# A Merge-and-Split Mechanism for Dynamic Virtual Organization Formation in Grids

Lena Mashayekhy, *Student Member, IEEE,* and Daniel Grosu, *Senior Member, IEEE*

**Abstract**—Executing large scale application programs in grids requires resources from several Grid Service Providers (GSPs). These providers form Virtual Organizations (VOs) by pooling their resources together to provide the required capabilities to execute the application. We model the VO formation in grids using concepts from coalitional game theory and design a mechanism for VO formation. The mechanism enables the GSPs to organize into VOs reducing the cost of execution and guaranteeing maximum profit for the GSPs. Furthermore, the mechanism guarantees that the VOs are stable, that is, the GSPs do not have incentives to break away from the current VO and join some other VO. We perform extensive simulation experiments using real workload traces to characterize the properties of the proposed mechanism. The results show that the mechanism produces VOs that are stable yielding high revenue for the participating GSPs.

**Index Terms**—Virtual organizations, Grid computing, VO formation, Coalitional game theory.

## 1 Introduction

Grid computing systems enable efficient collaboration among researchers and provide essential support for conducting cutting-edge science and engineering research. These systems are composed of geographically distributed resources (computers, storage etc.) owned by autonomous organizations. Resource management in such open distributed environments is a very complex problem which if solved leads to efficient utilization of resources and faster execution of applications. The existent grid resource management systems [1], [2], [3] do not explicitly address the formation and management of Virtual Organizations (VO) [4]. We argue that incentives are the main driving forces in the formation of VOs in grids, and thus, it is imperative to take them into account when designing VO formation mechanisms. To provide better performance and increase the efficiency it is essential to develop mechanisms for VO formation that take into account the behavior of the participants and provide incentives to contribute resources.

A VO is "a collection of geographically distributed functionally and/or culturally diverse entities that are linked by electronic forms of communication and rely on lateral, dynamic relationships for coordination" [5]. While this definition covers a wide range of VOs, this paper focuses on more specific types of VOs, those that emerge in the existing grids. The VOs we are focusing on are alliances among various organizations that collaborate and pool their resources to execute large scale applications [4]. More specifically, we will consider VOs that form dynamically and are short-lived, that is, they are formed in order to execute a given task and once the task is completed they are dismantled.

The life-cycle of a VO can be divided into four phases: identification, formation, operation, and dissolution. During the initiation phase the possible partners and the VO's objectives are identified. In the formation phase, the potential partners negotiate the exact terms, the goal, and the duration of collaboration. Once the VO is formed it enters the operation phase in which the members of the VO collaborate in solving a specific task. Once the VO completes the task, it dissolves. This paper focuses on designing mechanisms for the second phase, the formation of VOs. We model the VO formation as a coalitional game where GSPs decide to form VOs in such a way that each GSP maximizes its own profit, the difference between revenue and costs. The VO formation phase can be further divided into three subphases: coalitional structure generation, solving the task allocation problem of each coalition, and dividing the value among the members of the coalition. In the first sub-phase, the set of GSPs is partitioned into disjoint VOs. In the second sub-phase, the task assignment that maximizes the profit of the participating GSPs in each VO is determined. In the third sub-phase, the profit obtained by the VO is divided among the members of the VO. A GSP will choose to participate in a VO if its profit is not negative. The VOs provide the composite resource needed to execute applications. A VO is traditionally conceived for the sharing of resources, but it can also represent a business model [4]. In this work, a VO is a coalition of GSPs who desire to maximize their individual profits and are largely indifferent about the global welfare. We design a VO formation mechanism based on concepts from coalitional game theory [6]. The model that we consider consists of a set of GSPs and a grid user that submits a program and a specification

• *L. Mashayekhy and D. Grosu are with the Department of Computer Science, Wayne State University, Detroit, MI, 48202.*
*E-mail: mlena@wayne.edu, dgrosu@wayne.edu*

consisting of a deadline and payment. A subset of GSPs will form a VO in order to execute the program before its deadline. The objective of each GSP is to form a VO that yields the highest individual profit.

In Appendix A of the supplemental material we provide an extensive review of the related work.

### 1.1 Our Contribution

We address the problem of VO formation in grids by designing a mechanism that allows the GSPs to make their own decisions to participate in VOs. In this mechanism, coalitions of GSPs decide to merge and split in order to form a VO that maximizes the individual payoffs of the participating GSPs. The mechanism produces a stable VO structure, that is, none of the GSPs has incentives to merge to another VO or split from a VO to form another VO. We propose the use of a selfish split rule in order to find the optimal VO in the VO structure. The mechanism determines the mapping of the tasks to each of the VOs that minimizes the cost of execution by using a branch-and-bound method. As a result, in each step of the mechanism the mapping provides the maximum individual payoffs for the participating GSPs that make the decisions to further merge and split. We analyze the properties of our proposed VO formation mechanism and perform extensive simulation experiments using real workload traces from the Parallel Workloads Archive [7]. The results show that the proposed mechanism determines a stable VO that maximizes the individual payoffs of the participating GSPs.

### 1.2 Organization

The rest of the paper is organized as follows. In Section 2, we describe the VO formation problem and the system model we consider. In Section 3, we describe the game theoretic framework used to design the proposed VO formation mechanism, present the proposed mechanism, and characterize its properties. In Section 4, we evaluate the mechanism by extensive simulation experiments. In Section 5, we summarize our results and present possible directions for future research.

## 2 VO Formation as a Coalitional Game

In this section, we model the VO formation in grids as a coalitional game. We first describe the system model which considers that a user wants to execute a large-scale application program $\mathcal{T}$ consisting of $n$ independent tasks $\{T_1, T_2, \ldots, T_n\}$ on the available set of grid service providers (GSPs) by a given deadline $d$. Application programs consisting of several independent tasks are representative for a wide range of problems in science and engineering [8], [9]. Each task $T \in \mathcal{T}$ composing the application program is characterized by its workload $w(T)$, which can be defined as the amount of floating-point operations required to execute the task. Executing the application program $\mathcal{T}$ requires a large number of resources which cannot be provided by a single GSP. Thus, several GSPs pool their resources together to execute the application. We consider that a set of $m$ GSPs, $\mathcal{G} = \{G_1, G_2, \ldots, G_m\}$, are available and are willing to provide resources for executing programs. Here, we assume that the GSPs are driven by incentives in the sense that they will execute a task only if they make some profit out of it. More specifically, the GSPs are assumed to be self-interested and welfare-maximizing entities. Each service provider $G \in \mathcal{G}$ owns several computational resources which are abstracted as a single machine with speed $s(G)$. The speed $s(G)$ gives the number of floating-point operations per second that can be executed by GSP $G$. Therefore, the execution time of task $T$ at GSP $G$ is given by the *execution time function* $t : \mathcal{T} \times \mathcal{G} \rightarrow \mathbb{R}^+$, where $t(T, G) = \frac{w(T)}{s(G)}$. This corresponds to the execution time function used in the scheduling on related parallel machines model [10]. Another possibility would be to consider the function that corresponds to the scheduling on unrelated machines model, that is $t(T, G) = \frac{w(T)}{s(T,G)}$, where the speed of executing a task $s$ depends on both the task and the machine on which it is executed. Our proposed coalitional game and VO formation mechanism works with both types of functions, since the task mapping problem described in the next section is defined in terms of $t(T, G)$ and not in terms of workloads and speeds. Thus, without loss of generality, in the rest of the paper, we use the execution time function corresponding to the scheduling on related machines model. We also assume that once a task is assigned to a GSP, the task is neither preempted nor migrated.

