Selfish Grids: Game-Theoretic Modeling and NAS/PSA Benchmark Evaluation

Yu-Kwong Kwok, *Senior Member*, *IEEE*, Kai Hwang, *Fellow*, *IEEE*, and ShanShan Song, *Member*, *IEEE*

Abstract—Selfish behaviors of individual machines in a Grid can potentially damage the performance of the system as a whole. However, scrutinizing the Grid by taking into account the noncooperativeness of machines is a largely unexplored research problem. In this paper, we first present a new hierarchical game-theoretic model of the Grid that matches well with the physical administrative structure in real-life situations. We then focus on the impact of selfishness in intrasite job execution mechanisms. Based on our novel utility functions, we analytically derive the Nash equilibrium and optimal strategies for the general case. To study the effects of different strategies, we have also performed extensive simulations by using a well-known practical scheduling algorithm over the NAS (Numerical Aerodynamic Simulation) and the PSA (Parameter Sweep Application) workloads. We have studied the overall job execution performance of the Grid system under a wide range of parameters. Specifically, we find that the Optimal selfish strategy significantly outperforms the Nash selfish strategy. Our performance evaluation results can serve as a valuable reference for designing appropriate strategies in a practical Grid.

Index Terms—Grid computing, noncooperative games, virtual organizations, selfish behaviors, online scheduling, Nash equilibrium, optimal strategies, performance evaluation, NAS workload, parameter sweep application (PSA).

1 Introduction

The lofty goal of Grid computing [14], [17], [18] is to leverage on the interconnection of a large number of geographically distributed machines to solve computational problems faster at a gigantic scale [6]. However, this goal is based upon the premise that the interconnected machines are *cooperative* in the sense that they are willing to execute remote jobs. We believe that, as the Grid scales up, this premise may no longer hold. Notice that the Grid is a large-scale peer-to-peer (P2P) system at the server-level (rather than at the desktop level as in file-sharing P2P applications). Thus, the "peers," i.e., the Grid sites, owned and managed by different organizations, may not always want to cooperate with each other. Indeed, the various computers *within* a Grid site may not even cooperate with each other. This scenario resembles the situation of the noncooperation among states of a large country, or the noncooperation among departments in a large organization.

Thus, modeling the Grid and its constituents by taking into account the potential noncooperativeness at various levels is an important research problem. With such modeling, we can then study the impact of *selfishness* and subsequently design proper strategies to avoid its adverse impacts. This can, in turn, lead to a much more efficient utilization of the Grid processing resources. However, despite that there have been several recent attempts in scrutinizing the Grid from a so-called "market" oriented perspective [10], [15], [16], [46] (as detailed in Section 2), the modeling problem of the Grid with realistic selfishness concepts is relatively unexplored.

In this paper, we propose a new *game theoretic* modeling of the Grid and present our analytical as well as simulation performance results. Specifically, we make three contributions:

1. **A Hierarchical Game Theoretic Grid Model.** We consider that to manage the scalability of a Grid, a hierarchical structure must be used. Essentially, the hierarchy consists of three levels: the global scheduling level, the intersite level, and the intrasite level. We believe that this hierarchical structure matches well with the physical administrative structure of Grid sites.

   Based on this hierarchy, we introduce three different game theoretic scenarios: the intrasite job execution game, the intrasite bidding game, and the intersite bidding game. In this paper, we focus on the intrasite job execution game. The other games, and most importantly, the interplay among the three games, are presented elsewhere [26].
2. **Mathematical Analysis of the Intrasite Job Execution Game.** We first propose a novel but realistic utility function for each participating machine within a Grid site. We then formally derive the equilibrium strategies and the optimal strategies. Based on these analytical results, we design algorithms for the machines to achieve a high utility as well as high performance, despite the fact that the machines are selfish.

- *Y.-K. Kwok is with the Department of Electrical and Electronic Engineering, University of Hong Kong, Room 604, Chow Yei Ching Building, Pokfulam Road, Hong Kong. E-mail: ykwok@hku.hk.*
- *K. Hwang is with the Internet and Grid Research Laboratory, University of Southern California, Los Angeles, CA 90089. E-mail: kaihwang@usc.edu.*
- *S. Song is with Oracle Corporation.*

Manuscript received 1 Dec. 2004; revised 26 Feb. 2006; accepted 26 June 2006; published online 6 Jan. 2007.
Recommended for acceptance by X. Zhang.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-0288-1204.
Digital Object Identifier no. 10.1109/TPDS.2007.1013.

3. **Extensive Performance Evaluation of the Model.** We conducted extensive simulations to study the behaviors of the Grid under different strategies: heterogeneous strategies, Nash strategies, and optimal strategies. Specifically, based on a well-known practical scheduling algorithm, namely the MinMin algorithm [8], we studied the utility and job execution performance (in terms of makespan and slowdown ratio) of the Grid system under a wide range of parameters. Our performance evaluation results can serve as a valuable reference for designing appropriate strategies in a practical Grid.

