A Strategyproof Mechanism for Scheduling Divisible Loads in Bus Networks without Control Processors

Thomas E. Carroll and Daniel Grosu

Wayne State University
Department of Computer Science
5143 Cass Avenue, Detroit, MI 48202, USA.
{tec, dgrosu}@cs.wayne.edu

Abstract

Divisible Load Theory (DLT) considers the scheduling of arbitrarily partitionable loads in distributed systems. The underlying assumption of DLT is that the processors are obedient (i.e., they do not "cheat" the protocol), which is unrealistic when the processors are owned by autonomous, self-interested organizations that have no a priori motivation for cooperation and which strive to maximize their own welfare. In this scenario, they will manipulate the algorithm if it is beneficial to do so. In this paper we propose a strategyproof mechanism for scheduling divisible loads in bus networks without *control processors. We augment DLT with incentives so that it is to the benefit of a processor to truthfully report its processing capacity and to process its assignment at full capacity. The mechanism provides incentives to processors for reporting deviants and issues fines to deviants, which results in abated willingness to deviate.*

1. Introduction

Scheduling has important implications for the performance of distributed systems. Deficient scheduling leads to poorly utilized resources, inefficiencies, and suboptimal performance. In this paper we focus on the problem of scheduling divisible loads. *Divisible load problems* are characterized by large data sets where each element within the set requires an identical type of processing. The set can be partitioned into any number of independent fractions where each fraction requires scheduling.

Scheduling divisible loads is the subject of Divisible Load Theory (DLT) which was extensively studied in [3] where influences such as network architectures (*e.g.*, linear, tree, etc.), task arrangements, and optimality conditions are explored. The underlying assumption in DLT is that the processors are *obedient*, *i.e.*, under no circumstances will the processors "cheat". The assumption is unrealistic as the nodes may be owned by autonomous, self-interested entities that have no *a priori* motivation for cooperation and they are tempted to manipulate the algorithms in hope of increased benefits. In this environment, the processors are modeled as *strategic* agents. New protocols for DLT must account for this behavior. *Mechanism design theory* [17] — a subfield of economics that has recently garnered interest in computer science — provides the framework for solving such problems involving self-interested parties. The theory addresses *incentive compatibility*. Rational agents (self-interested, utility-maximizing) are provided incentives which induce a behavior that maximizes the social welfare. A mechanism produces an output based on agents' inputs. Each agent is characterized by its private values. An agent reports to the mechanism values that may not be equal to its private values. We are interested in *strategyproof mechanisms* in which the participants maximize their own utilities only if they report their true values to the mechanism.

In our previous work [9], we showed how DLT can be augmented with incentives. More specifically we designed a strategyproof mechanism called DLS-BL for scheduling divisible loads in *bus networks with a control processor*. In such systems the mechanism is executed by the control processor that is assumed to be trusted. The mechanism provides incentives to processors for participating. The agents maximize their profits by reporting their true values to the mechanism and executing their assignments as reported. DLS-BL is a Compensation-and-Bonus type mechanism [16].

In this paper we extend our research to include *bus networks without control processors*. These networks are harder to address due to the lack of the trusted control processor. We obtain the mechanism by having all the processors implement the DLS-BL mechanism. This results in a strategyproof mechanism as long as the processors cannot deviate from it. Since the processors are strategic, they will implement their own algorithm if it is beneficial to do so. To compensate for this situation, processors are provided incentives to monitor one another. In essence, processors are paid to fink. When a processor deviates from the algorithm, others immediately inform against it. The deviant is fined and the informers are rewarded with the collected sum.

Related work. The divisible load scheduling problem was studied extensively in recent years resulting in a cohesive theory called Divisible Load Theory (DLT). A reference book on DLT is [3]. Two recent surveys on DLT are [4] and [19]. Scheduling divisible loads in grids has been investigated in [21]. Scheduling divisible loads where the network and processing parameters are unknown is examined in [8]. A multiround divisible load scheduling algorithm is presented in [20]. New results and open research problems in DLT are presented in [2]. 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 [15], routing [5] and multicast transmission [6]. In their seminal paper, Nisan and Ronen [16] 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 problem of scheduling tasks on unrelated machines. A general framework for designing strategyproof mechanisms for one parameter agent was proposed by Archer and Tardos [1]. Grosu and Chronopoulos [10] derived a strategyproof mechanism that gives the overall optimal solution for the static load balancing problem in distributed systems. A strategyproof mechanism with verification combining incentives and DLT was proposed by Grosu and Carroll [9]. The results and the challenges of designing distributed mechanisms are surveyed in [7]. The *strategyproof computing* paradigm proposed in [13] considers the self-interest and incentives of participants in distributed computing systems. Ng *et al*. [14] proposed a strategyproof system for dynamic resource allocation in data staging. Mitchell and Teague [12] extended the distributed mechanism in [6] devising a new model where the agents themselves implement the mechanism, thus allowing them to deviate from the algorithm.