A GSP incurs cost for executing a task. The cost incurred by GSP $G \in \mathcal{G}$ when executing task $T \in \mathcal{T}$ is given by $c(T, G)$, where $c : \mathcal{T} \times \mathcal{G} \rightarrow \mathbb{R}^+$ is the *cost function*. Furthermore, we assume that a GSP has zero fixed costs and its variable costs are given by the function $c$. A user is willing to pay a price $P$ less than her available budget $B$ if the program is executed to completion by deadline $d$. If the program execution exceeds $d$, the user is not willing to pay any amount that is, $P = 0$.

Since a single GSP does not have the required resources for executing a program, GSPs form VOs in order to have the necessary resources to execute the program and more importantly, to maximize their profits. The profit is simply defined as the difference between the payment received by a GSP and its execution costs. If the profit is negative (i.e., a loss), the GSP will choose not to participate.

We model the VO formation problem as a coalitional game. A *coalitional game* [11] is defined by the pair $(\mathcal{G}, v)$, where $\mathcal{G}$ is the set of players and $v$ is a real-valued function called the *characteristic function*, defined on $S \subseteq \mathcal{G}$ such that $v : S \rightarrow \mathbb{R}^+$ and $v(\emptyset) = 0$. In our model, the players are the GSPs that form VOs which are coalitions of GSPs. In this work, we use the terms VO and coalition interchangeably.

Each subset $S \subseteq \mathcal{G}$ is a *coalition*. If all the players form a coalition, it is called the *grand coalition*. A coalition has a *value* given by the characteristic function $v(S)$ representing the profit obtained when the members of a coalition work as a group. For each coalition of GSPs $S \subseteq \mathcal{G}$, there exists a mapping $\pi_S : \mathcal{T} \rightarrow S$, which assigns task $T \in \mathcal{T}$ to service provider $G \in S$. To maximize the profit obtained by a VO, we need to find an optimal mapping of all the tasks on the members of VO in such a way that the mapping minimizes the execution cost. We call this task assignment problem the MIN-COST-ASSIGN problem.

MIN-COST-ASSIGN finds a mapping of $n$ tasks to $k$ GSPs in VO $S$, where $k = |S|$. The goal is to minimize the cost of execution. We consider the following decision variables:

$$\sigma_S(T,G) = \begin{cases} 1 & \text{if } \pi_S(T) = G, \\ 0 & \text{if } \pi_S(T) \neq G. \end{cases} \quad (1)$$

We formulate the MIN-COST-ASSIGN problem as an Integer Program (IP) as follows:

$$\text{Minimize } C(\mathcal{T},S) = \sum_{T\in\mathcal{T}} \sum_{G\in S} \sigma_S(T,G)c(T,G), \quad (2)$$

$$\text{Subject to: } \sum_{T\in\mathcal{T}} \sigma_S(T,G)t(T,G) \leq d, \ (\forall G \in S), \quad (3)$$

$$\sum_{G\in S} \sigma_S(T,G) = 1, \ (\forall T \in \mathcal{T}), \quad (4)$$

$$\sum_{T\in\mathcal{T}} \sigma_S(T,G) \geq 1, \ (\forall G \in S), \quad (5)$$

$$\sigma_S(T,G) \in \{0,1\}, \ (\forall G \in S \text{ and } \forall T \in \mathcal{T}). \quad (6)$$

The objective function (2) represents the costs incurred for executing the program $\mathcal{T}$ on $S$ under the mapping. Constraints (3) ensure that each GSP can execute its assigned tasks by the deadline. Constraints (4) guarantee that each task $T \in \mathcal{T}$ is assigned to exactly one GSP. Constraints (5) ensure that each GSP $G \in S$ is assigned at least one task. Constraints (6) represent the integrality requirements for the decision variables.

We define the following characteristic function for our proposed VO formation game:

$$v(S) = \begin{cases} 0 & \text{if } |S| = 0 \text{ or IP is not feasible,} \\ P - C(\mathcal{T},S) & \text{if } |S| > 0 \text{ and IP is feasible,} \end{cases} \quad (7)$$

where $|S|$ is the cardinality of $S$. Note that $v(S)$ can be negative if $C(\mathcal{T},S) > P$. That means GSPs in $S$ incur cost.

The objective of each GSP is to determine the membership in a coalition that gives the highest share of profit. There are different ways to divide the profit $v(S)$ earned by coalition $S$ among its members. Traditionally, the *Shapley value* [12] would be employed, but computing the Shapley value requires iterating over every partition of a coalition, an exponential time endeavor. Another rule for payoff division is *equal sharing* of the profit among members. Equal sharing provides a tractable way to determine the shares and has been successfully used as a payoff division rule in other systems where tractability is critical (e.g., [13]). For this reason we adopt here the equal sharing of the profit as the payoff division rule.

Due to their welfare-maximizing behavior, the GSPs prefer to form a low profit coalition if their profit divisions are higher than those obtained by participating in a high profit coalition. Therefore, a service provider $G$ determines its preferred coalition $S$, where $G \in S$ by solving:

$$\max_{(S)} \frac{P - C(\mathcal{T},S)}{|S|} \quad (8)$$

The GSPs' goal is to maximize their own profit by solving the optimization problem given in equation (8). Therefore, minimizing the cost $C(\mathcal{T},S)$ by solving the MIN-COST-ASSIGN problem maximizes the profit $P - C(\mathcal{T},S)$ earned by a VO. A VO obtains profit and then the profit is divided among participating GSPs. As a result, a GSP prefers a VO that provides the highest profit among all possible VOs.

The *payoff* or the *share* of GSP $G$ part of coalition $S$, denoted by $x_G(S)$ is given by $x_G(S) = \frac{v(S)}{|S|}$. Thus, the payoff vector $\mathbf{x}(\mathcal{G}) = (x_{G_1}(\mathcal{G}), \cdots, x_{G_m}(\mathcal{G}))$ gives the payoff divisions of the grand coalition. A solution concept for coalitional games is a payoff vector that divides the payoff among the players in some fair way. The primary concern for any coalitional game is stability (i.e., players do not have incentives to break away from a given coalition). One of the solution concepts used to asses the stability of coalitions is the *core*. In order to define the core we need to introduce first the concept of imputation.

*Definition 1 (Imputation):* An *imputation* is a payoff vector such that $x_G(\mathcal{G}) \geq v(G)$, for all GSPs $G \in \mathcal{G}$, and $\sum_{G\in\mathcal{G}} x_G(\mathcal{G}) = v(\mathcal{G})$.

The first condition says that by forming the grand coalition the profit obtained by each member $G$ participating in the grand coalition is not less than the one obtained when acting alone. The second condition says that the entire profit of the grand coalition should be divided among its members.

*Definition 2 (Core):* The *core* is a set of imputations such that $\sum_{G\in S} x_G(\mathcal{G}) \geq v(S), \forall S \subseteq \mathcal{G}$, i.e., for all coalitions, the payoff of any coalition is not greater than the sum of the payoffs of its members in the grand coalition.

