

A Strategyproof Mechanism for Scheduling Divisible Loads in Distributed Systems

Daniel Grosu and Thomas E. Carroll
Department of Computer Science
Wayne State University
Detroit, Michigan 48202, USA
{dgrosu, tec}@cs.wayne.edu

Abstract

An important scheduling problem is the one in which there are no dependencies between tasks and the tasks can be of arbitrary size. This is known as the divisible load scheduling problem and was studied extensively in recent years resulting in a cohesive theory called Divisible Load Theory (DLT). In this paper we augment the existing divisible load theory with incentives. We develop a strategyproof mechanism for scheduling divisible loads in distributed systems assuming a bus type interconnection and a linear cost model for the processors. The mechanism provides incentives to processors such that it is beneficial for them to report their true processing power and process the assigned load using their full processing capacity. We define the strategyproof mechanism and prove its properties. We simulate and study the implementation of the mechanism on systems characterized by different parameters.

1. Introduction

Scheduling tasks in a distributed computing system is one of the most challenging problems that need to be solved when running applications. By efficiently scheduling the tasks the performance and the efficiency of the system is improved. The task scheduling problem takes many forms depending on the characteristics of the tasks to be scheduled, the characteristics of the machines comprising the system and the objective function that needs to be optimized. One type of scheduling problem is the one in which there are no dependencies between tasks and the tasks can be of arbitrary size. This is the case for several applications in science and engineering in which the total load can be split into an arbitrary number of independent loads. These loads require the same type of processing and can be assigned to any computer in the system. In practice it corresponds to

the widely used master-slave model of parallel computation. The above scheduling problem can be characterized using the divisible load model which was studied extensively in recent years resulting in a cohesive theory called *Divisible Load Theory* (DLT). Divisible load theory provides analytical results and optimal algorithms for scheduling loads on various types of platforms such as bus, tree, star and linear networks [5].

The scheduling algorithms developed within DLT assume that the participants (in this case, processors) are obedient. Thus, they report to the scheduler the true parameters of their processing facilities (e.g. processing power). The scheduler makes the allocation decision according to the values reported by the processors or by the owners of these processors. This assumption is not valid in real life situations where these participants have no a-priori motivation for cooperation and they are tempted to manipulate the scheduling algorithm if it is beneficial to do so. This behavior may lead to poor system performance and inefficiency. Thus, we need to develop new algorithms and protocols that address the self interest of the participants. Unlike the traditional DLT algorithms, the new protocols must deal with the possible manipulations. Also, the system must provide incentives to agents to participate in the given algorithm. The solution of these kinds of problems comes from economics, more precisely from *mechanism design theory* [18]. The scope of this theory is to provide tools and methods to design protocols for self interested agents. Of interest are the so called *strategyproof mechanisms* in which the participants maximize their own utilities only if they report their true parameters and follow the given algorithm. In a general mechanism each participant has a privately known function called *valuation* which quantifies the agent's benefit or loss. Payments are designed and used to motivate the participants to report their true valuations. The goal of each participant is to maximize the sum of her valuation and payment. As an example consider several resource providers that offer

computer services. We assume that each resource is characterized by its job processing rate. An allocation mechanism is strategyproof if a resource owner maximizes her utility only by reporting the true resource processing rate to the mechanism. The optimal utility is independent of the values reported by the other participating processors.

In this paper we consider the design of strategyproof scheduling mechanisms in the context of divisible load theory. To our knowledge this is the first attempt to augment DLT with incentives. We develop a strategyproof mechanism that provides incentives to the processors to participate and report their true processing capabilities to the scheduler. The processors gain the maximum profit by executing the task only if they are truthfully reporting the private values characterizing their processing capabilities.

Related work. The divisible load scheduling problem was studied extensively in recent years resulting in a cohesive theory called Divisible Load Theory. A reference book on DLT is [5]. Two recent surveys on DLT are [6] and [19]. This theory has been used for scheduling loads on heterogeneous distributed systems in the context of different applications such as image processing [13], databases [7], linear algebra [8], and multimedia broadcasting [4]. Scheduling divisible loads in grids has been investigated in [22]. New results and open research problems in DLT are presented in [3]. All these works assumed that the participants in the load scheduling algorithms are obedient and follow the algorithm. Recently, several researchers considered the mechanism design theory to solve several computational problems that involve self interested participants. These problems include resource allocation and task scheduling [16, 20, 21], routing [9] and multicast transmission [10]. In their seminal paper, Nisan and Ronen [17] considered for the first time the mechanism design problem in a computational setting. They proposed and studied a VCG(Vickrey-Clarke-Groves) type mechanism for the shortest path in graphs where edges belong to self interested agents. They also provided a mechanism for solving the task scheduling on unrelated machines problem. A general framework for designing strategyproof mechanisms for one parameter agent was proposed by Archer and Tardos [1]. They developed a general method to design strategyproof mechanisms for optimization problems that have general objective functions and restricted form for valuations. In a subsequent paper [2] the same authors investigated the frugality of shortest path mechanisms. Grosu and Chronopoulos [12] derived a strategyproof mechanism that gives the overall optimal solution for the static load balancing problem in distributed systems. The results and the challenges of designing distributed mechanisms are surveyed in [11]. The *strategyproof computing* paradigm proposed in [14] considers the self-interest and incentives of participants in distributed

computing systems. Ng *et al.* [15] proposed a strategyproof system for dynamic resource allocation in data staging.

