A Load Balancing Mechanism with Verification *

Daniel Grosu and Anthony T. Chronopoulos
Department of Computer Science,
University of Texas at San Antonio,
6900 N. Loop 1604 West, San Antonio, TX 78249
{*dgrosu, atc*}*@cs.utsa.edu*

Abstract

In this paper we investigate the problem of designing load balancing mechanisms with verification for heterogeneous distributed systems. We derive a compensation and bonus type mechanism that solves the load balancing problem in distributed systems in which computers are characterized by linear latency functions. We prove that our mechanism is truthful and satisfies the voluntary participation condition. We present a simulation study to show the performance of our mechanism.

1. Introduction

The study in this paper is motivated by the recent increased interest in designing algorithms for resource allocation involving self interested participants [6, 16, 18]. We are especially interested in solving the load balancing problem in distributed systems where computational resources belong to self-interested parties (e.g. organizations, people). These participants called *agents* have no a-priori motivation for cooperation and they are tempted to manipulate the load allocation algorithm if it is beneficial to do so. This behavior may lead to poor system performance and inefficiency. Unlike the traditional protocols in distributed computing, the new protocols must deal with the possible manipulations. Thus the system must provide incentives to agents to participate in the given algorithm. The solution of this kind of problems comes from economics, more precisely from *mechanism design theory* [17]. The scope of this theory is to provide tools and methods to design protocols for self interested agents. Of interest are the so called *truthful mechanisms* in which agents are always forced to report their true parameters and follow the given algorithm.

*This research was supported, in part, by a grant from the Center for Infrastructure Assurance and Security at The University of Texas at San Antonio.

The goal of this paper is to design a load balancing mechanism with verification. We consider a distributed system in which computers are modeled by linear load-dependent latency functions [1, 19]. Solving the load balancing problem involves finding an allocation that minimizes the total latency of the system. The optimal allocation is obtained by assigning jobs to computers in proportion to their processing rates. This allocation algorithm is the basis for our mechanism. To design our mechanism we assume that each computer in the distributed system is characterized by its processing rate and that the true value of this rate is private knowledge. The *valuation* of each computer is the function that quantifies its benefit or loss and is equal to the negation of its latency.

The load balancing mechanism with verification works as follows. It first asks the computers to report their processing rates. Having obtained these rates, the mechanism computes the allocation and allocates the jobs to computers. Because we want to have a mechanism with verification the payment to each agent is computed and given to it after the assigned jobs were executed. Here we assume that the processing rate with which the jobs were actually executed is known to the mechanism. Each computer goal is to report a value for its processing rate such that its utility is maximized. The *utility* of each computer is defined as the sum of its valuation and its payment. The mechanism must be designed such that the agents maximize their utility only if they report the true values of the processing rates and execute the jobs using their full processing capacity. Also, if each computer reports its true value and executes the jobs at the full capacity then the minimum total latency is obtained.

In this paper we design a mechanism with verification based on a compensation and bonus type mechanism. This type of mechanism was initially studied by Nisan and Ronen [16] in the context of task scheduling on unrelated machines.

Related work

The load balancing problem in distributed systems has been extensively investigated under the classical assumption that

0-7695-1926-1/03/$17.00 (C) 2003 IEEE
Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'03)

the participants are obedient and follow the algorithm. The solution of this problem was obtained using different techniques and models [7, 10, 20].

Recently, several researchers considered the mechanism design theory to solve several computational problems that involve self interested agents. These problems include resource allocation and task scheduling [13, 21, 22], congestion control and routing [4, 11].

The closest work to our study is the paper of Nisan and Ronen [16]. They were the first to consider the mechanism design problem in a computational setting. They studied different types of mechanisms for shortest path and job scheduling on unrelated machines. They proposed a VCG(Vickrey-Clarke-Groves) mechanism for solving the shortest path in graphs where edges belong to self interested agents and mechanisms for solving the task scheduling on unrelated machines. The VCG mechanism allows arbitrary form for valuations and is restricted to objective functions defined by the sum of agents' valuations. They also proposed a mechanism with verification that solves the problem of task scheduling on unrelated machines. Their mechanism is a compensation and bonus type mechanism.

Archer and Tardos [2] derived a general framework for designing truthful mechanisms for optimization problems where the private data of each agent is described by one real valued parameter. Their method allows the design of truthful mechanisms for optimization problems that have general objective functions and restricted form for valuations. Using this method they designed truthful mechanisms for several problems in computer science. In [3] the same authors investigated the frugality of shortest path mechanisms.

Using the framework presented in [2], Grosu and Chronopoulos [8] designed a truthful mechanism that gives the overall optimal solution for the static load balancing problem in distributed systems in which computers are characterized by M/M/1 delay functions.

