Non-cooperative, Semi-cooperative, and Cooperative Games-based Grid Resource Allocation

Samee Ullah Khan and Ishfaq Ahmad
Department of Computer Science and Engineering
University of Texas at Arlington, TX-76019, U.S.A.
{sakhan, iahmad}@cse.uta.edu

Abstract

In this paper we consider, compare and analyze three game theoretical Grid resource allocation mechanisms. Namely, 1) the non-cooperative sealed-bid method where tasks are auctioned off to the highest bidder, 2) the semi-cooperative n-round sealed-bid method in which each site delegate its work to others if it cannot perform the work itself, and 3) the cooperative method in which all of the sites deliberate with one another to execute all the tasks as efficiently as possible.

To experimentally evaluate the above mentioned techniques, we perform extensive simulation studies that effectively encapsulate the task and machine heterogeneity. The tasks are assumed to be independent and bear multiple execution time deadlines. The simulation model is built around a hierarchical Grid infrastructure where machines are abstracted into larger computing centers labeled "federations," each of which are responsible for managing their own resources independently. These federations are then linked together with a primary portal to which Grid tasks would be submitted. To measure the effectiveness of these game theoretical techniques, the recorded performance is evaluated against a conventional baseline method in which tasks are randomly assigned to the sites without any task execution guarantee.

1. Introduction

Computational Grids provide transparent access to large-scale distributed computational resources and lend themselves, by their size and geographic distribution, to the creation of *federations* for sharing and aggregating large repertoire of resources [11], [14]. These grids consist of heterogeneous resources (PCs, workstations, clusters and supercomputers), with varying resource management requirements (single system image OS, queuing systems, etc.). The resource management systems for such grids should provide mechanisms and tools that facilitate the realization of the goals of both resource owners and users. Most of the existing work on resource management and scheduling problems in grids adopts a *conventional style* where a scheduling component decides which jobs are to be executed at which site based on certain cost functions (AppLeS [3], Netsolve [6], Legion [9]). Such cost functions are often driven by system-centric parameters that enhance system throughput and utilization rather than improving the utility of application processing.

This paper presents agent-based game theoretic resource management techniques, classified as cooperative, semi-cooperative and non-cooperative games. Using game theory, the behavior of these agents can be tailored to provide profitable contracts to the various federations of the grid, and, therefore, can provide allocation with a reasonable amount of processing and communications overhead. Within each federation care must be taken to ensure that the federation as a whole operates profitably. The individual resources that compose the federation also have their particular interests (*i.e.* some sites may not wish to execute jobs). Thus, the need to accommodate the wishes of the individual resources and the need of the federation as a whole to turn a profit must be balanced out.

The rules of resource management must decide whether the individuals are free to cater for their own interests or they have the *option* of planning as a group in advance of choosing their actions. This leads to the main question of whether or not the gaming mechanism involves cooperation. This paper seeks to find answers to these questions by proposing and comparing cooperative and non-cooperative gaming mechanism, using theoretical and experimental framework. The results are not surprising but the study highlights the pros and cons of each approach, describing as to how to best optimize the system resources. The main highlights of this paper are:

An economic model for grid environments: In this model an agent represents a grid site (or machine) and

1-4244-0054-6/06/$20.00 ©2006 IEEE

thus has the knowledge of the available computational resources, such as, CPU, memory, I/O, etc. Based on this information the agents compete individually or collectively to execute user submitted tasks.

Hierarchical policies for resource management in grid environments: The hierarchical resource management mechanism is a three-tier computational model. At the top level is the global portal to which the tasks are submitted to be executed by the Grid. This top level is a control structure for the middle-level federations, which are composed of individual sites. The proposed architecture has various advantages, such as ease of maintenance, decentralized control and fault tolerance. This approach is also analogous to many real life examples and consumer models, e.g. food chain models [7].

Gaming mechanisms for grid resource management: The paper proposes three game theoretical mechanisms for resource management: (a) A non-cooperative gaming mechanism where every site competes to maximize its profit without having the option for cooperation; (b) A cooperative method in which all of the sites deliberate with one another to execute all the tasks as efficiently as possible (each site negotiates with the other sites to maximize the profit of the group as a whole); (c) A semi-cooperative method in which each site tries to delegate its work to others in a limited fashion. All three methods are compared.

A mathematical foundation for the use of cooperative games in grid scheduling: Specifically, we derive mathematical game theoretical results that enable us to propose a cooperative Grid resource allocation mechanism. Based on this mathematical model, we prove that in any cooperative approach, global optima are only achievable when cooperation occurs among all the agents.

The remainder of this paper is organized as follows: Section 2 covers the related work pertaining to Grid resource allocation. In Section 3 we provide some necessary background game theoretical material and push the case of choosing different gaming environments. Section 4 focuses on describing the system architecture, while Section 5 illustrates the Grid computing environment. In Section 6 we focus on describing the four game theoretical resource allocation models and also describe the baseline method. We provide experimental results and final concluding remarks in Sections 7 and 8, respectively.

2. Related Work

Over the past several years, economic approaches to resource allocation have been developed quite successfully. They satisfy some basic requirements for a Grid setting [20], namely: 1) they are naturally decentralized, 2) decisions about whether to consume or provide resources are taken locally by the clients or service providers, 3) the use of currency provides incentives for service providers to contribute resources, and 4) clients have to act responsibly and cannot afford to waste resources due to their limited budget. A number of systems [1], [3], [10], [16], [26] have been built using a market mechanism to allocate the computational resources. However, all of them make the inherent assumption that a market-based approach is per se better, which is ad hoc, as the allocation depends on many factors besides demand and supply, such as communication delays, bandwidth, server speeds, etc.

