A Truthful Mechanism for Fair Load Balancing in Distributed Systems *

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 consider the problem of designing load balancing protocols in distributed systems where the participants (e.g. computers, users) are capable of manipulating the load allocation algorithm in their own interest. Using techniques from mechanism design theory we design a mechanism for fair load balancing in heterogeneous distributed systems. We prove that our mechanism is truthful and satisfies the voluntary participation condition. Based on the proposed mechanism we derive a fair load balancing protocol called FAIR-LBM. Finally, we study the effectiveness of our protocol by simulations.

1. Introduction

The problem of scheduling and load balancing in distributed systems has been extensively investigated under the assumption that the participants are obedient and follow a given algorithm [11]. In current distributed systems the computational resources are owned and operated by different agents or organizations. In such systems there is no a-priori motivation for cooperation and the agents may manipulate the resource allocation algorithm in their own interest leading to severe performance degradation and poor efficiency. Solving such problems involving selfish agents is the object of *mechanism design theory* [15]. 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*.

In a mechanism, each agent's benefit or loss is quantified by a function called *valuation*. This function is private information for each agent and depends on the outcome. The mechanism requests the agents to report their valuations, then it 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.

Our goal in this paper is to design a mechanism that gives a fair and Pareto optimal solution to the static load balancing problem in distributed systems. We can formulate this problem as follows: given a large number of jobs, find the allocation of jobs to computers such that the expected response time at each computer is minimized and all the jobs receive a fair treatment independent of the allocated computer. We use the fair load balancing algorithm proposed in [8] as a basis for our mechanism. 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 fair load balancing algorithm finds the fraction of load that is allocated to each computer such that the expected response time at each computer is minimized and each job is treated fairly. Each computer incurs a *cost* proportional to its utilization. The mechanism works as follows. First it asks each computer to report its processing rate. Then it computes the allocation using the fair load balancing algorithm. After computing the allocation the mechanism hands payments to computers. The computers receive the payments and evaluate their profits.

The goal of each computer 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 allocation is fair and Pareto optimal. Thus the goal of a

*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.

Proceedings of the Second IEEE International Symposium on Network Computing and Applications (NCA'03)
0-7695-1938-5/03 $17.00 © 2003 IEEE

mechanism designer is to find a payment function that motivates the agents to report their true value.

Related work

Recently, applications of game theory to computer science have attracted a lot of interest and have become a major trend. This was motivated by the need to design efficient protocols for self interested agents in current distributed systems. Mechanism design which is a sub-area of game theory was used in different settings such as market based protocols for resource allocation [3], routing [4] and mechanisms for trading CPU time [13]. A recent survey on distributed algorithmic mechanism design is [6].

The first work that considered the problem of mechanism design in a computational setting is the seminal paper of Nisan and Ronen [14]. They provided mechanisms for several concrete problems in computer science. They used the the Vickrey-Clarke-Groves (VCG) mechanism [15] to solve 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 also gave a mechanism that solves exactly the problem of scheduling jobs on unrelated machines in a model where the mechanism has more information. Feigenbaum *et al*. [5] studied two mechanisms for cost-sharing in multicast transmissions. Frugality of shortest paths mechanisms is investigated in [2].

A general method to design truthful mechanisms for optimization problems where each agent's secret data is represented by a single real valued parameter was proposed by Archer and Tardos in [1]. Their method allows general objective functions and restricted form for valuations. They gave truthful mechanisms for maximum flow, scheduling related machines, optimizing an affine function and special cases of uncapacitated facility location. Using this method, in [8] we designed a truthful mechanism that gives the overall optimal solution for the static load balancing problem in distributed systems.

Our contributions

We design a mechanism that gives a fair and Pareto optimal solution for the static load balancing problem in distributed systems. We prove that our mechanism is truthful and satisfies the voluntary participation condition. We derive a fair load balancing protocol (FAIR-LBM) that implements our mechanism and we investigate its effectiveness by simulations.

Organization

The paper is structured as follows. In Section 2 we present the distributed system model and the load balancing problem. In Section 3 we define our mechanism design problem. In Section 4 we design a truthful mechanism for fair load balancing in distributed systems. In Section 5 we investigate the effectiveness of our fair load balancing mechanism by simulation. In Section 6 we draw conclusions and present future directions.