The computational aspects of the mechanisms for cost sharing in multicast transmissions were studied by Feigenbaum *et al*. in [5]. In [4] a mechanism for low cost routing in networks is studied. In both these papers the authors considered a distributed mechanism which is suitable for implementation in large distributed systems. The results and the challenges of designing distributed mechanisms are surveyed in [6].

Our contributions

The focus of this paper is the design of a load balancing mechanism with verification in heterogeneous distributed systems. We model the computers of the distributed system using linear load-dependent latency functions. We formulate the problem of designing a load balancing mechanism with verification and we devise a truthful mechanism for this problem. We prove that our mechanism is truthful and satisfies the voluntary participation condition. A simulation study is performed to investigate the effectiveness of our mechanism.

Organization

The paper is structured as follows. In Section 2 we present the distributed system model and formulate the load balancing problem. In Section 3 we design a truthful mechanism with verification that solves the load balancing problem in distributed systems in which computers have linear load-dependent latency functions. In Section 4 we investigate the effectiveness of our load balancing mechanism by simulation. In Section 5 we draw conclusions and present future directions.

2. Model and problem formulation

We consider a distributed system that consists of a set $N = \{1, 2, \ldots, n\}$ of n heterogeneous computers. We assume that each computer is characterized by a load-dependent *latency function*. The latency function of computer i is linear on x_i and has the following form:

$$l_i(x_i) = a_i x_i \tag{1}$$

where x_i is the arrival rate of jobs allocated to computer i and a_i is a parameter inversely proportional to the processing rate of computer i. A small (big) a_i characterizes a fast (slow) computer. In other words $l_i(x_i)$ measures the time required to complete one job on computer i.

Models using linear load dependent latency functions were investigated in [1, 19]. This type of function could represent the expected waiting time in a M/G/1 queue, under light load conditions (considering a_i as the variance of the service time in the queue) [1].

We assume that there is a large number of jobs that need to be executed. These jobs arrive at the system with an arrival rate R. A feasible allocation of jobs to the computers is a vector $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ that satisfies two conditions:

(i) Positivity: $x_i > 0 \qquad i = 1, 2, \ldots, n$;

(ii) Conservation: $\sum_{i=1}^{n} x_i = R$;

The performance of the system is characterized by the *total latency*:

$$L(\mathbf{x}) = \sum_{i=1}^{n} x_i l_i(x_i) = \sum_{i=1}^{n} a_i x_i^2 \tag{2}$$

The load balancing problem can be stated as follows:

Definition 2.1 (Load balancing problem) Given the job arrival rate R at a distributed system with n heterogeneous computers characterized by the load-dependent latency functions $l_i(.)$, find a feasible allocation $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ that minimizes the total latency $L(\mathbf{x})$.

The following theorem gives the solution for this problem.

0-7695-1926-1/03/$17.00 (C) 2003 IEEE

Theorem 2.1 (Optimal allocation) The optimal allocation for the above load balancing problem is given by:

$$x_i = \frac{\frac{1}{a_i}}{\sum_{k=1}^{n} \frac{1}{a_k}} R \qquad i = 1, 2, \ldots, n \tag{3}$$

This allocation gives the minimum value for the total latency:

$$L^* = \frac{R^2}{\sum_{k=1}^{n} \frac{1}{a_k}} \tag{4}$$

Proof: In Appendix.

In other words this theorem says that if the latency functions for each computer are linear then the allocation of jobs in proportion to the processing rate of each computer gives the minimum total latency. Using this theorem, the following algorithm can be derived.

PR algorithm:

Input: Processing rates: $\frac{1}{a_1}, \frac{1}{a_2}, \ldots \frac{1}{a_n}$;
Arrival rate: R;
Output: Load allocation: $x_1, x_2, \ldots x_n$;
for $i = 1, \ldots, n$ **do**
$x_i \leftarrow R \frac{\frac{1}{a_i}}{\sum_{k=1}^{n} \frac{1}{a_k}}$;

The above algorithm solves the load balancing problem in the classical setting where the participants (computers) are assumed to follow the algorithm. Here we assume that computers are selfish agents characterized by their true values t_i, $i = 1, 2, \ldots, n$. The true value t_i corresponds to the parameter a_i in the latency function l_i, which is inversely proportional to the processing rate of computer i. We need to design a mechanism that obtains the optimal allocation and forces the participants to reveal their true values t_i and follow the PR algorithm. In the next section we design such a mechanism.

3. The load balancing mechanism with verification

In this section we formally describe the load balancing mechanism design problem considering our distributed system model. Then we present the design of our load balancing mechanism with verification and study its properties.