Market mechanisms provide a way of representing the system state. They value resources and achieve an efficient match of supply and demand. While some systems use only a price and match or offers and bids, others employ more sophisticated auction protocols [20]. Spawn [26], POPCORN [16], and CPM [4] are examples of systems which employ decentralized auctions with resource accounting. Dynasty [3] pursues a different approach: avoiding the communication overhead of auctions, it uses brokering without any ongoing negotiation. Prices are periodically fixed and there are fees for migration and data transport services. A different, yet interesting, approach is taken in Challenger [10], which implements load balancing with a market approach but without money. When a task is submitted, a request for bids containing its priority value and information that can be used to estimate its execution time is sent to the agents in the network. These agents provide bids by estimating the job execution time (incorporating certain important parameters such as, message delays etc.). A similar but more flexible approach is proposed in Nimrod-G Resource Broker [1], which is a resource management system for scheduling computations on globally distributed resources with varying QoS.

In a resource allocation mechanism proposed in [11] each computer optimizes its profits by considering the payment and cost involved in handling a task. Another approach, *GameMosix,* is adopted in [25], where agent behavior is modeled using a "friendship" model. Through this model, computers help each other (by sharing resources) only when they have previously established friendships. The main focus in both approaches is to provide a load balanced environment, which in many scenarios is a secondary issue.

Our work differs from all the above in: 1) providing a new hierarchical grid resource management infrastructure, 2) proposing various game theoretical resource allocation techniques that fit well in the proposed infrastructure, 3) extensively comparing the gaming strategies, and 4) deriving a cooperative method that focuses on optimizing the Grid usage as a whole.

3. The Case for Gaming Environments

Game theory is a collection of analytical tools designed to help understand the phenomena observable when players or decision makers interact. The basic assumptions that underlie the theory are that players pursue well-defined objectives, that are rational and take into account their knowledge or expectations of other player's behavior. There are two basic types of games [17]:

1. Non-cooperative game: It is a game structure in which the players do not have the *option* of planning as a group in advance of choosing their actions.

2. Cooperative game: A game structure in which the players have the *option* of planning as a group in advance of choosing their actions.

In essence, the three game theoretical methods proposed here fall in one of the two (or some combination of the two) basic categories. From the category of non-cooperative games we choose to formulate the game of Grid task allocation (or resource allocation) as a sealed-bid auction. In such a setup, a task is awarded to the highest bidder, meaning that the agent that is the best fit to execute the task wins. Thus, each agent competes in a non-cooperative environment to obtain the rights to execute tasks. For the work done, each agent is compensated by side-payments. If somehow an agent fails to execute a task, the game has no procedure to allow reallocation of the task.

To tackle the problem of immediate rejection of tasks, we propose a semi-cooperative *n*-round sealed-bid auction, which incorporates task reallocation. The agent who is unable to execute the task, on random, chooses an agent and passes the un-executable task to it. If the newly chosen agent can execute the task then the reallocating procedure finishes, otherwise, another agent is chosen on random. This process is repeated *n*-1 times, where *n* is the number of the agents. If even on the *n*-1 attempt the task still remains un-executable, then it is rejected.

The cooperative approach is the pin-up of this paper. All the agents collectively negotiate and deliberate to come up with a task allocation that is beneficial to the system as a whole. It is to be noted that some individual agents may not be content with the decision of the coalition, but the resulting allocation is efficient in the sense that it aims to reduce the makespan and provides a load balanced task allocation. We prove that this approach results in a society-efficient allocation that is superior to any non-cooperative or semi-cooperative approach.

4. Hierarchical Management of Grids

For managing large-scale grid systems, a multi-tier hierarchical organization is proposed which consists of:

Top level: A top-level resource manager is responsible for handling tasks submitted to the Grid. This manager could be either centralized or distributed.

Mid-level: Many mid-level federations: collections of computing resources grouped by particular interests (*i.e.* resources owned by an entity, resources within a geographic region, etc.).

Bottom-level: Within each federation, many individual computing resources (*e.g.* PCs, clusters and supercomputers).

When an application (metatask) is submitted to the Grid, the top level resource manager would give tasks to a particular federation, and that federation would then be responsible for giving the tasks to the particular computing resource that will execute it. By delegating responsibility for the tasks to the federation, the Grid can be viewed as a set of tens or hundreds of individual entities (the federations), instead of thousands or tens of thousands of resources, as would be the case with a flat model. This makes management of the overall system much easier. Additionally, each federation, due to the similar interests or common ownership of its participants, can follow the same administrative guidelines. This allows for ease of management and administration at both the federation level because each federation can have a set of unique administrative policies while maintaining its participation in the Grid through common interfaces, and at the Grid level by reducing the number of visible participants from view of the top-level resource manager.

5. Grid Computing Environment

To facilitate the management of this hierarchical grid architecture (as described in Section 4), we propose a multi-agent system. At the top level a Grid *broker* is responsible for handling the Grid job queue (we call it a job queue since it includes both user submitted tasks and local system tasks), similar to the setup of the Globus framework [11]. The federation level special agents called *ambassadors* are responsible for interaction with the broker. Underneath the ambassadors, each computing resource would be represented by an *agent*. By creating a grid economy, such as the one used in Nimrod-G [5], the ambassadors can submit bids for *execution contracts* to the broker, who would then select the winning federation by means of a sealed-bid auction. In order for the ambassadors to effectively estimate the execution time for a contract bid, each ambassador will query the agents in its federation for estimates on each task in order to generate an Estimated Time of Completion (ETC) matrix. (Details of how to obtain an ETC matrix are provided in Section 7.) The ambassador will then be responsible for selecting which tasks it can execute and the price at which it can execute those tasks.

The remainder of this paper will be concerned with how an ambassador efficiently distributes tasks from an awarded execution contract to its agents.

6. Resource Allocation Methods

In this section we first describe the conventional baseline method, followed by detailing the three game theoretical resource allocation techniques.

