Algorithmic Mechanism Design for Load Balancing in Distributed Systems

Daniel Grosu, *Student Member, IEEE,* and Anthony T. Chronopoulos, *Senior Member, IEEE*

***Abstract*—Computational grids are promising next-generation computing platforms for large-scale problems in science and engineering. Grids are large-scale computing systems composed of geographically distributed resources (computers, storage etc.) owned by self interested agents or organizations. These agents may manipulate the resource allocation algorithm in their own benefit, and their selfish behavior may lead to severe performance degradation and poor efficiency.**

In this paper, we investigate the problem of designing protocols for resource allocation involving selfish agents. Solving this kind of problems is the object of mechanism design theory. Using this theory, we design a truthful mechanism for solving the static load balancing problem in heterogeneous distributed systems. We prove that using the optimal allocation algorithm the output function admits a truthful payment scheme satisfying voluntary participation. We derive a protocol that implements our mechanism and present experiments to show its effectiveness.

***Index Terms*—Distributed systems, game theory, mechanism design, static load balancing.**

I. Introduction

In CURRENT distributed systems such as computational grids, resources belong to different self interested agents or organizations. These agents may manipulate the load allocation algorithm in their own benefit, and their selfish behavior may lead to severe performance degradation and poor efficiency. Solving such problems involving selfish agents is the object of *mechanism design theory* (also called *implementation theory*) [19]. This theory helps design protocols in which the agents are always forced to tell the truth and follow the rules. Such mechanisms are called *truthful* or *strategy-proof*.

Each participating agent has a privately known function called *valuation* which quantifies the agent benefit or loss. The valuation depends on the outcome and is reported to a centralized mechanism. The mechanism chooses an outcome that maximizes a given objective function and makes payments to the agents. The valuations and payments are expressed in some common unit of currency. The payments are designed and used to motivate the agents to report their true valuations. Reporting the true valuations leads to an optimal value for the objective function. The goal of each agent is to maximize the sum of her valuation and payment.

In this paper, we consider the mechanism design problem for load balancing in distributed systems. A general formulation of the *load balancing problem* is as follows: Given a large number of jobs, find the allocation of jobs to computers optimizing a given objective function (e.g., total execution time).

There are three typical approaches to load balancing problem in distributed systems.

1) *Global approach*: In this case, there is only one decision maker that optimizes the response time of the entire system over all jobs and the operating point is called *social optimum*. This is the classical approach and has been studied extensively using different techniques such as nonlinear optimization [21], [22] and polymatroid optimization [20].
2) *Cooperative approach*: In this case, there are several decision makers (e.g., jobs, computers) that cooperate in making the decisions such that each of them will operate at its optimum. Decision makers have complete freedom of preplay communication to make joint agreements about their operating points. This situation can be modeled as a cooperative game and game theory offers a suitable modeling framework [8].
3) *Noncooperative approach*: In this case, each of infinitely many jobs optimizes its own response time independently of the others, and they all eventually reach an equilibrium. This situation can be viewed as a noncooperative game among jobs. The equilibrium is called *Wardrop equilibrium* [12]. At the Wardrop equilibrium, a job cannot receive any further benefit by changing its own decision. If the number of jobs are finite, the Wardrop equilibrium reduces to the well-known *Nash equilibrium* [19].

Our goal is to design a mechanism that uses the optimal load balancing algorithm. The optimal algorithm belongs to the global approach in the classification presented above. To design our mechanism we use the framework derived by Archer and Tardos in [1]. We assume that each computer in the distributed system is characterized by its processing rate, and only computer i knows the true value of its processing rate. Jobs arrive at the system with a given arrival rate. The optimal algorithm finds the fraction of load that is allocated to each computer such that the expected execution time is minimized. The *cost* incurred by each computer is proportional to its utilization. The mechanism will ask each agent (computer) to report its processing rate and then compute the allocation using the optimal algorithm. After computing the allocation the mechanism hands payments to computers.

Manuscript received June 11, 2002. This work was supported in part by research grants from NASA NAG 2-1383 (1999–2001), the State of Texas Higher Education Coordinating Board under the Texas Advanced Research/Advanced Technology Program ATP 003658-0442-1999, and the Center for Infrastructure and Security at the University of Texas at San Antonio. This paper was recommended by Associate Editor S. Phoha.

The authors are with the Department of Computer Science, University of Texas at San Antonio, San Antonio, TX 78249 USA (e-mail: dgrosu@cs.utsa.edu; atc@cs.utsa.edu).

Digital Object Identifier 10.1109/TSMCB.2002.805812

Each computer goal is to chose a processing rate to report to the mechanism such that its profit is maximized. The *profit* is the difference between the payment handed by the mechanism and the true cost of processing the allocated jobs. The payments handed by the mechanism must motivate the computers to report their true value such that the expected response time of the entire system is minimized. Thus we need to design a payment function that guarantees this property.

Related Work: Recently, with the emergence of the Internet as a global platform for computation and communication, the need for efficient protocols that deals with self-interested agents has increased. This motivated the use of mechanism design theory in different settings such as market-based protocols for resource allocation in computational grids [3], [25], market-based protocols for scheduling [24], congestion control [13], routing [5], and mechanisms for trading CPU time [16]. A recent survey on distributed algorithmic mechanism design is [7]. For an introduction to general mechanism design theory, see [19].

The most important result in mechanism design theory is the Vickrey–Clarke–Groves (VCG) mechanism [4], [9], [23]. The VCG mechanism allows arbitrary form for valuations and its applicability is restricted to utilitarian objective functions (i.e., the objective function is the sum of agents' valuations).