Definition 3.1 (Mechanism design problem) The problem of designing a load balancing mechanism with verification is characterized by:

(i) A finite set $\mathbf{X}$ of allowed outputs. The output is a vector $\mathbf{x}(\mathbf{b}) = (x_1(\mathbf{b}), x_2(\mathbf{b}), \ldots, x_n(\mathbf{b}))$, $\mathbf{x}(\mathbf{b}) \in \mathbf{X}$, computed according to the agents' bids, $\mathbf{b} = (b_1, b_2, \ldots, b_n)$. 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 its *true value* (sometimes called its *type*) and a publicly known parameter $\tilde{t}_i \geq t_i$ called its *execution value*. The preferences of agent i are given by a function called *valuation* $V_i(\mathbf{x}, \tilde{\mathbf{t}}) = -\tilde{t}_i x_i^2$. The execution value $\tilde{t}_i$ determines the value of the latency function and thus the actual execution time for one job at agent i.

(iii) Each agent goal is to maximize its *utility*. The utility of agent i is $U_i(\mathbf{b}, \tilde{\mathbf{t}}) = P_i(\mathbf{b}, \tilde{\mathbf{t}}) + V_i(\mathbf{x}(\mathbf{b}), \tilde{\mathbf{t}})$, where P_i is the payment handed by the mechanism to agent i and $\tilde{\mathbf{t}}$ is the vector of execution values. The payments are handed to agents after the assigned jobs have been completed and the mechanism knows $\tilde{t}_i$, $i = 1, 2, \ldots, n$.

(iv) The goal of the mechanism is to select an output $\mathbf{x}$ that minimizes the total latency function $L(\mathbf{x}, \mathbf{b}) = \sum_{i=1}^{n} b_i x_i^2$.

An agent i may report a value (bid) b_i different from its true value t_i. The true value characterizes the actual processing capacity of computer i. In addition, agent i may choose to execute the jobs allocated to it with a different processing rate given by its execution value ($\tilde{t}_i \geq t_i$). Thus, an agent i may execute the assigned jobs at a slower rate than its true processing rate. The goal of a truthful mechanism with verification is to give incentives to agents such that it is beneficial for them to report their true values and execute the assigned jobs using their full processing capacity. Now we give a formal description of a mechanism with verification.

Definition 3.2 (Mechanism with verification) A *mechanism with verification* is a pair of functions:

(i) The *allocation function* $\mathbf{x}(\mathbf{b}) = (x_1(\mathbf{b}), x_2(\mathbf{b}), \ldots, x_n(\mathbf{b}))$. This function has as input the vector of agents' bids $\mathbf{b} = (b_1, b_2, \ldots, b_n)$ and returns an output $\mathbf{x} \in \mathbf{X}$.

(ii) The *payment function* $P(\mathbf{b}, \tilde{\mathbf{t}}) = (P_1(\mathbf{b}, \tilde{\mathbf{t}}), P_2(\mathbf{b}, \tilde{\mathbf{t}}), \ldots, P_n(\mathbf{b}, \tilde{\mathbf{t}}))$, where $P_i(\mathbf{b}, \tilde{\mathbf{t}})$ is the payment handed by the mechanism to agent i.

According to this definition, to obtain a load balancing mechanism with verification we must find an allocation function $\mathbf{X}(\mathbf{b})$ and a payment function $P(\mathbf{b}, \tilde{\mathbf{t}})$. The allocation function must minimize $L(\mathbf{x})$. The payment function must be designed such that the mechanism is truthful.

Here we consider a compensation and bonus type mechanism [16] to solve the load balancing problem. The optimal allocation function for this mechanism is given by the

0-7695-1926-1/03/$17.00 (C) 2003 IEEE

PR algorithm. The payment function for this type of mechanism is the sum of two functions: a compensation function and a bonus function.

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

In the following we define our load balancing mechanism with verification.

Definition 3.3 (The load balancing mechanism with verification) The mechanism with verification that solves the load balancing problem is defined by the following two functions:

(i) The allocation function given by the PR algorithm.

(ii) The payment function is given by:

$$P_i(\mathbf{b}, \tilde{\mathbf{t}}) = C_i(\mathbf{b}, \tilde{\mathbf{t}}) + B_i(\mathbf{b}, \tilde{\mathbf{t}}) \quad (5)$$

where the function $C_i(\mathbf{b}, \tilde{\mathbf{t}}) = \tilde{t}_i x_i^2(\mathbf{b})$ is called the *compensation* function for agent i; and the function $B_i(\mathbf{b}, \tilde{\mathbf{t}}) = L_{-i}(\mathbf{x}(\mathbf{b}_{-i}, \mathbf{b}_{-i})) - L(\mathbf{x}(\mathbf{b}), (\mathbf{b}_{-i}, \tilde{t}_i))$ is called the *bonus* for agent i. The function $L_{-i}(\mathbf{x}(\mathbf{b}_{-i}, \mathbf{b}_{-i}))$ is the optimal latency when agent i is not used in the allocation. Thus, the bonus for an agent is equal to its contribution in reducing the total latency.

