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

Lena Mashayekhy
Department of Computer Science
Wayne State University
Detroit, MI 48202, USA
Email: mlena@wayne.edu

Daniel Grosu
Department of Computer Science
Wayne State University
Detroit, MI 48202, USA
Email: dgrosu@wayne.edu

***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 simulations 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.**

## I. 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], [4] do not explicitly address the formation and management of Virtual Organizations (VO) [5]. 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.

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 objective 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. 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 [5]. 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. The model that we consider consists of a set of GSPs and a grid user that submits a program and a specification 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.

*Related Work.* Resource management in open distributed computing systems that span multiple administrative domains and organizations has been studied extensively. Several mechanisms and systems for resource management have been developed, examples include GRAM [1], SNAP [2], Condor-G [3], AppLeS [6], Nimrod/G [7], and Legion [4]. These mechanisms and systems do not explicitly take into account the autonomy of the resource owners and their incentives to contribute resources. Several economic-based models and systems for resource management in open distributed systems that address this issue have been proposed in the literature [8], [9], [10]. They do not explicitly address the problem of VO formation and management which is one of the key issues that need to be solved in large scale computing systems in order to facilitate collaboration among participating organizations. The requirements for establishing and managing dynamic VOs in grids are discussed in [11]. One of the most important requirements identified by the authors is the interoperability of the environment. This is usually satisfied in the current grids through common protocols for establishing and managing sharing relationships. Globus Toolkit [12] supports the operation and management of VOs by providing basic middleware for VO policy specification and enforcement, resource management, provisioning, and discovery. The toolkit does not provide mechanisms for VO formation and tools for VO management

and analysis. Dynamic VO formation among autonomous agents and the management of VOs are examined in [13]. The VO formation problem can be viewed as a coalition formation problem. Research on coalition formation has been conducted in the multi-agent systems community for problems such as allocating a task [14] and service composition [15].

To the best of our knowledge, the closest work to ours is presented in [16] which describes the mechanisms for VO formation used in the CONOISE-G project. The mechanisms used in CONOISE-G are based on Constraint Satisfaction Programming (CSP) techniques [17] while our proposed mechanism is based on coalitional game models and techniques. We believe that coalitional game theory is a powerful tool for modeling VO formation among autonomous organizations providing more efficient and scalable mechanisms compared to CSP. CSP techniques do not facilitate the stability and robustness analysis of the VO formation process, while this is intrinsic and represents one of the strengths of coalitional game theory. Furthermore, the CONOISE-G project does not address the issues of scheduling the applications within the VO, while our proposed framework addresses this issue and provides a mechanism for application scheduling within VOs. In our previous work [18], we developed a VO formation framework based on extensive form games. The approach we use here is based on coalitional games and merge-and-split operations which, unlike the one used in [18], guarantees the stability of the VOs formed by the proposed mechanism and is more computationally efficient.

*Our Contribution.* We address the problem of VO formation in grids by designing a mechanism that allows the GSPs to make their own decision 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 provides 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. We analyze the properties of our proposed VO formation mechanism and perform extensive simulation experiments to investigate its properties.

*Organization.* This paper is organized as follows. In Section II, we describe the VO formation problem and the system model we consider. In Section III, we describe the game theoretic framework used to design the proposed VO formation mechanism. Then, we present the proposed mechanism and characterize its properties. In Section IV, we evaluate the mechanism by extensive simulation experiments. In Section V, we summarize our results and present possible directions for future research.