Our contributions. The main contribution of this paper is to show how existing divisible load theory can be augmented with incentives. We develop a strategyproof mechanism for scheduling divisible loads in distributed systems assuming a bus type interconnection and a linear cost model for the processors. We define the mechanism and prove its properties. We simulate and study the implementation of the mechanism on systems characterized by different parameters.

Organization. The paper is structured as follows. In Section 2 we present a description of the divisible load scheduling problem. In Section 3 we present the framework used to design our mechanism. In Section 4 we present and discuss the proposed strategyproof mechanism. In Section 5 we study by simulation the proposed scheduling mechanism. In Section 6 we draw conclusions and present future directions.

2. Divisible Load Scheduling Problem

We consider a distributed system with a bus interconnection and a master processor. The set of processors is $S = \{P_0, P_1, \dots, P_m\}$ where P_0 is the master processor. Each processor P_i , $i = 1, \dots, m$, is characterized by w_i , the time taken by P_i to process a unit load. The fraction of load assigned to processor P_i is α_i . The amount of time in which P_i executes its assigned load is given by $\alpha_i w_i$. This corresponds to a linear cost model for the processors. To transfer α_i units of load from the master to P_i takes $\alpha_i z$ units of time, where z is the time to communicate a unit load from P_0 to any other processor. We assume that the master has no processing capability and it can communicate with a single processor at a given time (i.e. we assume the one-port model). We denote by $T_i(\alpha)$ the finish time of processor P_i , defined as the time required to send load α_i from P_0 to P_i plus the time of processing the load at P_i .

In Fig. 1 we present a diagram showing the execution on this system.

The scheduling problem denoted as BUS-LINEAR is to determine the load allocation $\alpha = (\alpha_1, \alpha_2, \dots, \alpha_m)$ which minimizes the total execution time (or makespan) $T(\alpha) = \max\{T_1(\alpha), T_2(\alpha), \dots, T_m(\alpha)\}$. The finish time of processor P_i is denoted by $T_i(\alpha)$ and is given by:

$$T_i(\alpha) = z \sum_{j=1}^i \alpha_j + \alpha_i w_i \quad (1)$$

The BUS-LINEAR problem can be formally described as follows:

$$\min_{\alpha} T(\alpha) \quad (2)$$

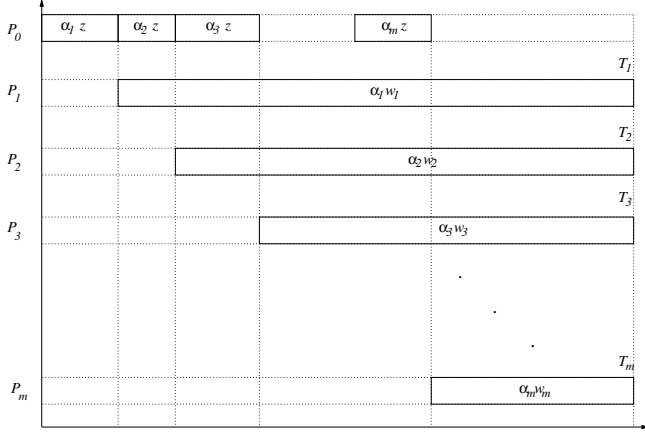


Figure 1. Timing diagram for the load execution.

subject to the constraints:

$$\alpha_i \geq 0, \quad i = 1, \dots, m \quad (3)$$

$$\sum_{i=1}^m \alpha_i = 1 \quad (4)$$

The following theorems proved in [5] characterize the optimal solution.

Theorem 2.1. The optimal solution for the BUS-LINEAR problem is obtained when all processors participate and they all finish executing their assigned load at the same time, i.e. $T_1(\alpha) = T_2(\alpha) = \dots = T_m(\alpha)$.

Theorem 2.2. Any load allocation order is optimal for the BUS-LINEAR problem.