We are interested in obtaining a truthful load balancing mechanism. A truthful mechanism can be defined as follows.

Definition 3.4 (Truthful mechanism) A mechanism is called *truthful* if for every agent i of type t_i and for every bids b_{-i} of the other agents, the agent's utility is maximized when it declares its real value t_i (i.e. truth-telling is a dominant strategy).

For our load balancing mechanism we can state the following theorem.

Theorem 3.1 (Truthfulness) The load balancing mechanism with verification is truthful.

Proof: In Appendix.

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

Definition 3.5 (Voluntary participation mechanism) We say that a mechanism satisfies the *voluntary participation condition* if $U_i((\mathbf{b}_{-i}, t_i), \tilde{t}_i) \geq 0$ for every agent i, true values t_i, execution values $\tilde{t}_i = t_i$, and other agents' bids $\mathbf{b}_{-i}$ (i.e. truthful agents never incur a loss).

Theorem 3.2 (Voluntary participation) The load balancing mechanism with verification satisfies the voluntary participation condition.

Proof: In Appendix.

We obtained a truthful load balancing mechanism with verification that satisfies the voluntary participation condition. Based on this mechanism a load balancing protocol can be derived. An informal description of this centralized protocol is as follows. The mechanism collects the bids from each computer, computes the allocation using PR algorithm and allocates the jobs. Then it waits for the allocated jobs to be executed. In this waiting period the mechanism estimates the actual job processing rate at each computer and use it to determine the execution value $\tilde{t}_i$. After the allocated jobs are completed the mechanism computes the payments and sends them to the computers. After receiving the payment each computer evaluates its utility. The total number of messages sent by the above protocol is $O(n)$.

4. Experimental results

In this section we study the effectiveness of the proposed mechanism by simulation. The simulated distributed system consists of 16 heterogeneous computers. The latency function of each computer is given by the parameter $a_i = t_i$ presented in Table 1. This parameter is inversely proportional to the computer's processing rate.

Computers	C1 - C2	C3 - C5	C6 - C10	C11 - C16
True value (t)	1	2	5	10

Table 1. System configuration.

We consider eight types of experiments depending on the bid and on the execution value of computer C1. In all the experiments we assume that all the computers except C1 bid their true values and that their execution values are the same as their true values. We classify these experiments in three main classes according to the bids of computer C1 as follows: *True*, when $b_1 = t_1$; *High*, when $b_1 > t_1$; and *Low*, when $b_1 < t_1$. We further divide these main classes considering the execution value of C1. We have two sets of experiments for the *True* class, four for the *High* class and two for the *Low* class. The parameters used in these experiments are presented in Table 2.

First, we consider the influence of false bids ($b_1 \neq t_1$) on the total latency. We assume that the job rate is $R = 20$ jobs/sec. In Figure 1 we present the total latency for the eight experiments. We now discuss the results of each experiment.

0-7695-1926-1/03/$17.00 (C) 2003 IEEE

Experiment	Characterization	t_1	b_1	$\tilde{t}_1$
True1	$\tilde{t}_1 = t_1 = b_1$	1	1	1
True2	$\tilde{t}_1 > t_1 = b_1$	1	1	3
High1	$\tilde{t}_1 = b_1 > t_1$	1	3	3
High2	$b_1 > t_1 = \tilde{t}_1$	1	3	1
High3	$b_1 > \tilde{t}_1 > t_1$	1	3	2
High4	$\tilde{t}_1 > b_1 > t_1$	1	3	4
Low1	$t_1 = t_1 > b_1$	1	0.5	1
Low2	$t_1 > t_1 > b_1$	1	0.5	2

Table 2. Types of experiments.

***True1*:** All the computers report their true values. The execution value is equal to the true value for each computer. As expected (from the theory) we obtain the minimum value for the total latency (L = 78.43).

***True2*:** The parameters are as in *True1* except that C1 has a higher execution value $\tilde{t}_1 > t_1$. This means C1 execution is slower increasing the total latency by 17%.

***High1*:** In this case C1 bids three times higher than its true value and the execution value is equal to the bid. Because C1 bids higher the other computers are overloaded thus increasing the total latency. C1 gets fewer jobs and executes them with a slower execution rate.