6.1. Conventional Baseline Method (BASE)

A conventional baseline (BASE) method is necessary to establish a lower bound on the performance of various game theoretical methods. We will use this method to visualize the difference in the solution quality between BASE and the studied game theoretical techniques. The BASE method was originally proposed in [18]. The basic idea of BASE is that when tasks arrive at the middle-tier, each agent is assigned a task on random by the ambassador. If an agent is unable to execute the task within the specified deadline, it is rejected without any reallocation. (Notice that if all the machines are heavily loaded, BASE can be a very effective strategy.) It is easy to see that BASE can allocate tasks in linear time $O(m)$.

6.2. Non-cooperative Method (NC)

The non-cooperative (NC) sealed-bid method relies on the solicitation of bids from the agents. Each agent submits a bid for a task that is at the *head* of the ambassador's job queue. After receiving bids from all the agents, the ambassador selects the agent which submitted the highest bid and awards the task for execution. For simplicity, we assume that an agent *j*'s bid (b_j) is inversely proportional to the ETC of a job onto its machine. (This is very common assumption; for instance, see [11] and [16].) That is, $b_j=[n\text{-}1/n]v_j$, where $v_j=1/\text{ETC}$ of task and n is the number of agents in the system. The following steps are involved in a successful run of NC:

Step 1: Ambassador de-queues the job queue and announces that a task *i* is ready for bidding.

Step 2: Each agent in parallel does the following:

Step 2a: Derive ETC for task *i*.

Step 2b: Artificially en-queues task *i* into its local job queue to observe if it is possible to execute the task within its associated deadline. (Details of how the deadlines are associated with a particular task is discussed in detail in Section 7. For the time being assume that each task has a deadline associated with it).

Step 2c: If answer to Step 2b is *yes* then submit $b_j=[n\text{-}1/n]v_j$, otherwise submit $b_j=0$ (since ETC=∞).

Step 3: Ambassador sorts the bids and chooses the agent with the highest bid.

Step 4: Ambassador allocates the task to the agent identified in Step 3, and the sequence repeats.

Notice that the ambassador does not immediately compensate the agent for executing the task. This is because the bidding process is entirely based on ETC. It is possible that after the allocation of a task *i*, an agent *j*'s job queue (which includes both local and Grid tasks) is overwhelmed with local (machine generated) tasks. This would delay the processing of task *i*, and probably to the extent that it is no longer possible to finish *i*'s execution before its deadline. If that happens then there is no option than to reject task *i*. Based on this observation, the ambassador compensates the agent if and only if it receives the required output (a guarantee on the actual execution of tasks) of the task. The compensation is equivalent to the bid posed by the agent who executed the task, making it a first-price sealed-bid auction [13]. By inspecting NC, we observe that tasks can be allocated in $O(mn\log n)$, where the most expensive computation takes place in Step 3 (sorting of the submitted bids).

6.3. Non-cooperative Method with Certain Degree of Cooperation (NNC)

The semi-distributed *n*-round non-cooperative (NNC) sealed bid method is similar to the NC method, but allows some limited cooperation. This is accomplished by handing a task to a different agent if the original agent cannot execute that task before the specified deadline, as opposed to immediately rejecting the task as is the case with NC. This handing off occurs *n*-1 times, after which the task is rejected. Since the initial task allocation procedure of NNC is exactly the same as NC, below we describe the task handoff steps involved in a successful run of NNC:

Step 1: For each assigned task repeatedly observe the deadline constraint in conjunction to the local job queue.

Step 2: If (at some point) the task cannot be executed, then randomly choose an agent (to differentiate we tag these agents as *helper* agents) and send the task for execution. (With the task attach a data structure (*D*) that holds the information about who is the original agent and which helpers have already been consulted.)

Step 3: After the job arrives at the helper's site, it artificially en-queues the task into its local job queue to observe if it is possible to execute the task within its associated deadline.

Step 3a: If it can execute the handoff task, then the process terminates.

Step 3b: Otherwise, choose on random another agent that is currently *not* present in *D* and send the task to it. If in *D* the number of helpers is equivalent to *n*-1, then reject the task. (The data structure remains associated with the task until the task is successfully executed or it is rejected.)

Notice that the compensation method of NC is fully applicable in NCC without any alteration. The agents (original or helper) are duly paid upon receiving the necessary output of tasks. Since NNC is similar to NC the initial allocation takes $O(mn\log n)$, but NNC allows reallocation of tasks so the running time of NNC becomes $O(mn\log n+m_r)$, where m_r is the number of reallocated tasks.

6.4. Cooperative Method (COOP)

In this setup all agents cooperate (COOP) to collectively progress towards the global goal of efficiently allocating tasks. In non-cooperative games, agents' sets of possible actions and their preferences over the possible outcomes, where an outcome is a profile of actions; each action is taken by a single agent autonomously. The primitive of the cooperative game model is the collection of set of joint actions that each group of agents (coalition) can take independently of the remaining agents. An outcome of a cooperative (or more appropriately a coalition) game is a specification of the coalition that forms and the joint action it takes. The other primitive of the model of a coalition game is the profile of the players' preferences over the set of all possible outcomes. Thus although actions are taken by coalitions, the theory is based on the individual's preferences.

Definition 1 [17]**:** *A coalition game consists of:*

1) a finite set of agents (players) N, and

2) a function v that associates with every nonempty subset S of N (which forms a coalition) a real number v(S) (which is the worth of S).

For each coalition S, $v(S)$ is the total payoff that is available for division among the members of S. In other words, the set of joint actions that the coalition S can take consists of all possible divisions of $v(S)$ among the members of S. It is important to understand that $v(S)$ should be interpreted as the maximum payoff that the coalition S can guarantee independently of the behavior of the coalition $N\backslash S$. Another important property (Definition 3) of the worth of the coalition of all players ($v(N)$) is that the worth is at least as large as the sum of the worth of the members of any partition of N.