Our contributions. The main contribution of this paper is the design of a strategyproof mechanism with verification for solving the divisible load scheduling problem in bus networks without control processors. The mechanisms provide incentives to processors for monitoring one another. When processors deviate from the algorithm, others inform against it resulting in the cheater being penalized and the informers rewarded. A minimally-trusted third party is authorized to collect fines and to distribute the sum among the processors. We define the mechanism and prove its properties.

Organization. The paper is structured as follows. In Section 2 we describe the divisible load scheduling problem in the context of systems with bus networks. In Section 3 we discuss the mechanism design foundations. In Section 4 we present our proposed mechanism. In Section 5 we prove the mechanism's properties. In Section 6 we draw conclusions and present future directions.

2. Divisible Load Scheduling Problem

We consider a distributed system consisting of m processors, $(P_1,\dots,P_m)$, interconnected by a bus network. Processor P_i $(i=1,\dots,m)$ is characterized by w_i, which is the time it takes to process a unit load. The processor is assigned α_i units of load and it takes time $\alpha_i w_i$ to compute the assignment. The cost incurred by P_i to process α_i units of load is $\alpha_i w_i$, which corresponds to a linear cost model. There is a *load-originating processor* responsible for distributing the load to the other processors. The load-originating processor transmits α_i units of load to P_i in time $\alpha_i z$, where z is the time it takes to communicate a unit load from the load-originating processor to any other processor. We denote by $\alpha=(\alpha_1,\dots,\alpha_m)$ the load allocation. Processor P_i finishes in time $T_i(\alpha)$, which is the total time taken to receive and process the assignment.

Depending on the existence of a control processor we have two classes of systems: bus network with control processor (CP) and bus network without control processor (NCP) [3]. Furthermore the bus network without control processor class can be divided into two subclasses depending on the existence of a front end: bus network without control processor, with front end (NCP-FE), and bus network without control processor, without front end (NCP-NFE). In the following we discuss these types of systems in the context of DLT.

Bus network with control processor (CP). There is an independent load-originating processor P_0. The processor P_0 does not have any processing capacity and can only communicate with one processor at a time (*i.e.*, the one-port model). Figure 1 shows a diagram representing the execution on this system. From the diagram, it is obvious that the finishing time $T_i(\alpha)$ is given by

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

We do not discuss this system further. This was the subject of our previous work [9] in which we designed a strat-

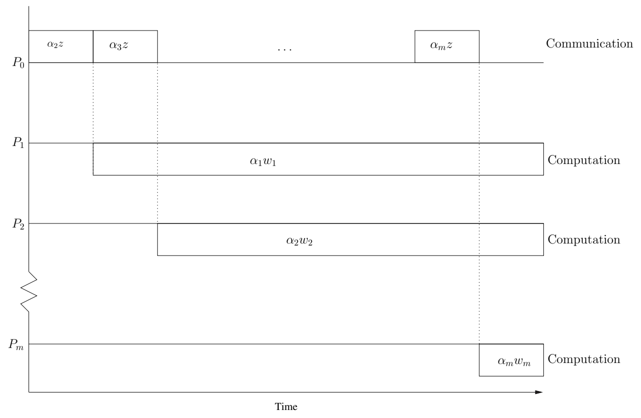

Figure 1. Bus network with control processor (CP)

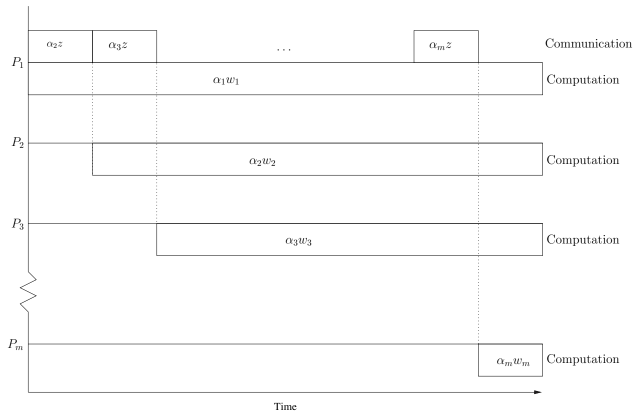

Figure 2. Bus network without control processor; load-originating processor with front end (NCP-FE)

egyproof mechanism for scheduling divisible loads in bus networks with control processor. The focus of this paper is the design of strategyproof mechanisms for the other two types of systems.