2. Distributed system model

Our distributed system model consists of n heterogeneous computers connected in an arbitrary fashion 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) [7] 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. $\Phi < \sum_{i=1}^{n} \mu_i$). 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 *loads* λ_i ($i = 1, \ldots, n$) assigned to computers minimizing the job expected response time at each computer. In addition we want that all jobs receive a fair treatment independent of the allocated computer. Thus we are interested in finding a fair and Pareto optimal allocation. The expected response time at computer i is given by:

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

In the rest of the paper we denote by $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_n)$ the vector of loads assigned to computers.

Because we are interested in finding a feasible allocation that is fair and Pareto optimal we need to define these concepts. We first define the concept of feasible allocation for our problem.

Definition 2.1 (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 interpretation of a Pareto optimal allocation is that it is impossible to find another allocation which leads to strictly lower expected job response times for all the computers simultaneously. We next define formally the concept of Pareto optimality in our context.

Definition 2.2 (Pareto optimal allocation) Let Λ be the set of feasible allocations and $F_i(\lambda)$ be the expected response time of computer i. Then $\lambda \in \Lambda$ is said to be a *Pareto optimal allocation* if for each $\lambda' \in \Lambda$, $F_i(\lambda') \leq F_i(\lambda)$, $i = 1, \ldots, n$ imply $F_i(\lambda') = F_i(\lambda), i = 1, \ldots, n$

Proceedings of the Second IEEE International Symposium on Network Computing and Applications (NCA'03)
0-7695-1938-5/03 $17.00 © 2003 IEEE

It can be easily seen that all the feasible allocations are Pareto optimal. Among these allocations we need to select those providing fairness. The fairness is a loose criterion and there are many notions of fairness. Here we consider the *fairness index* as a measure of fairness [10].

Definition 2.3 (Fairness index) The *fairness index* depends on the load allocation (λ) and is given by:

$$I(\lambda) = \frac{[\sum_{i=1}^{n} F_i(\lambda)]^2}{n \sum_{i=1}^{n} F_i^2(\lambda)} \quad (1)$$

This index is a measure of the 'equality' of the expected response times at different computers. So it is a measure of load balance. If all the computers have the same expected job response 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.

In [8] it is shown that a fair ($I(\lambda) = 1$) and Pareto optimal allocation for our problem is obtained by maximizing the following objective function:

$$Q(\lambda) = \sum_{i=1}^{n} ln(\mu_i - \lambda_i) \quad (2)$$

where $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_n)$ is the vector of loads assigned to computers. We use this optimization problem as a basis for designing our load balancing mechanism.

3. The mechanism design problem

We now describe our mechanism design problem. We assume that computers are agents and each of them has a *true value* t_i represented by the inverse of its processing rate, $t_i = \frac{1}{\mu_i}$. This value is private information, 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 allocation is fair and Pareto optimal. 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.

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

(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 $cost_i(t_i, \lambda(b)) = t_i \lambda_i(b)$. The cost of agent i is equivalent to computer i utilization.

(iii) Each agent goal is to maximize its 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 the objective function $Q(\lambda, b) = \sum_{i=1}^{n} ln(\mu_i - \lambda_i)$.

We can formulate our mechanism design problem as follows: design a truthful mechanism that gives a fair and Pareto optimal allocation. Solving this problem involves finding an allocation algorithm and a payment scheme that maximizes the objective function $Q(\lambda)$ according to the computer bids b_i and motivates all the computers to report their true values t_i. A mechanism can be formally characterized as follows:

Definition 3.2 (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 3.3 (Truthful mechanism) A mechanism is called *truthful* if for every agent i of type t_i and for every bids b_{-i} of the 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 3.4 (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 3.5 (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).

Proceedings of the Second IEEE International Symposium on Network Computing and Applications (NCA'03)
0-7695-1938-5/03 $17.00 © 2003 IEEE

4. The load balancing mechanism

In our mechanism design problem the computer's true value is represented by a single real valued parameter. A general method to design truthful mechanisms for such problems was proposed by Archer and Tardos in [1]. We use their method to design our load balancing mechanism. According to this method, to obtain a truthful mechanism we must find an output function satisfying two conditions: (a) it maximizes $Q(\lambda)$ 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 maximizes the objective function $Q(\lambda)$ and produces a feasible allocation. Then we will show that this output function is decreasing in the bids.

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

$$\max_{\lambda} Q(\lambda) \tag{3}$$

subject to the constraints defined by the feasibility conditions i)-iii) from Definition 2.1.