***High2*:** In this case C1 gets fewer jobs and the other computers are overloaded. Here the increase is not as big as in *High1* because C1 executes the jobs at its full capacity.

***High3*:** This case is similar to *High1* except that the execution on C1 is faster. It can be seen that the total latency is less than in the case *High1*.

***High4*:** Similar to *High1*, except that C1 executes the jobs slower, increasing the total latency.

***Low1*:** C1 bids 2 times less than its true value, getting more jobs and executing them at its full capacity. The increase in total latency is not big, about 11%.

***Low2*:** In this case C1 gets more jobs and executes them two times slower. This overloading of C1 leads to a significant increase in the total latency (about 66%).

From the results of these experiments, it can be observed that small deviations from the true value and from the execution value of only one computer may lead to large values of the total latency. We expect even larger increase if more than one computer does not report its true value and does not use its full processing capacity. The necessity of a mechanism that forces the participants to be truthful becomes vital in such situations.

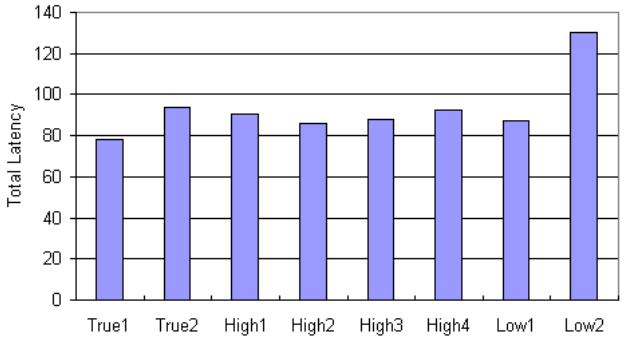

Figure 1. Performance degradation.

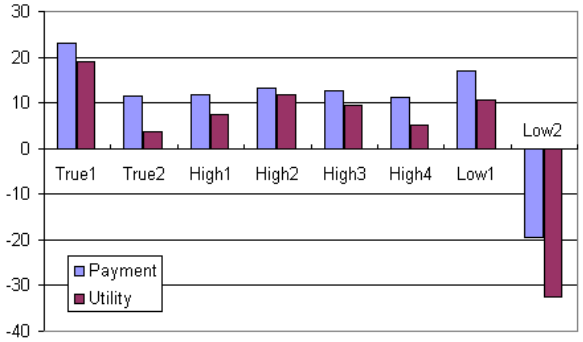

Figure 2. Payment and utility for computer C1.

In Figure 2 we present the payment and utility of computer C1 for each set of experiments. As expected, C1 obtains the highest utility in the experiment *True1*, when it bids its real value and uses its full processing capacity. In the other experiments C1 is penalized for lying and the payment that it receives is lower than in the case of *True1*. The utility is also lower in these experiments. An interesting situation occurs in the experiment *Low2* where the payment and utility of C1 are negative. This can be explained as follows. Computer C1 bids two times less than its true value and the PR algorithm allocates more jobs to it. In addition, its execution value $\tilde{t}_1$ is two times higher than t_1, that means the allocated jobs are executed two times slower than in the experiment *True1*(when $\tilde{t}_1 = t_1$). More allocated jobs to C1 combined with a slow execution increases the total latency L. The total latency becomes greater than the latency L_{-1} obtained when computer C1 is not used in the allocation (i.e. $L > L_{-1}$) and thus the bonus is negative. The absolute value of the bonus is greater than the compensation and from the definition of the payment it can be seen that the payment given to C1 is negative.

0-7695-1926-1/03/$17.00 (C) 2003 IEEE
Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS'03)

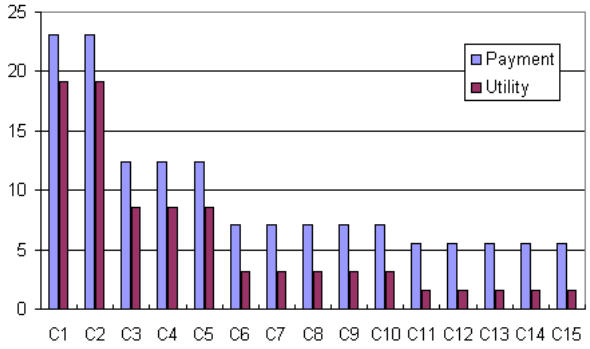

Figure 3. Payment and utility for each computer (*True1*).