The rest of the paper is organized as follows: Section 2 presents a brief review of related work. In Section 3, we describe our proposed hierarchical Grid model and the associated game theoretic research problems. We then describe in detail our modeling and analytical formulations of strategies for the intrasite job execution game in Section 4. Section 5 contains a detailed discussion about our simulation setup and the parameters used. We present the extensive simulation results and our interpretations in Section 6. The last section concludes the paper by suggesting some future research directions.

2 Related Work

Recently, we have witnessed an intensive interest in using game theoretic and market-oriented approaches in the analysis and design of distributed computing and networking algorithms [4], [20], [30], [35], [37], [48]. More notable examples include TCP congestion control [1], [2], routing [3], [7], [41], [42], bandwidth pricing [11], [47], contents delivery [16], file sharing [38], wireless caching [48], etc.

There have also been some recent results on game theoretic job allocation and scheduling reported in the literature [33]. Regev and Nisan [39] suggested the so-called POPCORN market for trading online CPU time among distributed computers. In their system, a virtual currency called "popcoin" was used between buyers and sellers of CPU times. The social efficiency and price stability were studied using the Vickrey auction theory [4], [39]. We believe that the major drawback of their approach is that some form of concrete currency is needed, and that is a system feature that we think would not be practical in a real-life situation where there is a tremendous number of machines involved. Similar approaches were also proposed by other researchers [9], [13], [15], [45].

Wolski et al. [46] proposed a model called G-Commerce, in which computational resources among different Grid sites are traded in a barter manner. The efficiency of two different economic models—commodities markets and auctions—were studied by simulations. They concluded that a commodity market is a better choice for controlling Grid resources compared with auctions. Ghosh et al. [19] study the load balancing issues in a mobile computational Grid. In their model, there is a *wireless access point* (WAP) which mediates the requests from different mobile devices constituting the Grid. Using the *Nash Bargaining Solution* (NBS) [31], they devised a framework for unifying network efficiency, fairness, utility maximization, and pricing. However, an explicit payment scheme must be enforced in the system.

Larson and Sandholm [27] pioneered the consideration of the computation cost involved in determining the valuations that are essential inputs to the auction system. They defined the notion of "miscomputing ratio," which characterizes the impact of selfishness on the efficiency of the auction. Nisan and Ronen [33], [34] formally defined the job allocation in a distributed system using a truthful mechanism framework. Here, "truthfulness" is a concept about the revealing of real information about each participant. That is, under a truthful mechanism, each player will not lie about its state. They proved that the MinWork mechanism—in which the "payment" to the job executing computer is the "time" valuation proposed by the second best computer—is a strongly truthful approximation mechanism.

Grosu et al. [21], [22], [23] recently performed some pioneering work in that they designed a load balancing system based on the truthful mechanism (a key component in Vickrey-Clarke-Groves (VCG) mechanisms [24], [36]), in which each computer optimizes its "profits" by considering the payment and cost involved in handling a job. They used the overall expected response time as the "social cost" of the whole system. The optimization of this metric reduces to a nonlinear optimization problem. With the help of a Lagrangian solution based on the Kuhn-Tucker conditions, Grosu et al. derived a set of useful conditions that are to be enforced algorithmically by each individual computers in a distributed manner. An imperfection in their work is that the physical meaning of the payment functions is unclear. Furthermore, in an open Grid environment, addressing load balancing alone is not enough.

Volper et al. [44] proposed a game-theoretic middleware called GameMosix. Selfish behaviors are modeled by "friendship relationships" in that computers will help each other only when they have established friendship relationships before. Quantitatively, a unit of friendship is accumulated if a computer takes a job from another computer. Sender and receiver algorithms were then devised to handle remote job executions based on friendship values.

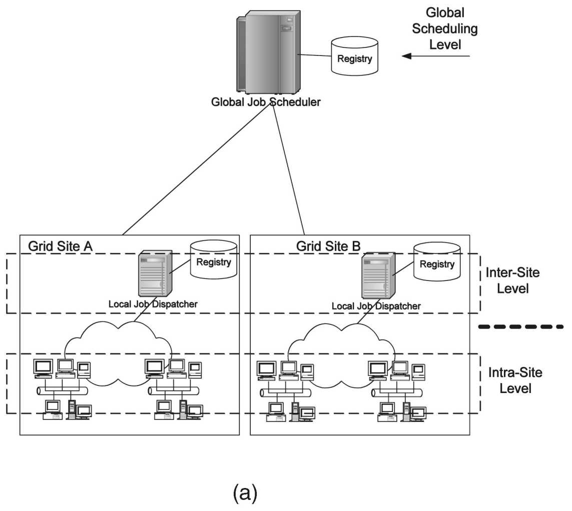
With reference to the above-mentioned related work, our proposed models and analytical formulations are novel in that we consider the hierarchical relationships among individual computers in a gigantic computational Grid. Our work is also the first of its kind in investigating the selfishness issues *within* a Grid site [26].