The following algorithm solves the BUS-LINEAR problem:

BUS-LINEAR Algorithm

Input: Time to process a unit load: w_1, w_2, \dots, w_m ;

Time to communicate a unit load: z ;

Output: Load fractions: $\alpha_1, \alpha_2, \dots, \alpha_m$;

1. **for** $j = 1, \dots, m - 1$ **do**

$$k_j \leftarrow \frac{w_j}{z + w_{j+1}}$$

2. $\alpha_1 \leftarrow \frac{1}{1 + \sum_{i=1}^{m-1} \prod_{j=1}^i k_j}$

3. **for** $i = 2, \dots, m$ **do**

$$\alpha_i = \alpha_1 \prod_{j=1}^{i-1} k_j$$

The algorithm is executed by the master processor P_0 when a new load needs to be processed. In order to compute the allocation P_0 requires information about the network and about the processing capabilities. The processors are assumed to report their true processing times for one unit load

to the master. This may not happen if the processors are owned and managed by different entities or agents. They may report different values in order to gain profit. In the next sections we present the design of a mechanism that gives incentive to the agents to report their true processing capabilities.

3. Mechanism Design Framework

In this section we introduce the main concepts of mechanism design theory. 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 [17]. In the following we define such problem.

A *mechanism design problem for one parameter agents* is characterized by:

- (i) A finite set L of allowed outputs. The output is a vector $\alpha(\mathbf{b}) = (\alpha_1(\mathbf{b}), \alpha_2(\mathbf{b}), \dots, \alpha_m(\mathbf{b}))$, $\alpha(\mathbf{b}) \in L$, computed according to the agents' bids, $\mathbf{b} = (b_1, b_2, \dots, b_m)$. Here, b_i is the value (bid) reported by agent i to the mechanism.
- (ii) Each agent i , ($i = 1, \dots, m$), has a privately known parameter t_i called the *true value* and a publicly known parameter $\tilde{t}_i \geq t_i$ called the *execution value*. The preferences of agent i are given by a function called *valuation* $V_i(\alpha, \tilde{\mathbf{t}})$.
- (iii) Each agent goal is to maximize its *utility*. The utility of agent i is $U_i(\mathbf{b}, \tilde{\mathbf{t}}) = Q_i(\mathbf{b}, \tilde{\mathbf{t}}) + V_i(\alpha(\mathbf{b}), \tilde{\mathbf{t}})$, where Q_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 mechanism knows \tilde{t}_i , $i = 1, 2, \dots, m$.
- (iv) The goal of the mechanism is to select an output α that optimizes a given cost function $g(\mathbf{b}, \alpha)$.

Definition 3.1. (*Mechanism*) A *mechanism with verification* is characterized by two functions:

- (i) The *output function* $\alpha(\mathbf{b}) = (\alpha_1(\mathbf{b}), \alpha_2(\mathbf{b}), \dots, \alpha_m(\mathbf{b}))$. This function has as input the vector of agents' bids $\mathbf{b} = (b_1, b_2, \dots, b_m)$ and returns an output $\alpha \in L$.
- (ii) The *payment function* $Q(\mathbf{b}, \tilde{\mathbf{t}}) = (Q_1(\mathbf{b}, \tilde{\mathbf{t}}), Q_2(\mathbf{b}, \tilde{\mathbf{t}}), \dots, Q_m(\mathbf{b}, \tilde{\mathbf{t}}))$, where $Q_i(\mathbf{b}, \tilde{\mathbf{t}})$ is the payment handed by the mechanism to agent i .

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) .

Definition 3.2. (*Strategyproof mechanism*) A mechanism is called *strategyproof* if for every agent i of type t_i and for every bids \mathbf{b}_{-i} of the other agents, the agent's utility is maximized when she declares her real type t_i (i.e. truth-telling is a dominant strategy).

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

Definition 3.3. (*Voluntary participation mechanism*) We say that a mechanism satisfies the *voluntary participation condition* if $U_i((\mathbf{b}_{-i}, t_i), (\mathbf{b}_{-i}, t_i)) \geq 0$ for every agent i , true values t_i , and other agents' bids \mathbf{b}_{-i} (i.e. truthful agents never incur a loss).

4. The proposed mechanism

Each processor (agent) P_i participating in the mechanism is characterized by the *true value* t_i which is equal to the time to process a unit load, i.e. $t_i = w_i$. Only P_i knows w_i . We denote by \mathbf{w} the vector of 'true' unit load processing times of the processors ($\mathbf{w} = (w_1, w_2, \dots, w_m)$). The mechanism asks each P_i to report its value b_i (the time to process a unit load). The processors may not report the true value. After all the processors report their values the mechanism computes an output function (i.e. the loads assigned to computers), $\alpha(\mathbf{b}) = (\alpha_1(\mathbf{b}), \alpha_2(\mathbf{b}), \dots, \alpha_m(\mathbf{b}))$, ($\sum_1^m \alpha_i(\mathbf{b}) = 1$) according to the processors' bids such that the makespan is minimized. A processor P_i may choose to process the load allocated to it at a different processing rate given by its execution value \tilde{w}_i , where $\tilde{w}_i \geq w_i$. Thus, processor P_i may process the assigned load at a slower rate than its true processing rate. The goal of a strategyproof mechanism with verification is to give incentives to agents such that it is beneficial for them to report their true values and process the assigned loads using their full processing capacity. After the loads are processed, the execution value \tilde{w}_i for every processor P_i is known to the mechanism. Once the mechanism knows the execution values it computes and hands a payment $Q_i(\mathbf{b}, \tilde{\mathbf{w}})$ to each processor P_i . All processors know the mechanism and the algorithm used to compute the output function (allocation). We assume an arbitrary order on the set of processors, which will not affect the optimality of allocation (according to Theorem 2.2).