Bus network without control processor, load-originating processor with front end (NCP-FE). There is no separate control processor. The load-originating processor P_1 has a front end permitting it to simultaneously communicate and compute. Again, we assume the one-port model. A diagram representing the execution on this system is shown in Figure 2. The finishing time $T_i(\alpha)$ is given by

$$T_i(\alpha) = \begin{cases} \alpha_1 w_1 & \text{if } i = 1 \\ z\sum_{j=1}^{i} \alpha_j + \alpha_i w_i & \text{if } i = 2, \ldots, m \end{cases} \tag{2}$$

Notice that P_1 does not experience any delay related to communicating the load.

Bus network without control processor, load-originating processor without front end (NCP-NFE). This is similar to the previous system in that a control processor is not present. But, the load-originating processor P_m does not have a front end, thus, it cannot simultaneously compute and communicate. As usual, we assume the one-port model. Figure 3 illustrates an execution on this system. We obtain the finishing time $T_i(\alpha)$

$$T_i(\alpha) = \begin{cases} z\sum_{j=1}^{i} \alpha_j + \alpha_i w_i & \text{if } i = 1, \ldots, m-1 \\ z\sum_{j=1}^{i-1} \alpha_j + \alpha_i w_i & \text{if } i = m \end{cases} \tag{3}$$

Processor P_m does not commence computing until it has communicated the loads to all the other processors.

With each of the three systems described above we associate a different scheduling problem. We call these problems: BUS-LINEAR-CP, BUS-LINEAR-NCP-FE, and BUS-LINEAR-NCP-NFE respectively. Each of these prob-

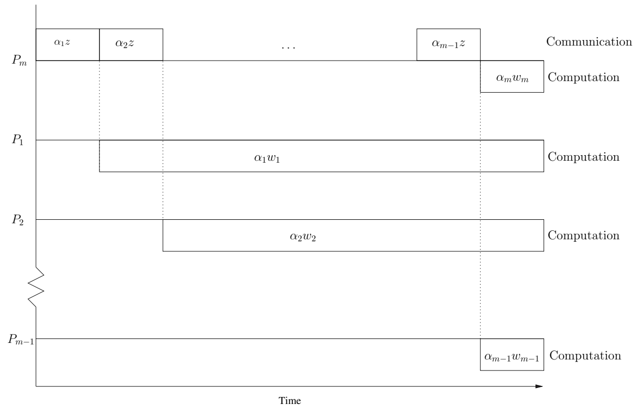

Figure 3. Bus network without control processor; load-originating processor without front end (NCP-NFE)

lems asks for the load allocation α which minimizes the total execution time (*i.e.*, $T(\alpha) = \max(T_1(\mathbf{b}), \ldots, T_m(\mathbf{b}))$) and are defined as follows.

$$\min_{\alpha} T(\alpha) \tag{4}$$

such that

$$\alpha_i \geq 0, \qquad i = 1, \ldots, m \tag{5}$$

and

$$\sum_{i=1}^{m} \alpha_i = 1. \tag{6}$$

The following theorems proved in [3] characterize the optimal solution for all three problems defined above.

Theorem 2.1. The optimal solution is obtained when all processors participate and they all finish executing their assigned load at the same time, *i.e.*, $T_1(\alpha) = \ldots = T_m(\alpha)$.

Theorem 2.2. Any load allocation order is optimal for the BUS-LINEAR-CP, BUS-LINEAR-NCP-FE, and BUS-LINEAR-NCP-NFE problems.

BUS-LINEAR-NCP-FE Problem. From Figure 2, the recursive equations

$$\alpha_i w_i = \alpha_{i+1} z + \alpha_{i+1} w_{i+1}, \qquad i = 1, \ldots, m-1 \tag{7}$$

compute the optimal allocation in the case of bus network without a control process; processor with front end. Using (7), the BUS-LINEAR-NCP-FE problem is solvable by the following algorithm.

Algorithm 2.1. *(BUS-LINEAR-NCP-FE Algorithm)*

```
Input: Time to process a unit load: $w_1, w_2, \ldots w_m$;
       Time to communicate a unit load: $z$;
Output: Load fractions: $\alpha_1, \alpha_2, \ldots \alpha_m$;
    1. for $j = 1, \ldots, m-1$ 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 $i = 2, \ldots, m$ do
           $\alpha_i = \alpha_1 \prod_{j=1}^{i-1} k_j$
```

BUS-LINEAR-NCP-NFE Problem. This problem is characterized by a different set of recursive equations as follows.

$$\alpha_i w_i = a_{i+1} z + \alpha_{i+1} w_{i+1}, \qquad i = 1, \ldots, m-2 \tag{8}$$

$$\alpha_{m-1} w_{m-1} = \alpha_m w_m \tag{9}$$

Using (8) and (9), the following algorithm that solves the BUS-LINEAR-NCP-NFE problem is derived.