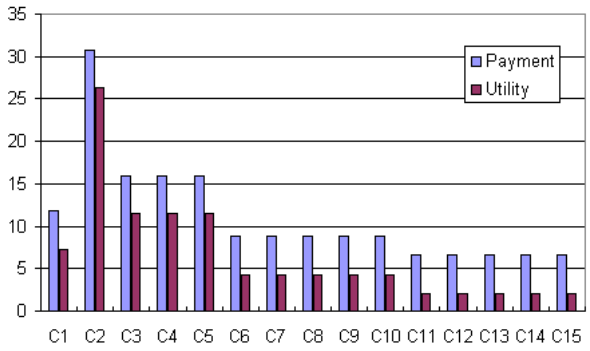

Figure 4. Payment and utility for each computer (*High1*).

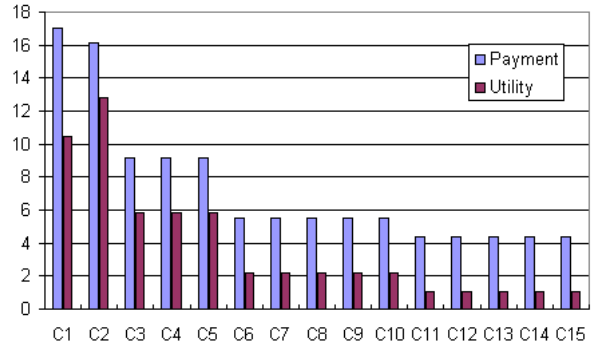

Figure 5. Payment and utility for each computer (*Low1*).

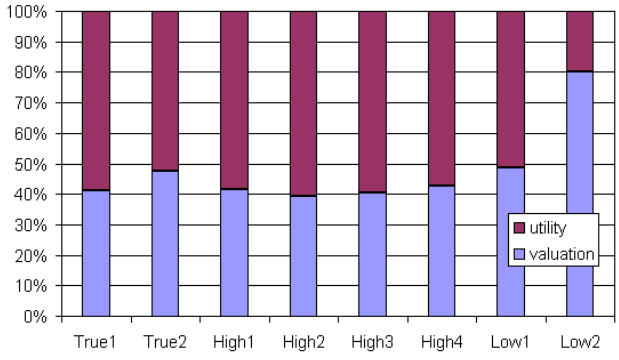

Figure 6. Payment structure.

In Figures 3-5 we present the payment structure for each computer in three of the experiments. In the experiment *Low1* (Figure 5) computer C1 obtains a utility which is 45% lower than in the experiment *True1*. The other computers (C2 - C16) obtain lower utilities. This is because all the computers except C1 receive fewer jobs and their payments are smaller than in the experiment *True1*.

In the experiment *High1* (Figure 4) computer C1 obtains a utility which is 62% lower than in the experiment *True1*. The other computers (C2 - C16) obtain higher utilities. The mechanism pays them more because they receive more jobs.

In Figure 6 we present the payment structure of our mechanism. It can be observed that the total payment given by our mechanism to the computers is at most 2.5 times the total valuation. The lower bound on the total payment is the total valuation. This is because our mechanism must preserve the voluntary participation property. The amount of the total payment given by a mechanism characterizes its frugality. A mechanism that gives small payments by some measure it is called frugal. Based on the experimental results presented here the payments are at most 2.5 times the total valuation, thus our mechanism can be considered frugal.

5. Conclusion

We designed a truthful mechanism with verification that solves the load balancing problem in a distributed system in which computers are characterized by linear load-dependent latency functions. We proved that this mechanism satisfies the voluntary participation condition. We studied the performance of this mechanism in terms of its payment structure.

Future work will address the problem of distributed handling of payments and the agents' privacy.

0-7695-1926-1/03/$17.00 (C) 2003 IEEE

A. Appendix

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

Proof of Theorem 2.1

The total latency function is a convex function and the optimal allocation can be obtained by using the Kuhn-Tucker conditions [12]. Let $\alpha \geq 0$, $\eta_i \geq 0$, $i = 1, \ldots, n$ denote the Lagrange multipliers [12]. We consider the Lagrangian function:

$$\mathcal{L}(\mathbf{x}, \alpha, \eta_1, \ldots, \eta_n) = \sum_{i=1}^{n} a_i x_i^2 - \alpha(\sum_{i=1}^{n} x_i - R) - \sum_{i=1}^{n} \eta_i x_i \tag{6}$$

The Kuhn-Tucker conditions imply that x_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:

$$\frac{\partial \mathcal{L}}{\partial x_i} = 0, \qquad i = 1, \ldots, n \tag{7}$$