The core contains payoff vectors that make the players want to form the grand coalition. The existence of a payoff vector in the core shows that the grand coalition is stable. Therefore, a payoff division is in the core if no player has an incentive to leave the grand coalition to join another coalition in order to obtain higher profit. The core of the VO formation game can be empty. If the

TABLE 1: The program settings.

| | cost | | speed | time | |
|---|---|---|---|---|---|
| | $T_1$ | $T_2$ | $s$ | $T_1$ | $T_2$ |
| $G_1$ | 3 | 4 | 8 | 3 | 4.5 |
| $G_2$ | 3 | 4 | 6 | 4 | 6 |
| $G_3$ | 4 | 5 | 12 | 2 | 3 |

deadline $d = 5$ payment $P = 10$

TABLE 2: The mappings for each coalition.

| $S$ | Mapping | $v(S)$ |
|---|---|---|
| $\{G_1\}$ | NOT FEASIBLE | 0 |
| $\{G_2\}$ | NOT FEASIBLE | 0 |
| $\{G_3\}$ | $T_1, T_2 \rightarrow G_3$ | 1 |
| $\{G_1, G_2\}$ | $T_2 \rightarrow G_1$; $T_1 \rightarrow G_2$ | 3 |
| $\{G_1, G_3\}$ | $T_1 \rightarrow G_1$; $T_2 \rightarrow G_3$ | 2 |
| $\{G_2, G_3\}$ | $T_1 \rightarrow G_2$; $T_2 \rightarrow G_3$ | 2 |
| $\{G_1, G_2, G_3\}$ | $T_2 \rightarrow G_1$; $T_1 \rightarrow G_2$ | 3 |

grand coalition does not form, independent and disjoint coalitions would form.

To show that the core of the proposed VO formation game can be empty we consider an example with three GSPs $\mathcal{G} = \{G_1, G_2, G_3\}$ and a two-task program $\mathcal{T} = \{T_1, T_2\}$, where $T_1$ and $T_2$ workloads are 24 and 36 million floating-point operations, respectively. In Table 1, we give the cost of executing each task on each GSP, the speed of each GSP, and the execution time of each task on each GSP. As an example, $G_1$ incurs 3 units of cost to execute $T_1$ and 4 units of cost to execute $T_2$. The speed of $G_1$ is 8 MFLOPS (Mega floating-point operations per second). Based on the definition of the time function, the execution time of task $T_1$ and $T_2$ are 3 and 4.5 seconds, respectively.

If $G_1$, $G_2$ and $G_3$ execute the entire program separately, then the program completes in 7.5, 10 and 5 units of time, respectively. We assume that the user has specified a deadline $d = 5$ and a payment $P = 10$. Please note that $\{G_1, G_2, G_3\}$ is not feasible based on constraint (5) which requires that each GSP be assigned at least one task. In this example, we relax constraint (5) to show that even if the grand coalition is considered feasible the core of the game is empty. The mapping and the values $v(S)$ are given in Table 2. Since $x_{G_1}(\mathcal{G}) + x_{G_2}(\mathcal{G}) \geq v(\{G_1, G_2\})$, $x_{G_3}(\mathcal{G}) \geq v(G_3)$, and $x_{G_1}(\mathcal{G}) + x_{G_2}(\mathcal{G}) + x_{G_3}(\mathcal{G}) = v(\{G_1, G_2, G_3\})$ are not satisfied, there is no payoff vector in the core, and thus, the core of the game is empty. More than this, since we are using equal sharing, the profit of $G_1$ and $G_2$ in coalition $\{G_1, G_2\}$ is 1.5 while the profit of each GSP in the grand coalition is 1. Thus, $\{G_1, G_2\}$ have an incentive to deviate from the grand coalition, and $G_3$ cannot be a member of the coalition. As a result, $\{G_1, G_2\}$ will execute the program.

In this paper, we assume that if a GSP does not execute a task it receives a payoff of 0. If there are some GSPs that do not participate in executing any task of the program, they are not considered members of the VO executing the application.

## 3 VO FORMATION MECHANISM

In this section, we introduce few concepts from coalitional game theory needed to describe the proposed mechanism and then present the mechanism.

### 3.1 Coalition Formation Framework

Coalition formation theory investigates the coalitional structures in games where the grand coalition does not form. *Coalition formation* [14] is the partitioning of the players into disjoint sets. A coalition structure $\mathcal{CS} = \{S_1, S_2, \ldots, S_h\}$ forms a partition of the set of GSPs $\mathcal{G}$ such that each player is a member of exactly one coalition, *i.e.*, $S_i \cap S_j = \emptyset$ for all $i$ and $j$, where $i \neq j$ and $\bigcup_{S_i \in \mathcal{CS}} S_i = \mathcal{G}$. The problem of finding the optimal coalition structure is NP-complete [15]. Enumerating all coalition structures to find the optimal coalition structure is not feasible since the possible number of coalition structures is $B_m$, the $m$-th Bell number [16] which gives the number of partitions of a set of size $m$, where $m = |\mathcal{G}|$.

In the VO formation game defined in the previous section only one of the coalitions in the coalition structure is selected to execute the application program. The selected coalition is the one that yields the highest individual payoff for all of its members. The coalitions that cannot complete the program within the deadline will not be considered since the payoff for such coalitions is zero.

The following concepts are used in the design of the VO formation mechanism.

*Definition 3 (Collection):* A *collection* in the grand coalition $\mathcal{G}$, is defined as the set $\mathcal{C} = \{S_1, \cdots, S_k\}$ of mutually disjoint coalitions. If $\cup_{j=1}^{k} S_j = \mathcal{G}$, the collection $\mathcal{C}$ is called a *partition* of $\mathcal{G}$.

*Definition 4 (Comparison):* A *collection comparison* $\triangleright$ is defined to compare two collections $A$ and $B$ that are partitions of the same subset $S \subseteq \mathcal{G}$. $A \triangleright B$ implies that the way $A$ partitions $S$ is preferred to the way $B$ partitions $S$.

In the proposed VO formation game, a welfare-maximizing GSP will determine its coalition by considering the profit it earns and not the coalition value. Thus, comparison relations among collections are defined based on the GSPs' individual payoffs. These comparison relations correspond to the merge and split rules which will be defined later in this section. We consider two collections $\hat{S} = \{\cup_{j=1}^{k} S_j\}$ and $\{S_1, \cdots, S_k\}$ from the same subset. We define two comparison relations, the *merge comparison* $\triangleright_m$ and the *split comparison* $\triangleright_s$, based on the individual payoffs as follows:

$$\begin{aligned} \hat{S} \triangleright_m \{S_1, \cdots, S_k\} \iff & \{\forall j \in \{1, \ldots, k\}, \forall G_i \in \hat{S} \cap S_j; \\ & x_i(\hat{S}) \geq x_i(S_j) \text{ and } \exists j \in \{1, \ldots, k\}, \\ & \exists G_r \in S_j \,; x_r(\hat{S}) > x_r(S_j)\} \end{aligned} \tag{9}$$

$$\{S_1, \cdots, S_k\} \triangleright_s \hat{S} \iff \{\exists j \in \{1, \ldots, k\}, \forall G_i \in \hat{S} \cap S_j; \; x_i(S_j) \geq x_i(\hat{S}) \text{ and } \exists G_r \in S_j \,; x_r(S_j) > x_r(\hat{S})\} \tag{10}$$

Equation (9) implies that collection $\hat{S}$ composed of only one coalition $\{\cup_{j=1}^{k} S_j\}$ is preferred over $\{S_1, \cdots, S_k\}$, if at least one player $G_r$ is able to improve its payoff without decreasing other players' payoffs. Note that, the merge comparison is a standard Pareto-dominance relation. Equation (10) implies that collection $\{S_1, \cdots, S_k\}$ is preferred over $\hat{S}$, if at least one coalition $S_j$ is able to keep the payoffs of its members while at least one of its members, $G_r$, is able to improve its payoff regardless of the effect on the other players outside of $S_j$.

