

Divisible Load Scheduling: An Approach Using Coalitional Games

Thomas E. Carroll and Daniel Grosu
Dept. of Computer Science
Wayne State University
5143 Cass Avenue, Detroit, MI 48202 USA.
{tec,dgrosu}@cs.wayne.edu

Abstract

Scheduling divisible loads in distributed systems is the subject of Divisible Load Theory (DLT). In this paper we show that coalitional game theory is a natural fit for modeling DLT as the participants in the scheduling algorithm must cooperate in order to execute a job. We devise a coalitional scheduling game in which the job owners and the independent organizations that own processors form coalitions in order to maximize their profits. We examine the payoffs to the participants and show that the core of the proposed coalitional scheduling game is non-empty. Then we examine the "fair sharing" of the payoffs among the participants using the Shapley value. Finally we study by simulation the properties of the proposed coalitional scheduling game considering different distributed systems configurations.

1. Introduction

Scheduling in a distributed system requires exploiting the characteristics of the jobs that will be executed. In this work we focus on scheduling jobs that are divisible. *Divisible load* problems exhibit *data parallelism*; they are characterized by large data sets where every element in the set requires an identical type of processing. The set can be partitioned into any number of fractions where each fraction requires a scheduling decision. These problems commonly arise in science and engineering. The scheduling of divisible loads is the subject of Divisible Load Theory (DLT). DLT is extensively studied in [3] where influences such as network architecture (*e.g.*, linear, bus, tree), task arrangement, and optimality conditions are explored. DLT requires a high degree of cooperation between the various processors. If we assume that these processors are owned and operated by independent organizations, these organizations along with the job owners must form a coalition in order to execute the job and to achieve profits.

In our previous work [6, 12] we showed that DLT can be modeled using noncooperative game theory. We designed several *strategyproof* mechanisms that schedule divisible loads in various network architectures. The processors — or more precisely the organizations who own processors — behave *strategically* and independently of one another. Incentives are employed so that the processors truthfully reveal their processing capacity and induce them to select the proposed strategy.

In the real world these resource-owning organizations will combine forces in order to increase their profit. In the cases in which the players cooperate, we use *cooperative* instead of *noncooperative* game theory. *Coalitional games* are studied within the cooperative game theory which models the interaction between groups of decision-makers. The cooperative game theory allows us to determine the stability of coalitions and how to share the profits among the players in a coalition.

Because of the high degree of cooperation needed among the players, we propose that the divisible load scheduling problem is better examined using coalitional instead of noncooperative game theory. In this work we study a DLT model in which the players want to maximize their profits. A description of the problem follows. The job owner earns money once the job is executed. Additionally, she prefers to have the job executed faster rather than slower. The processors incur costs executing the work. Thus, the job owner must compensate the processors or they, as they behave rationally, will not execute the load. As a result, the job owner and processors form a coalition and the profits earned are distributed to the coalition's members. We propose a coalitional game that models this situation.

Related work. The divisible load scheduling problem was studied extensively in recent years resulting in a cohesive theory called Divisible Load Theory (DLT) [3, 4]. Scheduling divisible loads where the network and processing parameters are not known is examined in [11]. A multi-round divisible load scheduling algorithm is presented in [15]. New results and open research problems in DLT are pre-

sented in [2]. The coalitional game theory, which examines the interactions between groups of decision-makers and thus is cooperative in nature, has been used to model many problems. A good reference on coalitional game theory is [13]. The complexity of different solution concepts for coalitional games is examined in [7]. One of the solution concepts is the *core*. A core solution to the binpacking problem is given in [9]. A general framework for studying the core of partition games is presented in [10]. Another concern is the sharing of the profits between the participants of the coalition. One method which rewards a participant depending on its value or power is the Shapley value [14].

Our contributions. The main contribution of this paper is the development of coalitional game models for the divisible load scheduling problem. We develop a model of scheduling divisible loads in distributed systems interconnected via a bus in which the job owner and the independent organizations that own processors form coalitions in order to maximize their profits. The job owner receives a payment once the job is executed. The owner prefers having the job executed faster than slower and thus, the faster the execution the higher her utility. The processors are characterized by a linear cost model and hence, their processing costs increase with the assignment size. We develop a coalitional game that models this scenario called *Divisible Load Scheduling Coalitional Game* (DLSCG). DLSCG goal is to minimize the sum of the makespan and the processing costs (otherwise stated as maximizing the profit). In certain circumstance, idling processors results in a reduction in total cost. We develop an algorithm that computes the load allocations and thus, the set of idled processors that minimizes the total cost. We show that the DLSCG is a *coalitional game* and that the associated *core* is non-empty. Afterward, we examine the payoffs given by the *Shapley value*. Finally, we simulate and study the properties of the model using different distributed system configurations.