3 A Hierarchical Semiselfish Grid Model

As mentioned in Section 1, the ultimate scale of a computational Grid is gigantic, and, thus, the Grid, pretty much like the Internet itself, will cross organizational and national boundaries. An open question is that of how such a gigantic distributed computing platform, which is likely to be composed of hundreds of thousands of processors, is to be structured and maintained. We believe that a hierarchical structure, as depicted in Fig. 1a, is the only feasible solution.

In our study, we envision that each "Grid site" is not going to be a single computer, but rather a network of computers,[1] each of which is a cluster of machines or a tightly coupled, massively parallel machine. Thus, eventually, we may have hundreds of Grid sites, each of which

1. Throughout this paper, we use the term "computer" and "machine" to refer to a monolithic autonomous computing platform that possibly consists of multiple CPUs.



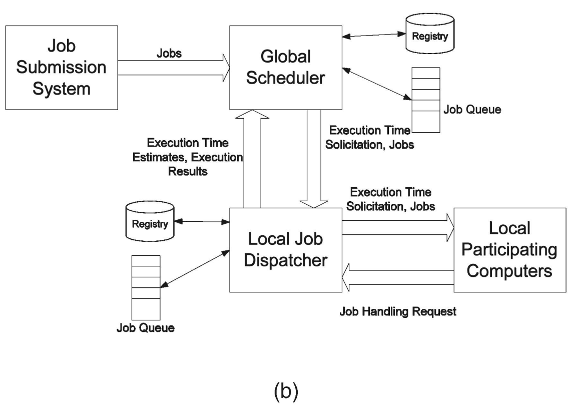

Job Submission System
Jobs
Global Scheduler
Registry
Job Queue
Execution Time Estimates, Execution Results
Execution Time Solicitation, Jobs
Execution Time Solicitation, Jobs
Registry
Local Job Dispatcher
Local Participating Computers
Job Queue
Job Handling Request
(b)

Fig. 1. System model of an open Grid computing platform. (a) The hierarchical structure of an open Grid. (b) The control flow in open Grid job scheduling.

consists of tens of multiprocessors (i.e., clusters and parallel machines). Indeed, such a structure, again resembling the Internet itself, closely matches the "administrative" structure of computing resources in organizations.

For instance, the computer science department of a university might own a large cluster of PCs, the electrical engineering department might possess another, and the physics department might manage a massively parallel supercomputer. Yet, all these computing resources participate in the global Grid community according to the university's mandate. Thus, at the intrasite level, the participating computers, each of which is autonomous, form a *federation*. At the intersite level, the participating Grid sites form another level of federation.

With the hierarchical structure shown in Fig. 1a, there are also two levels of job scheduling and dispatching, depicted in Fig. 1b. Specifically, the job submission system, which is implemented as a global middleware, channels user submitted jobs to the global scheduling system. We envision that such a job submission middleware can be easily constructed using Web services tools (e.g., WSDL and SOAP messages [5]). Equipped with a global Grid processing resources registry (which, again, could be based on the UDDI protocol), the global scheduler performs job allocation according to a certain scheduling algorithm.

Most importantly, at the intersite level, the scheduler has only the knowledge of the processing capability of each Grid site as a whole, without regard to the details within the site. In this manner, the scalability of scheduling at the global Grid level can be efficiently handled. Furthermore, this scheduling model conforms well to the administrative structure of the Grid community in the sense that the global scheduler probably should not "micromanage" the execution of jobs down to the machine level. The global scheduler makes use of the "capability parameters" supplied by the Grid sites as the inputs to the scheduling algorithm. These capability parameters are, in turn, mediated by the local job dispatcher at each Grid site based on its information about the local participating machines.

As described above, our hierarchical model, while capturing the realistic administrative features of a real-life, large-scale distributed computing environment, is also generic in nature. Indeed, this federation-based Grid model opens up a large variety of interesting research issues. First, any efficient online job scheduling algorithm can be used. Furthermore, it is important to study how the various parameters are generated and communicated. Indeed, from the hierarchical model, we can formulate three different game theoretic job allocation and execution problems:

1. **Intrasite Job Execution Strategies.** This problem concerns the strategies of the participating computers inside a Grid site. Specifically, although the various computers participate in the making up of the Grid site, each individual computer is selfish in that it only wants to execute jobs from local users but does not want to contribute to the execution of remote jobs.

   For example, even though a cluster of PCs in the computer science department is designated as a member computer of a university-based Grid site, the cluster's administrators and/or users may still prefer to dedicate the computing time to process local requests as much as possible. However, if every participating computer does not contribute, the Grid site as a whole will fail to deliver its promise as a serving member of the Grid community, thereby defying the original motive of forming the Grid. Thus, one of the participating computers eventually has to take up a job assigned to the Grid site by the global scheduler.