Nisan and Ronen [17] studied the mechanism design problem for several standard problems in computer science. Using the VCG mechanism, they solved the shortest path problem in a graph where each edge belongs to a different agent. For scheduling on unrelated machines, they designed an n-approximation truthful mechanism, where n is the number of agents. They proved a lower bound of 2 to the approximation ratio of any mechanism for this problem and conjectured that no truthful mechanism can yield an approximation ratio better than n. They also gave a mechanism that solves exactly the problem of scheduling jobs on unrelated machines in a model where the mechanism has more information. In this extended model, the payments are given after jobs' execution allowing the mechanism to verify the agent's declarations and penalize them for lying.

The computational feasibility of VCG mechanisms is addressed in [18]. The authors proved that all reasonable approximations or heuristics for a wide class of minimization problems yield nontruthful VCG-based mechanisms (i.e., mechanisms that use the VCG payment scheme and a suboptimal allocation algorithm). They showed that under reasonable assumption any VCG-based mechanism can be made feasible truthful.

A polynomial VCG mechanism for shortest paths is proposed in [10]. Feigenbaum *et al.* [6] studied two mechanisms for cost-sharing in multicast transmissions. Frugality of shortest paths mechanisms is investigated in [2].

Archer and Tardos [1] applied mechanism design theory to several combinatorial optimization problems where agent's secret data is represented by a single real valued parameter. They provided a method to design mechanisms for general objective functions and restricted form for valuations. For scheduling related parallel machines, they gave a three-approximation algorithm and used it to design a truthful mechanism. They also gave truthful mechanisms for maximum flow, scheduling related machines to minimize the sum of completion times, optimizing an affine function and special cases of uncapacitated facility location.

Our Contributions: We design a truthful mechanism for solving the static load balancing problem in distributed systems. We prove that using the optimal allocation algorithm the output function admits a truthful payment scheme satisfying voluntary participation. We derive a protocol that implements our mechanism and present experiments to show its effectiveness.

Organization: The paper is structured as follows. In Section II, we present the mechanism design terminology. In Section III, we present our distributed system model. In Section IV, we design a truthful mechanism for load balancing in distributed systems. In Section V, the effectiveness of our load balancing mechanism is investigated. In Section VI, we draw conclusions and present future directions.

II. Mechanism Design Concepts

In this section, we introduce some important mechanism design concepts. We limit our description to mechanism design problems for one parameter agents. In this type of mechanism design problems each agent has some private data represented by a single real valued parameter [1]. In the following, we define such problem.

Definition 2.1 (Mechanism Design Problem): A *mechanism design problem for one parameter agents* is characterized by the following.

i) A finite set Λ of allowed outputs. The output is a vector $\lambda(b) = (\lambda_1(b), \lambda_2(b), \ldots, \lambda_n(b))$, $\lambda(b) \in \Lambda$, computed according to the agents' bids $b = (b_1, b_2, \ldots, b_n)$. Here, b_i is the value (bid) reported by agent i to the mechanism.

ii) Each agent i, $(i = 1, \ldots, n)$ has a privately known parameter t_i called her *true value*. The cost incurred by each agent depends on the output and on her true value and is denoted as $cost_i(t_i, \lambda(b))$.

iii) Each agent goal is to maximize her profit. The profit of agent i is $profit_i(t_i, b) = P_i(b) - cost_i(t_i, \lambda(b))$, where P_i is the payment handed by the mechanism to agent i.

iv) The goal of the mechanism is to select an output λ that optimizes a given cost function $g(t, \lambda)$.

We assume that the cost functions have the following particular form: $cost_i(t_i, \lambda(b)) = t_i \lambda_i(b)$, i.e., t_i represents the cost per unit load.

Definition 2.2 (Mechanism): A *mechanism* is characterized by two functions:

i) The *output function* $\lambda(b) = (\lambda_1(b), \lambda_2(b), \ldots, \lambda_n(b))$. This function has as input the vector of agents' bids $b = (b_1, b_2, \ldots, b_n)$ and returns an output $\lambda \in \Lambda$.

ii) The *payment function* $P(b) = (P_1(b), P_2(b), \ldots, P_n(b))$ that gives the payment handed by the mechanism to each agent.

Notation: In the rest of the paper, we denote by b_{-i} the vector of bids, not including the bid of agent i. The vector b is represented as (b_{-i}, b_i).

Definition 2.3 (Truthful Mechanism): A mechanism is called *truthful* if for every agent i of type t_i and for every bids b_{-i} of the

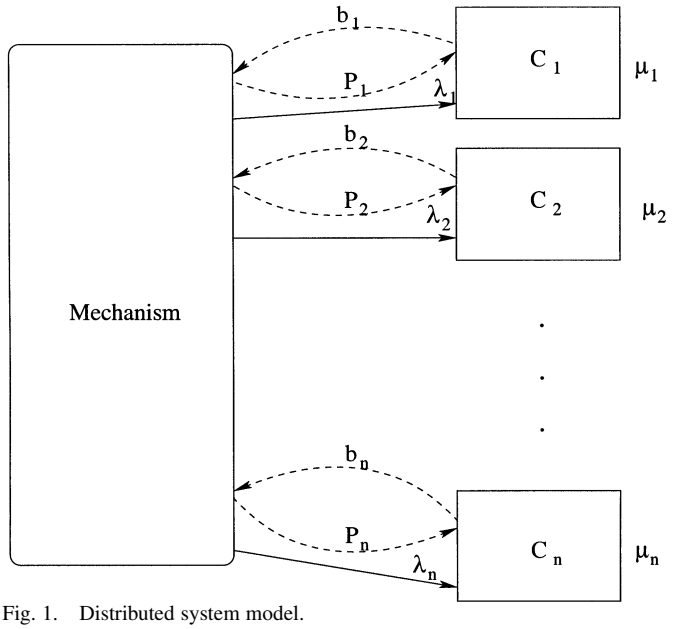

Fig. 1. Distributed system model.

other agents, the agent's profit is maximized when she declares her real type t_i. (i.e., truth-telling is a dominant strategy).