Organization. The paper is structured as follows. In Section 2 we present a description of the divisible load scheduling problem. In Section 3 we introduce the concepts of coalitional game theory. In Section 4 we define the Divisible Load Scheduling Coalitional Game (DLSCG) and study its properties. In Section 5 we study by simulation the DLSCG game. 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 $\mathbf{P} = \{P_0, P_1, \dots, P_m\}$ 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

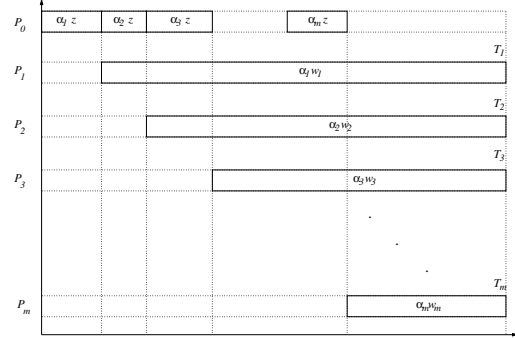


Figure 1. Timing diagram for the load execution.

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 Figure 1 we present a diagram representing 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: $\min_{\alpha} T(\alpha)$ subject to the constraints: $\alpha_i \geq 0$, $i = 1, \dots, m$ and $\sum_{i=1}^m \alpha_i = 1$. The following theorems proved in [3] characterize the optimal solution for this problem.

Theorem 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. *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. For P_0 to compute the allocation, it requires information about the network and the processing capabilities of the other participating processors.

Another method to compute the allocations is to recursively reduce processors into single equivalent processors. An equivalent processor has the same processing capacity as its constitute processors. To compute the equivalent execution time for processors P_1, P_2, \dots, P_k , we take the longest completion time $T_i(\alpha)$ and subtract the communication parameter from it; i.e., the equivalent execution time, w_{eq} , for processors P_1, P_2, \dots, P_k is

$$w_{\text{eq}} = \max(T_1(\alpha), T_2(\alpha), \dots, T_k(\alpha)) - z. \quad (2)$$

When the system is optimal and all processors finish executing at the same instant, (2) is simplified to

$$w_{\text{eq}} = w_1 \alpha_1 - z(1 - \alpha_1), \quad (3)$$

where P_1 is the first processor to be assigned load. The following recursive algorithm can be used in place of the above closed-form BUS-LINEAR algorithm to solve the BUS-LINEAR problem:

Recursive 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. $\alpha_1 \leftarrow 1$
2. $w_{\text{eq}} \leftarrow w_1$
3. **for** $j = 2, \dots, m$ **do**
 $\alpha_{\text{eq}}, \alpha_j \leftarrow \text{BUS-LINEAR}(w_{\text{eq}}, w_j; z)$
 $w_{\text{eq}} \leftarrow w_{\text{eq}} \alpha_{\text{eq}} - z(1 - \alpha_{\text{eq}})$
4. **for** $j = m - 1, \dots, 1$ **do**
 $\alpha_j \leftarrow 1 - \alpha_{j+1}$

In the above algorithm, P_1 and P_2 are reduced to an equivalent processor that replaces both of them. This processor is the first in order of load assignments. This equivalent processor and P_3 is then reduced to create another equivalent processor. This process continues until only one processor remains. It is evident that this algorithm operates in time $\Theta(m)$.

DLT requires a high degree of cooperation between the participants. In the next section we present the main concepts of coalitional game theory. We use these concepts to extend DLT so that allocations minimize the sum of the

makespan and the processing costs incurred when executing the task.

3. Coalitional Game Theory

In this section we introduce the main concepts of coalitional game theory. Coalitional game theory studies the interactions of groups of *decision-makers* (i.e., the *players*). The *grand coalition* \mathbf{N} is the set of all players; coalition $\mathbf{S} \subseteq \mathbf{N}$ results from partitioning the set of players. In the following we formally define a coalitional game.

Definition 1 (Coalitional game). [13] A n -person game is in the coalitional form (v, \mathbf{N}) when the characteristic function v is defined over the powerset of \mathbf{N} such that v satisfies the following two conditions: (i) $v(\emptyset) = 0$, and (ii) (Superadditivity) If $\mathbf{S}, \mathbf{T} \subset \mathbf{N}$ and $\mathbf{S} \cap \mathbf{T} = \emptyset$, then $v(\mathbf{S}) + v(\mathbf{T}) \leq v(\mathbf{S} \cup \mathbf{T})$.

The characteristic function $v(\mathbf{S})$ computes the *value* or *worth* of coalition \mathbf{S} . Generally, $v(\mathbf{S})$ is the sum of the payoffs of the participants; i.e., $v(\mathbf{S}) = \sum_{i \in \mathbf{S}} u_i$.

For games in coalitional form, we denote by $\mathbf{x} = (x_1, x_2, \dots, x_n)$ the *payoff vector*, where x_i is the payoff to player i . The *payoff* x_i quantifies the change in i 's utility. An *imputation* is a payoff vector that satisfies the *group rational* and *individually rational* conditions.