In an earlier paper [8] we obtained the solution of this problem and we derived an algorithm for computing it. For the clarity of presentation we describe the theorem that gives the solution using the notations in this paper. We also present a partial proof of this theorem because some of the equations will be used to prove our results in Theorem 4.2 and Theorem 4.3 below. We now present the theorem.

Theorem 4.1 [8] Assuming that computers are ordered in increasing order of their bids ($b_1 \leq b_2 \leq \ldots \leq b_n$) the solution of the optimization problem (3) is given by:

$$\lambda_i = \begin{cases} \frac{1}{b_i} - \frac{\sum_{i=1}^{c} 1/b_i - \Phi}{n} & \text{if} \quad 1 \leq i < c \\ 0 & \text{if} \quad c \leq i \leq n \end{cases} \tag{4}$$

where c is the minimum index that satisfies the inequality: $\frac{1}{b_c} \leq \frac{\sum_{k=1}^{c} 1/b_k - \Phi}{c}$

We maximize the convex function $Q(\lambda)$ 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 [12]. Let $\alpha \geq 0$, $\eta_i \geq 0$, $\gamma_i \geq 0$ $i = 1, \ldots, n$ denote the Lagrange multipliers [12]. We consider the Lagrangian function:

$$L(\lambda_1, \lambda_2, \ldots, \lambda_n, \alpha, \eta_1, \ldots, \eta_n, \gamma_1, \ldots, \gamma_n) = \sum_{i=1}^{n} ln(\mu_i - \lambda_i) - \alpha(\sum_{i=1}^{n} \Phi - \lambda_i) - \sum_{i=1}^{n} \eta_i(\lambda_i - \mu_i) - \sum_{i=1}^{n} \gamma_i(-\lambda_i)$$

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

$$\frac{\partial L}{\partial \lambda_i} = 0 \qquad \frac{\partial L}{\partial \alpha} = 0 \tag{5}$$

$$\eta_i(\lambda_i - \mu_i) = 0, \quad \eta_i \geq 0, \quad \lambda_i - \mu_i \leq 0, \qquad i = 1, \ldots, n \tag{6}$$

$$\gamma_i(-\lambda_i) = 0, \quad \gamma_i \geq 0, \quad -\lambda_i \leq 0, \qquad i = 1, \ldots, n \tag{7}$$

The stability condition requires $\mu_i > \lambda_i$, $i = 1, \ldots, n$. This implies $\eta_i = 0$, $i = 1, \ldots, n$. These conditions become:

$$\alpha = F_i(\lambda_i), \quad \text{if} \quad \lambda_i > 0 \qquad 1 \leq i \leq n \tag{8}$$

$$\alpha \leq F_i(\lambda_i), \quad \text{if} \quad \lambda_i = 0 \qquad 1 \leq i \leq n \tag{9}$$

$$\sum_{i=1}^{n} \lambda_i = \Phi, \quad \lambda_i \geq 0, \quad i = 1, \ldots, n \tag{10}$$

Based on the above theorem an algorithm that solves the optimization problem (3) can be derived. We now outline the algorithm in [8] using the notations above.

COOP algorithm:

Input: Bids submitted by computers: $b_1, b_2, \ldots b_n$;
Total arrival rate: Φ
Output: Load allocation: $\lambda_1, \lambda_2, \ldots \lambda_n$;

1. Sort the computers in increasing order of their bids ($b_1 \leq b_2 \leq \ldots \leq b_n$);
2. $c \leftarrow \frac{\sum_{i=1}^{n} 1/b_i - \Phi}{n}$;
3. **while** ($c > 1/b_n$) **do**
   $\lambda_n \leftarrow 0$;
   $n \leftarrow n - 1$;
   $c \leftarrow (c - \frac{\mu_{n+1}}{n+1})\frac{n+1}{n}$;
4. **for** $i = 1, \ldots, n$ **do**
   $\lambda_i \leftarrow 1/b_i - c$;

The allocation $\lambda(b) = (\lambda_1(b), \lambda_2(b), \ldots, \lambda_n(b))$ computed by the COOP algorithm provides Pareto optimality and fairness. This allocation 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. In the case some of the computers lie we expect worse performance (i.e. higher expected response times and lower fairness index).

We now present our load balancing mechanism.

Definition 4.1 (The fair load balancing mechanism) The mechanism that solves the load balancing problem is defined by the following two functions:

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

(ii) The payment function is given by:

$$P_i(b_{-i}, b_i) = b_i\lambda_i(b_{-i}, b_i) + \int_{b_i}^{\infty} \lambda_i(b_{-i}, x)dx \tag{11}$$

The COOP algorithm gives an output function that maximizes $Q(\lambda)$. In order to obtain a truthful mechanism satisfying voluntary participation we need to 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.