Algorithm 2.2. *(BUS-LINEAR-NCP-NFE Algorithm)*

```
Input: Time to process a unit load: $w_1, w_2, \ldots w_m$;
       Time to communicate a unit load: $z$;
Output: Load fractions: $\alpha_1, \alpha_2, \ldots \alpha_m$;
    1. for $j = 1, \ldots, m-2$ do
           $k_j \leftarrow \frac{w_j}{z + w_{j+1}}$
    2. $\alpha_1 \leftarrow \frac{1}{1 + \sum_{i=1}^{m-2} \prod_{j=1}^{i} k_j + \frac{w_{m-1}}{w_m} \prod_{j=1}^{m-2} k_j}$
    3. for $i = 2, \ldots, m-2$ do
           $\alpha_i = \alpha_1 \prod_{j=1}^{i-1} k_j$
    4. $\alpha_m = \frac{w_{m-1}}{w_m} \alpha_{m-1}$
```

In classical DLT, it is assumed that the load-originating processor faithfully executes the BUS-LINEAR algorithms and that the processors truthfully report their w_i. If the processors are owned by autonomous, self-interested organizations, the load-originating processor may deviate from the algorithm or the processors may misreport their processing capacities in hope of gaining additional profit. In the next sections we present the design of a mechanism that compensates for strategic processors. The mechanism ensures that the processors report their true processing capacities and, due to the lack of a trusted control processor, that the algorithm is faithfully executed.

3. Mechanism Design Framework

In this section we introduce the main concepts of mechanism design theory. We limit our focus to mechanism design for one parameter agents. Each agent in this mechanism design problem has private data represented by a single real value [16]. We define the problem in the following.

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

(i) A finite set L of allowed outputs. The output is a vector $\alpha(\mathbf{b}) = (\alpha_1(\mathbf{b}), \ldots, \alpha_m(\mathbf{b})) \in L$, computed according to the agents' bids, $\mathbf{b} = (b_1, \ldots, b_m)$. Here, b_i is the bid of agent i.

(ii) Each agent i ($i = 1, \ldots, m$) has a privately known value 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* $V_i(\alpha, \mathbf{t})$.

(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 the agents after the mechanism learns $\tilde{\mathbf{t}}$.

(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 with Verification)* A *mechanism with verification* is characterized by two functions.

(i) The *output function* $\alpha(\mathbf{b}) = (\alpha_1(\mathbf{b}), \ldots, \alpha_m(\mathbf{b}))$. The input to this function is the vector of agents' bids $\mathbf{b} = (b_1, \ldots, b_m)$ and returns an output $\alpha \in L$.

(ii) The *payment function*, $Q(\mathbf{b}, \tilde{\mathbf{t}}) = (Q_1(\mathbf{b}, \tilde{\mathbf{t}}), \ldots, Q_m(\mathbf{b}, \tilde{\mathbf{t}}))$, 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 excluding the bid of agent i. The vector $\mathbf{b}$ is represented by $(\mathbf{b}_{-i}, b_i)$.

The following definition characterizes an important property of a mechanism, the property that an agent obtains maximum utility when $\tilde{t}_i = b_i = t_i$ independent of the other agents' actions.

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 it declares its real type t_i (*i.e.*, truth-telling is a dominant strategy).

Another important property of a mechanism is the voluntary participation, who guarantees that truthful agents obtain non-negative utility. This property is desirable as agents participate in hope of profit.

Definition 3.3. *(Voluntary Participation Mechanism)* We say that a mechanism satisfies the *voluntary participation condition* if $U_i((\mathbf{b}_{-i}, \mathbf{\tilde{t}_i}) \geq 0$ for every agent i, true value t_i, and other agents' bids $\mathbf{b}_{-i}$ (*i.e.*, truthful agents never incur a loss).

In our previous work [9], we augmented DLT with incentives. More specifically we designed the strategyproof mechanism DLS-BL that solves the problem of scheduling divisible loads in bus networks with control processor (*i.e.*, BUS-LINEAR-CP problem). DLS-BL mechanism provides incentives to processors for participating and for truthfully reporting their processing capacity. Additionally, DLS-BL satisfies the voluntary participation condition as truthful processors will never receive non-negative utility. In the following we describe DLS-BL.

We assume a set of $m+1$ processors, $\mathbf{P} = (P_0, P_1, \ldots, P_m)$. A bus interconnects the processors. We designate P_0 as the control processor. As we mentioned in Section 2, P_0 does not have any load processing capacity. Its sole role is to compute the allocations and to distribute the loads to the other processors. Processor P_i $(i = 1, \ldots, m)$ is characterized by its *true value* w_i, *i.e.*, $t_i = w_i$, the time to execute one unit of load. The processor reports its bid b_i to P_0. The bid does not have to be equal to the true value, *i.e.*, $b_i \neq w_i$. Once P_0 collects all the bids, it computes BUS-LINEAR-CP algorithm [3] obtaining the load allocation $\alpha(\mathbf{b}) = (\alpha_1(\mathbf{b}), \ldots, \alpha_m(\mathbf{b}))$, where $\mathbf{b} = (b_1, \ldots, b_m)$ is the vector of the agents' bids and $\alpha_i(\mathbf{b})$ is the allocation for P_i.