Definition 2 (Group rational). [13] A *payoff vector* $\mathbf{x} = (x_1, x_2, \dots, x_n)$ is group rational if $\sum_{i \in \mathbf{N}} x_i = v(\mathbf{N})$.

Definition 3 (Individually rational). [13] A *payoff vector* $\mathbf{x} = (x_1, x_2, \dots, x_n)$ is individually rational if $x_i \geq v(\{i\})$ for all $i \in \mathbf{N}$.

The group rationality property ensures that whenever players decide to form a coalition they will share the worth of the coalition among themselves. The individually rational property states that an individual participating in a coalition would not receive a payoff less than if it acted alone.

Definition 4 (Imputation). A *payoff vector* \mathbf{x} is an imputation if it is group rational and individually rational.

We can write the set of imputations as

$$\{\mathbf{x} = (x_1, x_2, \dots, x_n) : \sum_{i \in \mathbf{N}} x_i = v(\mathbf{N}) \text{ and } x_i \geq v(\{i\}) \text{ for all } i \in \mathbf{N}\} \quad (4)$$

The *core* is a solution concept that characterizes the *stability* of the grand coalition \mathbf{N} .

Definition 5 (Stability). [13] An imputation \mathbf{x} is unstable through a coalition \mathbf{S} if $v(\mathbf{S}) > \sum_{i \in \mathbf{S}} x_i$. Imputation \mathbf{x} is unstable if a coalition \mathbf{S} exists such that \mathbf{x} is unstable; otherwise, \mathbf{x} is stable.

Definition 6 (Core). [13] The core is the set, \mathbf{C} , of stable imputations, i.e.,

$$\mathbf{C} = \{ \mathbf{x} = (x_1, x_2, \dots, x_n) : \sum_{i \in \mathbf{N}} x_i = v(\mathbf{N}) \text{ and} \\ \sum_{i \in \mathbf{S}} x_i \geq v(\mathbf{S}) \text{ for all } \mathbf{S} \subset \mathbf{N} \}. \quad (5)$$

The core measures the stability of the grand coalition \mathbf{N} . If the core is non-empty, then there exists imputations in which the players prefer the grand coalition over any other coalition. If the core is empty, then the players prefer a coalition other than the grand coalition.

Definition 7 (Essential game). [13] A coalition game (v, \mathbf{N}) is essential if $\sum_{i=1}^n v(\{i\}) < v(\mathbf{N})$ and inessential if $\sum_{i=1}^n v(\{i\}) = v(\mathbf{N})$.

In an essential game, the grand coalition's worth exceeds the sum of the individuals' worth, while in an inessential game, the grand coalition's worth is equal to the sum of the individuals' worth.

Theorem 3. [13] If the coalitional game (v, \mathbf{N}) is an inessential constant-sum game, then the imputation $\mathbf{x} = (v(\{1\}), v(\{2\}), \dots, v(\{n\}))$ is the only imputation in the core.

Proof: It follows directly from Definition 6. ■

An inessential game will always have the unique imputation $\mathbf{x} = (v(\{1\}), v(\{2\}), \dots, v(\{n\}))$ in the core, but the core of an essential game may be empty or contain one or more imputations.

The next solution concept that we employ in this paper is the Shapley value. The *Shapley value* is a measure of the value or power of a player i , for all $i = 1, 2, \dots, n$. It computes a unique imputation that may or may not be in the core.

Definition 8 (Value function). The value function $\phi(v)$ is a function that assigns to the coalitional game (v, \mathbf{N}) a vector $\phi(v) = (\phi_1(v), \phi_2(v), \dots, \phi_n(v))$, i.e., $\phi : v \rightarrow \mathbb{R}^n$.

The Shapley value is usually computed using the following definition.

Definition 9 (Shapley value). [13] The Shapley value is given by $\phi = (\phi_1, \phi_2, \dots, \phi_n)$, where for $i = 1, 2, \dots, n$,

$$\phi_i(v) = \sum_{\mathbf{S} \subseteq \mathcal{P}(\mathbf{N}) | i \in \mathbf{S}} \frac{(|\mathbf{S}| - 1)! (n - |\mathbf{S}|)!}{n!} (v(\mathbf{S}) - v(\mathbf{S} - \{i\})), \quad (6)$$

where $\mathcal{P}(\mathbf{N})$ is the powerset of \mathbf{N} .

An imputation that is computed using (6) satisfies the following conditions, called the Shapley Axioms.

Definition 10 (Shapley Axioms). [13] A value function $\phi(v)$ for coalitional game (v, \mathbf{N}) must satisfy the following Shapley Axioms:

- (i) (Efficiency) $\sum_{i \in \mathbf{N}} \phi_i(v) = v(\mathbf{N})$
- (ii) (Symmetry) If there exists an i and j such that $v(\mathbf{S} \cup \{i\}) = v(\mathbf{S} \cup \{j\})$ for every coalition \mathbf{S} not containing i and j , then $\phi_i(v) = \phi_j(v)$
- (iii) (Dummy Axiom) If there exists an i such that $v(\mathbf{S}) = v(\mathbf{S} \cup \{i\})$ for every coalition \mathbf{S} not containing i , then $\phi_i(v) = 0$
- (iv) (Additivity) If u and v are characteristic functions, then $\phi(u + v) = \phi(u) + \phi(v)$

We use the concepts described in this section and DLT to devise a coalitional game that models divisible load scheduling. The objective of this game is to minimize the sum of the makespan, which is the usual object of DLT, and the costs incurred by the processors in executing their assignments. We study the scheduling game and characterize its solutions.