Definition 2.4 (Truthful Payment Scheme): We say that an output function admits a *truthful payment scheme* if there exists a payment function P such that the mechanism is truthful.

A desirable property of a mechanism is that the profit of a truthful agent is always non-negative. The agents hope for a profit by participating in the mechanism.

Definition 2.5 (Voluntary Participation Mechanism): We say that a mechanism satisfies the *voluntary participation condition* if $profit_i(t_i, (b_{-i}, t_i)) \geq 0$ for every agent i, true values t_i, and other agents' bids b_{-i} (i.e., truthful agents never incur a loss).

III. Distributed System Model

We consider a distributed system that consists of n heterogeneous computers connected by a communication network. Computers have the same processing capabilities in the sense that a job may be processed from start to finish at any computer in the system. We assume that each computer is modeled as an M/M/1 queueing system (i.e., Poisson arrivals and exponentially distributed processing times) [14] and is characterized by its *average processing rate* μ_i, $i = 1, \ldots, n$. Jobs are generated by users and arrive at the system according to a time invariant Poisson process with average rate Φ. We call Φ the *total job arrival rate* in the system. The total job arrival rate must be less than the aggregate processing rate of the system (i.e., $\Phi < \sum_{i=1}^{n} \mu_i$). The system model is presented in Fig. 1. The system has to decide on how to distribute jobs to computers such that it will operate optimally. We assume that the decision to distribute jobs to computers is *static* i.e., it does not depend on the current state of the system. Thus we need to find the *load* λ_i that is assigned to computer i ($i = 1, \ldots, n$) such that the expected response time of all jobs is minimized. The expected response time at computer i is given by

$$F_i(\lambda_i) = \frac{1}{\mu_i - \lambda_i}. \tag{1}$$

Thus, the overall expected response time is given by

$$D(\lambda) = \frac{1}{\Phi} \sum_{i=1}^{n} \lambda_i F_i(\lambda_i) = \frac{1}{\Phi} \sum_{i=1}^{n} \frac{\lambda_i}{\mu_i - \lambda_i} \tag{2}$$

where $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_n)$ is the vector of loads assigned to computers.

We assume that computers are agents and each of them has a *true value* t_i represented by the inverse of its processing rate $t_i = 1/\mu_i$. Only computer i knows t_i. The mechanism will ask each computer i to report its value b_i (the inverse of its processing rate). The computers may not report the true value. After all the computers report their values the mechanism computes an output function (i.e., the loads assigned to computers), $\lambda(b) = (\lambda_1(b), \lambda_2(b), \ldots, \lambda_n(b))$, according to the agents' bids such that the overall expected execution time is minimized. The mechanism also hands a payment $P_i(b)$ to each computer. All computers know the algorithm used to compute the output function (allocation) and the payment scheme.

Each computer incurs some cost:

$$\text{cost}_i(t_i, \lambda(b)) = t_i \lambda_i(b). \tag{3}$$

The cost is equivalent to computer utilization. The greater the utilization, the greater the cost. We assume each computer wants to choose its strategy (what value b_i to report) such that its profit is maximized. The profit for each computer is defined as the payment received from the mechanism minus the cost incurred in running the jobs allocated to it:

$$\text{profit}_i(t_i, b) = P_i(b) - \text{cost}_i(t_i, \lambda(b)). \tag{4}$$

Our goal is to design a truthful mechanism that minimizes the overall expected response time of the system. This involves finding an allocation algorithm and a payment scheme that minimizes the overall expected response time according to the computer bids b_i and motivates all the computers to report their true values t_i.

IV. Designing the Mechanism

To design our load balancing mechanism we use the framework proposed by Archer and Tardos in [1]. They provided a method to design mechanisms where each agent's true data is represented by a single real valued parameter. According to this method, to obtain a truthful mechanism we must find an output function satisfying two conditions: a) It minimizes $D(\lambda)$, and b) it is decreasing in the bids. In addition, we want a mechanism satisfying voluntary participation. To guarantee this property we must find a payment function satisfying voluntary participation.

First, we are interested in finding an output function $\lambda(b)$ that minimizes the expected execution time over all jobs $D(\lambda)$ and produces a feasible allocation. Then we will show that this output function is decreasing in the bids.

Definition 4.1 (Feasible Allocation): A *feasible allocation* $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_n)$ is a load allocation that satisfies the following conditions:

i) Positivity: $\lambda_i \geq 0$, $i = 1, \ldots, n$;
ii) Conservation: $\sum_{i=1}^{n} \lambda_i = \Phi$;
iii) Stability: $\lambda_i < \mu_i$, $i = 1, \ldots, n$. □

The optimal load allocation can be obtained solving the following nonlinear optimization problem:

$$\min_{\lambda} D(\lambda) \tag{5}$$

subject to the constraints defined by the conditions i)-iii).

Thus, obtaining the solution to this problem involves minimizing the convex function $D(\lambda)$ over a convex feasible region defined by the conditions i)-iii). In this case, the first order Kuhn–Tucker conditions are necessary and sufficient for optimality [15].