Definition 2 [17]**:** *A coalition game (N,v) is cohesive if* $v(N) \geq \sum_{k=1}^{K} v(S_k)$ *for every partition* $\{S_1, \ldots, S_K\}$ *of N.*

In essence, Definition 2 is a special case of the condition of superadditivity, which requires that $v(S\cup T)\geq v(S)+v(T)$ for all coalitions S and T with $S\cap T = \varnothing$. This property is also important to understand the central concept of coalition games – the *core*. The main idea behind the core is similar to that of the Nash equilibrium of a non-cooperative game, *i.e.*, an outcome is stable (or in equilibrium) if no deviation is profitable. In the case of the core, the outcome is stable if no coalition can deviate and obtain an outcome better for all of its members. Note that in a coalition game there can be as many as 2^N possible coalitions. It would be impractical to go through each and every coalition in order to find which coalition is the most beneficial. But from Definitions 1 and 2 it is clear that the coalition game profits the most when a coalition of N agents forms – called the *grand coalition* [22]. Thus, if we are able to (some how) guarantee that in a particular coalition game the grand coalition is always the most beneficial, then we can simply confine our search to the grand coalition. To be technically precise, we need to prove that the core of a coalition game is nonempty [17]. If the core is nonempty, then there exists at least one outcome that is stable via the grand coalition. On the other hand if the core is empty then there is no outcome that can guarantee stability. Below we proceed in that direction.

Let (N,M,U,σ_0) be a coalition Gird task allocation game, where $N = \{1,\ldots,n\}$ is the set of agents, $M = \{1,\ldots, m\}$ is the set of tasks in the ambassador's job queue, U is a non-negative $n\times m$ matrix that gives the utility (u_{ij}) of each task i for each agent j (the utility is to be interpreted as the ETC of a task on an agent's machine, *i.e.*, the smaller the ETC the larger is the utility), and σ_0 is the order of the ambassador's job queue. We define the worth $v(S)$ of a coalition $S\subseteq N$ as the maximum total profit (utility) it can guarantee itself without any help from $N\backslash S$. This utility can be determined in two stages. In the first stage, all players sequentially choose a task, respecting σ_0. In the second stage, the members of S reallocate the chosen tasks among themselves to reach coalitional efficiency. Obviously, the outcome of this reallocation depends on the tasks chosen by the members of S, and therefore also on the tasks chosen by the members of $N\backslash S$.

In order to describe the value $v(S)$ of a coalition $S\subseteq N$, we make use of a very common technique to describe coalition structure formation – the *extensive form game* [13]. We define an extensive form coalition task allocation game $(\{S,N\backslash S\},T,C^S,u^S)$ with agent set $\{S,N\backslash S\}$. The various components of this game are as follows.

1. The Root: For the root of the tree T, let $1\leq k\leq m$ and the set of bijective maps from $\{1,\ldots,k\}$ to M is denoted by S_k. A map $\pi\in S_k$ is interpreted as a situation where task $\pi(i)$ is chosen by agent i for each $1\leq i\leq k$. Similarly, we define S_0 as the situation where none of the tasks is chosen yet. Let T be the rooted tree with node set S_k and root S_0. There is an arc between $\pi\in S_k$ and $\tau\in S_{k+1}$ with $0\leq k\leq m$-1, if and only if $\pi(i)=\tau(i)$ for all $1\leq i\leq k$. That is, there is an arc between π and τ if π can be extended to τ by assigning an appropriate task to player k+1. So, $V_1=S_k$ and $V_2=S_m$ are the sets of non-terminal and terminal nodes, respectively.

2. Control: The control function $C^S:S_k\rightarrow\{S,N\backslash S\}$ is defined as follows. Let $\pi\in S_k$ for some $0\leq k\leq m$-1. Then we define $C^S(\pi)=S$ if and only if $k+1\in S$. So coalition S controls the nodes at which one of its members is to choose a task for scheduling. Let Σ_S and $\Sigma_{N\backslash S}$ be the set of all possible strategies of agents S and $N\backslash S$, respectively.

3. Utility function: Finally, we describe the utility function $u^S: \Sigma_S \times \Sigma_{N\backslash S} \rightarrow \Re^{\{S,N\backslash S\}}$. Let $y=(y_S,y_{N\backslash S}) \in \Sigma_S \times \Sigma_{N\backslash S}$. Let $\tau \in S_m$ be the terminal node reached by strategy profile y, and let $H_S(\tau)=\{\tau(i): i \in S\}$ be the corresponding set of tasks identified for scheduling by S. Now define $u_S^S(y)=\max\{\Sigma_{i \in S}: \pi \in \Pi(S,H_S(\pi))\}$, and $u^S_{N\backslash S}(y)=-u_S^S(y)$. So, the payoff of S at terminal node $\tau \in S_m$ is the maximum utility S obtains after reallocating the initially chosen tasks and the payoff for $N\backslash S$ is just the opposite of the payoff of S. Hence, $N\backslash S$ maximizes its payoff at the extensive form game by minimizing the payoff of S and vice versa.

Based on that above discussion the coalition grid task allocation game (N,v) is defined by:

$$v(S) = \max_{yS \in \sum_S} \min_{yN\backslash S \in \sum_{N\backslash S}} u_S^S(y), \quad \forall\ S \subseteq N \cdot \tag{1}$$

Notice that $v(S)$ is precisely the maximum utility that coalition S can guarantee itself, without any help from $N\backslash S$. Using Equation 1, we show the non-emptiness of the core of the coalition grid task allocation game.

