schedulers (GSPs) of clusters dedicated to that kind of applications. The second scenario, i.e., Scenario 2, considers enterogenous tasks and there are as many GRBs as many tasks. While in Scenario 1 there is a single GRB that interacts with the GSPs considering one task at a time according to a given task ordering (e.g., FIFO), in Scenario 2 there are many GRBs interacting at the same time with the GSPs. Therefore, in the latter scenario the GSP of a cluster may receive awards from many GRBs, and it will respond with an *acceptance* only to the award related to the most useful announcement for the cluster, and with a *refusal* to the other awards. In both the above described scenarios we use the following data set for the Grid simulation.

We consider a Grid system constituted by 10 clusters. Each cluster has 10 machines or resource units (processors), with the same speed equal to 400 MIPS (million instructions per second). For all the clusters, the minimum and maximum price of a machine per time unit (i.e. second) is 6 and 8 G$ (Grid $) per time unit, respectively. Tasks arrive according to a Poisson arrival process where λ is the average arrival rate (i.e., number of tasks per time unit). On average, 45% of the arriving tasks are *background* tasks, that is, tasks generated inside the clusters by the resource owners, and 55% are external tasks generated by the Grid users. Background tasks of a cluster have priority over external tasks submitted to that cluster, and they are scheduled immediately on the available resources of the cluster in order to be finished as earliest as possible. The size O_i of a task is equal to 10000 MI (million instructions) plus a uniformly generated number between $\pm$ 10% of 10000 MI. The due-date d_i of a task is equal to $r_i + run\_time + wait\_time$ plus a uniformly generated number between $\pm$ 10% of $(run\_time + wait\_time)$, where r_i is the task arrival date, $run\_time = 5$ time units is the expected task run time supposing that half of the computational resources of a cluster is allocated to the task, and $wait\_time$ is the allowed task waiting time. The budget B_i of a task is equal to 250 G$ (Grid $) plus a uniformly generated amount between $\pm$ 10% of 250 G$. Finally, the penalty cost (task weight) w_i for each time unit exceeding the task due date is equal to $B_i/(d_i - r_i - run\_time)$. The length of each simulation is 100 time units. During the first and last 10 time units no measurements are made to ensure the evaluation of the system at its steady state. We have experimented with different values of λ and $wait\_time$ parameters. In the following we report results with $\lambda = 1, 2, \ldots, 10$, and with a fixed value for $wait\_time = 5$ time units. Accordingly, the average number of tasks generated in each simulation is $100, 200, \ldots, 1000$ tasks, respectively. The simulator was coded in the C language and the time required to finish a simulation run is not greater that 1 second on a Pentium IV PC. Figure 1(a) shows the average cluster load (in percentage) due to background tasks, the total load with the economic model (ECO) both for Scenario 1 (ECO1) and Scenario 2 (ECO2), and with the Round-Robin (RR) protocol. For low congested cases ($\lambda \leq 3$), we have no significant difference between the results of ECO (in both the two simulated scenarios) and of RR, and the total average load reach 70% with $\lambda = 3$. For medium/high congested cases, and in particular with $\lambda \geq 4$, the overall load is greater than 70% in all the cases; nevertheless, while with RR it is always less than 90%, with ECO it riches 95.5% for $\lambda = 7$ (both in ECO1 and ECO2), with an improvement of more than 19% with respect to RR. For greater arrival rates (i.e., $\lambda \geq 8$) the dif-

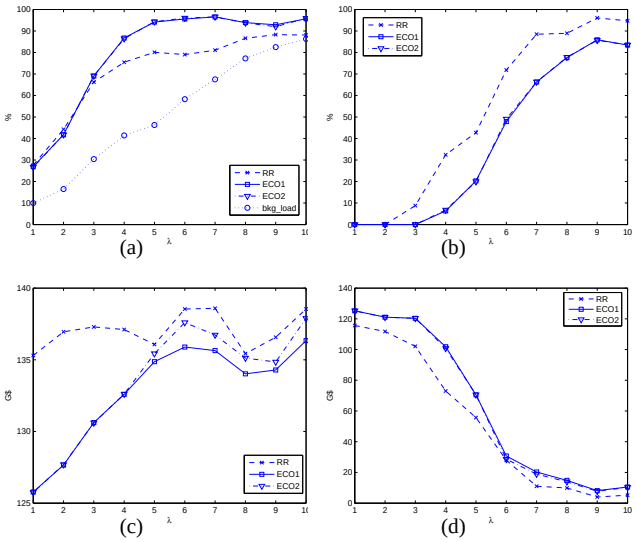

Figure 1: Computational results.