Let $\alpha \geq 0, \eta_i \geq 0, i = 1, \ldots, n$ denote the Lagrange multipliers [15]. The Lagrangian function is

$$L(\lambda_1, \lambda_2, \ldots, \lambda_n, \alpha, \eta_1, \ldots, \eta_n) = \sum_{i=1}^{n} \lambda_i F_i(\lambda_i) - \alpha \left(\sum_{i=1}^{n} \lambda_i - \Phi \right) - \sum_{i=1}^{n} \eta_i \lambda_i. \tag{6}$$

The Kuhn–Tucker conditions imply that $\lambda_i, i = 1, \ldots, n$ is the optimal solution to our problem if and only if there exists $\alpha \geq 0, \eta_i \geq 0, i = 1, \ldots, n$ such that

$$\frac{\partial L}{\partial \lambda_i} = 0 \tag{7}$$

$$\frac{\partial L}{\partial \alpha} = 0 \tag{8}$$

$$\eta_i \lambda_i = 0, \quad \eta_i \geq 0, \quad \lambda_i \geq 0, \qquad i = 1, \ldots, n. \tag{9}$$

These conditions become

$$\lambda_i F_i'(\lambda_i) + F_i(\lambda_i) - \alpha - \eta_i = 0, \qquad i = 1, \ldots, n \tag{10}$$

$$\sum_{i=1}^{n} \lambda_i = \Phi \tag{11}$$

$$\eta_i \lambda_i = 0, \quad \eta_i \geq 0, \quad \lambda_i \geq 0, \qquad i = 1, \ldots, n. \tag{12}$$

These are equivalent to

$$\alpha = \lambda_i F_i'(\lambda_i) + F_i(\lambda_i), \text{ if } \lambda_i > 0 \quad 1 \leq i \leq n \tag{13}$$

$$\alpha \leq \lambda_i F_i'(\lambda_i) + F_i(\lambda_i), \text{ if } \lambda_i = 0 \quad 1 \leq i \leq n \tag{14}$$

$$\sum_{i=1}^{n} \lambda_i = \Phi, \quad \lambda_i \geq 0, \quad i = 1, \ldots, n. \tag{15}$$

By solving these conditions, one can obtain the optimal algorithm but this is not our focus in this paper. Algorithms for this kind of optimization problem were proposed in the past [21], [22]. For the clarity of presentation, we describe a variant of the algorithm in [21] using our notations. Our derivation of this algorithm is different from their approach and was partially presented above because some of the equations will be used to prove our results in Theorems 4.1 and 4.2 below. We now present the algorithm.

OPTIM algorithm:
Input: Bids $b_1, b_2, \ldots b_n$ submitted by computers
Total arrival rate: Φ
Output: Load allocation: $\lambda_1, \lambda_2, \ldots \lambda_n$;
1. Sort the computers in increasing order of their bids
($b_1 \leq b_2 \leq \cdots \leq b_n$);
2. $c \leftarrow (\sum_{i=1}^{n} 1/b_i - \Phi)/(\sum_{i=1}^{n} \sqrt{1/b_i})$;
3. **while** ($c > \sqrt{1/b_n}$) **do**
$\lambda_n \leftarrow 0$;
$n \leftarrow n - 1$;
$c \leftarrow (\sum_{i=1}^{n} 1/b_i - \Phi)/(\sum_{i=1}^{n} \sqrt{1/b_i})$;
4. **for** $i = 1, \ldots, n$ **do**
$\lambda_i \leftarrow 1/b_i - c\sqrt{1/b_i}$;

Remark: In step 2, the coefficient c is computed. This coefficient is used in step 3 (while-loop) to determine which computers have slow processing rates. If $c > \sqrt{1/b_n}$, then computer C_n is too slow and will not be used for the allocation (i.e., $\lambda_n = 0$). The number of computers n is decremented by 1, and a new c is computed. Step 4 computes the loads λ_i that will be allocated to each computer.

This algorithm computes an allocation $\lambda(b) = (\lambda_1(b), \lambda_2(b), \ldots, \lambda_n(b))$ that provides the overall optimum for the expected execution time. This optimum is obtained according to the bids reported by computers. If some of the computers declare values different than their true values ($t_i = 1/\mu_i$), this optimum may not be the same as the "true optimum" obtained when all the computers declare their true values. Therefore, if some of the computers lie, we expect worse performance (i.e., higher overall expected execution time).

We found an output function (given by OPTIM) that minimizes $D(\lambda)$. Now, we must prove that this output function admits a truthful mechanism. In the following we state and prove two theorems: i) that the output function is decreasing in the bids (thus guaranteeing truthfulness) and ii) that our mechanism admits a truthful payment scheme satisfying voluntary participation.

Definition 4.2 (Decreasing Output Function): An output function $\lambda(b) = (\lambda_1(b), \lambda_2(b), \ldots, \lambda_n(b))$ is *decreasing* if each $\lambda_i(b_{-i}, b_i)$ is a decreasing function of b_i for all i and b_{-i}.

Theorem 4.1: The output function $\lambda(b)$ computed by the optimal algorithm is decreasing.

Proof: See the Appendix .

Theorem 4.2: The output function $\lambda(b)$ computed by the optimal algorithm admits a *truthful* payment scheme satisfying *voluntary participation* and the payment for each computer i ($i = 1, 2, \ldots, n$) is given by

$$P_i(b_{-i}, b_i) = b_i \lambda_i(b_{-i}, b_i) + \int_{b_i}^{\infty} \lambda_i(b_{-i}, x) dx \tag{16}$$

Proof: See the Appendix .