Using the defined comparison relations, we propose a VO formation mechanism involving two types of rules as follows [14]:

*Merge Rule:* Merge any set of coalitions $\{S_1, \cdots, S_k\}$, where $\{\cup_{j=1}^{k} S_j\} \triangleright_m \{S_1, \cdots, S_k\}$.

*Split Rule:* Split any coalition $\hat{S} = \{\cup_{j=1}^{k} S_j\}$, where $\{S_1, \cdots, S_k\} \triangleright_s \{\cup_{j=1}^{k} S_j\}$.

Coalitions decide to merge only if at least one GSP is able to strictly improve its individual payoff through the merge rule without decreasing the other GSPs' payoffs. Therefore, the merge rule is an agreement among the GSPs to operate together if it is beneficial for them.

As we mentioned before, one of the formed coalitions, the final coalition, executes the program, thus, the formation of the rest of the coalitions is not important. The reason for this is that the rest of the GSPs which are not in the final coalition can participate again in another coalition formation process for executing another application program. Therefore, a coalition decides to split only if there is at least one sub-coalition that strictly improves the individual payoffs of its constituent GSPs. Under the split rule, the individual payoffs of the other sub-coalitions may decrease. The split rule can be seen as the implementation of a *selfish* decision by a coalition, which does not take into account the effect of the split on the other coalitions.

Two coalitions $S_i$ and $S_j$ decide to merge based on the merge comparison defined by (9) where all of GSPs in $S_i \cup S_j$ are able to keep or improve their individual payoffs. A GSP individual payoff is computed based on (8) while satisfying the deadline constraint. As a result, the merge occurs if the following two inequalities are satisfied where at least one of them must be strict.

$$\frac{P - C(\mathcal{T}, S_i \cup S_j)}{|S_i \cup S_j|} \geq \frac{P - C(\mathcal{T}, S_i)}{|S_i|} \tag{11}$$

$$\frac{P - C(\mathcal{T}, S_i \cup S_j)}{|S_i \cup S_j|} \geq \frac{P - C(\mathcal{T}, S_j)}{|S_j|} \tag{12}$$

Since $|S_i \cup S_j| > |S_i|$ and $|S_i \cup S_j| > |S_j|$, in order for a GSP in $S_i$ to keep or improve its payoff, $P - C(\mathcal{T}, S_i \cup S_j) \geq P - C(\mathcal{T}, S_i)$, and it should be the same for a GSP in $S_j$. Thus, $C(\mathcal{T}, S_i \cup S_j) \leq C(\mathcal{T}, S_i)$ and $C(\mathcal{T}, S_i \cup S_j) \leq C(\mathcal{T}, S_j)$. That means that coalitions can only merge when the cost of the formed coalition by merge is less than their cost.

For the split rule, a coalition $\hat{S}$ decides to split into two coalitions $S_i$ and $S_j$ based on the split comparison defined by (10), where all GSPs in $S_i$, $S_j$, or both are able to keep or improve their individual payoffs. Thus, $\hat{S}$ splits if at least one of the following inequalities is satisfied.

$$\frac{P - C(\mathcal{T}, \hat{S})}{|\hat{S}|} < \frac{P - C(\mathcal{T}, S_i)}{|S_i|} \tag{13}$$

$$\frac{P - C(\mathcal{T}, \hat{S})}{|\hat{S}|} < \frac{P - C(\mathcal{T}, S_j)}{|S_j|} \tag{14}$$

That means that the individual payoff of each GSP in at least one of the splitted coalitions, $S_i$ or $S_j$, should be higher than its individual payoff in $\hat{S}$.

The stability of the resulting coalition structure is characterized using the concept of defection function $\mathbb{D}$ [14].

*Definition 5 (Defection function):* A *defection function* $\mathbb{D}$ is a function which associates with each partition $\mathcal{P}$ of $\mathcal{G}$ a group of collections in $\mathcal{G}$.

A partition $\mathcal{P}$ is $\mathbb{D}$-stable if no group of players is interested in leaving $\mathcal{P}$. Thus, the players can only form the collections allowed by $\mathbb{D}$. A defection function $\mathbb{D}_P$ which allows the formation of all partitions of the grand coalition was proposed by Apt and Witzel [14]. $\mathbb{D}_P$-stability is defined based on this function. $\mathbb{D}_P$ allows any group of players to leave the partition $\mathcal{P}$ of $\mathcal{G}$ through merge-and-split rules to create another partition in $\mathcal{G}$. Therefore, $\mathbb{D}_P$-stability means that no coalition has an incentive to merge or split.

In order to clarify the above concepts, let us consider the example in Section 2 with three GSPs $\mathcal{G} = \{G_1, G_2, G_3\}$ and a two-task program $\mathcal{T} = \{T_1, T_2\}$. $G_1$ and $G_2$ cannot perform the program by acting alone since the deadline is exceeded, thus they receive zero. $G_3$ receives 1 by acting alone. Consider that $G_3$ communicates with $G_2$ in order to merge. Based on the values in Table 2, $\{G_2, G_3\} \triangleright_m \{\{G_2\}, \{G_3\}\}$, since $G_2$ improves its payoff while $G_3$ keeps its original payoff. Thus, the merge is optimal. Now, there are two coalitions $\{G_1\}$ and $\{G_2, G_3\}$. $G_1$ communicates with $\{G_2, G_3\}$ in order to merge and since $\{G_1, G_2, G_3\} \triangleright_m \{\{G_1\}, \{G_2, G_3\}\}$, the merge occurs. In this case, $G_1$ improves its payoff while $G_2$ and $G_3$ keep their previous payoff. Now, $\{G_1, G_2, G_3\}$ tries to split. The only sub-coalition that can split is $\{G_1, G_2\}$ since $\{\{G_1, G_2\}, \{G_3\}\} \triangleright_s \{G_1, G_2, G_3\}$, i.e., $G_1$ and $G_2$ improve their payoffs by splitting. None of $G_1$ and $G_2$ wants to split from coalition $\{G_1, G_2\}$. There are no coalitions able to merge or split any further. Even if GSPs use a different order of merging, at the end of the merge step the grand coalition forms, and then $\{G_1, G_2\}$ splits. As a result, partition $\{\{G_1, G_2\}, \{G_3\}\}$ is $\mathbb{D}_P$-stable.

### 3.2 Merge-and-Split VO Formation Mechanism (MSVOF)

The proposed merge-and-split VO formation mechanism (MSVOF) is given in Algorithm 1. The mechanism is executed by a trusted party that also facilitates the communication among VOs/GSPs. The design of the mechanism assumes that the players report their true execution speeds and costs to the trusted party executing the mechanism. This is needed in order to guarantee the stability of the VO formed by the mechanism. In practice, the mechanism will require the verification of these parameters as part of each GSP's agreement to participate in the mechanism. Achieving truthfulness (i.e., incentivizing agents to report their true costs and execution times through payments) in this context without such verification procedures, is a difficult task (if not impossible) that needs significant further research.