Theorem 4.2 The output function $\lambda(b) = (\lambda_1(b), \lambda_2(b), \ldots, \lambda_n(b))$ computed by the COOP algorithm is decreasing, that means each $\lambda_i(b_{-i}, b_i)$ is a decreasing function of b_i for all i and b_{-i}.

Proof: In Appendix.

Theorem 4.3 (Truthfulness and voluntary participation) The fair load balancing mechanism is truthful and satisfies the voluntary participation condition.

Proof: In Appendix.

For our mechanism we use a payment function similar to [1]. The first term, $b_i\lambda_i(b_{-i}, b_i)$, of the payment function in equation (11) 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:

$$P_t = profit_i(t_i, (b_{-i}, t_i)) = \int_{t_i}^{\infty} \lambda_i(b_{-i}, x)dx$$

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:

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

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

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

Since 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.

Based on the above theorems we derived a protocol called FAIR-LBM that implements our fair load balancing mechanism. We now give a high level description of the FAIR-LBM protocol.

FAIR-LBM Protocol (executed periodically):

Dispatcher (mechanism):

1. Send a request for bid message (`ReqBid`) to each computer in the system.
2. When the bids from all the computers are received then:
   2.1. Compute the allocation using COOP algorithm.
   2.2. Compute the payments P_i for each computer using equation (11).
   2.3. Send P_i to each computer i.

Computer (agent) i:

1. If a `ReqBid` message is received then send bid b_i to the dispatcher.
2. Receive P_i from dispatcher.
3. Compute $profit_i$.

Because the COOP 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. The FAIR-LBM protocol is executed periodically or when there is a change in the total job arrival rate. Between two executions of this protocol the jobs will be allocated to computers by the dispatcher according to the allocation computed by COOP. Computers will receive the maximum profit only when they report the true value.

5. Experimental results

In this section we investigate the effectiveness of our load balancing mechanism by simulation. We simulate a system consisting of 16 heterogeneous computers with four different processing rates. The distributed system configuration is presented in Table 1. The first row of this table 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 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.

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

Table 1. System configuration.

Response time

First we study the influence of false bidding on the expected response times at each computer. As we pointed out in the previous section, if all the computers report their true values then the fair Pareto optimal allocation is obtained. This means all the computers have the same minimum possible expected response time. We expect large differences in the

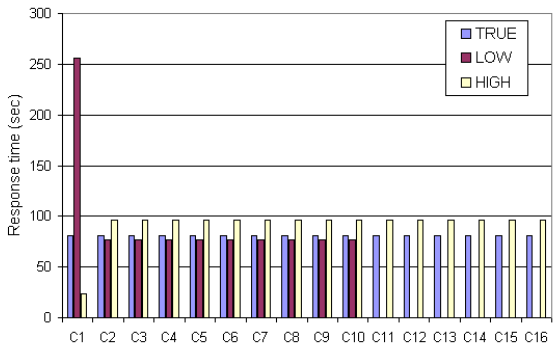

Figure 1. Response time at each computer (high system load)

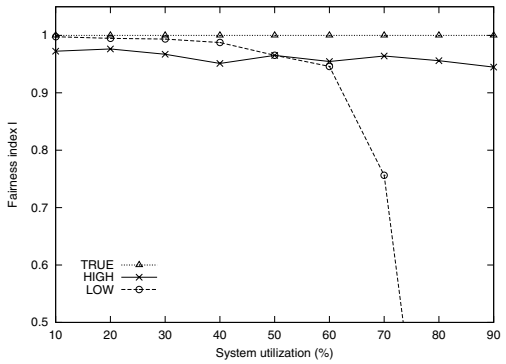

Figure 2. Fairness index vs. system utilization.

response times if one or more computers lie. In our experiments we consider that the fastest computer C_1 declares false bids. The fastest computer, C_1 has $t_1 = 1/\mu_1 = 7.69$ as its true value.

In Figure 1 we present the expected response time at each computer for high system utilization ($\rho = 70\%$). and three types of bidding for C_1: true bidding, underbidding and overbidding. *System utilization* (ρ) is defined as the ratio of total arrival rate to aggregate processing rate of the system: $\rho = \frac{\Phi}{\sum_{i=1}^{n} \mu_i}$.

In the first experiment all computers including C_1 bid their true value and we obtain equal expected response times of 80.44 sec for all the computers.