## II. 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 by a given deadline $d$. Application programs consisting of several independent tasks are representative for a wide range of problems in science and engineering [19], [20], [21]. 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 instructions required by 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 instructions 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 $t(T, G) = \frac{w(T)}{s(G)}$. 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 the cost function, $c : \mathcal{T} \times \mathcal{G} \rightarrow \mathbb{R}^+$. 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* [22] is defined by the pair $(N, v)$, where $N$ is the set of players and $v$ is a real-valued function called the *characteristic function*, defined on $S \subseteq N$ such that $v : S \rightarrow \mathbb{R}^+$ and $v(\emptyset) = 0$. In our model, the players are the GSPs (i.e., $N = \mathcal{G}$) 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 $G \in S$. The costs incurred for executing the program $\mathcal{T}$ on $S$ under mapping $\pi_S$ is given by

$$C(\mathcal{T}, S) = \sum_{T \in \mathcal{T}} \sum_{G \in S} \sigma_S(T, G) c(T, G), \qquad (1)$$

where

$$\sigma_S(T,G) = \begin{cases} 1 & \text{if } \pi_S(T) = G, \\ 0 & \text{if } \pi_S(T) \neq G. \end{cases} \quad (2)$$

The execution time of the program is given by its *makespan* (*i.e.,* completion time) as induced by the mapping $\pi_S$. The execution time is given by

$$E(\mathcal{T},S) = \max_{G \in S} \sum_{T \in \mathcal{T}} \sigma_S(T,G) t(T,G). \quad (3)$$

where

$$t(T,G) = \frac{w(T)}{s(G)} \quad (4)$$

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

$$v(S) = \begin{cases} 0 & \text{if } |S| = 0 \text{ or } E(\mathcal{T},S) > d, \\ P - C(\mathcal{T},S) & \text{if } |S| > 0 \text{ and } E(\mathcal{T},S) \leq d, \end{cases} \quad (5)$$

where $|S|$ is the cardinality of $S$. Note that $v(S)$ satisfies the constraint $v(\emptyset) = 0$.

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* [23] 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 an allocation rule in other systems where tractability is critical (*e.g.,* [14]). 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 $S$ by solving:

$$\max_{(S)} \frac{P - C(\mathcal{T},S)}{|S|} \quad (6)$$

subject to: (i) $E(\mathcal{T},S) \leq d$, and (ii) $G \in S$. The first constraint specifies that the program is completed before the deadline, while the second constraint says that $G$ is a member of the coalition.

The *payoff* or the *share* of GSP $G$ part of coalition $S$, denoted by $x_G(S)$ is given by

$$x_G(S) = \frac{v(S)}{|S|}. \quad (7)$$

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 allocates the payoff among the players in some fair way. The primary concern for any coalition game is the stability. 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. 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. 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 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 grand coalition does not form, independent and disjoint coalitions would form.

TABLE I: 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 |

TABLE II: 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 |

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 instructions, respectively. In Table I, 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 MIPS. 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 $P = 10$. The mapping and $v(S)$ are given in Table II. 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 $G_3$ can not 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 participate to perform a task, it should receive zero as a payoff. If there are some GSPs that do not participate in performing any task of the program, they should not be considered as members of the VO.

## III. 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.

### A. Coalition Formation Framework

*Coalition formation* [24] is the partitioning of the players into disjoint sets. A coalition structure $\mathcal{CS} = \{S_1, S_2, \ldots, S_h\}$ forms a partition 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 $\bigcup_{S_i \in \mathcal{CS}} S_i = N$. 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 1 (Collection):* A *collection* in the grand coalition $N$, is defined as the set $\mathcal{C} = \{S_1, \cdots, S_k\}$ of mutually disjoint coalitions. If $\cup_{j=1}^{k} S_j = N$, the collection $\mathcal{C}$ is called a *partition* of $N$.

*Definition 2 (Comparison):* A *collection comparison* $\triangleright$ is defined to compare two collections $A$ and $B$ that are partitions of the same subset $S \subseteq N$. $A \triangleright B$ 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:

$$\begin{aligned} \hat{S} \triangleright_m \{S_1, \cdots, S_k\} \iff & \{\forall j \leq k, \forall G_i \in \hat{S} \cap S_j; \\ & x_i(\hat{S}) \geq x_i(S_j) \text{ and } \exists j \leq k, \\ & \exists G_r \in S_j \,; x_r(\hat{S}) > x_r(S_j)\} \end{aligned} \tag{8}$$

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

Equation (8) implies that coalition $\hat{S}$ 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. Equation (9) 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 merge and split rules as follows [24]:

> *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 by the individual order is an agreement among the GSPs to operate together if it is beneficial for them.