Processor P_i 's ($i = 1, 2, \dots, m$) valuation is defined as:

$$V_i(\alpha(\mathbf{b}), \tilde{\mathbf{w}}) = -\alpha_i \tilde{w}_i \quad (5)$$

which is equivalent to the negation of P_i 's actual time required to process load α_i . The greater the processing time, the smaller the valuation. This can be considered to be the cost incurred by P_i in processing α_i . We assume that each processor wants to choose its strategy (what value b_i to report) such that its utility is maximized. The utility for each

processor is defined as the payment received from the mechanism plus the valuation determined by the load allocated to it:

$$U_i(\mathbf{b}, \tilde{\mathbf{w}}) = Q_i(\mathbf{b}, \tilde{\mathbf{w}}) + V_i(\alpha(\mathbf{b}), \tilde{\mathbf{w}}) \quad (6)$$

The goal is to design a strategyproof mechanism that minimizes the makespan of the system composed of the m processors. This involves finding an allocation algorithm and a payment scheme that minimizes the makespan according to the processors' bids \mathbf{b} and motivates all the processors to bid their true values w_i and process the load at their full processing capacity (i.e. $\tilde{w}_i = w_i$). For our mechanism we use the optimal allocation algorithm described in Section 2. Using this algorithm and the method presented in [17] we designed a strategyproof mechanism with verification that minimizes the makespan. In the following we define the mechanism.

Definition 4.1. (*DLS-BL Mechanism*) The DLS-BL mechanism is defined by the following two functions:

- (i) The allocation function given by the BUS-LINEAR algorithm.
- (ii) The payment function given by:

$$Q_i(\mathbf{b}, \tilde{\mathbf{w}}) = C_i(\mathbf{b}, \tilde{\mathbf{w}}) + B_i(\mathbf{b}, \tilde{\mathbf{w}}) \quad (7)$$

where the function $C_i(\mathbf{b}, \tilde{\mathbf{w}}) = \alpha_i \tilde{w}_i$ is called the *compensation* function for processor P_i ; and the function $B_i(\mathbf{b}, \tilde{\mathbf{w}}) = T_{-i}(\alpha(\mathbf{b}_{-i}), \mathbf{b}_{-i}) - T(\alpha(\mathbf{b}), (\mathbf{b}_{-i}, \tilde{w}_i))$ is called the *bonus* for processor P_i .

The function $T_{-i}(\alpha(\mathbf{b}_{-i}), \mathbf{b}_{-i})$ is the optimal total execution time when processor P_i is not used in the allocation. Thus, the bonus for a processor is equal to its contribution in reducing the total execution time.

For our scheduling mechanism we can state the following theorems.

Theorem 4.1. (*Strategyproofness*) The DLS-BL mechanism is strategyproof.

Proof. Assuming a vector of bids \mathbf{b} , the utility of processor P_i is:

$$\begin{aligned} U_i(\mathbf{b}, \tilde{\mathbf{w}}) &= Q_i(\mathbf{b}, \tilde{\mathbf{w}}) + V_i(\alpha(\mathbf{b}), \tilde{\mathbf{w}}) \\ &= T_{-i}(\alpha(\mathbf{b}_{-i}), \mathbf{b}_{-i}) - T(\alpha(\mathbf{b}), (\mathbf{b}_{-i}, \tilde{w}_i)) + \alpha_i(\mathbf{b}) \tilde{w}_i - \alpha_i(\mathbf{b}) \tilde{w}_i \\ &= T_{-i}(\alpha(\mathbf{b}_{-i}), \mathbf{b}_{-i}) - T(\alpha(\mathbf{b}), (\mathbf{b}_{-i}, \tilde{w}_i)) \quad (8) \end{aligned}$$

We consider two possible situations:

- (i) $\tilde{w}_i = w_i$ i.e. processor P_i processes its assigned load using its full processing capability.

If processor P_i bids its true value w_i then its utility U_i^t is:

$$U_i^t = T_{-i}(\alpha(\mathbf{b}_{-i}), \mathbf{b}_{-i}) - T(\alpha(\mathbf{b}_{-i}, w_i), (\mathbf{b}_{-i}, \tilde{w}_i))$$

$$= T_{-i}(\alpha(\mathbf{b}_{-i}), \mathbf{b}_{-i}) - T_i^t \quad (9)$$

If processor P_i bids lower ($b_i^l < w_i$) then its utility U_i^l is:

$$\begin{aligned} U_i^l &= T_{-i}(\alpha(\mathbf{b}_{-i}), \mathbf{b}_{-i}) - T(\alpha(\mathbf{b}_{-i}, b_i^l), (\mathbf{b}_{-i}, \tilde{w}_i)) \\ &= T_{-i}(\alpha(\mathbf{b}_{-i}), \mathbf{b}_{-i}) - T_i^t \quad (10) \end{aligned}$$

We want to show that $U_i^l \geq U_i^h$, which reduces to show that $T_i^l \geq T_i^h$. Because T_i^t is the minimum possible value for the processing time (from the optimality of BUS-LINEAR algorithm), by bidding a lower value, processor P_i gets more load and the total execution time is increased, thus $T_i^l \geq T_i^t$.