Theorem 1: *Let (N,M,U,σ_0) be the coalition gird task allocation game and let (N,v) be its corresponding coalition game. Let the core of (N,v) be the set $C(v) = \{x \in \Re^N: \Sigma_{i\in S} x_i \geq v(S)\}$ for every $S \subseteq N$ and $\Sigma_{i\in N} x_i \geq v(N)$. Let $(u,w) \in C(v)$ and let the mapping $\tau:\{1,\ldots,m\} \rightarrow M$ be a bijection such that $w_{\tau(1)} \geq \ldots \geq w_{\tau(m)}$. Define $x_i=u_i+ w_{\tau(i)}$ for all $i \in N$. Then, the allocation x belongs to the core of (N,v), i.e., $x \in C(v)$.*

Proof: By definition of x, $\Sigma_{i\in N} x_i = v(N \cup M)$. Since $v(N \cup M) = v(N)$, $\Sigma_{i\in N} x_i = v(N)$. It remains only to show stability. Consider the extensive form game $(\{S,N\backslash S\},T,C^S,u^S)$, with strategy $z_{N\backslash S} \in \Sigma_{N\backslash S}$ for the agents in set $N\backslash S$: "always schedule the task with highest w_i that is still available." More precisely, let $z_{N\backslash S} \in \Sigma_{N\backslash S}$ be such that $z_{N\backslash S}(\sigma) = \pi$ for each $\sigma \in S_k$, $k+1 \in N\backslash S$, and $\pi \in S_{k+1}$ with $w_\pi(k+1) \geq w_j$ for all $j \in M\backslash\{\sigma(1),\ldots,\sigma(k)\}$.

Now if the agents in set S would use a similar strategy in the strategic form game as the agents in set $N\backslash S$, *i.e.*, also "always pick the highest w_i that remains," then the agents in set S would acquire $\{\pi(i): i \in S\}$ as its set of tasks. If the agents in set S use a different strategy, then, given the agents in set $N\backslash S$'s strategy $z_{N\backslash S}$, it would obtain a set of tasks A with lower w_i values. Formally,

$$\sum_{a \in A} w_a \leq \sum_{i \in S} w_\pi(i) \cdot \tag{2}$$

In particular, let the agents in set S play a best reply against strategy $z_{N\backslash S}$. Let A^* be the set of tasks scheduled by the agents in set S. Let $\pi: S \rightarrow A^*$ be the optimal rescheduling of the tasks. From Equation 2 it follows that:

$$\sum_{i \in S} w_\pi(i) = \sum_{a \in A^*} w_a \leq \sum_{i \in S} w_\sigma(i) \cdot \tag{3}$$

Hence,

$$\begin{aligned} &\sum_{i \in S} x_i = \\ &\sum_{i \in S} u_i + \sum_{i \in S} w_\sigma(i) \geq \sum_{i \in S} u_i + \sum_{i \in S} w_\pi(i) \geq v_A(S \bigcup \{\pi(i): i \in S\}) \\ &= \sum_{i \in S} U_{i\pi(i)} \end{aligned} \tag{4}$$

.

The first inequality is due to Equation 3. The second inequality is satisfied because $(u,w) \in C(v_A)$. The last equality is satisfied since the matching $\{(i,\pi(i)): i \in S\}$ is an optimal rescheduling, and hence optimal for coalition $S \cup \{\pi(i): i \in S\}$ at the assignment game (N,v_A).

From the definition of the game (N,v) it follows that:

$$\sum_{i \in S} U_{i\pi(i)} = \max_{yS \in \sum_S} u_S^S(yS, z_{N\backslash S}) \geq \max_{yS \in \sum_S} \min_{yN\backslash S \in \sum_{N\backslash S}} u_S^S(y) = v(S) \cdot \tag{5}$$

Now the theorem follows immediately from Equations 4 and 5.■

The above results assert that in the coalition gird task scheduling game, the agents can only gain a superior utility if they all cooperate to find an allocation for the tasks. Thus, rather than investigating all the 2^N possible forms of coalition, we can confine ourselves with evaluating the outcome of the grand coalition. However, it remains to be seen how one can optimally match n agents to the m tasks, since the total possible combinations are of magnitude $O(m!/(n-m)!)$. It turns out that this computationally infeasible problem can be solved in $O(nm^2)$ time using the widely cited Hungarian method [14] . Below we detail the Hungarian method.

Step 0: Take as input the ETC matrix U.

Step 1: Subtract the smallest entry in each row form all the entries in that row. (Each row will have at least one zero entry and all other entries will remain positive.)

Step 2: Subtract the smallest entry in each column from each entry in that column. (Each row and column will have at least one zero entry.)

Step 3: Cover the zeros identified by Steps 1 and 2 by crossing out the rows and columns of U. This cover should be obtained by the minimal number of crossings.

Step 4: Check for optimality.

Step 4a: If the number of crossings is n, then optimality is reached. Go to Step 6.

Step 4b: If the number of crossing is less than n, then an optimal assignment need to be found. Go to Step 5.

Step 5: Determine the smallest entry not covered by any crossing. Subtract this entry from all uncovered entries and add it to all entries covered by both a horizontal and vertical crossings. Go to Step 3.

Step 6: Scan each row. The first zero is the task to agent allocation provided that the column does not already contain an allocation.

Below we detail the steps involved in a successful run of COOP:

Step 1: Ambassador de-queues the job queue n times, where n is the number of agents in the system. This is done for the following two reasons. First, in a real life system, the number of tasks would be much larger than the number of agents in the system. To permit a realistic negotiation process, limiting the number of tasks for a single round of negotiation is extremely important. Second, coalition formation requires perfect information, if we do not limit the size of the utility matrix for the Hungarian method, then we run into the problem of how to obtain ETC for every task that arrives in real-time at the ambassador's job queue. Essentially we want the agents to meet, discuss, negotiate, agree on the allocation, send their allocated tasks to their local job queues and repeat the process for the next set of tasks.

Step 2: Use the Hungarian method to find the task to agent mapping.

Step 3: Ambassador allocates the tasks to the agents that are identified in Step 3, and the sequence repeats.

Once again notice that the compensation method of NC is fully applicable in COOP without any alteration. The agents are duly paid upon receiving the necessary output of tasks. To cater for the case when an agent is unable to execute an allocated task due to deadline constraints, that particular task is sent to the ambassador who inserts that task in front of its job queue so that it can be immediately included in the next round of task allocation. Observe that the Hungarian method is called at least $O(m/n)$ times in a successful rum of COOP. The Hungarian method itself takes $O(nm^2)$. Therefore including the number of reallocated tasks, the total running time of COOP becomes $O((m/n)nm^2+m_r)$.