4. Divisible Load Scheduling as a Coalitional Game

In this section we explore divisible load theory using coalitional game models. We believe that these models are a natural fit for DLT as a high degree of cooperation between the job owner and the processors is necessary to achieving a small makespan. There exists several studies in modeling divisible loads and scheduling in general using *noncooperative* games. Three basic models are used to represent the noncooperative scenario: (i) processors behave *strategically*, tasks *obediently* [6, 12]; (ii) tasks behave *strategically*, processors *obediently* [5]; and (iii) processors and tasks behave *strategically* [1]. In this work we consider a different approach and advocate the use of a coalitional game model. Denote by J the job that needs to be processed. We use the distributed system model as outlined in Section 2 with one modification: we let job J 's owner control processor P_0 . Consequently, we exclude P_0 from the processor set \mathbf{P} and we refer to the entity that performs the master processor's duties as J . We now formally describe the coalitional game that models the divisible load scheduling problem.

Players The grand coalition \mathbf{N} comprises the job J and processors P_1, P_2, \dots, P_m

Actions The set of actions of a coalition $\mathbf{S} \subseteq \mathbf{N}$ consisting of J and k processors is the set of all divisible load allocations of J to the processors in \mathbf{S}

Preferences Processor P_i 's preferences are represented by the following utility function

$$u_i = r_i - \alpha_i w_i, \quad (7)$$

where α_i units of load are processed by P_i and r_i is the difference between the amount of money P_i initially has and the amount it has at the end of the execution. If P_i loses money, then $r_i < 0$; if P_i gains money, then $r_i > 0$. Similarly, J 's preferences are represented by

$$u_J = \begin{cases} C_J + r_J - T & \text{if } J \text{ is completed} \\ r_J & \text{otherwise,} \end{cases} \quad (8)$$

where C_J is a constant that represents J 's payoff if J could be instantaneously processed, T is the time it takes to process J (i.e., the makespan), and r_J is similarly defined as r_i .

Processor P_i 's preferences, for $i = 1, 2, \dots, m$, reflect that as the size of its load assignment increases, there is a corresponding increase in its processing cost. As such, P_i 's preferences consider both the sum of the processing costs and any change in money. Job J 's preferences considers both the length of the makespan and processing costs, which differs from our previous work [6, 12] in which J only considered makespan length.

From the above utility functions, we construct the characteristic function for the *Divisible Load Scheduling Coalitional Game* (DLSCG) as follows:

$$v(\mathbf{S}) = \begin{cases} 0 & \text{if } |\mathbf{S}| < 2 \text{ or } J \notin \mathbf{S} \\ \max(C_J - T(\alpha_{\mathbf{R}}) - \sum_{P_i \in \mathbf{R}} \alpha_i w_i, 0) & \text{otherwise} \end{cases} \quad (9)$$

where $\mathbf{R} \in \arg \max_{\mathbf{R} \in \mathcal{P}(\mathbf{S})} \{C_J - T(\alpha_{\mathbf{R}}) - \sum_{P_i \in \mathbf{R}} \alpha_i w_i\}$ such that $\sum_{P_i \in \mathbf{R}} \alpha_i = 1$ and $\alpha_j = 0$, for all $P_j \in \mathbf{S} - \mathbf{R}$. The set $\mathcal{P}(\mathbf{S})$ is the powerset of \mathbf{S} . Denote by $\alpha_{\mathbf{R}} = (\alpha_1, \alpha_2, \dots, \alpha_m)$ the vector of allocations such that $\alpha_i > 0$ for all $i \in \mathbf{R}$. The expression $C_J - T(\alpha_{\mathbf{R}})$ is the payoff J obtains when it is executed by the processors in \mathbf{R} with makespan $T(\alpha_{\mathbf{R}})$. A processor P_i incurs cost $\alpha_i w_i$ in executing its assignment. As we mentioned before, this corresponds to a linear cost model. The sum of the *processing costs* incurred by the processors in \mathbf{R} when processing J is $\sum_{P_i \in \mathbf{R}} \alpha_i w_i$. We express the *total cost* as the sum of the makespan and the processing costs. In some circumstances the total cost for a coalition may exceed C_J , resulting in a negative value. Due to the constraints of $v(\emptyset) = 0$ and superadditivity, we must set the value of these coalitions to zero.