   This problem is interesting in that we need to determine how a participating computer should formulate its job execution strategy so as to maximize its own utility (i.e., execute more local jobs) in a selfish manner without rendering the whole site nonoperational. We focus on this problem in this paper.
2. **Intrasite Bidding.** This problem concerns the determination of the advertised "execution capabilities"

TABLE 1
Notation in the Game-Theoretic Formulation

Symbol	Definition
$T(J_k)$	Execution time of job J_k
a_k	Serial fraction of job J_k
b_k	Parallel fraction of job J_k
U_i	Utility function of machine i (i.e., player i)
s_i	Degree of cooperation (DoC) of machine i (i.e., the *mixed strategy* [36] of player i)
P_i^t	Total number of processors available at machine i
P_i^r	Number of processors used for executing a job at machine i
τ	Duration of a job dispatching round
P_o	Fixed overhead component of P^r
Q	Variable component of P^r
P	The minimum number of processors used to finish a job
P_w	Extra number of processors needed for a job after τ units of time
α	Selfishness penalty factor
R_j	Reputation Index (RI) of Grid site j
n	Number of players at each Grid site
m	Number of Grid sites
W_j^1	Workload accepted in the first round by site j
W_j^2	Workload accepted in the second round by site j
W_j^r	Workload rejected eventually by site j
β_1, β_2, γ	Weighting factors in updating RI

for jobs submitted to the global scheduler. Recall that, for the scheduler to allocate jobs using a certain scheduling algorithm, it needs to know all the sites' execution capabilities—in our study, these are modeled as the execution times needed for the pending jobs. To determine the execution time needed for a certain job, within a Grid site each participating computer can make a "declaration"—a notification to the local job dispatcher specifying the time needed to execute the job.

The local job dispatcher can then "moderate" all these declarations to come up with a single value to be sent to the global scheduler. For example, if the local job dispatcher is aggressive in job execution, it could use the "minimization" approach—taking the minimum value of the declarations from all the member computers. On the other hand, a conservative approach is to perform "maximization"—taking the maximum value instead.

This problem is also interesting in that we need to determine, possibly using auction theory, the best "bidding" (i.e., making execution time declarations) strategies for each member computer. Specifically, we need to determine whether *truthful revelation* is the best approach in the bidding process.

3. **Intersite Bidding.** Similar to the intrasite situation, at the intersite level, the various local job dispatchers also need to formulate game theoretic strategies for computing the single representative value of the job execution time to be sent to the global scheduler.

Another exciting avenue of research is to study the interplay of these three games, i.e., how the selfishness of each individual computer affects the intrasite bidding, which, in turn, will impact the intersite bidding in a complicated manner.

Indeed, different combinations of the above games will result in different Grid structures. For a *semiselfish* Grid, the intrasite games are noncooperative while the intersite game is cooperative. This model fits most present-day Grid situations because a Grid is usually formed after some cooperative negotiations at the organization level. However, the individual machines operated by bottom-level departments may not cooperate with each other. For a *fully selfish* Grid, the games are assumed to be noncooperative at all levels. This model is the most general model. Finally, the *ideal* Grids are modeled by cooperative games at all levels.

In this paper, we focus our formulation, analysis, and results on the first problem introduced above. Specifically, to simplify the model, we assume that, for the intersite and intrasite bidding processes, truthful mechanisms [28] are used. In subsequent papers, we will present our results on the untruthful revelation of participating machines within each Grid site and the intersite auction problem.

4 Semiselfish Mechanisms and Mixed Strategies

In this section, we present our analytical formulation of the game theoretic framework for the intrasite job execution mechanism. We first describe the job model and execution policies. We then formulate the two-player case, followed by the general n-player case. Game theoretic algorithms induced by our analysis are formalized at the end of this section. Table 1 summarizes the notation used throughout this paper.

In our game theoretic study of Grid job scheduling, we consider a class of malleable jobs [25], each of which has the following execution time model: $T(J_k) = a_k + \frac{b_k}{P}$, where a_k is the serial portion of the job J_k and b_k is the parallel portion that can be shortened (hence, malleable) if more processors are available. That is, the execution time decreases in a linear manner as the number of processors allocated to the job increases. Thus, we assume that each job is a parallel application that can be executed using multiple processors. Consequently, the "cost" for each participating computer (e.g., possibly a cluster of PCs) in executing a job is the number of processors, denoted by P, devoted to the job during its execution time period.

To model the "selfish" behavior of each participating computer (i.e., each player) in a Grid site j, we propose the following *utility function*:

$$U_i = \frac{P_i^t}{P_i^r}, \tag{1}$$

where U_i is the utility of player i, P_i^t is the total number of processors of player i, and P_i^r is the total number of processors it used for a remote job. Here, we assume that $P_i^r > 0$ because there is always some overhead for a computer to participate in the Grid (e.g., the need to expend some processing resources to monitor the Grid status, or to advertise its capabilities, and so on).