Finally, we quote from literature the following result which enables us to prove that the COOP method is a society-efficient method for task scheduling in a computational grid.

Lemma 1 [22]: *A coalition game is society-efficient if and only if the core is nonempty.* ■

Theorem 2 [22]: *COOP is a society-efficient method.*

Proof: Follows from the result of Theorem 1 by applying Lemma 1. ■

7. Experiments and Discussion of Results

A hierarchical Grid infrastructure is simulated using a discrete event-driven simulator, where tasks are submitted to a centralized broker. The broker then advertises the job queue to be bid on, after which each federation submits a bid to the broker. The bids are generated by soliciting estimates from the federation's sites. As mentioned in Section 5, we only confine ourselves with the allocation of tasks at the middle and bottom tier levels. Therefore, the simulation encapsulates a federation of workstations.

Note that the ETC values may differ from actual times, e.g., actual times may depend on input data and communication delays. Therefore, for the simulation studies, the *Actual Time to Complete* (ATC) values were calculated using the ETC values as the mean. (The ATC values were used for the evaluation of the techniques.) The tasks were assumed to be independent with multiple deadlines. The worth of a submitted task degrades according to a degradation scheme, if the task misses a certain deadline. More specifically, let w_i be a deadline factor for task i, where

$$w_i = \begin{cases} 1.00 & \text{if } t_i \text{ finished at or below its primary deadline,} \\ 0.50 & \text{if } t_i \text{ finished at or below its 50\% deadline,} \\ 0.25 & \text{if } t_i \text{ finished at or below its 25\% deadline,} \\ 0.00 & \text{if } t_i \text{ is never executed.} \end{cases}$$

(Note that w_i indicates the degradation scheme of the worth a task according to when the task finishes.)

A grid system with 16 machines and an average of 200,000 tasks was simulated for a period of 300 minutes. For each of the scenarios that are discussed later in this section, 50 trials were run. (A trial is defined as one such simulation of the Grid system.) The period from 0 to 10 minutes was the system start-up period. The period between 10 to 250 minutes was considered the evaluation period (*i.e.*, the period where the scheduling techniques' performances were measured). Within the simulation period (*i.e.*, the system start up period and the evaluation period), the arrival times of the tasks were randomly generated using a Poisson distribution. To better simulate an overloaded system, the mean task inter-arrival time was faster (3.2 seconds) during the system start-up period than during the evaluation period (7.6 seconds). In addition, random bursty arrival rate periods were introduced during the evaluation period, where the arrival rate was increased. These periods did not overlap with each other and had a mean task inter-arrival time of 7 seconds. The duration of a bursty period was 10 minutes.

The estimated execution times of all tasks taking heterogeneity into consideration were generated using the gamma distribution method described in [2]. Four different cases of ETC heterogeneities were used in this study, the high task and high machine heterogeneity case, the low task and low machine heterogeneity case, the high task and low machine heterogeneity case and the low task and high machine heterogeneity case. (The data presented in this article is the average of the 4 cases over all the trials.)

The deadline of each task was calculated by incorporating the arrival time of the task, plus the median execution time of the task (across all machines), plus a multiplier times the median execution time of all tasks (*i.e.*, 20 minutes of simulation study). Two types of deadlines, *i.e.*, loose and tight, were used in the simulation. The multiplier was changed to make the deadlines (*i.e.*, the 100%, 50%, and 25% deadline) for the two types of deadlines. For the loose deadline, the

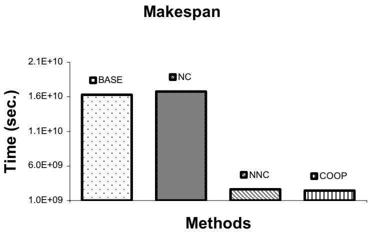

Figure 1. Makespan.

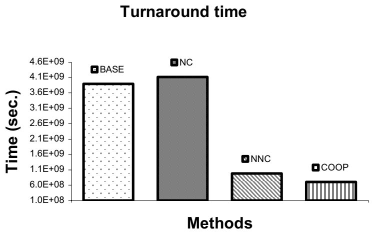

Figure 2. Turnaround time.

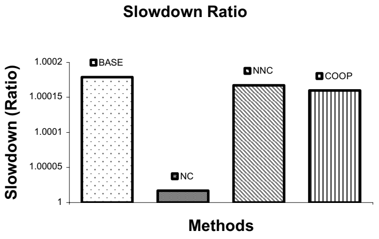

Figure 3. Slowdown ratio.

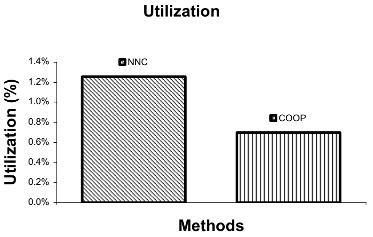

Figure 4. Utilization.

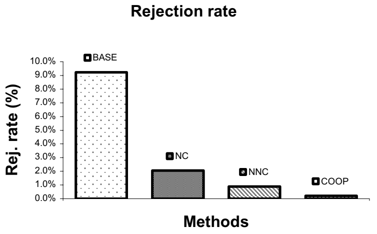

Figure 5. Task rejection rate.

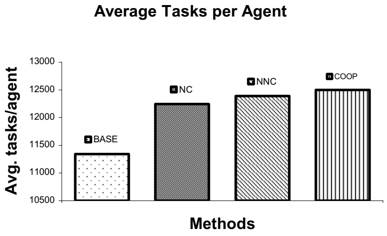

Figure 6. Average Tasks per agent.

Table 1. Load balancing: Tasks per agent and standard deviation on the distribution of tasks.