If $v(\mathbf{S}) = 0$, for any $\mathbf{S} \in \mathcal{P}(\mathbf{N})$, no processing is performed and the participants' utility is unchanged. If the utility did change then one of the following two cases would be true: for any player i , if i 's utility is negative (i.e., $u_i < 0$), then i must have paid the other players and thus, $r_i < 0$. Player i can increase her utility by not paying any of the other players, resulting in $u_i = 0$. In the other case, player i 's utility is positive (i.e., $u_i > 0$) and thus, $r_i > 0$. The sum of the payments of the players without i must be negative, i.e., $\sum_{j \in \mathbf{S} - \{i\}} r_j < 0$. If $v(\mathbf{S}) > 0$ then, by (9), J

is processed to completion. The intuition behind the second branch of (9) is that work will be assigned to minimize the length of the makespan *and* the processing costs, unlike DLT which only considers minimizing the length of the makespan.

In the following we discuss the load assignments that satisfy the conditions set forth in the definition of DLSCG. First, we show that the assignment order influences processing cost.

Theorem 4. *Out of all arbitrary assignment orders of processors P_1, P_2, \dots, P_m , the total cost is minimized when the BUS-LINEAR algorithm is used and the processors are arranged in order of their speed from the fastest to the slowest, i.e., $w_1 \leq w_2 \leq \dots \leq w_m$.*

Proof: The sum of the processor costs is a linear combination of the processor execution times. BUS-LINEAR assigns more load to the fast processors when these processors are early in the assignment order rather than later. Since these processors have greater load, they are weighted greater in the sum and thus, the ordering minimizes the total cost. ■

To simplify the discussion, we assume that the load is assigned to the processors in order of non-decreasing execution times, i.e., the load is assigned in order of $w_1 \leq w_2 \leq \dots \leq w_m$. This simplification does not result in loss of generality.

In certain circumstance, idling processors (i.e., assigning them zero load) results in a reduction of total cost. The next theorems specify when processors are idled so that the total cost is minimized.

Theorem 5. *Let P_{eq} be the equivalent processor obtained by reducing processors P_1, P_2, \dots, P_{i-1} , for $i = 2, \dots, m$. If reducing P_i with P_{eq} to form equivalent processor P'_{eq} results in an increase in total cost, then processors P_i, P_{i+1}, \dots, P_m must be idled (i.e., $\alpha_i = \alpha_{i+1} = \dots = \alpha_m = 0$) in order to reduce the total cost.*

Proof: Compute the load assignments, α , using BUS-LINEAR. This allocation minimizes the makespan length. Beginning with processors P_1 and P_2 , if J decreases P_2 's assignment by ϵ units and assigns them to P_1 , then the change in total cost, Δcost , is

$$\begin{aligned} \Delta\text{cost} &= \text{change in makespan} + \text{change in processor cost} \\ &= \epsilon(2w_1 + z - w_2). \end{aligned} \quad (10)$$

If $w_2 > 2w_1 + z$, then idling P_2 (i.e., $\alpha_2 = 0$) and assigning the load to P_1 will decrease cost by $\alpha_2(w_2 - 2w_1 - z)$. We idle P_3, P_4, \dots, P_m as these processors have equal or greater execution times when compared to P_2 . Otherwise, we reduce P_1 and P_2 to P_{eq} with processing costs $c_{eq} = w_1\alpha_1 + w_2\alpha_2$ and execution time $w_{eq} = T(\alpha) - z$,

where $T(\alpha)$ is the completion time. We now compare the execution time of P_3 with P_{eq} . If J allocates ϵ units of load from P_3 to P_{eq} , then the change in total cost is

$$\begin{aligned} \Delta \text{cost} &= \text{change in makespan} + \text{change in processor cost} \\ &= \epsilon(T(\alpha) + pc_{\text{eq}} - w_3). \end{aligned} \quad (11)$$

If $w_3 > T(\alpha) + pc_{\text{eq}}$, then idling P_3 and assigning the load to P_{eq} will reduce the total cost by $\alpha_3(w_3 - T(\alpha) - pc_{\text{eq}})$. If P_3 is not idled, then it is reduced with P_{eq} to generate P'_{eq} . For equivalent processor P_{eq} comprising processors P_1, P_2, \dots, P_{i-1} , for $i = 2, 3, \dots, P_m$, and P_i , if $w_i > T(\alpha) + pc_{\text{eq}}$, then idling P_i results in a decrease in total cost of $\alpha_i(w_i - T(\alpha) - pc_{\text{eq}})$. Since $w_{i+1} \geq w_i$, idling $P_{i+1}, P_{i+2}, \dots, P_m$ results in further decrease in cost. ■

If the execution time of the slower processor is greater than the sum of the processing costs, the communication parameter, and the execution time of the equivalent processor comprising the faster processors, then all processors whose execution time is equal to or greater than the execution time of the slower processor should be idled to reduce total cost. In effect, if a slower processor exceeds this constraint for any single faster processor, then it and the slower processors should be idled to reduce the total cost.