In the second experiment C_1 bids 7% lower than its true value. In this case C_1's response time increases drastically (256.41 sec). This is due to computer C_1 overloading. The overloading occurs because C_1 bids lower, that means it reports a higher value for its processing rate. The algorithm will allocate more jobs to C_1 increasing its response time. Computers C_2 to C_{10} obtain the same response time, lower than in the first experiment (in which all the computers reported their true values). The slowest computers are not utilized, they receive no jobs. This variations in individual response times is reflected in the overall expected response time of the system. The value of the overall expected response time increases from 80.44 sec (true bidding) to 125.69 sec (underbidding).

In the third experiment C_1 bids 33% higher than its true value. In this case C_1's response time decreases to 23.31 sec but the response times of all the other computers increase to 96.15 sec. This is because c_1 is underloaded. The value of the overall expected response time increases from 80.44 sec to 82.48 sec. It can be observed that small deviations from the true value of only one computer may lead to large variations in individual response times and in the overall expected response time. 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.

Fairness

In Figure 2 we present the variations in the fairness index. 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. In the case of overbidding the fairness index is maintained around 96% for the whole range of the system utilization. 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.

Payment structure and frugality

The profit gained by each computer at high system loads (ρ = 70%) is presented in Figure 3. It can be observed that the profit at C_1 is maximum if it bids the true value, 2% lower if it bids higher and 1% lower if it bids lower. Computer C_1 is penalized by our mechanism because 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, thus they will not gain anything. These computers will be utilized in the case when C_1 overbids and when it bids the true value, 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 property of a mechanism is its *frugality*. An informal definition of frugality is as follows. We say that a mechanism is *frugal* if the mechanism's payments are small by some measure [2]. In other words this property tells us how efficient a mechanism is. The mechanism is interested in keeping its payments as small as possible.

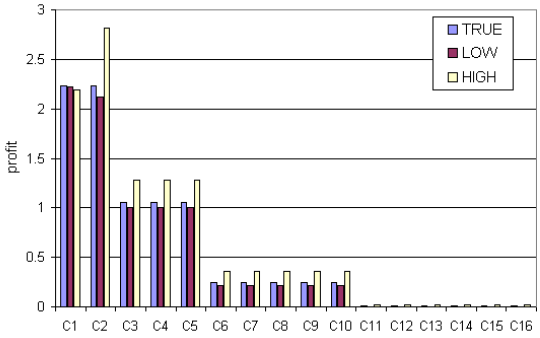

Figure 3. Profit for each computer (high system load)

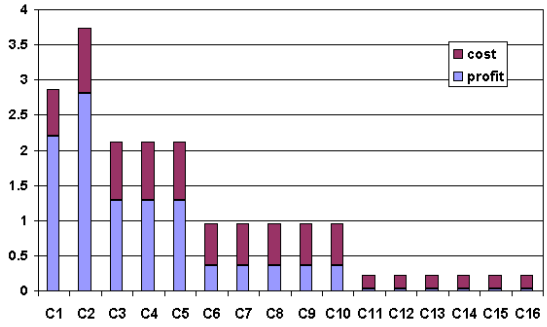

Figure 4. Payment structure for each computer (C_1 bids higher)

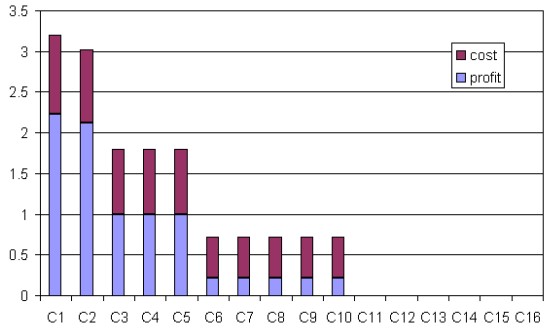

Figure 5. Payment structure for each computer (C_1 bids lower)

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 Figure 4 and 5 we present the cost and profit as fractions of the payment received by each computer at high system load. In these figures we are interested to see how close to the cost is the payment to each computer. It can be observed that the cost incurred by C_1 when it bids higher is about 24% of the payment. In the case when C_1 bids lower its cost is about 31% of the payment. For the other computers the cost is between 25% and 99% when C_1 bids higher and between 30% and 75% when C_1 bids lower. For the distributed system considered in these experiments (high system load) the highest payment given to a computer is about 3 times its cost.