Remarks: Analogously to [1], we obtained $P_i(b_{-i}, b_i)$ for our mechanism. The first term $b_i \lambda_i(b_{-i}, b_i)$ of the payment function in (16) compensates the cost incurred by computer

i. The second term $\int_{b_i}^{\infty} \lambda_i(b_{-i}, x)dx$ represents the expected profit of computer i. If computer i bids its true value t_i, then its profit P_t is

$$\begin{aligned} P_t =& profit_i(t_i, (b_{-i}, t_i)) \\ =& t_i\lambda_i(b_{-i}, t_i) + \int_{t_i}^{\infty} \lambda_i(b_{-i}, x)dx - t_i\lambda_i(b_{-i}, t_i) \\ =& \int_{t_i}^{\infty} \lambda_i(b_{-i}, x)dx. \end{aligned}$$

If computer i bids its true value, then the expected profit is greater than in the case it bids other values. We can explain this as follows. If computer i bids higher ($b_i^h > t_i$), then the expected profit P_h is

$$\begin{aligned} P_h =& profit_i(t_i, (b_{-i}, b_i^h)) \\ =& b_i^h\lambda_i(b_{-i}, b_i^h) + \int_{b_i^h}^{\infty} \lambda_i(b_{-i}, x)dx - t_i\lambda_i(b_{-i}, b_i^h) = \\ =& (b_i^h - t_i)\lambda_i(b_{-i}, b_i^h) + \int_{b_i^h}^{\infty} \lambda_i(b_{-i}, x)dx. \end{aligned}$$

Because $\int_{b_i}^{\infty} \lambda_i(b_{-i}, x)dx < \infty$ and $b_i^h > t_i$, we can express the profit when computer i bids the true value as follows:

$$P_t = \int_{t_i}^{b_i^h} \lambda_i(b_{-i}, x)dx + \int_{b_i^h}^{\infty} \lambda_i(b_{-i}, x)dx.$$

Because λ_i is decreasing in b_i and $b_i^h > t_i$, we have the following equation:

$$(b_i^h - t_i)\lambda_i(b_{-i}, b_i^h) < \int_{t_i}^{b_i^h} \lambda_i(b_{-i}, x)dx.$$

From this relation, it can be seen that $P_h < P_t$. The same argument applies to the case when computer i bids lower.

Because the optimal algorithm assumes a central dispatcher, the mechanism will be implemented in a centralized way as part of the dispatcher code. We assume that the dispatcher is run on one of the computers and is able to communicate with all the other computers in the distributed system.

In the following, we present the protocol that implements our load balancing mechanism (LBM). This protocol has two phases: bidding and completion.

Protocol LBM:

Phase I: Bidding

1) The dispatcher sends a request for bids message (*ReqBid*) to each computer in the system.
2) When a computer receives a *ReqBid*, it replies with its bid b_i to the dispatcher.

Phase II: Completion

1) After the dispatcher collects all the bids, it does the following:
    1) Computes the allocation using OPTIM algorithm.
    2) Computes the payments P_i for each computer using (16).
    3) Sends P_i to each computer i.
2) Each computer receives its payment and evaluates its profit.

This protocol is executed periodically or when there is a change in the total job arrival rate. During two executions of this protocol, the jobs will be allocated to computers by the dispatcher according to the allocation computed by OPTIM. Computers will receive the maximum profit only when they report the true value.

TABLE I
SYSTEM CONFIGURATION

Relative processing rate	1	2	5	10
Number of computers	6	5	3	2
Processing rate (jobs/sec)	0.013	0.026	0.065	0.13

V. EXPERIMENTAL RESULTS

To study the effectiveness of our truthful mechanism, we simulated a heterogeneous system consisting of 16 computers with four different processing rates. In Table I, we present the system configuration. The first row contains the relative processing rates of each of the four computer types. Here, the relative processing rate for computer C_i is defined as the ratio of the processing rate of C_i to the processing rate of the slowest computer in the system. The second row contains the number of computers in the system corresponding to each computer type. The last row shows the processing rate of each computer type in the system. We choose 0.013 jobs/s as the processing rate for the slowest computer because it is a value that can be found in real distributed systems [21]. In addition, we consider only computers that are at most ten times faster than the slowest because this is the case in most of the current heterogeneous distributed systems.

We study the performance degradation due to false bids declarations. We define *performance degradation (PD)* as follows: $PD = ((D_F - D_T)/D_T)100\%$, where D_F is the response time of the system when one or more computers report false values; and D_T is the "true" optimal response time of the system when all computers report their true values. *PD* quantifies the increase in the execution time due to false bidding. As we pointed out in the previous section, if all the computers report their true values, then the "true" optimum is obtained and $PD = 0$. We expect an increase in the response time and in *PD* when one or more computers lie.

In our experiments we consider that the fastest computer C_1 declares false bids. C_1 has $t_1 = 1/\mu_1 = 7.69$ as its true value. In Fig. 2, we present the degradation in expected response time of the system for different values of system utilization (ranging from 10% to 90%) and two types of bidding: overbidding and underbidding. *System utilization* (ρ) is defined as the ratio of total arrival rate to aggregate processing rate of the system: $\rho = \Phi/(\sum_{i=1}^{n} \mu_i)$. In the first experiment, C_1 bids 7% lower than the true value. In this case the performance degradation is around 2% for low and medium system utilization, increasing drastically (around 300%) for high system utilization. This increase is due to computer C_1 overloading. The overloading occurs because C_1 bids lower, which means that it reports a higher value for its processing rate. The algorithm will allocate more jobs to C_1, increasing its response time. This increase is reflected in the expected response time of the system. In the second experiment,

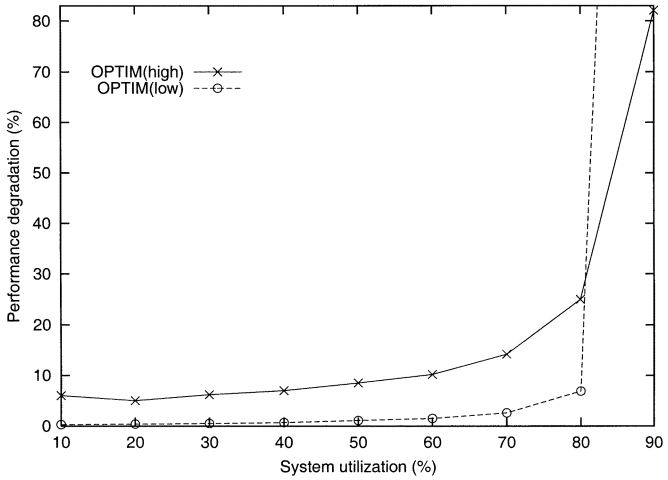

Fig. 2. Performance degradation versus system utilization.