The control processor P_0 transmits α_i units of load to P_i. Processor P_i processes the assigned load. It may decide to process the load at a slower rate, with $\tilde{w}_i$ being the time to process a unit load, where $\tilde{w}_i \geq w_i$. To overcome this situation, we employ a mechanism with verification. The execution time $\tilde{w}_i$ is observed by P_0. The control processor P_0 computes and sends the payment Q_i to P_i.

The goal of P_i is to maximize its utility

$$U_i(\mathbf{b}, \mathbf{\tilde{w}_i}) = V_i(\alpha(\mathbf{b}), \mathbf{\tilde{w}}) + Q_i(\mathbf{b}, \mathbf{\tilde{w}}). \tag{10}$$

Processor P_i has a *valuation function*

$$V_i(\alpha(\mathbf{b}), \tilde{w}) = -\alpha_i \tilde{w}_i \tag{11}$$

which is the negation of the time required for P_i to process α_i units of load. V_i can be considered as the cost incurred by P_i when processing the assigned load. There is a direct relationship between processing time and cost. $Q_i(\mathbf{b}, \mathbf{\tilde{w}})$ is the payment to P_i for executing the load. The payment Q_i is defined as

$$Q_i(\mathbf{b}, \mathbf{\tilde{w}}) = C_i(\mathbf{b}, \mathbf{\tilde{w}}) + B_i(\mathbf{b}, \mathbf{\tilde{w}}) \tag{12}$$

where

$$C_i(\mathbf{b}, \mathbf{\tilde{w}}) = \alpha_i \tilde{w}_i$$

is the *compensation function* for P_i and

$$B_i(\mathbf{b}, \mathbf{\tilde{w}}) = T_{-i}(\alpha(\mathbf{b}_{-i}), \mathbf{b}_{-i}) - T(\alpha(\mathbf{b}), (\mathbf{b}_{-i}, \tilde{w}_i))$$

is the *bonus function* for P_i. The function $T_{-i}(\alpha(\mathbf{b}_{-i}), \mathbf{b}_{-i})$ is the optimal execution time when P_i does not participate. Thus, the bonus is equivalent to its contribution in reducing the total execution time. Notice that the compensation is the negation of the valuation. In effect, the compensation function reimburses P_i for the work it performs. The bonus provides incentive to report the true processing capacity. B_i may not be positive, especially in cases where $\tilde{w}_i < b_i$ (*i.e.*, the processor executes the load slower than what it bid). In [9] we proved the following theorems that characterize the DLS-BL mechanism.

Theorem 3.1. *(Strategyproofness)* DLS-BL is a strategyproof mechanism.

Theorem 3.2. *(Voluntary Participation)* DLS-BL is a voluntary participation mechanism

4. The Proposed Mechanism

We propose the mechanism DLS-BL-NCP for scheduling divisible loads on bus networks without control processors. The system model comprises m strategic processors interconnected via a bus. No *control processor* exists. We assume that the network is *obedient* and that the network and communication protocols are tamper-proof. An entity is *tamper-proof* [12] if its operation (*i.e.*, the algorithm which it is executing) is completely controlled by the designers and not by the entity itself. We further assume that the network has a reliable, atomic mechanism for broadcasting information. Since the transmission media (*i.e.*, the bus) is shared among all processors and the distance between any pair of processors is constant, we believe that the assumption is reasonable. The processor with access to the load is designated to be the *load-originating processor* and it is either P_1 or P_m depending on the existence of a front end. Besides the processors, there are two other entities. The first is the *user* who submits jobs to be processed and has funds to compensate the processors for the work performed on her behalf. The second entity is the *referee*. In the standard mechanism design model, the agents provide inputs to a central authority which faithfully executes the algorithm. The agents are able to lie to the central authority, but they are unable to alter the algorithm. In our mechanism, the agents themselves implement the mechanism. They will alter the algorithm if it is beneficial to do so. The main role of the referee is to prosecute cheating processors. The referee is isolated and remains passive until signaled by a processor that presumes cheating. If sufficient proof is brought forth, the referee imposes fines and distributes the proceeds among the other processors.

The referee is different from the control processor considered in [9]. The control processor is a trusted central authority that possesses the processor parameters, computes the load allocation, and transmits the work units to the processors. In DLT and DLS-BL mechanism, the control processor is assumed to be obedient, *i.e*, it will not cheat or deviate from the prescribed protocol. The referee, on the other hand, is used to resolve conflicts and if no conflicts arise, it does not posses any processor parameters.