As we mentioned before, one of the formed coalitions, final coalition, performs the program, thus, the formation of the rest of the coalitions is not important. The reason for that is the rest of the GSPs which are not in the final coalition can participate again in another coalition formation process for performing another assigned application. Therefore, a coalition decides to split only if there is at least one subcoalition that strictly improves the individual payoffs of its constituent GSPs. Under the split rule, the individual payoffs of the other subcoalitions 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 (8) 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 (6) while satisfying the deadline constraint. As a result, the merge occurs if the following two inequalities are satisfied.

$$\frac{P - C(\mathcal{T}, S_i \cup S_j)}{|S_i \cup S_j|} \geq \frac{P - C(\mathcal{T}, S_i)}{|S_i|} \tag{10}$$

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

Since $|S_i \cup S_j| > |S_i|$ and $|S_i \cup S_j| > |S_j|$, in order to 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 (9) 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.

$$\frac{P - C(\mathcal{T}, \hat{S})}{|\hat{S}|} \leq \frac{P - C(\mathcal{T}, S_i)}{|S_i|} \quad (12)$$

$$\frac{P - C(\mathcal{T}, \hat{S})}{|\hat{S}|} \leq \frac{P - C(\mathcal{T}, S_j)}{|S_j|} \quad (13)$$

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 or equal to its individual payoff in $\hat{S}$.

The stability of the resulting coalition structure is characterized using the concept of defection function $\mathbb{D}$ [24].

*Definition 3 (Defection function):* A *defection function* $\mathbb{D}$ is a function which associates with each partition $\mathcal{P}$ of $N$ a group of collections in $N$. 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 $\mathbb{D}_P$ is discussed in [24] which allows formation of all partitions of the grand coalition. $\mathbb{D}_P$-stability is defined based on this function. $\mathbb{D}_P$ allows any group of players to leave the partition $\mathcal{P}$ of $N$ through merge-and-split rules to create another partition in $N$. Therefore, $\mathbb{D}_P$-stability means no coalition has an incentive to merge or split.

In order to clarify the above concepts, let us consider the example in section II 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 II, $\{G_2, G_3\} \rhd_m \{\{G_2\}, \{G_3\}\}$, 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 $\{G_1, G_2, G_3\} \rhd_m \{\{G_1\}, \{G_2, G_3\}\}$, 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 subcoalition that can split is $\{G_1, G_2\}$ since $\{\{G_1, G_2\}, \{G_3\}\} \rhd_s \{G_1, G_2, G_3\}$, i.e., $G_1$ and $G_2$ improve their payoff by splitting. None of $G_1$ and $G_2$ wants to split from the coalition $\{G_1, G_2\}$. There are no coalitions to be able to merge or split any further. Even if GSPs try different order to merge, 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.

### *B. Merge-and-Split VO Formation Mechanism (MSVOF)*

The proposed VO formation mechanism is given in Algorithm 1. The mechanism is executed by a trusted party that also facilitates the communication among VOs/GSPs. MSVOF starts with a coalition structure $\mathcal{CS}$ consisting of every singleton $G_i \in N$ as a coalition $S_i$ in $\mathcal{CS}$. MSVOF executes the mapping algorithm for each coalition $S_i \in \mathcal{CS}$ to find an allocation for the application program $\mathcal{T}$ on GSPs in $S_i$. The mapping algorithm solves a variant of the general assignment problem (GAP) that can be viewed as the problem of scheduling parallel machines with cost. In this paper, we use the GAP algorithm described in [25]. This algorithm solves the LP relaxation of the problem and performs rounding by building a bipartite graph based on the relaxed solution. The integer solution is obtained by finding an integer matching of the bipartite graph. Other mapping algorithms can also be used by the coalitions to decide on the mapping.