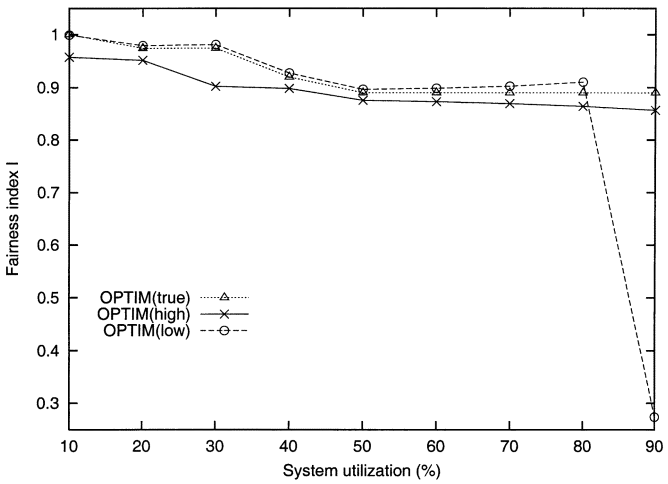

Fig. 3. Fairness index versus system utilization.

C_1 bids 33% higher than the true value. In this case the performance degradation is about 6% at low system utilization, about 15% at medium system utilization, and more than 80% at high system utilization. It can be observed that small deviations from the true value of only one computer may lead to large values of performance degradation. If we consider that more than one computer does not report its true value then we expect very poor performance. This justifies the need for a mechanism that will force the computers to declare their true values. In Fig. 3, we present the variations in the fairness index. The *fairness index*

$$I = \frac{\left[\sum_{i=1}^{n} F_i\right]^2}{n \sum_{i=1}^{n} F_i^2} \tag{17}$$

was proposed in [11] to quantify the fairness of load balancing schemes. This index is a measure of the "equality" of execution times at different computers. Therefore, it is a measure of load balance. If all the computers have the same expected job execution times, then $I = 1$, and the system is 100% fair to all jobs and is load balanced. If the differences on F_i increase, I decreases, and the load balancing scheme favors only some tasks.

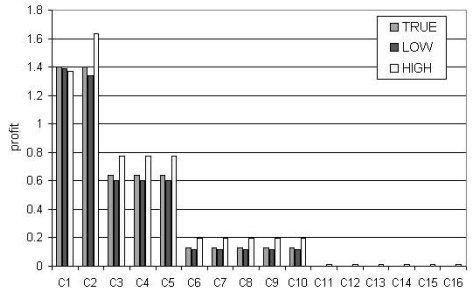

Fig. 4. Profit for each computer (medium system load).

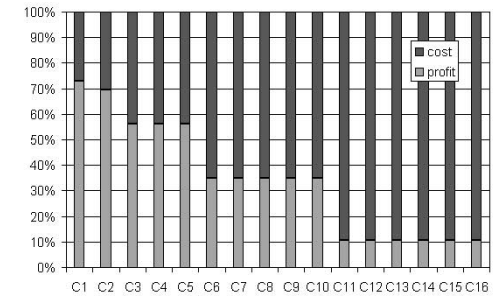

Fig. 5. Payment structure for each computer (C_1 bids higher).

It can be observed that in the case of underbidding the fairness index decreases drastically at high system loads. This is because C_1's response time is much higher than that of the other computers. The fairness index is maintained around 90% for all other values. If more than one computer does not report its true value, then we expect small variations in the fairness index. This can also be seen from the definition of this index.

In Fig. 4, we present the profit for each computer at medium system loads ($\rho = 50\%$). It can be observed that the profit at C_1 is maximum if it bids the true value, 3% lower if it bids higher and 1% lower if it bids lower. The mechanism penalizes C_1 if it does not report the true value. When C_1 bids lower, the other computer's profits are lower because their payments decrease. Computers C_{11} to C_{16} are not utilized when C_1 underbids and when it reports the true value; therefore, they gain nothing ($profit_i = 0$, $i = 11, 12, \ldots, 16$). These computers will be utilized in the case when C_1 overbids, getting a small profit. When C_1 overbids, the profit for all the computers except C_1 is higher than in the case when C_1 bids the true value. This is because the payments increase for these computers.

An important issue is the *frugality* of our mechanism. We say that a mechanism is *frugal* if the mechanism's payments are small by some measure [2]. This property gives us an idea of how efficient a mechanism is. The mechanism is interested in keeping its payments as small as possible. Each payment consists of an execution cost plus a profit for the computer which runs the jobs. Our mechanism must preserve voluntary participation, so the lower bound on its payments is the total cost incurred by the computers.

In Figs. 5 and 6, we present the cost and profit as fractions of the payment received by each computer at medium loads. In these figures, we are interested to see how close to the cost

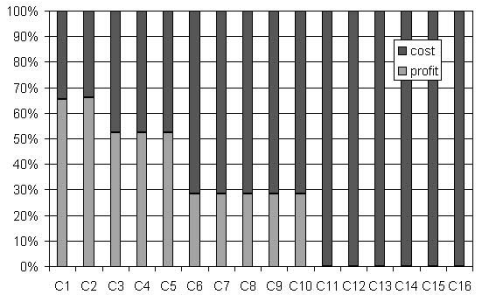

Fig. 6. Payment structure for each computer (C_1 bids lower).