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

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

We consider $x_i > 0$ and this implies that the last condition is satisfied only if $\eta_i = 0$. The conditions become:

$$2a_i x_i - \alpha = 0 \qquad i = 1, \ldots, n \tag{10}$$

$$\sum_{i=1}^{n} x_i - R = 0 \tag{11}$$

Solving these equations we get:

$$x_i = \frac{\alpha}{2a_i} \qquad i = 1, \ldots, n \tag{12}$$

$$\alpha = \frac{2R}{\sum_{k=1}^{n} \frac{1}{a_k}} \tag{13}$$

From these, we obtain the optimal allocation x_i:

$$x_i = \frac{\frac{1}{a_i}}{\sum_{k=1}^{n} \frac{1}{a_k}} R \qquad i = 1, \ldots, n \tag{14}$$

Using this allocation we compute the minimum value for the latency function as follows:

$$L^* = \sum_{i=1}^{n} a_i x_i^2 = \sum_{i=1}^{n} a_i \frac{(\frac{1}{a_i})^2}{(\sum_{k=1}^{n} \frac{1}{a_k})^2} R^2 = \frac{R^2}{\sum_{k=1}^{n} \frac{1}{a_k}} \tag{15}$$

□

Proof of Theorem 3.1

Assuming a vector of bids $\mathbf{b}$, the utility of agent i is:

$$U_i(\mathbf{b}, \tilde{\mathbf{t}}) = P_i(\mathbf{b}, \tilde{\mathbf{t}}) + V_i(\mathbf{x}(\mathbf{b}), \tilde{\mathbf{t}})$$

$$= L_{-i}(\mathbf{x}(\mathbf{b}_{-i})) - L(\mathbf{x}(\mathbf{b}), (\mathbf{b}_{-i}, \tilde{t}_i)) + \tilde{t}_i x_i^2(\mathbf{b}) - \tilde{t}_i x_i^2(\mathbf{b})$$

$$= L_{-i}(\mathbf{x}(\mathbf{b}_{-i})) - L(\mathbf{x}(\mathbf{b}), (\mathbf{b}_{-i}, \tilde{t}_i)) \tag{16}$$

We consider two possible situations:

i) $t_i = \tilde{t}_i$ i.e. agent i executes its assigned jobs using its full processing capability.

If agent i bids its true value t_i then its utility U_i^t is:

$$U_i^t = L_{-i}(\mathbf{x}(\mathbf{b}_{-i})) - L(\mathbf{x}(\mathbf{b}_{-i}, t_i), (\mathbf{b}_{-i}, \tilde{t}_i))$$

$$= L_{-i}(\mathbf{x}(\mathbf{b}_{-i})) - L_i^t \tag{17}$$

If agent i bids lower ($b_i^l < t_i$) then its utility U_i^l is:

$$U_i^l = L_{-i}(\mathbf{x}(\mathbf{b}_{-i})) - L(\mathbf{x}(\mathbf{b}_{-i}, b_i^l), (\mathbf{b}_{-i}, \tilde{t}_i))$$

$$= L_{-i}(\mathbf{x}(\mathbf{b}_{-i})) - L_i^l \tag{18}$$

We want to show that $U_i^t \geq U_i^l$, that reduces to show that $L_i^l \geq L_i^t$. Because L_i^t is the minimum possible value for the latency (from the optimality of PR algorithm), by bidding a lower value, agent i gets more jobs and the total latency is increased, thus $L_i^l \geq L_i^t$.

If agent i bids higher ($b_i^h > t_i$) then its utility U_i^h is:

$$U_i^h = L_{-i}(\mathbf{x}(\mathbf{b}_{-i})) - L(\mathbf{x}(\mathbf{b}_{-i}, b_i^h), (\mathbf{b}_{-i}, \tilde{t}_i))$$

$$= L_{-i}(\mathbf{x}(\mathbf{b}_{-i})) - L_i^h \tag{19}$$

By bidding a higher value agent i gets fewer jobs and so more jobs will be assigned to the other agents. Due to the optimality of allocation the total latency increases i.e. $L_i^h \geq L_i^t$ and thus we have $U_i^t \geq U_i^h$.

ii) $\tilde{t}_i > t_i$ i.e. agent i executes its assigned jobs at a slower rate thus increasing the total latency. A similar argument as in the case i) applies.

□

Proof of Theorem 3.2

The utility of agent i when it bids its true value t_i is:

$$U_i^t = L_{-i}(\mathbf{x}(\mathbf{b}_{-i})) - L(\mathbf{x}(\mathbf{b}_{-i}, t_i), (\mathbf{b}_{-i}, \tilde{t}_i)) \tag{20}$$

The latency L_{-i} is obtained by using all the other agents except agent i. By allocating the same number of jobs, we get a higher latency L_{-i} than in the case of using all the agents, with agent i bidding its true value (from the optimality of allocation). Thus $U_i^t \geq 0$. □

References

[1] E. Altman, T. Basar, T. Jimenez and N. Shimkin, "Routing in Two Parallel Links: Game-Theoretic Distributed Algorithms", *Journal of Parallel and Distributed Computing*, vol. 61, no. 9, pp. 1367-1381, September 2001.