Notation. We use the following notation in this section.

- The set of processors is $\mathbf{P} = (P_1, \ldots, P_m)$ such that $|\mathbf{P}| = m$.
- The load-originating processor is P_{lo}, where $P_{lo} \in \mathbf{P}$.
- Let SK_β be the private key of β. $SIG_\beta(m)$ is the secure digital signature of m under SK_β. $S_\beta(m) = (m, SIG_\beta(m))$ is the digitally signed message m under SK_β.

The description of DLS-BL-NCP follows. We assume the existence of a *payment infrastructure* and a *public key infrastructure* (*PKI*), to which the participants have access.

DLS-BL-NCP Mechanism

Initialization: Each participant has a public cryptographic key set. We do not dictate the specific cryptosystem, but it must minimally support digital signatures. The public key is registered under the participant's identity with the aforementioned PKI. The user prepares her data by dividing it into small, equal-sized blocks. Each block B has a unique identifier I_B appended to it and then the aggregate is signed by the user, *i.e.*, $S_{\text{user}}(B, I_B)$.

Bidding: An all-to-all broadcast occurs in which processor P_i $(i = 1, \ldots, m)$ communicates its digitally-signed bid $S_{P_i}(b_i, P_i)$ to P_j $(j \neq i)$. Commitments are not required according to our atomic broadcast assumption[1]. If P_i does not wish to participate, it does not broadcast a bid and it receives a utility of 0. Without loss of generality, we assume that P_i participates.

P_j $(j = 1, \ldots, m)$ verifies the authenticity and integrity of $S_{P_i}(b_i, P_i)$. If the message fails verification, it is discarded. If P_j receives multiple authenticated messages from P_i, it signals the referee providing the messages as evidence of cheating. If in fact cheating has occurred, the referee fines P_i an amount F. If the concerns are unfounded, P_j is penalized F. Fine F must be large to dissuade cheating and to induce finking. Furthermore, F must be larger than the sum of the compensations, *i.e.*, $F \gg \sum_{j=1}^{m} \alpha_j w_j$. All parties are aware of the magnitude of F. Let P_k be the party that is fined. The referee rewards $\frac{F}{m-1}$ to P_i $(i = 1, \ldots, m,\ i \neq k)$ thus terminating the protocol.

Allocating Load: Every processor computes the allocation (either Algorithm 2.1 or 2.2 respective to network type) obtaining load allocations $\alpha(\mathbf{b}) = (\alpha_1(\mathbf{b}), \ldots, \alpha_m(\mathbf{b}))$.

Processor P_{lo} transmits $\tilde{\alpha}_i$ units of load to P_i $(i \neq lo)$. If $\tilde{\alpha}_i \neq \alpha_i$ (*i.e.*, the assignment of P_i is incorrect), P_i signals the referee. Processors P_{lo} and P_i submit their vector of bids, $\mathbf{b}$, to the referee who verifies the authenticity of the bids and computes the allocation $\alpha(\mathbf{b})$. Both processors must submit their vector of bids as P_j, $j = 1, lo$, may alter its bid in its vector $\mathbf{b}$ (*i.e.*, $(S_{P_1}(b_1, P_1), \ldots, S'_{P_j}(b'_j, P_j),\ \ldots, S_{P_m}(b_m, P_m)))$. If the vector $\mathbf{b}$ submitted by P_j is inconsistent or fails authentication, P_j is fined. It is possible that both P_{lo} and P_i are penalized. If P_i claims that $\tilde{\alpha}_i > \alpha_i$, the referee attempts to substantiate the claim by comparing the blocks that P_i possesses with the original data set. If the claim is true, P_{lo} is fined. If the claim is unfounded, P_i is fined. The case in which $\tilde{\alpha}_i < \alpha_i$ is more difficult to resolve primarily due to the absence of credible evidence. There are three cases in which $\tilde{\alpha}_i < \alpha_i$ may occur: (i) P_{lo} communicated to few load units, (ii) the load unit integrity check failed, or (iii) P_i is lying. In all cases, the referee acts as an intermediary receiving load units from P_{lo}, verifying their integrity, and transmitting them to P_i. If P_{lo} refuses to transmit the correct number of load units or load unit integrity fails, P_{lo} is fined. If P_i claims that it did not receive enough load units, P_i is fined. In all situations where fines are raised, the protocol is immediately terminated. The total fine collected is xF, where x is the number of penalized processors. The referee distributes $\alpha_i \tilde{w}_i$ to the $i-1$ processors that have commenced work. The procedure for determining $\tilde{w}_i$ is detailed in the following stages. The remainder is evenly distributed among the $m-x$ non-deviating processors.