Agents	Methods (Tasks per Agent)			
	BASE	NC	NNC	COOP
1	11066	11736	1097	12443
2	11350	12014	1806	12474
3	11385	12479	3393	12517
4	11025	11773	1378	12437
5	11644	12431	6048	12549
6	10886	11585	934	12417
7	11845	12750	17104	12560
8	11413	12607	4377	12543
9	11143	11919	1599	12483
10	11572	12347	3288	12527
11	11300	12103	1917	12490
12	11476	12251	2289	12514
13	11190	11933	1647	12473
14	11797	13014	147496	12579
15	11053	11937	1497	12476
16	11387	12125	2347	12503
Standard deviation	278.50	394.00	36239.22	46.02

multiplier was set to four, eight, and twelve for the primary (100%), 50%, and 25% deadline, respectively. the tight deadline, the multiplier was set to one, two, and four for the primary (100%), 50%, and 25% deadline, respectively. (By loosening the deadlines the depreciation of the task as its execution is delayed is reduced, thereby increasing the task's after-execution worth. For example, a multiplier of 4 would increase a task's after-execution worth by 25%; a multiplier of 8 would increase it by 50%, etc.) To evaluate the various techniques discussed in this paper, we made use of the following performance matrices:

Makespan: The latest finish time among all tasks.

Turnaround time: The average time spent by a task in the Grid.

Slowdown ratio: The ratio of the average turnaround time to the average waiting time of all tasks.

Utilization: The fraction of resources used by reallocated tasks.

Task rejection rate: The percentage of tasks rejected by all the agents in the system.

To begin, we study (Figure 1) the makespan achieved by the techniques. Some very interesting results are observable. First, surprisingly BASE outperforms NC by producing a smaller makespan. This is because in NC the task allocation is biased in favor of more powerful (faster machines) agents. Second, we can observe tremendous reduction in makespan even with minor cooperation among agents. The makespan is reduced by as much as 85% in case of NCC. COOP outperforms all the techniques by producing a makespan of 2409119610. Although this is only an improvement of 8.52% compared to NCC's makespan of 2633596204, yet COOP's task allocation is superior in load balancing, reduced task rejection and other important performance matrices which will be discussed subsequently.

Turnaround time is an important factor in determining how fast an application enters and exits the Grid system. This measure includes: 1) the time a task takes to come to the front of the queue (of the broker) so that it can be considered for scheduling, 2) the time required for a task to propagate through the network from the broker to the ambassador queue, 3) the time taken for a task to move to the front of the ambassador queue, 4) propagation delay from the ambassador to the agent's machine (site), 5) the time a task takes to reach the front of the site queue at which it is to be executed, 6) the time needed to schedule a task on the site's local scheduler, 7) the time a task takes if reallocation is performed, 8) the time a task actually takes for execution, and 9) the time it takes to go back to the broker (possibly through the ambassador). We seek to identify methods that can effectively reduce the cumulative time that a task spends in the Grid system. A technique that exhibits a small turnaround time makes itself available faster to process other inline tasks. Figure 2 illustrates the average turnaround time of the studied techniques. These results are almost identical to the makespan results. The techniques based on shortest turnaround time are ranked as: 1) COOP, 2) NNC, 3) BASE, 4) NC.

The slowdown ratio encapsulates the average total time taken by a job in the Gird infrastructure due to various task allocation decisions. A method that makes allocation decisions correctly and in a timely fashion would have a reduced slowdown ratio. Figure 3 portrays the results obtained by observing the slowdown ratio. Clearly NC, which basically takes decisions locally in a greedy fashion, exhibits the minimum slowdown ratio. COOP performed surprisingly well with a slowdown ratio of 1.0016 (third best). It is natural to think that COOP would depict the worst of the slowdown time; however, COOP's decision quality is unparallel. No decision on task allocation is made unless it is negotiated with all the agents in the system. On the other hand BASE which uses no informed decision on task allocation performs the worst among all the techniques. The techniques ranked according to the smallest slowdown ratio are as follows: 1) NC, 2) COOP, 3) NNC, 4) BASE.

If an agent cannot executed a given task, it may become necessary to reallocate that task to another agent. However, this necessity comes at a cost of increase in turnaround time and slowdown ratio. A technique that exhibits a smaller utilization factor ensures superior allocation. Note that BASE and NC cannot be included in this comparison because they have no facility for reallocation of tasks. Figure 4 shows COOP with 0.69% utilization compared to 1.2% of NNC. This is analogous to the measure of task rejection rate (Figure 5), where COOP again outperforms the other methods, followed by NNC and NC.

The average tasks per node or agent (Figure 6) is a mirror inverse of the rejection rate measurement, as it is based on the total number of tasks completed by the federation. Load balancing (Table 1), however, provides important information about how well the methods can distribute the tasks among the various agents in order to create the most efficient use of the federation's resources. The entries in Table 1 represent the number of tasks executed by each agent averaged (and rounded off) over the number of trials. Due to the heterogeneous nature of the federation, NNC has the worst load balancing in our study (based on standard deviation). In this case one agent (Agent 14) completely outclasses the other agents in the system, and as a result more tasks are allocated there. This in turn increases the reallocation of tasks, hence the worst load balancing. BASE demonstrated acceptable load balancing because of the random assignment of tasks to sites. NC provides a mediocre load balancing but worst than the naïve BASE method. COOP demonstrated the best load balancing of any method due to the desire of sites to ensure the most efficient execution scheme possible. The techniques ranked according to the best load balanced workload are: 1) COOP, 2) BASE, 3) NC, 4) NNC.

8. Conclusions

This paper proposed and compared various game theoretical resource allocation techniques in the Grid computing environment. The cooperation among the agents needed when an agent is unable to guarantee the execution of task can occur in a number of ways as illustrated by the techniques discussed in this paper. On one extreme was a technique that did not allow any cooperation among agents, while on the other extreme was a method that utilized the concept of coalition formation to collectively approach the problem.