MSVOF uses a branch-and-bound method to solve the MIN-COST-ASSIGN problem for the formed VOs, and thus, to obtain the mapping of the tasks to GSPs. We will denote by B&B-MIN-COST-ASSIGN($S_i$) the function that implements the branch-and-bound method for solving the MIN-COST-ASSIGN problem for a VO $S_i$. Branch-and-bound are methods for solving global optimization problems [17]. They are commonly used for solving integer programs by implicit enumeration in which linear programming relaxations provide the bounds [18]. These methods are based on the observation that a systematic enumeration of integer solutions has a tree structure [19]. The main idea in branch-and-bound is to avoid growing the whole tree as much as possible by permanently discarding nodes, or any of its descendants, which will never be either feasible or optimal. A branch-and-bound method requires two procedures. The first one is a branching procedure that returns two or more smaller sets of the problem. The result of this step is a tree structure (the search tree) whose nodes are the subsets of the problem. The second one is a bounding procedure that computes upper and lower bounds for the objective function within a given subset of the problem by solving the relaxed linear program. Updating bounds for active nodes in a tree enables the method to prune some non-promising branches. We use the branch-and-bound method but any other mapping algorithms such as those solving variants of the General Assignment Problem (GAP) can also be used by the VOs to find the minimum cost mapping of the tasks on the GSPs.

MSVOF starts with a coalition structure $\mathcal{CS}$ consisting of every singleton $G_i \in \mathcal{G}$ as a coalition $S_i$ in $\mathcal{CS}$ (line 1) and it checks if all tasks can be executed by the individual GSPs (line 2). Then, $v(S_i)$ is computed based on the allocation. MSVOF uses a matrix $visited$ to keep track of all pairs of coalitions in $\mathcal{CS}$ that are visited for merging. By using this matrix, all possible combinations of two coalitions in $\mathcal{CS}$ are visited during the merge process (lines 8-26). The merge process starts every time by choosing two non-visited coalitions in $\mathcal{CS}$ randomly, e.g., $S_i$ and $S_j$. B&B-MIN-COST-ASSIGN is called to find an optimal allocation for the application program $\mathcal{T}$ on $S_i \cup S_j$. If $S_i \cup S_j \rhd_m \{S_i, S_j\}$, then coalitions $S_i$ and $S_j$ decide to merge. Therefore, all the members receive higher profit by merging since equal sharing is used to divide the profit. $S_i \cup S_j$ is saved in $S_i$, and $S_j$ is removed from $\mathcal{CS}$. Since $S_i$ is changed, it can be selected in the next merge steps. Thus, $visited[S_i][S_k]$ for all $S_k \in \mathcal{CS}$, $k \neq i$ is set to false. The merge process tries to find another pair of non-visited coalitions suitable for merging. If all the coalitions are tested and a merge does not occur, or the grand coalition forms, the merge process ends.

**Algorithm 1** Merge-and-Split VO Formation Mechanism (MSVOF)

1: $\mathcal{CS} = \{\{G_1\}, \cdots, \{G_m\}\}$
2: Map program $\mathcal{T}$ on each $S_i \in \mathcal{CS}$
3: **repeat**
4: $stop \leftarrow True$
5: **for all** $S_i, S_j \in \mathcal{CS}, i \neq j$ **do**
6: $visited[S_i][S_j] \leftarrow False$
7: **end for**
8: {Merge process starts:}
9: **repeat**
10: $flag \leftarrow True$
11: Randomly select $S_i, S_j \in \mathcal{CS}$ for which $visited[S_i][S_j] = False, i \neq j$
12: $visited[S_i][S_j] \leftarrow True$
13: B&B-MIN-COST-ASSIGN($S_i \cup S_j$) {Map program $\mathcal{T}$ on $S_i \cup S_j$}
14: **if** $S_i \cup S_j \rhd_m \{S_i, S_j\}$ **then**
15: $S_i \leftarrow S_i \cup S_j$ {merge $S_i$ and $S_j$}
16: $S_j \leftarrow \emptyset$ {$S_j$ is removed from $\mathcal{CS}$}
17: **for all** $S_k \in \mathcal{CS}, k \neq i$ **do**
18: $visited[S_i][S_k] \leftarrow False$
19: **end for**
20: **end if**
21: **for all** $S_i, S_j \in \mathcal{CS}, i \neq j$ **do**
22: **if** not $visited[S_i][S_j]$ **then**
23: $flag \leftarrow False$
24: **end if**
25: **end for**
26: **until** $(|\mathcal{CS}| = 1)$ or $(flag = True)$
27: {Split process starts:}
28: **for all** $S_i \in \mathcal{CS}$ where $|S_i| > 1$ **do**
29: **for all** partitions $\{S_j, S_k\}$ of $S_i$, where $S_i = S_j \cup S_k, S_j \cap S_k = \emptyset$ **do**
30: B&B-MIN-COST-ASSIGN($S_j$) {Map program $\mathcal{T}$ on $S_j$}
31: B&B-MIN-COST-ASSIGN($S_k$) {Map program $\mathcal{T}$ on $S_k$}
32: **if** $\{S_j, S_k\} \rhd_s S_i$ **then**
33: $S_i \leftarrow S_j$ {that is $\mathcal{CS} = \mathcal{CS} \setminus S_i$}
34: $\mathcal{CS} = \mathcal{CS} \bigcup S_k$
35: $stop \leftarrow False$
36: Break (one split occurs; no need to check other splits)
37: **end if**
38: **end for**
39: **end for**
40: **until** $stop = True$
41: Find $k = \arg\max_{S_i \in \mathcal{CS}} \{v(S_i)/|S_i|\}$
42: Map and execute program $\mathcal{T}$ on VO $S_k$

The coalition structure $\mathcal{CS}$ obtained by the merge process is then subject to splits. In the split process, all coalitions that have more than one member are subject to splitting (lines 27-40). The mechanism tries to split $S_i$ that has more than one member into two disjoint coalitions $S_j$ and $S_k$ where $S_j \cup S_k = S_i$. The for loop in line 29 iterates over all possible partitions in the co-lexicographical order introduced by Knuth [16]. The problem of finding all partitions of a set, is modeled as the problem of partitioning of an integer. Since the split checks the partitions of a set into two subsets, we use the partitioning of an integer into two parts, i.e., two positive integers, where the sum of those is equal to the integer. For example, to partition a set of four GSPs, we partition 15 into two integers. The binary representations of those integers indicates which GSPs are selected in a subset (e.g., $15 = 4 + 11$ which is equivalent to $1111 = 0010 + 1101$, that means that the set $\{G_1, G_2, G_3, G_4\}$ is partitioned into $\{\{G_3\}, \{G_1, G_2, G_4\}\}$). To increase the speed of the splitting process, we check the subsets with the largest number of GSPs of these partitions first and if they are not feasible there is no reason to check their subsets. This way we reduce the number of partitions that have to be checked and implicitly the execution time of the mechanism. B&B-MIN-COST-ASSIGN is called twice to find an optimal allocation on $S_j$ and an optimal allocation on $S_k$ for application $\mathcal{T}$. Since the split is a selfish decision, the splitting occurs even if only one of the members of coalition $S_j$ or $S_k$ can improve its individual value. As a result, the coalition with the higher individual payoff is the decision maker for the split.

If one or more coalitions split, then the merging process starts again. To do so, the *stop* flag is set to false (line 35). Multiple successive merge-and-split operations are repeated until the mechanism terminates. That means that there are no choices for merge or split for all existing coalitions in $\mathcal{CS}$. Let's consider $\mathcal{CS}_{final}$ as the final coalition structure. The mechanism selects one of the coalitions in the $\mathcal{CS}_{final}$ that yields the highest individual payoff for its members (line 41). The selected coalition will execute the program $\mathcal{T}$.

### 3.3 MSVOF Properties

We now investigate the properties of MSVOF. We will first show that MSVOF always terminates and produces stable VOs, and then investigate its complexity.