When MSVOF starts, all tasks are considered to be assigned to each of the GSPs. Then, $v(S_i)$ is computed based on the allocation. MSVOF uses a matrix $visited$ for checking if all pairs of coalitions in $\mathcal{CS}$ are visited for merge or not. By using this matrix, all possible combinations of two coalitions in $\mathcal{CS}$ are visited during the merge step. The merge process starts every time by choosing two non-visited coalitions in $\mathcal{CS}$ randomly, e.g., $S_i$ and $S_j$. The mapping algorithm is called to find an allocation for the application program $\mathcal{T}$ on $S_i \cup S_j$. If

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

$\mathcal{CS} = \{\{G_1\}, \cdots, \{G_m\}\}$
Map program $\mathcal{T}$ on each $S_i \in \mathcal{CS}$
**repeat**
  $stop \leftarrow True$
  **for all** $S_i, S_j \in \mathcal{CS}, i \neq j$ **do**
    $visited[S_i][S_j] \leftarrow False$
  **end for**
  {Merge process starts:}
  **repeat**
    $flag \leftarrow True$
    Randomly select $S_i, S_j \in \mathcal{CS}$ for which $visited[S_i][S_j] = False, i \neq j$
    $visited[S_i][S_j] \leftarrow True$
    Map program $\mathcal{T}$ on $S_i \cup S_j$ using GAP
    **if** $S_i \cup S_j \rhd_m \{S_i, S_j\}$ **then**
      $S_i \leftarrow S_i \cup S_j$ {merge $S_i$ and $S_j$}
      $S_j \leftarrow \emptyset$ {$S_j$ is removed from $\mathcal{CS}$}
      **for all** $S_k \in \mathcal{CS}, k \neq i$ **do**
        $visited[S_i][S_k] \leftarrow False$
      **end for**
    **end if**
    **for all** $S_i, S_j \in \mathcal{CS}, i \neq j$ **do**
      **if** not $visited[S_i][S_j]$ **then**
        $flag \leftarrow False$
      **end if**
    **end for**
  **until** ($|\mathcal{CS}| = 1$) or ($flag = True$)
  {Split process starts:}
  **for all** $S_i \in \mathcal{CS}$ where $|S_i| > 1$ **do**
    **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**
      Map program $\mathcal{T}$ on $S_j$ using GAP
      Map program $\mathcal{T}$ on $S_k$ using GAP
      **if** $\{S_j, S_k\} \rhd_s S_i$ **then**
        $S_i \leftarrow S_j$ {that is $\mathcal{CS} = \mathcal{CS} \setminus S_i$}
        $\mathcal{CS} = \mathcal{CS} \bigcup S_k$
        $stop \leftarrow False$
        Break (one split occurs; no need to check other splits)
      **end if**
    **end for**
  **end for**
**until** $stop = True$
Find $k = \arg\max_{S_i \in \mathcal{CS}} \{v(S_i)/|S_i|\}$
Map and execute program $\mathcal{T}$ on VO $S_k$

$S_i \cup S_j \triangleright_m \{S_i, S_j\}$, then coalitions $S_i$ and $S_j$ decide to merge. Therefore, all the members receive higher profit by merging due to using of the equal sharing of profit. $S_i \cup S_j$ is saved in $S_i$, and $S_j$ is deleted from $\mathcal{CS}$. Since $S_i$ is changed, it can be selected in 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 for merge. If all the coalitions are tested and a merge does not occur, or the grand coalition forms, the merge process ends.

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 for splitting. The mechanism tries to split $S_i$ that has more than one member in two disjoint coalitions $S_j$ and $S_k$ where $S_j \cup S_k = S_i$. The mapping algorithm is called twice to find an allocation on $S_j$ and an allocation on $S_k$ for application $\mathcal{T}$. Since the split is a selfish decision, the splitting occurs even if 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. Multiple successive merge-and-split operations are repeated until the mechanism terminates. That means 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 has the highest individual value for its members. The selected coalition will perform the program $\mathcal{T}$.