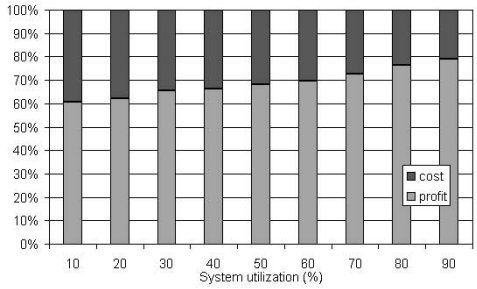

Fig. 7. Total payment versus system utilization.

is the payment to each computer. It can be observed that the cost incurred by C_1 when it bids higher is about 25% of the payment. In the case when C_1 bids lower, its cost is about 35% of the payment. For the other computers, the cost is between 50 and 90% when C_1 bids higher and between 53 and 100% when C_1 bids lower. For the distributed system considered in these experiments (medium loads) the highest payment given to a computer is about three times its cost.

In Fig. 7, we present the total cost and profit as fractions of the total payment for different values of system utilization when C_1 reports its true value. The total cost is about 21% of the payment at 90% system utilization, which is the smallest percentage. The percentage of cost increases to 40% at 10% system utilization. At medium loads the mechanism pays less than three times the total cost. When C_1 bids lower and higher, the percentages are similar and are not presented here. We expect that these values are also valid considering other parameters of the distributed system.

VI. Conclusion

In current distributed systems such as computational grids, resources belong to different self interested agents or organizations. These agents may manipulate the load allocation algorithm in their own benefit, and their selfish behavior may lead to severe performance degradation and poor efficiency.

In this paper, we investigated the problem of designing protocols for resource allocation involving selfish agents. Solving this kind of problems is the object of mechanism design theory. Using this theory, we designed a truthful mechanism for solving the load balancing problem in heterogeneous distributed systems. We proved that using the optimal allocation algorithm the output function admits a truthful payment scheme satisfying voluntary participation. We derived a protocol that implements our mechanism and presented experiments to show its effectiveness.

Future work will address the implementation of our load balancing mechanism in a real distributed system as well as the design of a distributed load balancing mechanism.

Appendix

In this section, we present the proofs of the results used in the paper.

A. Proof for Theorem 4.1

We fix the other bids and consider $\lambda_i(b_{-i}, b_i)$ as a single variable function of b_i. Let $\tilde{b}_i$ and b_i be any two bids such that $\tilde{b}_i > b_i$. In terms of processing rates, we have $1/\tilde{\mu}_i > 1/\mu_i$ i.e., $\tilde{\mu}_i < \mu_i$. Let $\tilde{\lambda}_i > 0$ and $\lambda_i > 0$ be the loads allocated by the optimal algorithm when computer i bids $\tilde{b}_i$ and b_i, respectively. We must prove that $\tilde{\lambda}_i < \lambda_i$, i.e., the allocation function computed by the optimal algorithm is decreasing in b_i.

Assume by contradiction that $\tilde{\lambda}_i \geq \lambda_i$. This implies $1/(\mu_i - \lambda_i) < 1/(\tilde{\mu}_i - \lambda_i) \leq 1/(\tilde{\mu}_i - \tilde{\lambda}_i)$. This means that $\tilde{F}_i > F_i$. Since $\tilde{\lambda}_i > 0$ is higher than λ_i and $\sum_{i=1}^{n} \lambda_i = \Phi$, there must be a computer l such that $\tilde{\lambda}_l \leq \lambda_l$, $\lambda_l > 0$. Since $\tilde{\lambda}_i \geq \lambda_i > 0$ the Kuhn–Tucker conditions for optimality (13), (14) imply that

$$\lambda_i F_i' + F_i = \alpha \tag{18}$$

$$\tilde{\lambda}_i \tilde{F}_i' + \tilde{F}_i = \tilde{\alpha}. \tag{19}$$

Since $\lambda_l \geq \tilde{\lambda}_l$ and $\lambda_l > 0$, the Kuhn–Tucker conditions for optimality in (13) and (14) imply that

$$\lambda_l F_l' + F_l = \alpha \tag{20}$$

$$\tilde{\lambda}_l \tilde{F}_l' + \tilde{F}_l \geq \tilde{\alpha}. \tag{21}$$

Combining (18)–(21), we obtain

$$\lambda_i F_i' + F_i = \lambda_l F_l' + F_l \tag{22}$$

$$\tilde{\lambda}_l \tilde{F}_l' + \tilde{F}_l \geq \tilde{\lambda}_i \tilde{F}_i' + \tilde{F}_i. \tag{23}$$

Because $\tilde{\lambda}_l \leq \lambda_l$, we have $1/(\mu_l - \tilde{\lambda}_l) \leq 1/(\mu_l - \lambda_l)$. This implies $\tilde{F}_l \leq F_l$. Also, using $\tilde{\lambda}_l \leq \lambda_l$, we obtain the following equation:

$$\lambda_i F_i' + F_i \geq \tilde{\lambda}_i \tilde{F}_i' + \tilde{F}_i. \tag{24}$$

This is a contradiction because $\tilde{\lambda}_i \geq \lambda_i$ and $\tilde{F}_i > F_i$. □

B. Proof of Theorem 4.2

We use the result of Archer and Tardos [1] that states that if the output function is decreasing in the bids then it admits a truthful payment scheme. We proved in Theorem 4.1 that the load function $\lambda(b)$ is decreasing in the bids; therefore, it admits a truthful mechanism.

We next use another result from [1], stating that if the area under the work curve is finite, the mechanism admits voluntary participation. For feasible bids, the area under the work curve is finite, i.e.,

$$\int_{b_{i\,\min}}^{\infty} \lambda_i(b_{-i}, u)du < \infty$$

where $b_{i\min}$ is the bid that corresponds to $\lambda_i = \Phi$. Thus, our mechanism admits voluntary participation and the payments are given by (16). □

ACKNOWLEDGMENT

The authors wish to express their thanks to the editor and the anonymous referees for their helpful and constructive suggestions, which considerably improved the quality of the paper.