[2] A. Archer and E. Tardos, "Truthful Mechanism for One-Parameter Agents", *Proc. of the 42nd IEEE Symp. on Foundations of Computer Science*, pp. 482-491, October 2001.

0-7695-1926-1/03/$17.00 (C) 2003 IEEE

[3] A. Archer and E. Tardos, "Frugal Path Mechanisms", *Proc. of the 13th Annual ACM-SIAM Symp. on Discrete Algorithms*, pp. 991-999, January 2002.

[4] J. Feigenbaum, C. Papadimitriou, R. Sami and S. Shenker, "A BGP-based Mechanism for Lowest-Cost Routing", *Proc. of the 21st ACM Symposium on Principles of Distributed Computing*, pp. 173-182, July 2002.

[5] J. Feigenbaum, C. Papadimitriou and S. Shenker, "Sharing the Cost of Multicast Transmissions", *Journal of Computer and System Sciences*, vol. 63, no. 1, pp. 21-41, August 2001.

[6] J. Feigenbaum and S. Shenker, "Distributed Algorithmic Mechanism Design: Recent Results and Future Directions", *Proc. of the 6th ACM Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, pp. 1-13, September 2002.

[7] D. Grosu, A. T. Chronopoulos, and M. Y. Leung, "Load Balancing in Distributed Systems: An Approach Using Cooperative Games", *Proc. of the 16th IEEE International Parallel and Distributed Processing Symposium*, pp. 501-510, April 2002.

[8] D. Grosu and A. T. Chronopoulos, "Algorithmic Mechanism Design for Load Balancing in Distributed Systems", *Proc. of the 4th IEEE International Conference on Cluster Computing*, pp. 445-450, September 2002.

[9] J. Hershberger and S. Suri, "Vickrey Prices and Shortest Paths: What is an Edge Worth?", *Proc. of the 42nd IEEE Symp. on Foundations of Computer Science*, pp. 252-259, October 2001.

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

[11] R. Karp, E. Koutsoupias, C. H. Papadimitriou, and S. Shenker, "Optimization Problems in Congestion Control", *Proc. of the 41st IEEE Symp. on Foundations of Computer Science*, pp. 66-74, November 2000.

[12] D. G. Luenberger, *Linear and Nonlinear Programming*, Addison-Wesley, Reading, Mass., 1984.

[13] N. Nisan, S. London, O. Regev, and N. Camiel, "Globally Distributed Computation over Internet - The POPCORN Project", *Proc. of the 18th IEEE International Conference on Distributed Computing Systems*, pp. 592-601, May 1998.

[14] N. Nisan and A. Ronen, "Algorithmic Mechanism Design", *Proc. of the 31rd Annual ACM Symp. on Theory of Computing*, pp. 129-140, May 1999.

[15] N. Nisan and A. Ronen, "Computationally Feasible VCG Mechanism", *Proc. of the 2nd ACM Conference on Electronic Commerce*, pp. 242-252, October 2000.

[16] N. Nisan and A. Ronen, "Algorithmic Mechanism Design", *Games and Economic Behavior*, vol. 35, no. 1-2, pp. 166-196, April 2001.

[17] M. Osborne and A. Rubinstein, *A Course in Game Theory*, MIT Press, Cambridge, Mass., 1994.

[18] A. Ronen, "Algorithms for Rational Agents", *Proc. of the 27th Conf. on Current Trends in Theory and Practice of Informatics*, LNCS 1963, Springer Verlag, pp. 56-70, November 2000.

[19] T. Roughgarden, "Stackelberg Scheduling Strategies", *Proc. of the 33rd Annual ACM Symp. on Theory of Computing*, pp. 104-113, July 2001.

[20] X. Tang and S. T. Chanson, "Optimizing static Job Scheduling in a Network of Heterogeneous Computers", *Proc. of the Intl. Conf. on Parallel Processing*, pp. 373-382, August 2000.

[21] W. E. Walsh, M. P. Wellman, P. R. Wurman, and J. K. MacKie-Mason, "Some Economics of Market-based Distributed Scheduling", *Proc. of the 18th IEEE International Conference on Distributed Computing Systems*, pp. 612-621, May 1998.

[22] R. Wolski, J. S. Plank, T. Bryan, and J. Brevik, "G-commerce: Market Formulations Controlling Resource Allocation on the Computational Grid", *Proc. of the 15th IEEE International Parallel and Distributed Processing Symposium*, April 2001.

0-7695-1926-1/03/$17.00 (C) 2003 IEEE