We believe that this simplistic selfish utility function is able to model a real-life situation. Essentially, each machine is selfish in the sense that it does not want to contribute to the Grid community if possible by minimizing the utilization of the machine by remote jobs. Thus, the machine can spend more time to handle local jobs. This is realistic even in current Grid computing environments because, although on the institutional or departmental level machines are "assigned" to contribute in a Grid, the local users may not care too much about this and would simply like to utilize the machines as much as they can.

However, the Grid site as a whole would like to maximize its *Reputation Index* (RI), which quantifies the contributions of the site (intuitively, a higher RI would lead to a better reputation of the organization as a whole). Specifically, the RI value R_j will be incremented if an assigned job is successfully executed at site j and decremented if the job fails (the failure of a job will be elaborated below). In the following, we propose our novel formulation of this assigned job execution mechanism as a noncooperative game [32] to study the dynamics of the conflicting goals of the selfish machines and the Grid site as a whole.

In our model, we assume that, after a job is assigned to a Grid site, the job is associated with an *execution deadline* in that the job can be held in the job queue at the local job dispatcher for a certain period of time. Let us denote this time by 2τ. We elaborate the rationale behind this policy in Section 4.2. Thus, in the execution game, there are two rounds of "moves." Within each round, each computer acts according to its selfish strategy and can choose to either ignore the job or take it up.

We consider *mixed strategies* [31], [36] in our study. Essentially, each computer uses a probabilistic "wait-and-see" approach—try to avoid the work by waiting, with a certain probability, for some other computer to take it up. Now, consider that if a job is taken up immediately after it is assigned, the amount of resources occupied is given by $P^r = P_o + Q$, where P_o is the fixed overhead component of resources and Q is a variable component that depends on how much time is left for the job (here, the player index indicated by the subscript is dropped for clarity).

Specifically, if the job is taken up immediately after assignment, then $Q = P$, where P is the number of processors needed in order to finish the job using the amount of time advertised by the Grid site to the global scheduler. On the other hand, if the job is executed after one round (i.e., τ units of time) because no computer takes it up in the first round, then the number of processors involved becomes $P^r = P_o + Q = P_o + P + P_w$. That is, the waiting time τ has to be compensated by "throwing in" P_w more processors to the job so that the deadline of the job can still be met. Let us consider a simple scenario first—only two computers are involved.

4.1 The Two-Player Game

Let us consider two participating computers, denoted by M_1 and M_2, having mixed strategies s_i, where $0 \leq s_i \leq 1$ for $i = 1, 2$. Here, s_i, called the *degree of cooperation* (DoC) in our study, is the probability (i.e., the mixed strategy) that the assigned job is taken by computer M_i. Now, in the first round, if M_1 chooses not to take up the job, there are two possible outcomes: 1) M_2 takes it up, or 2) M_2 also does not take it up. Suppose that, after the first round, if the job is not taken up, M_1 will take it up with probability 1. As such, we have

$$Q = s_1 P + (1 - s_1)(1 - s_2)(P + P_w). \tag{2}$$

By symmetry, a similar expression can also be derived for M_2. Suppose $P_w = \alpha P$, where $0 < \alpha < 1$ (i.e., τ is not a long period of time with respect to the job's execution time elaborated in Section 4.2). Here, α is called the *selfishness penalty factor* because it quantifies the amount of extra resources incurred should the machine refuse to take up the job earlier. Differentiating U_1 with respect to s_1 gives

$$\frac{\partial U_1}{\partial s_1} = -\frac{P^t P}{(P^r)^2}(s_2(1+\alpha) - \alpha). \tag{3}$$

Depending on the value of s_2, $\frac{\partial U_1}{\partial s_1}$ takes on different values:

1. $s_2 < \frac{\alpha}{1+\alpha} \Longrightarrow \frac{\partial U_1}{\partial s_1} > 0$: M_1's best "execution strategy" is "always do it," i.e., $s_1 = 1$.
2. $s_2 > \frac{\alpha}{1+\alpha} \Longrightarrow \frac{\partial U_1}{\partial s_1} < 0$: M_1's best "execution strategy" is "always ignore," i.e., $s_1 = 0$.
3. $s_2 = \frac{\alpha}{1+\alpha} \Longrightarrow \frac{\partial U_1}{\partial s_1} = 0$: M_1's best "execution strategy" is either of the two possible actions, i.e., it is indifferent.