If processor P_i bids higher ($b_i^h > w_i$) then its utility U_i^h is:

$$\begin{aligned} U_i^h &= T_{-i}(\alpha(\mathbf{b}_{-i}), \mathbf{b}_{-i}) - T(\alpha(\mathbf{b}_{-i}, b_i^h), (\mathbf{b}_{-i}, \tilde{w}_i)) \\ &= T_{-i}(\alpha(\mathbf{b}_{-i}), \mathbf{b}_{-i}) - T_i^h \quad (11) \end{aligned}$$

By bidding a higher value processor P_i gets less load and thus more load will be assigned to the other processors. Due to the optimality of allocation the total execution time increases i.e. $T_i^h \geq T_i^t$ and thus we have $U_i^t \geq U_i^h$.

(ii) $\tilde{w}_i > w_i$ i.e. processor P_i processes its assigned load at a slower rate thus increasing the total execution time. A similar argument as in case i) applies. \square

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.

Theorem 4.2. (*Voluntary participation*) The DLS-BL mechanism satisfies the voluntary participation condition.

Proof. The utility of processor P_i when it bids its true value w_i is:

$$U_i^t = T_{-i}(\alpha(\mathbf{b}_{-i}), \mathbf{b}_{-i}) - T(\alpha(\mathbf{b}_{-i}, w_i), (\mathbf{b}_{-i}, \tilde{w}_i)) \quad (12)$$

The total execution time T_{-i} is obtained by using all the other processors except processor i . By allocating the same amount of load, we get a higher execution time T_{-i} than in the case of using all the processors, with processor P_i bidding its true value (from the optimality of allocation). Thus $U_i^T \geq 0$. \square

We obtained a strategyproof scheduling mechanism that satisfies the voluntary participation condition. Because the optimal algorithm assumes a central dispatcher (master), the mechanism will be implemented in a centralized way as part of the master's code. The protocol assumes the existence of a payment infrastructure. In the following we present the protocol that implements the DLS-BL mechanism.

Protocol DLS-BL:

1. After the master processor P_0 collects all the bids it computes the allocation using the BUS-LINEAR algorithm.
2. Once the assigned load is finished the Master processor P_0 does the following:
 - 2.1. Determines the execution values for each processor, \tilde{w}_i .
 - 2.2. Computes the payments Q_i for each processor P_i using equation (7).
 - 2.3. Sends Q_i to each P_i .
3. Each processor receives its payment and evaluates its profit.

Processors will obtain the maximum profit only when they report the true value of their processing time.

5. Experimental results

In this section we study by simulation the proposed strategyproof scheduling mechanism. We consider two distributed systems composed of sixteen processors (P_1, \dots, P_{16}) and one master processor (P_0). The first system, called the 'fast' system, has a fast P_1 with $w_1 = 0.1$. The second system, called the 'slow' system, has a slow P_1 with $w_1 = 0.7$. The times to process a unit load w_i , $i = 1, \dots, 16$ are presented in Table 1. For both systems we assume that only P_1 cheats by reporting values different than its true processing time and by processing the load at a different rate than the true rate. The time to communicate a unit load from P_0 to any other processor is $z = 0.01$ for both systems. The low communication latency ensures that the system is computationally bound. If z is equal in magnitude or larger than the time to process a unit load at the fast processors, the system becomes communication bound, resulting in fewer processors performing work and negligible bonuses.

For each system, we examine eight cases:

- (1) $w_1 = b_1 = \tilde{w}_1$ (i.e., P_1 bids truthfully and it processes the load as reported);
- (2) $\tilde{w}_1 > w_1 = b_1$ (i.e., P_1 bids truthfully, but processes the load at a slower rate);
- (3) $w_1 < b_1 = \tilde{w}_1$ (i.e., P_1 bids a rate slower than its true rate, but processes the load at the reported rate);
- (4) $w_1 = \tilde{w}_1 < b_1$ (i.e., P_1 bids a rate slower than its true rate, but it processes the load at its true rate);

- (5) $w_1 < \tilde{w}_1 < b_1$ (i.e., P_1 processes its load slower than its true rate and bids a rate slower than its execution rate);
- (6) $w_1 < b_1 < \tilde{w}_1$ (i.e., P_1 processes the load slower than it bid and bids a rate slower than its true rate);
- (7) $b_1 < w_1 = \tilde{w}_1$ (i.e., P_1 bids a rate faster than its true rate, but processes the load at its true rate);
- (8) $b_1 < w_1 < \tilde{w}_1$ (i.e., P_1 bids a rate faster than its true rate and processes the load slower than its true rate).