*Theorem 1:* Every partition resulting from our proposed VO formation is $\mathbb{D}_P$-stable.

*Proof:* (Sketch) Since the merge and split operations are based on the comparison relations $\triangleright_m$ and $\triangleright_s$, the resulting partition after each merge or split is more preferred than a previous partition. As a result, there is no cycle of partitions in any sequence of merge and split operations. Since the number of different partitions is finite, the merge-and-split iterations terminate. The final partition cannot be subject to any further merge or split. As a result, the final partition is $\mathbb{D}_P$-stable. ■

The time complexity of the mechanism is determined by the number of attempts for merge and split. In the worst case scenario, each coalition needs to make a merge attempt with all the other coalitions in $\mathcal{CS}$. The total worst case number of merge attempts for all coalitions is in $O(m^2)$. However, the merge process requires a significantly less number of attempts since once a coalition is 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 in $O(2^{|S|})$ 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 specially 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.

TABLE III: Simulation Parameters

| Param. | Description | Value(s) |
|---|---|---|
| $m$ | Number of GSPs | 16 |
| $n$ | Number of tasks | [50, 225] |
| $s$ | GSP's speeds ($m \times 1$ vector) | 159 $\times[1, 2m]$ GIPS |
| $w$ | Tasks' workload ($n \times 1$ vector) | [20,000, 320,000] GI |
| $t$ | Execution time matrix ($m \times n$) | $\frac{w}{s}$ seconds |
| $c$ | Cost matrix ($m \times n$) | $[1, \phi_b \times \phi_r]$ |
| $d$ | Deadline | 10,000 seconds |
| $P$ | Payment | 20,000 units |
| $\phi_b$ | Maximum baseline value | 100 |
| $\phi_r$ | Maximum row multiplier | 10 |

## IV. Experimental Results

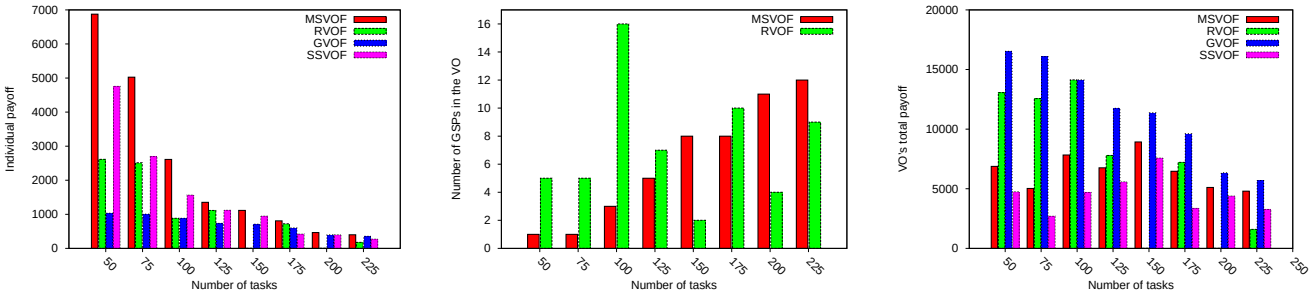
We perform a set of simulation experiments which allows us to investigate how effective the MSVOF mechanism is in determining the stable VOs.

### A. Simulation Parameters

We consider 16 GSPs. Since each GSP is a provider not an abstract machine, considering this number of GSPs is reasonable. We performed runs for eight different application program sizes, ranging from 50 to 225 tasks. The simulation parameters and their values are listed in Table III. The deadline and payment are kept the same in all the experiments. The values for deadline and payment are large enough to make sure that there is a feasible solution in each experiment.