Theorem 1. *The strategy combination* $(s_1, s_2) = (\frac{\alpha}{1+\alpha} \frac{\alpha}{1+\alpha})$ *achieves a* Nash Equilibrium *[31], [36] in the two-player game in that no player can benefit by unilaterally deviating from this strategy combination (i.e.,* $U_i(s_i^*) \leq U_i(\frac{\alpha}{1+\alpha})$ *for any* $s_i^* \neq \frac{\alpha}{1+\alpha}$*).*

Proof. The theorem is true because $Q = P$ by (4) and, thus, U_i does not depend on the value of s_i (for $i = 1, 2$) under this symmetric combination. Thus, any deviation of s_i from $\frac{\alpha}{1+\alpha}$ would not affect U_i. □

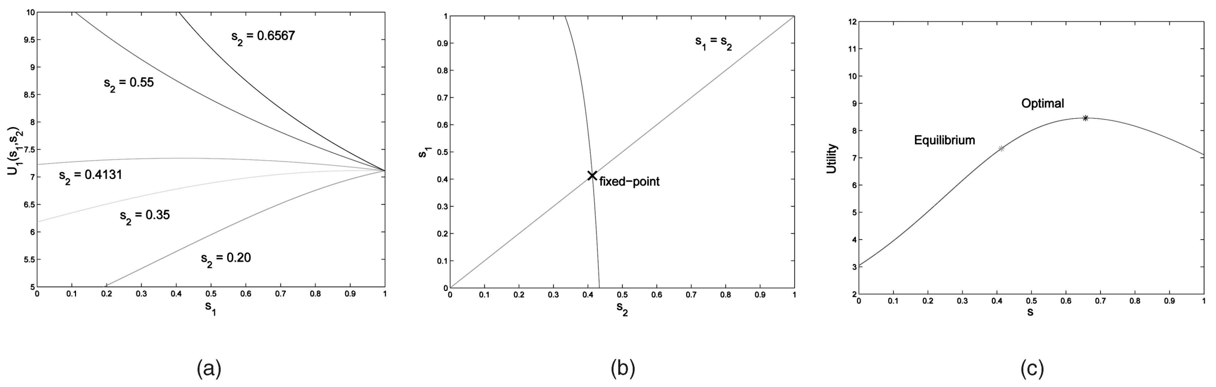


Fig. 2. Relationships among the utility function U_1 and DoC s_1 of machine M_1, and the DoC s_2 of machine M_2. (a) $U_1(s_1, s_2)$ versus s_1 with various values of s_2. (b) s_1 versus s_2. (c) Utility versus s.

It should be noted that deviating from the Nash equilibrium strategy does not make the utility worse. In fact, the only requirement of the Nash equilibrium is that unilateral deviation does not lead to a better utility. However, this equilibrium, albeit unique, is a weak one (i.e., unstable), and the solution is degenerated [36], [40] in that each player i can choose any strategy provided the other player fixes its strategy to be $\frac{\alpha}{1+\alpha}$.

Now, let us consider the case where each of the two computers is patient enough to wait for one more time interval τ (i.e., the absolute deadline) before committing itself to take up the job. Thus, the variable component of the number of processors involved becomes

$$Q = s_1P + (1 - s_1)(1 - s_2)[s_1P + P_w + (1 - s_1)(1 - s_2)(P + 2P_w)]. \tag{4}$$

The terms in the square brackets account for the total cost of the two-round waiting. With some sample numerical values (i.e., $P_{\text{total}} = 256$, $P_o = 4$, $P = 32$, and $\alpha = 0.5$), Fig. 2a shows the relationships among U_1, s_1, and s_2. We can draw a number of conclusions:

- If M_1 always takes up the job, i.e., $s_1 = 1$, its utility is independent of M_2's strategy s_2.
- The maximum value of the utility increases with s_2.
- For small values of s_2, the optimal strategy for M_1 is to always take up the job.
- For large values of s_2, the optimal strategy for M_1 is to always wait.
- For some values of s_2, the optimal strategy for M_1 is the interior of the strategy space, i.e., $s_1 \in (0, 1)$.

Furthermore, by performing partial differentiation with respect to s_1, we can see that the best execution strategy with variable s_2 for M_1 is

$$s_1 = \frac{2(1 + 2\alpha)(1 - s_2)^2 - (1 - \alpha)(1 - s_2) - 1}{2(1 + 2\alpha)(1 - s_2)^2 - 2(1 - s_2)}. \tag{5}$$

If we take $s_1 = s_2$ and $\alpha = 0.5$ so as to solve this cubic equation, we can get only one real root: $s_1 = s_2 = 0.4131$. Indeed, Fig. 2b shows a plot of (5) when $\alpha = 0.5$. We can see that there is only one *fixed point* solution within the feasible strategy space, i.e., $s_1 = s_2 = 0.4131$, which is the unique equilibrium strategy of the game.

However, this Nash equilibrium is again suboptimal, as is evident by the following analysis: To obtain the global optimal utility value, let us take $s_1 = s_2 = s$ in (4) and substitute it into the utility function U (here, the subscript is dropped because of symmetry). We then consider the case of setting $\frac{\partial U}{\partial s} = 0$ under this "enforced" symmetrical strategy combination. We have

$$4(1 + 2\alpha)s^3 - 3(3 + 8\alpha)s^2 + 2(4 + 13\alpha)s - 2(1 + 5\alpha) = 0. \tag{6}$$

Solving this cubic equation with $\alpha = 0.5$, we also get only one real root: $s = 0.6567$. Fig. 2c shows the variation of the utility function with symmetrical strategies (i.e., $s_1 = s_2$). We can see that the optimal strategy is $s_1 = s_2 = \hat{s} = 0.6567$, while the Nash equilibrium strategy is $s_1 = s_2 = 0.4131$. Thus, the Nash equilibrium utility is *Pareto inefficient* [36], which is a common characteristic in noncooperative game models. Fortunately, under our hierarchical scheduling model, we can make use of the local job dispatcher to guide the players (i.e., the participating computers) to use the optimal strategy. This is elaborated in Section 4.4 below.