The simulation study, which was built around a newly proposed hierarchical Grid infrastructure, used a diverse workload that captures task to machine heterogeneity extremely well. The hierarchical Grid infrastructure consists of machines that abstract into larger computing centers labeled "federations," each of which is responsible for managing its own resources independently. These federations are then linked together with a primary portal to which Grid jobs would be submitted. Using this simulation model we extensively evaluated the proposed game theoretical techniques and studied their behaviors under various performance metrics, such as, makespan, turnaround time, slowdown ratio, utilization, task rejection rate and load balancing. Based on our experimental findings, we conclude that the cooperation among agents is not only important but extremely necessary in order to execute tasks that bear multiple execution time deadlines. Although the proposed cooperative method has high computational complexity, yet the task allocation has low: 1) task rejection, 2) utilization, 3) slowdown ratio, and 4) turn around time. Moreover, the allocation has near perfect load balancing and minimum makespan. For applications that are of critical nature this cooperative approach is the best choice. For other applications the simple conventional baseline method would be best suitable as other game theoretical approaches such as, the non-cooperative and semi-cooperative are only *just* better.

References

[1] D. Abramson, R. Buyya, and J. Giddy, "A Computational Economy for Grid Computing and its Implementation in the Nimrod-G Resource Broker," *Future Generation Computer Systems Journal*, vol. 18, no. 8, pp. 1061-1074, 2002.

[2] S. Ali, H. J. Siegel, M. Maheswaran and D. Hensgen, "Representing Task and Machine Heterogeneities for Heterogeneous Computing Systems," *Tamkang Journal of Science and Engineering*, Special 50th Anniversary Issue, vol. 3, no. 3, 2000, pp. 195-207, 2000.

[3] M. Backschat, A. Pfaffinger, and C. Zenger, "Economic-based Dynamic Load Distribution in Large Workstation Network," in *Proc. of the 2nd International. Euro-Par Conference*, vol. 2, 1996, pp. 631-634.

[4] F. Berman and R. Wolski, "The AppLes Project: A Status Report," in *Proc. of the 8th NEC Research Symposium*, 1997.

[5] J. Brooke, M. Foster, S. Pickles, K. Taylor, T. Hewitt, "Mini-Grids: Effective test-beds for Grid Application," in *Proc. of the 1st IEEE/ACM International Workshop on Grid Computing*, 2000, pp. 158-169.

[6] R. Buyya, D. Abramson and J. Giddy, "Nimrod-G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid," *International Conference on High Performance Computing in Asia-Pacific Region*, 2000, pp. 283-289.

[7] H. Casanova and J. Dongarra, "NetSolve: A network server for solving computational science problems," *International Journal of Supercomputing Applications and High Performance Computing*, vol. 11, no. 3, pp. 212-223, 1997.

[8] T. Casavant and J. Kuhl, "A Taxonomy of Scheduling in General-purpose Distributed Computing Systems," *IEEE Trans. on Software Engineering*, vol. 14, no. 2, pp. 141-154, 1988.

[9] S. Chaplin, J. Karpovich and A. Grimshaw, "The Legion Resource Management System," in *Proc. of the 5th Workshop on Job Scheduling Strategies for Parallel Processing*, 1999, pp. 162-178.

[10] A. Chavez, A. Moukas, and P. Maes, "Challenger: A Multi-agent System for Distributed Resource Allocation," in *Proc. of the 1st ACM International Conference on Autonomous Agents*, 1997, pp. 323-331.

[11] I. Foster, C. Kesselman, "Globus: A Metacomputing Infrastructure Toolkit," *International Journal of Supercomputing Applications*, vol. 11, no. 2, pp. 115-128, 1997.

[12] D. Grosu and A. T. Chronopoulos, "Algorithm Mechanism Design for Load Balancing in Distributed Systems," *IEEE Trans. Systems, Man, and Cybernetics*, vol. 34, no. 1, pp. 77–84, 2004.

[13] V. Krishna. *Auction Theory*, Academic Press, San Diego, U.S.A., 2002.

[14] H. Kuhn, "The Hungarian Method for the Assignment Problem," *Naval Res. Logistics Quarterly*, vol. 2, pp. 83-97, 1955.

[15] B. Lesyng, P. Bała, D. Erwin, "EUROGRID: European Computational Grid Test bed," *Journal of Parallel and Distributed Computing*, vol. 63 no. 5, pp. 590-596, May 2003

[16] N. Nisan, S. London, O. Regev, and N. Camiel, "Globally Distributed Computation over the Internet: The POPCORN Project," in *Proc. of the 18th ICDCS*, 1998, pp. 592-601.

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

[18] T. Quint, "On One-sided Versus Two-sided Matching Markets," *Games and Economic Behavior*, vol. 16, pp. 124-134, 1996.

[19] K. Ramamritham, J. Stankovic and W. Zhao, "Distributed Scheduling of Tasks with Deadlines and Resource Requirements," *IEEE Trans. on Computers*, vol. 38, no. 4, pp. 1110-1123, 1989.

[20] T. Sandholm, "Distributed Rational Decision Making", *Multi-agent Systems*. MIT Press, 2000.

[21] H. Scarf, "The Allocation of Resources in the Presence of Indivisibilities," *Journal of Economic Prespectives*, vol. 4, pp. 111-128, 1994.

[22] L. Shapley, "On Balanced Sets and Cores," *Naval Res. Logistics Quaterly*, vol. 14, pp. 453-460, 1967.

[23] L . Shapley and M. Shubik, "The Assignment Game I: The Core," *International Journal of Game Theory*, vol. 1, pp. 111-130, 1971.

[24] L. Svenson, "Large Indivisibilities: An Analysis with Indivisibilities," *Econometrica*, vol. 51, pp. 939-954, 1983.

[25] D. E. Volper, J. C. Oh, and M. Jung, "GameMosix: Game-Theoretic Middleware for CPU Sharing in Un trusted P2P Environment," in *Proc. of 17th ICDCS*, 2004.

[26] C. Waldspurger, T. Hogg, B. Huberman, J. Kephart, and W. Stornetta, "Spawn: A Distributed Computational Economy," *IEEE Trans. on Software Engineering*, vol. 18, no. 2, pp. 103-117, 1992.