In Figure 6 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 30% of the payment at 90% system utilization which is the smallest percentage. The percentage of cost increases to 72% at 10% system utilization. The total payment given by our mechanism to the computers is less than 3 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.

6. Conclusion

In this paper we studied the problem of fair load balancing in heterogeneous distributed systems where computational resources are owned by self interested agents. We derived a truthful mechanism that gives a fair and Pareto optimal solution to this problem. We proved that our mechanism satisfies the voluntary participation condition. We derived a fair load balancing protocol (FAIR-LBM) that implements our mechanism. Finally, we investigated the effectiveness of the proposed mechanism by simulation. The results show that our mechanism is frugal.

We view this work as a preliminary step in developing load allocation mechanisms in more complex settings. In our future work we plan to address this issue as well as the implementation of our mechanism in a real distributed system.

A. Appendix

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

Proof of Theorem 4.2

We consider $\lambda_i(b_{-i}, b_i)$ as a single variable function of b_i by fixing the other bids 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 $\frac{1}{\tilde{\mu}_i} > \frac{1}{\mu_i}$ i.e. $\tilde{\mu}_i < \mu_i$. Let $\tilde{\lambda}_i > 0$ and $\lambda_i > 0$ be the

Proceedings of the Second IEEE International Symposium on Network Computing and Applications (NCA'03)
0-7695-1938-5/03 $17.00 © 2003 IEEE

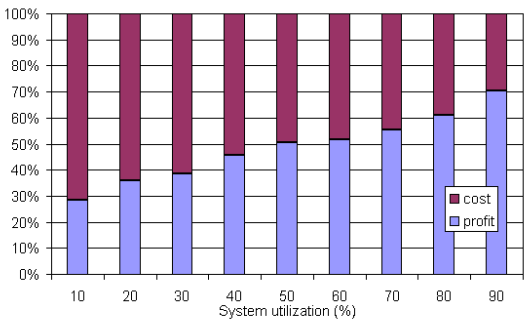

Figure 6. Total payment vs. system utilization.

loads allocated by the COOP 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 COOP 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 $\tilde{F}_i(\tilde{\lambda}_i) > F_i(\lambda_i)$. 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 (8), (9) imply that:

$$F_i(\lambda_i) = \alpha \quad \text{and} \quad \tilde{F}_i(\tilde{\lambda}_i) = \tilde{\alpha} \tag{12}$$

Since $\lambda_l \geq \tilde{\lambda}_l$ and $\lambda_l > 0$ the Kuhn-Tucker conditions for optimality (8), (9) imply that:

$$F_l(\lambda_l) = \alpha \quad \text{and} \quad \tilde{F}_l(\tilde{\lambda}_l) \geq \tilde{\alpha} \tag{13}$$

Combining equations (12)-(13) we obtain:

$$F_i(\lambda_i) = F_l(\lambda_l) \quad \text{and} \quad \tilde{F}_l(\tilde{\lambda}_l) \geq \tilde{F}_i(\tilde{\lambda}_i) \tag{14}$$

Because $\tilde{\lambda}_l \leq \lambda_l$ we have $1/(\mu_l - \tilde{\lambda}_l) \leq 1/(\mu_l - \lambda_l)$. This implies $\tilde{F}_l(\tilde{\lambda}_l) \leq F_l(\lambda_l)$. Using this we obtain the following equation: $F_i(\lambda_i) \geq \tilde{F}_i(\tilde{\lambda}_i)$. This is a contradiction because $\tilde{\lambda}_i \geq \lambda_i$ and $\tilde{F}_i(\tilde{\lambda}_i) > F_i(\lambda_i)$. □

Proof of Theorem 4.3

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, so 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. $\int_{b_{imin}}^{\infty} \lambda_i(b_{-i}, x)dx < \infty$ where b_{imin} is the bid that corresponds to $\lambda_i = \Phi$. Thus, our mechanism admits voluntary participation and the payments are given by equation (11). □

References

[1] 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.

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

[3] R. Buyya, D. Abramson, and J. Giddy, "A Case for Economy Grid Architecture for Service-Oriented Grid Computing", *Proc. of the 10th IEEE Heterogeneous Computing Workshop*, pp. 776-790, April 2001.

[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", *Proc. of the 32nd Annual ACM Symp. on Theory of Computing*, pp. 218-227, May 2000.

[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. Gross and C. M. Harris, *Fundamentals of Queueing Theory*, Wiley-Interscience, New York, NY, 1998.

[8] 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. 52-61, April 2002.

[9] 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.

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

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

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