It should be noted that "optimality" is defined with respect to the utility of the machines. Thus, an optimal strategy is one that "satisfies" optimally the "selfishness" of the local machines. Consequently, according to the "self-interest" of the local machines, each machine would then be willing to use the "optimal" strategy values for local job execution decision making.

In the above analysis, we assume that there are only exactly two rounds of moves. We can easily extend the analysis to the general case where there are an infinite number of rounds. In this general case, the variable resource component Q of M_1 is given by

$$Q = P\left(\frac{\alpha\Omega}{(1 - \Omega)^2} + \frac{s_1}{(1 - \Omega)}\right), \tag{7}$$

where $\Omega = (1 - s_1)(1 - s_2)$. Here, again by partial differentiation with respect to s_1, we find that the equilibrium strategy of M_1 is given by

TABLE 2
Some Sample s_i Values of the Nash and Optimal Strategies

α	$n=2$		$n=4$		$n=6$		$n=8$		$n=10$	
	Nash	Optimal	Nash	Optimal	Nash	Optimal	Nash	Optimal	Nash	Optimal
0.1	0.2572	0.4516	0.1121	0.2656	0.0720	0.1986	0.0531	0.1621	0.0421	0.1386
0.2	0.3384	0.5361	0.1508	0.3210	0.0975	0.2411	0.0721	0.1971	0.0572	0.1686
0.3	0.3936	0.5890	0.1783	0.3572	0.1157	0.2691	0.0857	0.2202	0.0681	0.1884
0.4	0.4360	0.6276	0.2002	0.3846	0.1303	0.2903	0.0967	0.2377	0.0768	0.2035
0.5	0.4707	0.6579	0.2185	0.4066	0.1426	0.3075	0.1059	0.2520	0.0843	0.2157

$$s_1 = \frac{(\alpha-1)(s_2)^2 - 3\alpha s_2 + 2\alpha}{(\alpha-1)(s_2)^2 + (1-2\alpha)s_2 + \alpha}. \tag{8}$$

Solving (8) with $s_1 = s_2$ (i.e., symmetric equilibrium) and $\alpha = 0.5$, we have $s_1 = s_2 = 0.4707$. On the other hand, the optimal strategy for the "enforced" symmetric case is given by the solution of the following equation:

$$s^4 - 2(\alpha+1)s^3 + 6\alpha s^2 - 8\alpha s + 4\alpha = 0. \tag{9}$$

For $\alpha = 0.5$, the only legitimate root of this equation is $s = 0.6579$. We can see that the Nash and optimal strategy values for the 2-round case are both slightly smaller than those of the ∞-round case. Nevertheless, as argued from a practical perspective in Section 4.2 below, in our model the system would consider a job as rejected by the assigned site if it was not taken up after two rounds. Thus, the job execution game would not be played indefinitely for each allocated job at a site.

4.2 The Two-Round Policy

In the above analysis, the selfishness penalty factor α is defined as $\alpha = \frac{P_w}{P}$. It can be shown that $\frac{\alpha}{1+\alpha} = \frac{\tau}{\Gamma}$, where Γ is the execution time for the parallel fraction of the job. Here, first of all, we can see that, with a fixed value of α, τ takes on different values for different jobs. Secondly, as α gets larger, τ becomes a larger fraction of Γ.

Indeed, with $\alpha = 0.1$ (i.e., 10 percent more processors are needed to finish the job after each round), τ is equal to 9.1 percent of Γ. On the other hand, with $\alpha = 0.5$ (i.e., 50 percent more processors are needed to finish the job after each round), τ is equal to 33.3 percent of Γ. Thus, with $\alpha = 0.5$, after two rounds of waiting, 66.7 percent of originally useful execution time is wasted and 200 perecent of the originally needed resources are needed to finish the job. In view of this, it is deemed to be reasonable to consider that the job is rejected if it is not taken up by any player after two rounds. As detailed in Section 4.4, a rejected job is rescheduled by the global scheduler to a possibly new site in the next batch.