The speed vector $s$ is generated relative to the fastest current Intel processor, Intel Core i7 Extreme Edition 990x which can perform 159 GIPS. Each GSP has a speed chosen within the range [159, $2\times m\times 159$] GIPS. The reason is that each GSP can have several nodes capable of performing 159 GIPS. Each task has a workload expressed in Giga Instructions, randomly selected from [20000, 320000]. The workload vector, $w$, contains the workload of each task of the application program. Based on the speed vector and the workload vector, the execution time of each task $T_j$ on each GSP $G_i$ is obtained using equation (4). The execution time matrix is consistent if a GSP $G_i$ that executes any task $T_j$ faster than GSP $G_k$, executes all tasks faster than GSP $G_k$ [26]. The generated time matrix is consistent due to the fact that $w(T_j)$ is fixed for $T_j \in \mathcal{T}$, 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 in [26]. 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 generated by the element $i$ of the baseline vector multiplied by a uniform random number within $[1, \phi_r]$, a row multiplier.




(a) (b) (c)

Fig. 1: (a) GSPs Individual Payoff; (b) Size of Final VO; (c) Total Payoff of the Final VO.

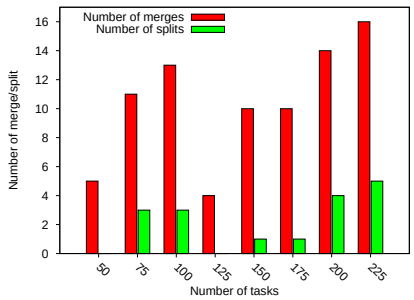



Fig. 2: Total Number of Merge and Split Operations

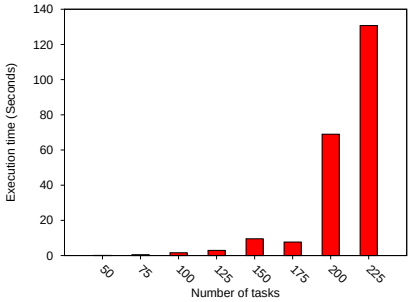



Fig. 3: MSVOF's Execution Time

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.

### *B. 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. In the RVOF mechanism all tasks are mapped to a random size VO where GSPs are randomly selected to be part of that VO. The SSVO 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 GAP algorithm to map the tasks to GSPs. This allows us to focus on the VO formation and not on the choice of the mapping algorithms.

In Fig. 1a, we show the performance of MSVOF, in terms of the individual GSP's payoff 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. Since the equal sharing method is used, in the GVOF mechanism where the number of GSPs in the VO is the highest, GSPs get the smallest payoff. The significant differences between the MSVOF and the SSVOF shows the importance of decisions to merge and split to form the best VO. As the number of tasks increases, the individual payoff decreases, this is due to the fact that we consider the same payment in all experiments. Thus, increasing the number of tasks increases the cost. For example for 75 tasks, MSVOF obtains a payoff of 5025 while RVOF, GVOF and SSVOF obtain 2512.2, 1005.12 and 2699, respectively. On average, the individual payoff of MSVOF is 2.32, 3.26 and 1.53 times better than RVOF, GVOF and SSVOF, respectively.

In Fig. 1b, we show the size of the final VO obtained by MSVOF. This figure shows that as the number of tasks increases the VO size increases. It means that the more tasks the more GSPs pool their resources to form a VO in order to execute the program. In this figure, we also show the size of the VO determined by RVOF.

In Fig. 1c, we compare the total payoff obtained by MSVOF and the other three mechanisms. These results show that GSPs prefer smaller VOs (shown in Fig. 1b) in order to obtain higher individual profits. As a result, the VO resulting from MSVOF may not provide the highest total payoff. For example, for 75 tasks, MSVOF obtains a total payoff of 5,025 for the final

VO, while RVOF, GVOF and SSVOF obtain a total payoff of 12,561, 16,082 and 2,699 for their final VOs of size 5, 16, and 1, respectively.

In Fig. 2, we show the total number of merge and split operations performed by MSVOF. For example for 75 tasks, 11 merge and 3 split operations are performed in order to form a stable coalition structure, but a VO with size one is selected among all coalitions in the coalition structure to perform the tasks because it provides the highest individual payoff.