Corollary 1. *BUS-LINEAR computes the optimal allocations minimizing the total cost when processors satisfying Theorem 5 are idled.*

Using the above results, we modify the recursive BUS-LINEAR algorithm such that it minimizes the total cost of the system.

BUS-LINEAR-COST 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. Sort the processors in order of
 $w_{\pi(1)} \leq w_{\pi(2)} \leq \dots \leq w_{\pi(m)}$,
where $\pi(i)$ is the ordering permutation
2. $\alpha_1 \leftarrow 1, w_{\text{eq}} \leftarrow w_1, pc_{\text{eq}} \leftarrow w_1$
3. **for** $j = 2, 3, \dots, m$ **do**
 if $w_j > pc_{\text{eq}} + w_{\text{eq}} + z$ **break**
 $\alpha_{\text{eq}}, \alpha_j \leftarrow \text{BUS-LINEAR}(w_{\text{eq}}, w_j; z)$
 $pc_{\text{eq}} \leftarrow \alpha_{\text{eq}}w_{\text{eq}} + \alpha_j w_j$
 $w_{\text{eq}} \leftarrow \alpha_{\text{eq}}\alpha_{\text{eq}} - z(1 - \alpha_{\text{eq}})$
4. **for** $i = j - 2, \dots, 1$ **do**
 $\alpha_i \leftarrow \alpha_{i+1}$
5. **for** $i = j, \dots, m$ **do**
 $\alpha_i \leftarrow 0$

The complexity of this algorithm increases from $\Theta(m)$ for BUS-LINEAR to $\Theta(m \lg m)$ due to the sorting of processing times.

We now consider a special case in which all processors have the same speed (i.e., $w_1 = w_2 = \dots = w_m$). In this context, we can make stronger claims about the load allocations.

Theorem 6. *For the grand coalition \mathbf{N} comprising equal-speed processors, if $v(\mathbf{N}) > 0$ then all processors are assigned load.*

Proof: Since all speeds are equal, processors will not be assigned zero load due to the conditions set in Theorem 5. As a consequence of Theorem 1, BUS-LINEAR-COST assigns load to every processor. Thus, all processors will be assigned load. ■

The BUS-LINEAR-COST algorithm computes the load allocations that satisfy the conditions for the characteristic function defined in (9). Without the BUS-LINEAR-COST algorithm, an expensive combinatorial program would be needed. Since the characteristic function for DLSCG is defined, we can formally show that DLSCG is a coalitional game and consequently, use coalitional theory to study its properties.

Theorem 7. *The DLSCG game defined by (v, \mathbf{N}) is a coalitional game.*

Proof: We claim that (9) satisfies the conditions in Definition 1. The condition $v(\emptyset) = 0$ is satisfied by the first branch of (9). To show superadditivity, we let \mathbf{S} and \mathbf{T} be two disjoint sets from \mathbf{N} such that $T \in \mathbf{T}$. We have $v(\mathbf{S}) + v(\mathbf{T}) \leq v(\mathbf{S} \cup \mathbf{T})$ and $v(\mathbf{T}) \leq v(\mathbf{S} \cup \mathbf{T})$. The expression $v(\mathbf{T}) \leq v(\mathbf{S} \cup \mathbf{T})$ is satisfied as the load assignments are always optimally computed. In the very least, the system $\mathbf{S} \cup \mathbf{T}$ will use the same allocations as in the system \mathbf{T} . Since both conditions of a coalitional game are satisfied, the DLSCG game (v, \mathbf{N}) is a coalitional game. ■

One of the important solution concepts for coalitional games is the core. Simply put, the core is the set of stable imputations (Definition 6). The players prefer participating in the grand coalition over any other coalitions if we employ the imputations in the core. In the context of DLSCG, we want all the processors to participate in the grand coalition as this provides the best opportunities in obtaining optimality. We now show that the core is non-empty for any set of processors participating in the DLSCG game.

Notation. In the following, the imputation \mathbf{x} is a vector of the form $(x_J, x_1, x_2, \dots, x_m)$, where x_J is J 's payoff and x_i , for $i = 1, 2, \dots, m$, is P_i 's payoff.

Lemma 1. *Assume the DLSCG game defined by (v, \mathbf{N}) . If $v(\mathbf{N}) = 0$, then the imputation $\mathbf{x} = (0, 0, \dots, 0)$ is the only imputation in the core.*

Proof: This game is a zero-sum, inessential coalitional game. Thus, by applying Theorem 3, we see that $\mathbf{x} = (0, 0, \dots, 0)$ is in the core. ■

Lemma 2. *Assume the DLSCG game defined by (v, \mathbf{N}) . If $v(\mathbf{N}) > 0$, then the imputation $\mathbf{x} = (C_J - T(\alpha_{\mathbf{R}}) - \sum_{i \in \mathbf{R}} \alpha_i w_i, 0, 0, \dots, 0)$ is in the core.*

Before proving the theorem, we explain why the payoffs to the processors are zero. If a processor P_i performs work, it incurs cost $\alpha_i w_i$. Processor P_i is paid $r_i = \alpha_i w_i$ to compensate it for the incurred processing cost and thus, the payoff $x_i = 0$. In the case of $\alpha_i = 0$, no costs are incurred. If we set $r_i = 0$, then P_i 's payoff is $x_i = 0$.