Processing Load: The processors execute their assignments. Processor P_i may process its load at a slower rate which means a unit load is processed in time $\tilde{w}_i$, where $\tilde{w}_i \geq w_i$. We cope with this situation by employing a *strategyproof mechanism with verification*. The goal of a strategyproof mechanism with verification is to give incentives to agents such that it is beneficial for them to report their values and process the assigned loads using their full processing capacity. We assume that the processors are augmented with a tamper-proof

[1]Commitments are required when atomic broadcast facilities are not available. When atomic facilities are not available, a sender distinctly transmits a message to each recipient. The sender may transmit different messages even though broadcasting by definition means sending the same message to all the recipients. Before broadcasting, the sender publicizes a commitment computed for the message. The recipient checks the commitment to ensure that it has received the proper message.

meter that reports the time executing the assigned load. The referee has access to the meters and records ϕ_i, the time to execute the load assigned to P_i. Once the processors complete computing their tasks, the referee broadcasts $(\phi_1,\ldots,\phi_m)$ to all.

Computing Payments: Processor P_i $(i=1,\ldots,m)$ computes $\tilde{w}_j=\frac{\phi_i}{\alpha_i}$ $(j=1,\ldots,m)$. We denote the vector of execution values by $\tilde{\mathbf{w}}=(\tilde{w}_1,\ldots,\tilde{w}_m)$. Processor P_i uses (12) to compute payment $Q_j(\mathbf{b},\tilde{\mathbf{w}})$. We denote the vector of payments $(Q_1,\ldots,Q_m)$ by $\mathbf{Q}$. Processor P_i submits $S_{P_i}(P_i,\mathbf{Q})$ to the referee. If there are multiple contradictory messages from P_i, the referee fines it. The referee verifies all vectors $\mathbf{Q}$ for equality. If there is inequality among the vectors, the bids are provided to the referee which computes the payments. The referee fines F to the x processors who incorrectly computed the payments or who provided contradictory messages. The referee distributes $\frac{xF}{m-x}$ to each of the $m-x$ correct processors. The referee forwards $\mathbf{Q}$ to the payment infrastructure. The bill is presented to the user who remits payment. More sophisticated methods such as quorums [11, 18] may not be used to resolve payments as these methods require a minimum number of obedient players and in our mechanism, all the processors are strategic.

This completes the mechanism description. We now examine the penalties associated with the mechanism. There are three stages of the mechanism in which we inspect for algorithm deviation. The offenses are: (i) multiple, inconsistent bids broadcasted in the Bidding phase; (ii) incorrect load assignments in Allocating Work phase; (iii) incorrect payment computation in Computing Payments phase; (iv) manipulated bid vectors transmitted to the referee; (v) unsubstantiated claims. All the above result in penalizing the cheating processors. The penalties are engineered so that processors that have already performed computation are compensated.

5. Properties and Complexity

In this section, we study the properties and the complexity of DLS-BL-NCP. The first property we investigate is the agents' aversion to deviate from the protocol.

Lemma 5.1. *(Utility Maximization)* A processor maximizes its utility by following DLS-BL-NCP.

Proof. (Sketch) A processor can cheat by deviating from the algorithm and by misreporting its processing capacity. Incentives are provided for processors to monitor one another. A processor that deviates will be detected resulting in large fines that greatly reduce its utility. If the processor misreports its processing capacity, it will experience reduced utility due to the strategyproofness of the DLS-BL mechanism (Theorem 3.1). □

Lemma 5.2. *(Fines)* A processor receives a fine only if it has deviated from DLS-BL-NCP.

Proof. (Sketch) Processor P_i is fined either for not executing the protocol faithfully or another processor P_j produces contradictory messages signed by P_i. In the first case, P_i clearly deviates from DLS-BL-NCP. In the second case, P_j sends the messages either by successfully forging signatures or by possessing the private key of P_i. We assume that the forging of signatures is impossible. Processor P_j obtains the private key either by P_i sharing it or by stealing it from P_i. It is a violation of the mechanism for a second party to possess a private key. Thus, P_i is fined for the protocol deviation. □

Corollary 5.1. A processor cannot receive a reward if no other processor has cheated.

Theorem 5.1. *(Compliance)* The processors will faithfully execute the mechanism.

Proof. (Sketch) By Lemma 5.2, it is more profitable for a processor to report deviation than to deviate itself. Therefore, all deviations will be reported. Furthermore, Lemma 5.1 shows that a processor will maximize its utility by faithfully executing the mechanism. Therefore, we can conclude that the processors will faithfully execute the prescribed mechanism. □

The second property we investigate is strategyproofness. If a mechanism is strategyproof, an agent maximizes its utility by being truthful.