REFERENCES

[1] A. Archer and E. Tardos, "Truthful mechanism for one-parameter agents," *Proc. 42nd IEEE Symp. Foundations Comput. Sci.*, pp. 482–491, Oct. 2001.

[2] ——, "Frugal path mechanisms," in *Proc.13th Annu. ACM-SIAM Symp. Discrete Algorithms*, Jan. 2002, pp. 991–999.

[3] R. Buyya, D. Abramson, and J. Giddy, "A case for economy grid architecture for service-oriented grid computing," *Proc. 10th IEEE Heterogeneous Comput. Workshop*, pp. 776–790, Apr. 2001.

[4] E. Clarke, "Multipart pricing of public goods," *Public Choice*, vol. 15, pp. 17–33, 1971.

[5] J. Feigenbaum, C. Papadimitriou, and S. Shenker, "A BGP-based mechanism for lowest-cost routing," in *Proc. 21st ACM Symp. Principles Distributed Comput.*, July 2002, pp. 173–182.

[6] J. Feigenbaum, C. Papadimitriou, R. Sami, and S. Shenker, "Sharing the cost of multicast transmissions," in *Proc. 32nd Annu. ACM Symp. Theory Comput.*, May 2000, pp. 218–227.

[7] J. Feigenbaum and S. Shenker, "Distributed algorithmic mechanism design: Recent results and future directions," in *Proc. ACM Workshop Discrete Algorithms Methods Mobile Comput. Commun.*, Sept. 2002.

[8] D. Grosu, A. T. Chronopoulos, and M. Y. Leung, "Load balancing in distributed systems: An approach using cooperative games," *Proc. 16th IEEE Int. Parallel Distributed Processing Symp.*, pp. 52–61, Apr. 2002.

[9] T. Groves, "Incentive in teams," *Econometrica*, vol. 41, no. 4, pp. 617–631, 1973.

[10] J. Hershberger and S. Suri, "Vickrey prices and shortest paths: What is an edge worth?," *Proc. 42nd IEEE Symp. Foundations Comput. Sci.*, pp. 252–259, Oct. 2001.

[11] R. Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. New York: Wiley-Interscience, 1991.

[12] H. Kameda, J. Li, C. Kim, and Y. Zhang, *Optimal Load Balancing in Distributed Computer Systems*. London, U.K.: Springer-Verlag, 1997.

[13] R. Karp, E. Koutsoupias, C. H. Papadimitriou, and S. Shenker, "Optimization problems in congestion control," *Proc. 41st IEEE Symp. Foundations Comput. Sci.*, pp. 66–74, Nov. 2000.

[14] L. Kleinrock, *Queueing Systems – Volume 1: Theory*. New York: Wiley, 1975.

[15] D. G. Luenberger, *Linear and Nonlinear Programming*. Reading, MA: Addison-Wesley, 1984.

[16] N. Nisan, S. London, O. Regev, and N. Camiel, "Globally distributed computation over internet – The POPCORN project," *Proc. 18th IEEE Int. Conf. Distributed Comput. Syst.*, pp. 592–601, May 1998.

[17] N. Nisan and A. Ronen, "Algorithmic mechanism design," in *Proc. 31rd Annu. ACM Symp. Theory Comput.*, May 1999, pp. 129–140.

[18] ——, "Computationally feasible VCG mechanism," in *Proc. 2nd ACM Conf. Electron. Commerce*, Oct. 2000, pp. 242–252.

[19] M. Osborne and A. Rubinstein, *A Course in Game Theory*. Cambridge, MA: MIT Press, 1994.

[20] K. W. Ross and D. D. Yao, "Optimal load balancing and scheduling in a distributed computer system," *J. ACM*, vol. 38, no. 3, pp. 676–690, July 1991.

[21] X. Tang and S. T. Chanson, "Optimizing static job scheduling in a network of heterogeneous computers," in *Proc. Intl. Conf. Parallel Processing*, Aug. 2000, pp. 373–382.

[22] A. N. Tantawi and D. Towsley, "Optimal static load balancing in distributed computer systems," *J. ACM*, vol. 32, no. 2, pp. 445–465, Apr. 1985.

[23] W. Vickrey, "Counterspeculation, auctions, and competitive sealed tenders," *J. Finance*, vol. 16, no. 1, pp. 8–37, Mar. 1961.

[24] W. E. Walsh, M. P. Wellman, P. R. Wurman, and J. K. MacKie-Mason, "Some economics of market-based distributed scheduling," *Proc. 18th IEEE Int. Conf. Distributed Comput. Syst.*, pp. 612–621, May 1998.

[25] R. Wolski, J. S. Plank, T. Bryan, and J. Brevik, "G-commerce: Market formulations controlling resource allocation on the computational grid," *Proc. 15th IEEE Int. Parallel Distributed Processing Symp.*, Apr. 2001.

Daniel Grosu (S'99) received the Diploma of engineering in automatic control and industrial informatics from the Technical University of Iasi, Iasi, Romania, in 1994 and the M.Sc. degree in computer science from The University of Texas at San Antonio in 2002. He is currently pursuing the Ph.D. degree with the Department of Computer Science, The University of Texas at San Antonio.

His research interests include load balancing, distributed systems, and topics at the border of computer science, game theory, and economics.

Mr. Grosu is a student member of the ACM.

Anthony T. Chronopoulos (SM'98) received the Ph.D. degree from the University of Illinois at Urbana-Champaign in 1987.

He is now with the Department of Computer Science, University of Texas at San Antonio. He has published 31 journal and more than 35 refereed conference proceedings publications in the areas of scientific computation and parallel and distributed computing. He has received 12 federal/state government research grants. His work is cited in more than 130 research articles.