The bid b_1 and the execution value \tilde{w}_1 for each case are presented in Table 2 (fast system) and Table 3 (slow system).

Figure 2 and 3 show the makespan (T) for the eight cases and the two types of systems, fast and slow. Notice that case (1) ($w_1 = \tilde{w}_1 = b_1$) results in the minimum makespan, while all other cases result in larger makespan. When $\tilde{w}_1 \leq b_1$ (cases (3), (4), and (5)), the makespan is increased by a small amount (0.6% for the slow system and 11% for the fast system) as the load allocated to P_1 is reduced and the load allocated to the other processors is increased. The impact is larger if the number of processors is reduced as there are fewer processors to distribute the load. In the remaining cases ($b_1 < \tilde{w}_1$), the system performance dramatically degrades as P_1 is overloaded and the other processors are underutilized. In these cases, P_1 is the processor which is slowing down the entire system. The increase in makespan is large (between 57% and 209%) and it is due to the impact of \tilde{w}_1 . Comparing the fast system

Table 1. The times to process a unit load.

w_1		w_2	$w_3 - w_5$	$w_6 - w_{10}$	$w_{11} - w_{16}$
fast	slow	0.1	0.2	0.5	1.0
0.1	0.7				

Table 2. Bids and execution values (fast system).

Case	1	2	3	4	5	6	7	8
b_1	0.1	0.1	0.3	0.3	0.3	0.3	0.05	0.05
\tilde{w}_1	0.1	0.3	0.3	0.1	0.2	0.4	0.1	0.2

Table 3. Bids and execution values (slow system).

Case	1	2	3	4	5	6	7	8
b_1	0.7	0.7	0.9	0.9	0.9	0.8	0.5	0.5
\tilde{w}_1	0.7	0.9	0.9	0.7	0.8	0.9	0.7	0.8

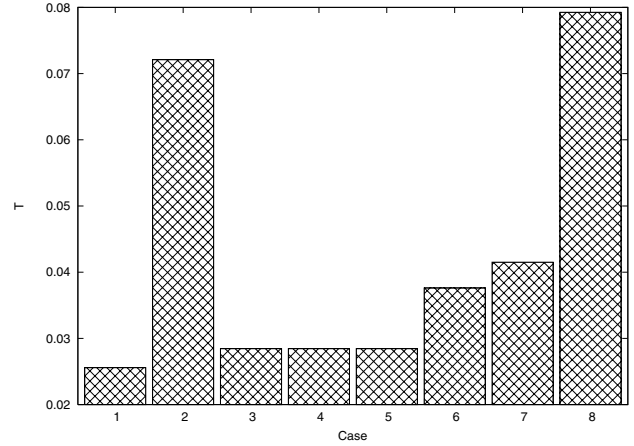


Figure 2. Makespan when P_1 cheats (fast system).

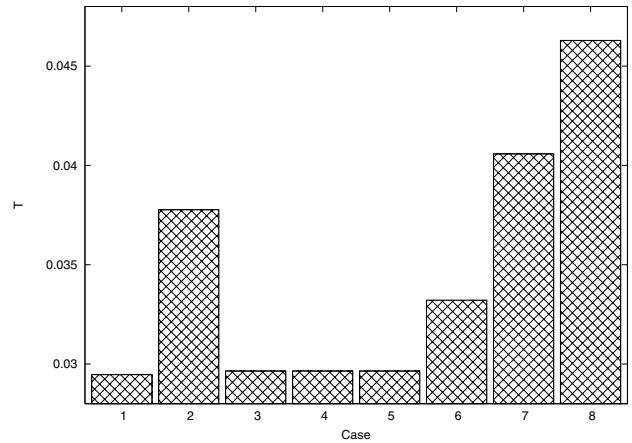


Figure 3. Makespan when P_1 cheats (slow system).

with the slow system, we notice that the performance degradation is *greater* for the fast system even though the relative rate change is similar or smaller than that for the slow system. This is because BUS-LINEAR algorithm allocates more work to faster processors than to slower processors.