Theorem 5.2. *(Strategyproofness)* DLS-BL-NCP is strategyproof.

Proof. The allocation function α and the payment function $\mathbf{Q}$ are identical to the ones used in DLS-BL. By Theorem 3.1, we know that DLS-BL is a strategyproof mechanism. Theorem 5.1 ensures that the processors will not deviate from the mechanism. Therefore, DLS-BL-NCP is strategyproof. □

We now investigate voluntary participation. A truthful agent never incurs a loss when partaking in a mechanism that satisfies the condition.

Theorem 5.3. *(Voluntary participation)* DLS-BL-NCP satisfies the voluntary participation condition.

Proof. DLS-BL-NCP uses the allocation function α and the payment function $\mathbf{Q}$ of DLS-BL. By Theorem 3.2, the DLS-BL mechanism satisfies the voluntary participation condition. Theorem 5.1 states that the processors will not deviate from the algorithm. Therefore, DLS-BL-NCP satisfies voluntary participation. □

We examine the communication complexity of DLS-BL-NCP. We define communication cost as the product between the number of messages transmitted and the message size. We do not include the communication necessary for transferring the load units.

Theorem 5.4. *(Communication Complexity)* The communication complexity of DLS-BL-NCP for m processors is $\Theta(m^2)$.

Proof. The communication cost is dominated by the Computing Payment phase. Each of m processors transmits a vector of size m to the referee. Therefore, the complexity is $\Theta(m^2)$. □

6. Conclusion

In this paper we proposed the strategyproof mechanism DLS-BL-NCP that schedules divisible loads on bus networks without control processors. We base the mechanism on the DLS-BL mechanism [9]. DLS-BL mechanism is a strategyproof mechanism that satisfies voluntary participation. We prevent deviations from the mechanism by providing incentives to the processors to monitor one another. When a processor deviates from the mechanism, other processors inform against it resulting in the deviant being fined. The fine is large enough to dissuade all deviations and all parties know the amount. The collected sum is distributed among the non-deviating processors. We show that the DLS-BL-NCP mechanism is strategyproof and it satisfies the voluntary participation condition.

For future work, we are planning to investigate other network architectures. We are hoping to complete a cohesive theory that combines DLT with incentives.

Acknowledgment. This research was supported in part by Wayne State University Faculty Research Grant.

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, Oct. 2001.

[2] 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, Mar. 2005.

[3] V. Bharadwaj, D. Ghose, V. Mani, and T. G. Robertazzi. *Scheduling Divisible Loads in Parallel and Distributed Systems*. IEEE Computer Society Press, Los Alamitos, CA, USA, 1996.

[4] V. Bharadwaj, D. Ghose, and T. G. Robertazzi. Divisible load theory: A new paradigm for load scheduling in distributed systems. *Cluster Computing*, 6(1):7–17, Jan. 2003.

[5] 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.

[6] J. Feigenbaum, C. H. Papadimitriou, and S. Shenker. Sharing the cost of multicast transmissions. *Journal of Computer and System Sciences*, 63(1):21–41, Aug. 2001.

[7] 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, Sept. 2002.

[8] D. Ghose, H. J. Kim, and T. H. Kim. Adaptive divisible load scheduling strategies for workstation clusters with unknown network resources. *IEEE Trans. Parallel and Distributed Syst.*, 16(10):897–907, Oct. 2005.

[9] D. Grosu and T. E. Carroll. A strategyproof mechanism for scheduling divisible loads in distributed systems. In *Proc. of the 4th International Symposium on Parallel and Distributed Computing (ISPDC 2005), 4–7 July, 2005, Lille, France*. IEEE Computer Society, July 2005.

[10] 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, Feb. 2004.

[11] L. Lamport, R. Shostak, and M. Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, July 1982.

[12] J. C. Mitchell and V. Teague. Autonomous nodes and distributed mechanisms. In *Proc. of the Mext-NSF-JSPS International Symp. on Software Security - Theories and Systems*, pages 58–83, Nov. 2003.

[13] 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.

[14] 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.

[15] 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.

[16] N. Nisan and A. Ronen. Algorithmic mechanism design. *Games and Economic Behaviour*, 35(1/2):166–196, Apr. 2001.

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

[18] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–234, Apr. 1980.

[19] T. G. Robertazzi. Ten reasons to use divisble load theory. *IEEE Computer*, 36(5):63–68, May 2003.

[20] Y. Yang, K. van der Raadt, and H. Casanova. Multiround algorithms for scheduling divisible loads. *IEEE Trans. Parallel and Distributed Syst.*, 16(11):1092–1102, Nov. 2005.

[21] D. Yu and T. G. Robertazzi. Divisible load scheduling for grid computing. In *Proc. of the 15th International Conference on Parallel and Distributed Computing and Systems*, Nov. 2003.