*Theorem 1:* Every partition of $\mathcal{G}$ determined by MSVOF is $\mathbb{D}_P$-stable.

We provide the proof of this theorem in Appendix B of the supplemental material.

Next, we investigate the complexity of MSVOF. The time complexity of the mechanism is determined by the number of attempts for merge and split. Each such attempt involves solving the MIN-COST-ASSIGN problem using the B&B-MIN-COST-ASSIGN, which requires exponential time in the number of tasks. As a result, the complexity of the mechanism is given by the number of merge and split attempts multiplied by the complexity of B&B-MIN-COST-ASSIGN. We now analyze the complexity of a single iteration of the main MSVOF loop (lines 3-40). In the worst case scenario, each coalition needs to make a merge attempt with all the other coalitions in $\mathcal{CS}$. At the beginning, all GSPs form singleton coalitions, thus there are $m$ coalitions in $\mathcal{CS}$. In the worst case, the first merge occurs after $\frac{m(m-1)}{2}$ attempts, the second requires $\frac{(m-1)(m-2)}{2}$ attempts and so on. The total worst case number of merge attempts is in $O(m^3)$. However, the merge process requires a significantly less number of attempts since once two coalitions are found for merge and the merge occurs, it does not always require to go through all the merge attempts. In the worst case scenario, splitting a coalition $S$ is $O(2^{|S|})$ which involves finding all the possible partitions of size two of the participating GSPs in that coalition. In practice, this split operation is restricted to the formed coalitions in $\mathcal{CS}$ and is not performed over all GSPs in $\mathcal{G}$. That means, the complexity of the split operation depends on the size of the coalitions in $\mathcal{CS}$ and not on the total number $m$ of GSPs. The coalitions in $\mathcal{CS}$ are small sets especially since we apply selfish split decisions that keep the size of the coalitions as small as possible. As a result, the split is reasonable in terms of complexity. In addition, once a coalition decides to split, the search for further splits is not needed. The worst case scenario occurs if the split of VO $S$ is not possible. This is due to checking all possible partitions into two sub-coalitions of that VO. To avoid this case, we check if at least one of the two sub-coalitions of size $|S-1|$, and respectively 1, is feasible. If none of them is feasible, we do not need to check any other partitions of $S$. As a result, in some cases that reduces the number of configurations to be considered for the split to $O(|S|)$.

The complexity of the MSVOF can be further reduced by limiting the size of the formed VOs. In Appendix C of the supplemental material we present a variant of the mechanism called $k$-MSVOF that restrict the size of VO's to $k$ GSPs, where $k$ is an integer.

## 4 Experimental Results

We perform a set of simulation experiments in order to investigate how effective the MSVOF mechanism is in determining stable VOs.

### 4.1 Experimental Setup

For our experiments we consider 16 GSPs which is a reasonable estimation of the number of GSPs in real grids. The number of GSPs is small since each GSP is a provider and not a single machine. We use real workloads from the Parallel Workloads Archive [7], [20] to drive our simulation experiments. More specifically, we use the log from the Atlas cluster at Lawrence Livermore National Laboratory (LLNL). This log consists of recently collected traces (from November 2006 to Jun 2007) that

TABLE 3: Simulation Parameters

| Param. | Description | Value(s) |
|---|---|---|
| $m$ | Number of GSPs | 16 |
| $n$ | Number of tasks | $[8, 8832]$ |
| $s$ | GSP's speeds ($m \times 1$ vector) | 4.91 $\times[16, 128]$ GFLOPS |
| $w$ | Tasks' workload ($n \times 1$ vector) | $[17676, 1682922.14]$ GFLOP |
| $t$ | Execution time matrix ($m \times n$) | $\frac{w}{s}$ seconds |
| $c$ | Cost matrix ($m \times n$) | $[1, \phi_b \times \phi_r]$ |
| $d$ | Deadline | $[0.3, 2.0]$ $\times$ Runtime $\times$ $n/1000$ seconds |
| $P$ | Payment | $[0.2, 0.4] \times max_c \times n$ units |
| $\phi_b$ | Maximum baseline value | 100 |
| $\phi_r$ | Maximum row multiplier | 10 |
| Runtime | Runtime of a job from log | $\geq 7200$ seconds |
| $max_c$ | Maximum amount of cost | $\phi_b \times \phi_r$ |

contain a good range of job sizes, from 8 to 8832. We used the cleaned log LLNL-Atlas-2006-2.1-cln.swf which has 43,778 jobs. We selected 21,915 jobs that completed successfully out of all the jobs in the log. About 13% of the total completed jobs are large jobs having runtimes greater than 7200 seconds.

The Atlas cluster [21] contains 1152 nodes, each with 8 processors for a total of 9,216 processors. Each processor is an AMD dual-core Opteron with a clock speed of 2.4 GHz. The theoretical system peak performance of the Atlas cluster is 44.24 TFLOPS (Tera FLoating-point OPerations per Second). As a result, the peak performance of each processor is 4.91 GFLOPS.

We selected six different sizes (i.e., number of tasks) of the application program from the Atlas log, ranging from 256 to 8192 tasks. For each program, the number of allocated processors the job uses gives the number of tasks, while the average CPU time used gives the average runtime of a task. We used the peak performance of a processor to convert the runtime to workload for each task. We generated the values of the other parameters based on the extracted data from the Atlas log. The parameters and their values are listed in Table 3. The values for deadline and payment were generated in such a way that there exists a feasible solution in each experiment.

Each task has a workload expressed in Giga Floating-point Operations (GFLOP). To generate a workload, we extract the runtime of a job (in seconds) from the logs, and multiply that by the performance (GFLOPS) of a processor in the Atlas system. This number gives the maximum amount of giga floating-point operations for a task. We assume that the workload of each task is within $[0.5, 1.0]$ of the maximum GFLOP of the job. The workload vector, $w$, contains the workload of each task of the application program.

The speed vector $s$ is generated relative to the Atlas system. Each GSP has a speed chosen within the range $4.91 \times [16, 128]$ GFLOPS. This is due to the fact that each GSP can have several processors capable of performing $4.91$ GFLOPS. The reason that we chose this range is that the number of processors of the Atlas cluster is 9,216. If all $16$ GSPs have the highest performance of $128 \times 4.91$, we would have 2048 processors that is 22.2 percent of the power of the Atlas system. As a result the deadline is generated at most 16 times larger than the runtime to make sure there is a feasible solution for the task allocation.

Based on the speed vector and the workload vector, the execution time of each task $T_j$ on each GSP $G_i$ is obtained. The execution time matrix is consistent if GSP $G_i$ that executes any task $T_j$ faster than GSP $G_k$, executes all tasks faster than GSP $G_k$ [22]. The generated time matrix is consistent due to the fact that for every task $T_j \in \mathcal{T}$, $w(T_j)$ is fixed for all GSPs $G_i \in \mathcal{G}$, thus, for any task $T_j$ if $t(T_j, G_i) < t(T_j, G_k)$ is true, then we have $s(G_i) > s(G_k)$ which means $G_i$ is faster than $G_k$. As a result, $t(T_q, G_k) > t(T_q, G_i)$ is satisfied for all tasks $T_q \in \mathcal{T}$.