The relationship between P_1 's utility and payment versus the eight cases for the fast and slow system is depicted in Figure 4 and Figure 5 respectively. As expected, case (1) ($w_1 = \tilde{w}_1 = b_1$) results in the greatest U_1 which means that when P_1 is not cheating it gets the maximum utility. In all the other cases P_1 obtains a lower U_1 . When $b_1 < \tilde{w}_1$ (cases (2), (6), (7), and (8)), the utility is negative due to the impact of \tilde{w}_1 on the makespan. In these cases, $B_1 < 0$ as $T_{-1}(\alpha(\mathbf{b}_{-1}), \mathbf{b}_{-1}) < T(\alpha(\mathbf{b}), (\mathbf{b}_{-1}, \tilde{w}_1))$. In the remain-

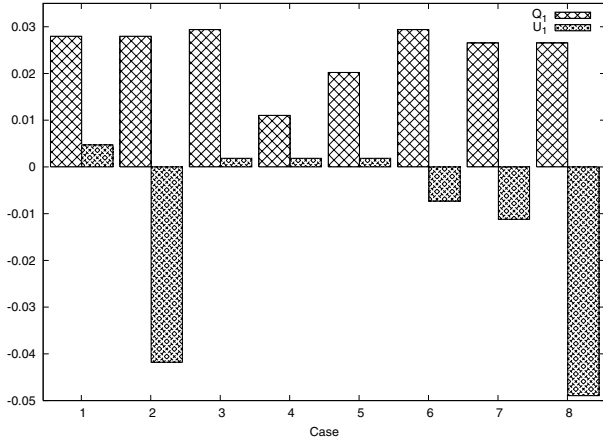


Figure 4. Payment and Utility of P_1 (fast system).

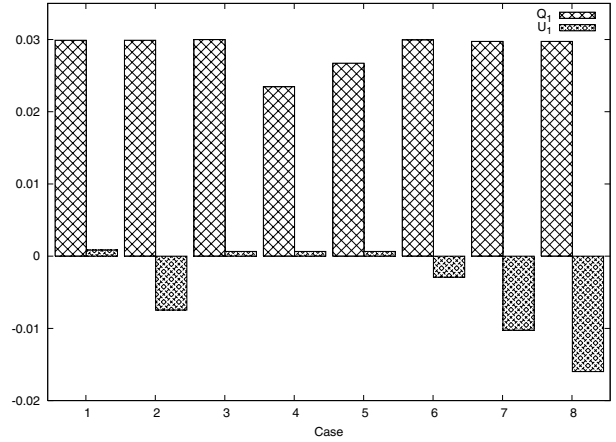


Figure 5. Payment and Utility of P_1 (slow system).

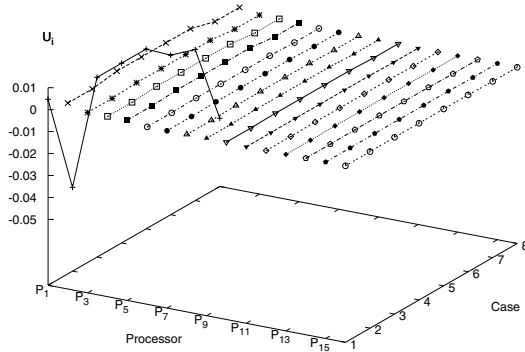


Figure 6. Utility of each processor when P_1 cheats (fast system).

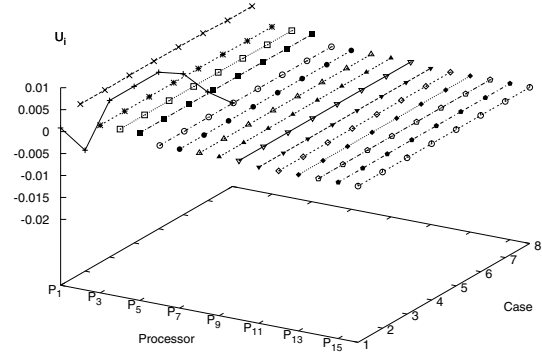


Figure 7. Utility of each processor when P_1 cheats (slow system).

ing cases, B_1 and thus U_1 is reduced as α_1 is smaller than in the optimal case. As anticipated, the utility of P_1 is much greater for the fast system than for the slow system, due to larger allocations to faster processors.

Figure 6 and 7 show the utility for all processors and all cases for the two types of systems we considered. When $b_1 > w_1$ (cases (3), (4), (5), and (6)), α_j (for $j = 2, \dots, 16$) is increased resulting in greater U_j . For example U_2 is increased by 36% in the fast system and by 1% in the slow system. When $b_1 < w_1$ (cases (7) and (8)), the reduced α_j results in decreased U_j . For example U_2 is decreased by 30% in the fast system and by 3% in the slow system. In the remaining cases, α_j and U_j are unchanged as $b_1 = w_1$. The impact of $b_1 \neq \hat{w}_1$ is felt unevenly among the processors. The effects of cheating diminishes as the processor index increases. For example in the case of the fast system and cases (3), (4), (5), and (6), the increase in U_2 is 36%

while the increase in U_{16} is 31%. This behavior is due to the allocation computed by the scheduling algorithm.