Proof: We claim that $\mathbf{x} = (C_J - T(\alpha_{\mathbf{R}}) - \sum_{i \in \mathbf{R}} \alpha_i w_i, 0, 0, \dots, 0)$. The imputation trivially satisfies the condition $\sum_{i \in \mathbf{N}} x_i = v(\mathbf{N})$. Applying the definition of $v(\mathbf{S})$ in (9), we note that $v(\mathbf{N}) \geq v(\mathbf{S})$, for any $\mathbf{S} \in \mathcal{P}(\mathbf{N})$. Thus, \mathbf{x} is in the core. ■

Since we show that the game under all scenarios contains at least one imputation, we know that the core is non-empty.

Theorem 8. *The core for the DLSCG game defined by (v, \mathbf{N}) is non-empty.*

Proof: It follows directly from Lemma 1 and 2. ■

Since the core of the DLSCG game is non-empty, there exists a stable imputation and if the imputation is used, the job's owner and the processors' organizations prefer the grand coalition over any other coalition. We desire this property as the grand coalition provides the best opportunity to reduce total cost.

Generally when $v(\mathbf{N}) > 0$, there is more than one imputation in the core. Any imputation \mathbf{x} satisfying the following linear program will also be in the core: $\sum_{i \in \mathbf{N}} x_i = v(\mathbf{N})$, $\sum_{i \in \mathbf{S}} x_i \geq v(\mathbf{S})$, for all $\mathbf{S} \subset \mathbf{N}$.

Since exploring all the imputations is of limited value, we do not attempt to solve the linear program and characterize the solutions.

The core solution concept gives a set of imputations that are stable, some of which have better characteristics than others. We are interested in the unique imputation satisfying the Shapley Axioms (Definition 10). The Shapley value implements the concept of "fair sharing" by which a player is paid a sum equal to its power within the coalition. In the definition of Shapley value (6) the quantity $v(\mathbf{S}) - v(\mathbf{S} - \{i\})$ is the amount by which the value of coalition $\mathbf{S} - \{i\}$ increases when i joins it. By superadditivity, we know that this quantity will never be negative.

Note that from the definition of DLSCG that the job J must be part of a coalition \mathbf{S} for $v(\mathbf{S})$ to be positive. Thus, $v(\mathbf{S}) - v(\mathbf{S} - \{J\}) = v(\mathbf{S})$. In the case of a processor P_i and any coalitions \mathbf{S} , $v(\mathbf{S}) - v(\mathbf{S} - \{P_i\}) \leq v(\mathbf{S})$.

We know that in general the exact computation of the Shapley value requires exponential time [8]. Deng and Papadimitriou [7] gave a polynomial time algorithm for computing the Shapley value for a specific graph problem. The graph has the property that a coalition is a subgraph and that the value of the coalition is the sum of the weighted edges. Unfortunately, the DLSCG model cannot be reduced to this graph problem. We desire a polynomial time algorithm for the Shapley value that will allow the computation

	w_1	w_2-w_8	w_9-w_{12}	$w_{13}-w_{16}$
fast	0.1	0.2	0.2	0.2
intermediate	0.1	0.2	0.4	0.8
slow	0.1	0.2	0.8	0.8

Table 1. Processor execution times for the three system types.

of the payoffs after the allocation is computed using BUS-LINEAR-COST.

5. Experimental Results

In this section we study by simulation the DLSCG coalitional game. We consider three distributed systems, each composed of sixteen processors (P_1, P_2, \dots, P_{16}). We characterize each system by the relative speed of the processor set. Table 1 gives the execution time of the processors for each system. Notice that the first eight processors, P_1, P_2, \dots, P_8 , are the same in all three systems. The "fast" system has the remaining eight processors with equal execution times $w_2 = w_3 = \dots = w_{16} = 0.2$. The "intermediate" system has the eight processors partitioned into two sets, with execution times 0.4 and 0.8. The "slow" system has the eight processors with execution times 0.8.

Figure 2 shows the total cost of the three systems when the communication parameter $z = 0.05$. The horizontal axis indicates the number of slowest processors that are idled, (*i.e.*, 0 indicates that all processors are assigned load, 1 that $\alpha_{16} = 0$, 2 that $\alpha_{15} = \alpha_{16} = 0$, *etc.*). All the processors in the fast system have low execution times; thus, the lowest total cost is achieved when all processors are assigned load. As processors are idled, the makespan increases, which correspondingly results in an increase in total cost. In the intermediate system, processors with execution times 0.4 and 0.8 are greater than the total cost, 0.25, of P_1 alone. When these processors are idled (number of idle processors = 8 in Figure 2), the total cost is minimized. Similar to the intermediate case, the slow system is configured to minimize the total cost when idling processors P_9-P_{16} . Note when number of idle processors equals 8 the three curves converge as processors P_9-P_{16} are idled. At this point, the three systems have the same execution times for processors P_1-P_8 .