Each cost matrix $c$ is generated using the method described by Braun et al. [22]. First, a baseline vector of size $n$ is generated where each element is a random uniform number within $[1, \phi_b]$. Then, the rows of the cost matrix are generated based on the baseline vector. Each element $j$ in row $i$ of the matrix, $c(i, j)$, is obtained by multiplying the element $i$ of the baseline vector with a uniform random number within $[1, \phi_r]$, a row multiplier. Therefore, one row requires $m$ different row multipliers. As a result, each element in the cost matrix is within the range $[1, \phi_b \times \phi_r]$.

We consider that the costs of GSPs are unrelated to each other, i.e., if $s(G_i) > s(G_k)$, for any task $T_j$, either $c(T_j, G_i) \leq c(T_j, G_k)$ or $c(T_j, G_k) \leq c(T_j, G_i)$ is true. This is due to GSPs policies. However, we consider that the costs are related to the workload of the tasks, i.e., for two tasks $T_j$ and $T_q$ where $w(T_j) > w(T_q)$, we have $c(T_j, G_i) > c(T_q, G_i)$ for all $G_i \in \mathcal{G}$. A task with the smallest workload has the cheapest cost on all GSPs.

We use the CPLEX branch-and-bound solver provided by IBM ILOG CPLEX Optimization Studio for Academics Initiative [23] for solving the MIN-COST-ASSIGN problem. In all experiments we use the default configuration of CPLEX for generating the bounds for the branch-and-bound solver.

### 4.2 Analysis of Results

We compare the performance of our VO formation mechanism, MSVOF, with that of three other mechanisms: Grand Coalition VO Formation (GVOF), Random VO Formation (RVOF), and Same-Size VO Formation (SSVOF). The GVOF mechanism maps the application program on all GSPs, that is, the grand coalition forms as a VO to perform the program. The RVOF mechanism maps all tasks to a random size VO, where GSPs are randomly selected to be part of that VO. The SSVOF mechanism maps all tasks to a VO with the same size as the VO formed by MSVOF. However, in this case, GSPs are selected randomly to be part of the coalition. All the mechanisms use the branch-and-bound method for solving MIN-COST-ASSIGN and finding the mapping of

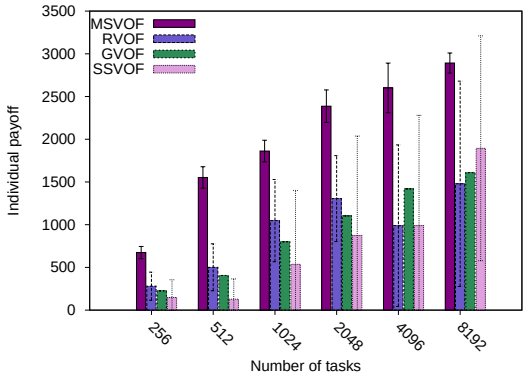


Fig. 1: GSPs' individual payoff

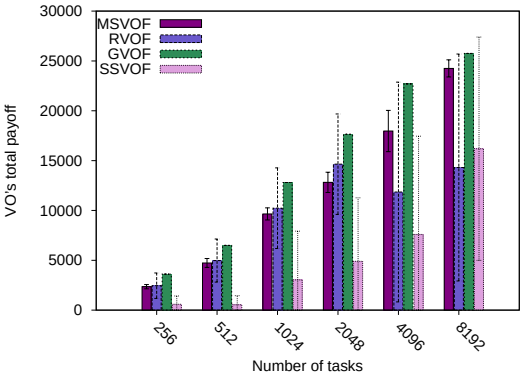


Fig. 3: Total payoff of the final VO

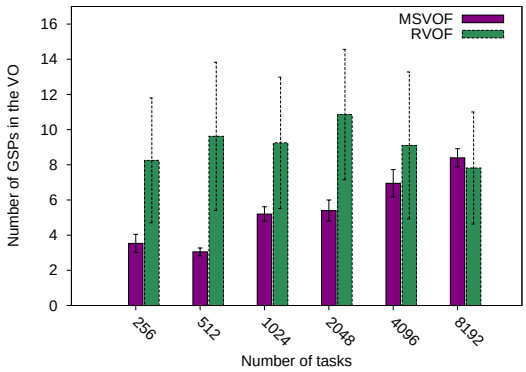


Fig. 2: Size of final VO

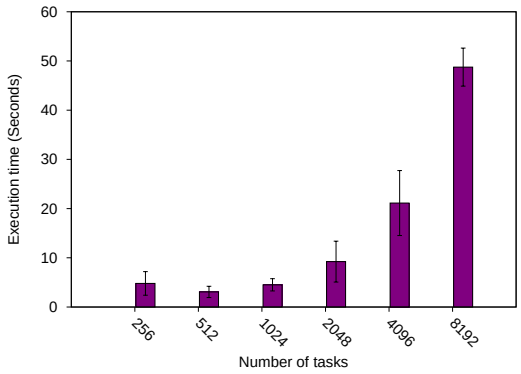


Fig. 4: MSVOF's execution time

the tasks to GSPs in a VO. Using the same mapping algorithm for all four mechanisms allows us to focus on the VO formation and not on the choice of the mapping algorithms. We performed a series of ten experiments in each case, and we represented the average of the obtained results.

In Fig. 1, we show the performance of MSVOF in terms of the individual GSP's payoffs in the final VO as a function of the number of tasks. The figure shows that the MSVOF provides the highest individual payoff for GSPs in the final VO among all four mechanisms. The significant differences between the MSVOF and the SSVOF shows the importance of decisions to merge and split to form the best VO. Both of these mechanisms form VOs having the same size, but our proposed mechanism selects the GSPs based on the merge-and-split rules. In SSVOF, GSPs are selected randomly. Standard deviation is very high in SSVOF and RVOF since in many cases the formed VO is unable to execute the program and the participating GSPs receive zero. The reason that GVOF does not have the error bars is that we ran the experiments for the same number of tasks and GVOF always forms the grand coalition, and thus, the individual profit is the same for all experiments. On average, the individual GSP's payoff in the case of MSVOF is 2.13, 2.15 and 1.9 times greater than that obtained in the case of RVOF, GVOF and SSVOF, respectively.

In Fig. 2, we compare the size of the final VO obtained by MSVOF and RVOF. We do not represent the sizes of the VOs obtained by SSVOF and GVOF since they are fixed to the size determined by MSVOF and the maximum size, respectively. This figure shows that as the number of tasks increases the size of the VO obtained by MSVOF increases. It means that the more tasks, the more GSPs pool their resources to form a VO in order to execute the program.

In Fig. 3, we compare the total payoff obtained by MSVOF with that obtained by the other three mechanisms. In all cases GVOF provides the highest total payoff. These results show that GSPs prefer smaller VOs (shown in Fig. 2) in order to obtain higher individual profits. The global welfare does not have an impact on the formation of VOs. As a result, the VO resulting from MSVOF may not provide the highest total payoff but it will provide the highest individual payoff. For example, for 256 tasks, MSVOF obtains a total payoff of 2351.6 for the final VO, while RVOF, GVOF and SSVOF obtain a total payoff of 2456, 3610 and 591.6 for their final VOs of average size 8.25, 16, and 3.53, respectively.

In Appendix D of the supplemental material, we show the average total number of merge and split operations performed.

From the above results, we conclude that the proposed VO formation mechanism is able to form stable VOs that

ensure the program is completed before its deadline and provide the highest individual payoff for the GSPs.