ference between the cluster load with ECO and with RR decreases. Nevertheless, for $\lambda \geq 8$ the Grid system becomes very high congested and more than 80% of the incoming tasks has been rejected as shown in Figure 1(b) where the ratio (in percentage) between the number of rejected tasks and the number of submitted tasks are plotted for different values of λ. In particular, Figure 1(b) shows that even for low congested cases (i.e., $\lambda \leq 4$) a significant amount of submitted tasks has been rejected by the Grid with RR (more than 32.4% with $\lambda = 4$), in opposition to a very small fraction (6.7%) with ECO. In medium/high congested case (e.g., $\lambda = 7$), RR rejects more than 88.5% tasks while with ECO the fraction of rejected tasks is non more than 66.3%. Also for very high congested cases ($\lambda \geq 8$) there is a significant gap between the fraction of rejected tasks with RR protocol and ECO model. Finally, there is a negligible difference in the performance of ECO comparing Scenarios 1 (ECO1) and 2 (ECO2), that shows a high level of robustness of the economic model. Figure 1(c) shows the trend of the average computational cost per scheduled task (among the scheduled tasks), as a function of task arrival rate λ. With RR protocol the task computational cost is almost independent from λ and on average equal to 137 G$; this is due to the logic of RR protocol where all the available resources of the selected cluster is assigned to the submitted task in order to finish it as earliest as possible. With ECO the average task computational cost increases with λ from 126 G$ (with $\lambda = 1$) to 138 G$ (with $\lambda = 10$, and Scenario 2 (ECO2)), and it is always less than the value obtained with RR. In particular, there is a non-negligible difference between the two experimented scenarios (see ECO1 and ECO2 curves) for $\lambda \geq 5$, with a greater computational cost in Scenario 2, where many GRBs interacts with each GSP, and hence each GSP accept the most profitable award for the GSP itself, resulting in a greater average profit for the resource owners than that of Scenario 1. Figure 1(d) shows the average utility of submitted task as a function of λ. Both scheduled and rejected tasks are considered in this evaluation, with the utility of rejected tasks fixed to zero, and the utility of scheduled tasks equal to the difference between the task budget (fixed to 250 G$ per task) and the task execution cost (computational cost plus penalty cost). The figure shows that task utility decreases with λ, but with ECO the average task utility is always greater than that with RR protocol in both the two evaluated scenarios. In particular there is a significant gap among task utilities obtained with ECO and RR in the medium/high congested case (i.e., λ between 3 and 5). Finally, we note that the average tardiness is very small (less than 1.5 time units).

References:

[1] R. Buyya, D. Abramson, J. Giddy and H. Stockinger, Economic Models for Resource Management and Scheduling in Grid Computing, Concurrency Computat.: Pract. Exper. 14 (2002) 1507–1542.

[2] H. Casanova and J. Dongarra, NetSolve: A Network Server for Solving Computational Science Problems, International Journal of Supercomputing Applications and High Performance Computing 11 (1997) 212–223.

[3] I. Foster and C. Kesselman, The Grid: blueprint for a new computing infrastructure (2nd edition), Morgan Kaufmann, 2004.

[4] N. Nisan, S. London, O. Regev and N. Camiel, Globally distributed computation over the Internet - the POPCORN project, Proceedings of the 18th International Conference on Distributed Computing Systems (ICDCS 1998), Amsterdam, The Netherlands, May 1998. IEEE Computer Society, 1998.

[5] K. Ranganathan and I. Foster, Decoupling computation and data scheduling in distributed data-intensive applications, Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing (HPDC-11), Edinburgh, Scotland, July 23-26, IEEE Computer Society (2002) 352–358.

[6] A. Su, F. Berman, R. Wolski and M.M. Strout, Using AppLeS to schedule simple SARA on the computational Grid, International Journal of High Performance Computing Application 13 (1999) 253–262.