We now compute the Shapley value for the above systems with $C_J = 2.0$. Figure 3 depicts the payoffs for the participants in the distributed system with communication parameter $z = 0.05$. As expected, the job J obtains most of the payoff ($v(\mathbf{S}) - v(\mathbf{S} - \{J\}) = v(\mathbf{S})$). When the performance of the system is reduced, the payoff to J is reduced. Note that the payoffs to the processors are non-zero. This means that, unlike the simple imputations provided in Sec-

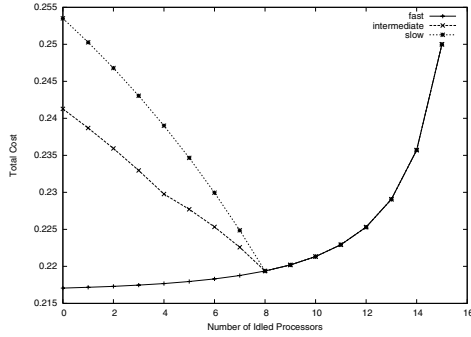


Figure 2. The total cost versus the number of idled processors for the distributed system with $z = 0.05$.

tion 4, the processors are making profit. Processors with identical execution times receive the same payoff which is a consequence of the symmetry condition of the Shapley Axioms (Definition 10). As the fast processors execute more load in the slower systems, they receive a greater payoff than they do in the fast system.

6. Conclusion

In this paper we designed a coalitional game that models the divisible load scheduling problem on a distributed system interconnected with a bus network. The objective of the game is to minimize the total cost, which is the sum of the makespan and the costs incurred in executing the job. The job's owner is paid upon execution of the job. Additionally, the owner must compensate the processors for the incurred costs. We show that the game has a non-empty core, which signifies that the job and the processors will not leave the grand coalition. This is fortuitous as the grand coalition provides the best opportunities for minimizing the total cost. We then examined the "fair sharing" of the Shapley value. Finally, we studied by simulation the properties of the proposed coalitional game considering different system configurations. For future work we plan to investigate coalitional game models for other resource allocation problems in distributed computing.

References

[1] N. Andelman and Y. Mansour. A sufficient condition for truthfulness with single parameter agents. In *Proc. of the 7th ACM Conference on Electronic Commerce (EC '06)*, pages 8–17, June 2006.

[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.

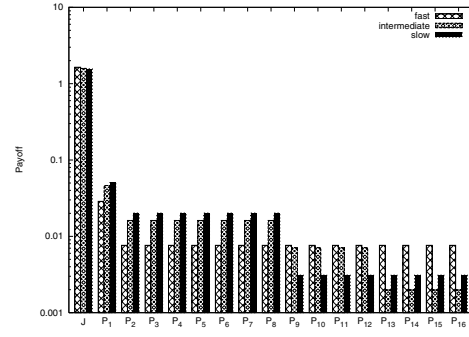


Figure 3. The Shapley values for the distributed system with $z = 0.05$.

[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] T. E. Carroll and D. Grosu. Selfish multi-user task scheduling. In *Proc. of the 5th International Symp. on Parallel and Distributed Computing (ISPDC '06)*, pages 99–106, July 2006.

[6] T. E. Carroll and D. Grosu. A strategyproof mechanism for scheduling divisible loads in tree networks. In *Proc. of the 20th IEEE International Parallel and Distributed Processing Symp. (IPDPS 2006)*, Apr. 2006.

[7] X. Deng and C. H. Papadimitriou. On the complexity of cooperative solution concepts. *Math. Operational Research*, 19(2):257–266, 1994.

[8] U. Faigle and W. Kern. The shapley value for cooperative games under precedence constraints. *International J. of Game Theory*, 21(3):249–266, Sept. 1992.

[9] U. Faigle and W. Kern. Approximate core allocation for binpacking games. *SIAM J. Discrete Math*, 11(3):387–399, Aug. 1998.

[10] U. Faigle and W. Kern. On the core of ordered submodular cost games. *Math. Programming*, 87:483–499, 2000.

[11] 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.

[12] D. Grosu and T. E. Carroll. A strategyproof mechanism for scheduling divisible loads in distributed systems. In *Proc. of the 4th International Symp. on Parallel and Distributed Computing (ISPDC'05)*, pages 83–90. IEEE Computer Society, July 2005.

[13] G. Owen. *Game Theory*. Academic Press, New York, NY, USA, 3rd edition, 1995.

[14] L. S. Shapley. *A Value for n-Person Games*, pages 307–317. Contributions to the Theory of Games II, H. W. Kuhn and A. W. Tucker (eds.). Princeton University Press, 1953.

[15] 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.