Fig. 4 shows the execution time of MSVOF. These results were obtained when running MSVOF on a 3.00GHz Intel quad-core PC with 8GB of memory. The MSVOF's execution time is reasonable given that the application program would require several hours to execute. The reason for getting higher execution times for 4096 and 8192 tasks is that the VOs explored by the mechanism are larger in size. As a result, the split operation takes more time to test the possible cases. The execution times of the other mechanisms are negligible compared to that of MSVOF, and thus, we chose not to present them in the figure.

We investigate the performance of the $k$-MSVOF mechanism (the variant of MSVOF that restricts the size of the VO to $k$) in Appendix E of the supplemental material.

## 5 Conclusion

We proposed a novel mechanism for VO formation in grids. In the proposed mechanism, GSPs cooperate to form VOs in order to execute application programs. We modeled the problem as a coalitional game and derived a centralized VO formation mechanism based on merge-and-split operations. To find the optimal mapping of the tasks on the participating GSPs in a VO, we used a branch-and-bound method. We showed that our proposed mechanism produces stable VOs. We performed extensive experiments with data extracted from real workload traces to investigate its properties. Experimental results showed that the VO obtained by MSVOF maximizes the individual payoffs of the participating GSPs. In addition, most of the time MSVOF determines the final VO with the smallest number of participating GSPs. The mechanism's execution time is reasonable given that application programs would require several hours to execute. We believe that this research will encourage grid service providers to adopt VO formation mechanisms for allocating their resources in order to execute application programs.

In future work, we would like to incorporate the trust relationships among GSPs in our VO formation model and design new mechanisms for VO formation that take them into account. In addition, we would like to extend this research to cloud federation formation, where cloud providers cooperate in order to provide the resources requested by users.

## Acknowledgments

This paper is a revised and extended version of [24] presented at the 30th IEEE International Performance Computing and Communications Conference (IPCCC 2011). The authors wish to express their thanks to the editor and the anonymous referees for their helpful and constructive suggestions, which considerably improved the quality of the paper. This research was supported, in part, by NSF grants DGE-0654014 and CNS-1116787.

## References

[1] K. Czajkowski, I. Foster, C. Kesselman, V. Sander, and S. Tuecke, "SNAP: A protocol for negotiating service level agreements and coordinating resource management in distributed systems," in *Proc. of the 8th Workshop on Job Scheduling Strategies for Parallel Processing*, 2002, pp. 153–183.

[2] J. Frey, T. Tannenbaum, I. Foster, M. Livny, and S. Tuecke, "Condor-g: A computation management agent for multi-institutional grids," *Cluster Computing*, vol. 5, no. 3, pp. 237–246, 2002.

[3] A. S. Grimshaw and W. A. Wulf, "The legion vision of a worldwide virtual computer," *Commun. ACM*, vol. 40, no. 1, January 1997.

[4] I. Foster and C. Kesselman, *The grid: blueprint for a new computing infrastructure*. Morgan Kaufmann, 2004.

[5] G. DeSanctis and P. Monge, "Communication processes for virtual organizations," *Organization Science*, vol. 10, no. 6, pp. 693–703, 1999.

[6] M. J. Osborne and A. Rubinstein, *A Course in Game Theory*. Cambridge, MA, USA: MIT Press, 1994.

[7] Parallel workloads archive. [Online]. Available: http://www.cs.huji.ac.il/labs/parallel/workload/

[8] W. Cirne, D. Paranhos, L. Costa, E. Santos-Neto, F. Brasileiro, J. Sauvé, F. Silva, C. Barros, and C. Silveira, "Running bag-of-tasks applications on computational grids: The mygrid approach," in *Proc. of the Intl. Conf. on Parallel Processing*, 2003, pp. 407–416.

[9] C. Weng and X. Lu, "Heuristic scheduling for bag-of-tasks applications in combination with QoS in the computational grid," *Future Generation Computer Systems*, vol. 21, no. 2, pp. 271–280, 2005.

[10] M. Pinedo, *Scheduling: Theory, Algorithms, and Systems*. Upper Saddle River, NJ: Prentice Hall, 2002.

[11] G. Owen, *Game Theory*, 3rd ed. New York, NY, USA: Academic Press, 1995.

[12] L. Shapley, "A Value for n-person Games," in *Contributions to the Theory of Games*, H. Kuhn and A. Tucker, Eds. Princeton University Press, 1953, vol. II, pp. 307–317.

[13] O. Shehory and S. Kraus, "Task allocation via coalition formation among autonomous agents," in *Proc. of Intl. Joint Conf. on Artificial Intelligence*, vol. 14, 1995, pp. 655–661.

[14] K. Apt and A. Witzel, "A generic approach to coalition formation," *Intl. Game Theory Review*, vol. 11, no. 3, pp. 347–367, 2009.

[15] T. Sandholm, K. Larson, M. Andersson, O. Shehory, and F. Tohme, "Coalition structure generation with worst case guarantees," *Artificial Intelligence*, vol. 111, pp. 209–238, 1999.

[16] D. Knuth, *The Art of Computer Programming, Volume 4, Fascicle 3: Generating All Combinations and Partitions*. Addison-Wesley Professional, 2005.

[17] E. Lawler and D. Wood, "Branch-and-bound methods: A survey," *Operations research*, pp. 699–719, 1966.

[18] L. Wolsey, "Integer programming," *IIE Transactions*, vol. 32, pp. 273–285, 2000.

[19] ——, *Integer programming*, ser. Wiley-Interscience series in discrete mathematics and optimization. Wiley, 1998.

[20] S. Chapin, W. Cirne, D. Feitelson, J. Jones, S. Leutenegger, U. Schwiegelshohn, W. Smith, and D. Talby, "Benchmarks and standards for the evaluation of parallel job schedulers," in *Proc. of 5th Workshop on Job Scheduling Strategies for Parallel Processing*, 1999, pp. 67–90.

[21] Atlas. [Online]. Available: https://www.llnl.gov/news/newsreleases/2007/NR-07-04-05.html

[22] T. Braun, H. Siegel, N. Beck, L. Boloni, M. Maheswaran, A. Reuther, J. Robertson, M. Theys, B. Yao, D. Hensgen *et al.*, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," *J. of Parallel and Distributed Computing*, vol. 61, no. 6, pp. 810–837, 2001.

[23] IBM ILOG CPLEX optimization studio for academics initiative. [Online]. Available: http://www01.ibm.com/software/websphere/products/optimization/academic-initiative/

[24] L. Mashayekhy and D. Grosu, "A merge-and-split mechanism for dynamic virtual organization formation in grids," in *Proc. 30th IEEE Intl. Performance Computing and Communications Conf.*, 2011.

**Lena Mashayekhy** received her B.Sc. degree in Computer Engineering (Software) from Iran University of Science and Technology (IUST), Tehran, Iran and her M.Sc. degree from the University of Isfahan, Isfahan, Iran. She is currently a PhD candidate in the Department of Computer Science at Wayne State University, Detroit, Michigan. Her research interests include distributed systems, parallel computing, game theory and mechanism design. She is a student member of the IEEE.

**Daniel Grosu** received the Diploma in engineering (automatic control and industrial informatics) from the Technical University of Iaşi, Romania, in 1994 and the MSc and PhD degrees in computer science from the University of Texas at San Antonio in 2002 and 2003, respectively. Currently, he is an associate professor in the Department of Computer Science, Wayne State University, Detroit. His research interests include distributed systems and algorithms, resource allocation, computer security and topics at the border of computer science, game theory and economics. He has published more than seventy peer-reviewed papers in the above areas. He has served on the program and steering committees of several international meetings in parallel and distributed computing. He is a senior member of the ACM, the IEEE, and the IEEE Computer Society.