Fig. 3 shows the execution time of MSVOF. 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 200 and 225 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.

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.

## V. Conclusion

Simulation 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 the smallest number of participating GSPs. The mechanism's execution time is reasonable given that applications 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.

## Acknowledgment

This work was partially supported by NSF grants DGE-0654014 and CNS-1116787.

## References

[1] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke, "A resource management architecture for metacomputing systems," in *Proc. of the Workshop on Job Scheduling Strategies for Parallel Processing*, 1998, pp. 62–82.

[2] 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.

[3] 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.

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

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

[6] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman, "Heuristics for scheduling parameter sweep applications in grid environments," in *Proc. of the 9th Heterogeneous Computing Workshop*, April 2000, pp. 349–363.

[7] R. Buyya, D. Abramson, and J. Giddy, "Nimrod/g: An architecture for a resource management and scheduling system in a global computational grid," in *Proc. of the 4th Intl. Conf. on High Performance Computing in Asia-Pacific Region*, May 2000.

[8] R. Buyya, D. Abramson, J. Giddy, and H. Stockinger, "Economic models for resource allocation and scheduling in grid computing," *Concurrency and Computation: Practice and Experience*, vol. 14, no. 13-15, pp. 1507–1542, 2002.

[9] L. Kang and D. Parkes, "A decentralized auction framework to promote efficient resource allocation in open computational grids," in *Proc. of the Joint Workshop on The Economics of Networked Systems and Incentive-Based Computing (NetEcon+IBC 2007)*, June 2007.

[10] R. Wolski, J. S. Plank, J. Brevik, and T. Bryan, "Analyzing market-based resource allocation strategies for the computational grid," *International Journal of High Performance Computing Applications*, vol. 15, no. 3, pp. 258–281, Aug. 2001.

[11] I. Foster, C. Kesselman, and S. Tuecke, "The anatomy of the grid: Enabling scalable virtual organizations," *Int. J. Supercomputer Applications*, vol. 15, no. 3, pp. 200–222, 2001.

[12] Globus, "`http://www.globus.org`."

[13] I. Foster, N. R. Jennings, and C. Kesselman, "Brain meets brawn: Why grid and agents need each other," in *Proc. of the 3rd Intl. Joint Conf. on Autonomous Agents and Multiagent Systems*, 2004, pp. 8–15.

[14] 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.

[15] I. Müller, R. Kowalczyk, and P. Braun, "Towards agent-based coalition formation for service composition," in *Proc. of the IEEE/WIC/ACM Intl. Conf. on Intelligent Agent Technology*, Dec. 2006, pp. 73–80.

[16] J. Patel, W. T. L. Teacy, N. R. Jennings, M. Luck, S. Chalmers, N. Oren, T. J. Norman, A. Preece, P. M. D. Gray, G. Shercliff, P. J. Stockreisser, J. Shao, W. A. Gray, N. J. Fiddian, and S. Thompson, "Agent-based virtual organisations for the grid," *Multiagent Grid Syst.*, vol. 1, no. 4, pp. 237–249, 2005.

[17] K. Apt, *Principles of Constraint Programming*. New York, USA: Cambridge University Press, 2003.

[18] T. E. Carroll and D. Grosu, "Formation of virtual organizations in grids: A game-theoretic approach," *Concurrency and Computation: Practice and Experience*, vol. 22, no. 14, pp. 1972–1989, 2010.

[19] 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.

[20] 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.

[21] F. da Silva, S. Carvalho, and E. Hruschka, "A scheduling algorithm for running bag-of-tasks data mining applications on the grid," in *Euro-Par 2004 Parallel Processing*. Springer, 2004, pp. 254–262.

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

[23] 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.

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

[25] D. Shmoys and É. Tardos, "An approximation algorithm for the generalized assignment problem," *Mathematical Programming*, vol. 62, no. 1, pp. 461–474, 1993.

[26] 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.