6. Conclusion

In this paper we considered the design of strategyproof scheduling mechanisms in the context of divisible load theory. To our knowledge this is the first attempt to augment DLT with incentives. We developed a strategyproof mechanism that provides incentives to the processors to participate and report their true processing capabilities to the scheduler. The processors gain the maximum profit by executing the load only if they are truthfully reporting their private values characterizing their processing capabilities. We proved and studied the properties of the mechanism. We performed an extensive simulation study in order to show the properties of our mechanism. Future work include the development

of distributed mechanisms for task scheduling and also the study of agents' privacy in these mechanisms.

References

- [1] A. Archer and E. Tardos. Truthful mechanism for one-parameter agents. In *Proc. of the 42nd IEEE Symp. on Foundations of Computer Science*, pages 482–491, October 2001.
- [2] A. Archer and E. Tardos. Frugal path mechanisms. In *Proc. of the 13th Annual ACM-SIAM Symp. on Discrete Algorithms*, pages 991–999, January 2002.
- [3] O. Beaumont, H. Casanova, A. Legrand, Y. Robert, and Y. Yang. Scheduling divisible loads on star and tree networks: Results and open problems. *IEEE Trans. Parallel and Distributed Syst.*, 16(3):207–218, March 2005.
- [4] V. Bharadwaj and G. Barlas. Access time minimization for distributed multimedia applications. *Multimedia Tools and Applications*, 12(2-3):235–256, November 2000.
- [5] V. Bharadwaj, D. Ghose, V. Mani, and T. Robertazzi. *Scheduling Divisible Loads in Parallel and Distributed Systems*. IEEE Press, Los Alamitos, California, 1996.
- [6] V. Bharadwaj, D. Ghose, and T. Robertazzi. Divisible load theory: A new paradigm for load scheduling in distributed systems. *Cluster Computing*, 6(1):7–17, 2003.
- [7] J. Blazewicz, M. Drozdowski, and M. Markiewicz. Divisible task scheduling - concept and verification. *Parallel Computing*, 25(1):87–98, January 1999.
- [8] S. Chan, V. Bharadwaj, and D. Ghose. Large matrix-vector products on distributed bus networks with communication delays using the divisible load paradigm: Performance and simulation. *Mathematics and Computers in Simulation*, 58:71–92, 2001.
- [9] J. Feigenbaum, C. Papadimitriou, R. Sami, and S. Shenker. A BGP-based mechanism for lowest-cost routing. In *Proc. of the 21st ACM Symp. on Principles of Distributed Computing*, pages 173–182, July 2002.
- [10] J. Feigenbaum, C. Papadimitriou, and S. Shenker. Sharing the cost of multicast transmissions. *Journal of Computer and System Sciences*, 63(1):21–41, August 2001.
- [11] J. Feigenbaum and S. Shenker. Distributed algorithmic mechanism design: Recent results and future directions. In *Proc. of the 6th ACM Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, pages 1–13, September 2002.
- [12] D. Grosu and A. T. Chronopoulos. Algorithmic mechanism design for load balancing in distributed systems. *IEEE Trans. Systems, Man and Cybernetics - Part B: Cybernetics*, 34(1):77–84, February 2004.
- [13] X. Li, V. Bharadwaj, and C. Ko. Distributed image processing on a network of workstations. *Intl. Journal of Computers and Their Applications*, 25(2):1–10, 2003.
- [14] C. Ng, D. Parkes, and M. Seltzer. Strategyproof computing: Systems infrastructures for self-interested parties. In *Proc. of the 1st Workshop on Economics of Peer-to-Peer Systems*, June 2003.
- [15] C. Ng, D. Parkes, and M. Seltzer. Virtual worlds: Fast and strategyproof auctions for dynamic resource allocation. In *Proc. of the ACM Conference on Electronic Commerce*, pages 238–239, June 2003.
- [16] N. Nisan, S. London, O. Regev, and N. Camiel. Globally distributed computation over Internet - The POPCORN project. In *Proc. of the 18th IEEE International Conference on Distributed Computing Systems*, pages 592–601, May 1998.
- [17] N. Nisan and A. Ronen. Algorithmic mechanism design. *Games and Economic Behaviour*, 35(1/2):166–196, April 2001.
- [18] M. Osborne and A. Rubinstein. *A Course in Game Theory*. MIT Press, Cambridge, Mass., 1994.
- [19] T. Robertazzi. Ten reasons to use divisible load theory. *IEEE Computer*, 36(5):63–68, May 2003.
- [20] W. E. Walsh, M. P. Wellman, P. R. Wurman, and J. K. MacKie-Mason. Some economics of market-based distributed scheduling. In *Proc. of the 18th IEEE International Conference on Distributed Computing Systems*, pages 612–621, May 1998.
- [21] R. Wolski, J. S. Plank, T. Bryan, and J. Brevik. G-commerce: market formulations controlling resource allocation on the computational grid. In *Proc. of the 15th IEEE International Parallel and Distributed Processing Symposium*, April 2001.
- [22] D. Yu and T. Robertazzi. Divisible load scheduling for grid computing. In *Proc. of the 15th International Conference on Parallel and Distributed Computing and Systems*, November 2003.