4.3 The *n*-Player Game

We can easily extend the general case (i.e., with an infinite number of rounds) of the two-player game to the n-player scenario—there are n participating computers in a Grid site. Specifically, we have the following theorem:

Theorem 2. *The variable component of the number of processors Q involved for player i in the n-player game is*

$$Q = P\left(\frac{\alpha\Lambda}{(1-\Lambda)^2} + \frac{s_i}{(1-\Lambda)}\right), \tag{10}$$

where $\Lambda = \prod_j^n (1-s_j)$. Thus, the symmetric Nash equilibrium strategy s^* is given by

$$s^* = \frac{(1-\alpha)\xi^2 - (\alpha+2)\xi + 1}{(1-\alpha)\xi^2 - \xi}, \tag{11}$$

where $\xi = (1-s^*)^{n-1}$.

Proof. We provide an outline of the proof here. We can derive (10) by generalizing (7) using mathematical induction on n. We can then derive (11) by partial differentiation of the utility function with respect to one of the n mixed strategy variables s. □

Furthermore, the following theorem formalizes the existence of an equilibrium strategy for the n-player intrasite job execution game:

Theorem 3. *There exists an equilibrium strategy for the n-player intrasite job execution game.*

Proof. First of all, each player's strategy space, $\{s_i\} \in \Re^1$, is nonempty, convex, and compact. Secondly, the utility function, U_i, is continuous on $\Pi_{1\le j\le n}\{s_j\} \subset \Re^n$. Furthermore, the best execution strategy mapping is single-valued. Thus, we can conclude that there exists an equilibrium strategy for the n-player game [36]. □

Although Theorem 3 does not rule out the possibility of more than one equilibrium, we observe, from simulations, that the equilibrium strategy (11) appears to be unique. On the other hand, the optimal symmetric strategy $\hat{s}$ is given by the following theorem:

Theorem 4. *The optimal symmetric strategy $\hat{s}$ is given by the unique legitimate real root*[2] *of the following equation:*

$$\begin{aligned} 1 + [(1-\alpha n) + (n-2)s](1-s)^{2n-1} \\ -[(\alpha n + 2) + (n-2)s](1-s)^{n-1} = 0. \end{aligned} \tag{12}$$

Proof. This can be easily shown by generalizing (6) using mathematical induction on n. □

Table 2 lists some sample s_i values of the Nash equilibrium and optimal strategies. As can be seen, the strategy values get smaller with a larger number of players or with a smaller selfishness penalty factor.

2. When n is odd, there is only one real root; when n is even, there are three real roots but only one of them is within the (0, 1) interval.

4.4 Algorithms for Intrasite Game Players

With the formulations described above, we can formalize the algorithms for each participating computer and the local job dispatcher.

Using Algorithm 1, the local job dispatcher continuously communicates with the global scheduler (e.g., via some SOAP messages) to check if the global scheduler is soliciting execution time estimates for pending jobs. If so, the local job dispatcher will in turn solicit such estimates within its jurisdiction. Here, we assume that the local job dispatcher uses a conservative approach in that it uses the maximum value of the local estimates as the representative value for the global scheduler. Determining an "accurate" representative value (with respect to the actual resulting execution time) is a challenging problem outside the scope of this paper. Finally, we assume that the global scheduler just uses the *earliest completion time* (ECT) algorithm, i.e., it assigns the job to the site that can finish the job at the earliest time.

Algorithm 1 Local Job Dispatcher

```
1: if global scheduler solicits "execution time estimates"
   for a job, $J_k$ then
2:    Solicit execution time estimates from all
      participating computers, $T_i(J_k)$;
3:    Return $\max\{T_i(J_k)\}$ to the global scheduler;
4: end if
5: if a job is assigned to the site then
6:    Check the currently active number of participating
      computers, $n$;
7:    Broadcast the value of optimal strategy $\hat{s}$ according
      to (12) to all the participating computers;
8:    for $\mathrm{round} = 1, 2$ (i.e., a total of $2\tau$ units of time) do
9:       if a computer $M_i$ takes up the job (ties are broken
         randomly) then
10:         Send the job to $M_i$;
11:         Declare that the job is unavailable;
12:      end if
13:   end for
14:   if no computer takes up the job then
15:      Declare that the job fails;
16:   end if
17: end if
```

If there is a job assigned to the Grid site, the local job dispatcher will then coordinate the intrasite job execution game. Specifically, to enforce the participating sites to use the optimal strategy, it first determines the value of $\hat{s}$ based on the current number of active participants (i.e., n) in the site. Then, it waits to see if, within two rounds (i.e., 2τ units of time), the job is taken up by some participant. If so, the job is handed over to the volunteer; otherwise, the job is declared as failed and the global scheduler is notified. Consequently, the global scheduler needs to reschedule the job in the next batch, inevitably leading to a longer overall makespan for the whole set of jobs. Furthermore, the global scheduler will deduct the RI value of the concerned Grid site, which in turn will hurt the reputation of the Grid site. Corresponding to Algorithm 1, each participating machine uses Algorithm 2 to play the intrasite game.

Algorithm 2 Participating Machine (Player i)

```
1: if local job dispatcher solicits "execution time
   estimates" for a job, $J_k$ then
2:    Return the local estimate of $T_i(J_k)$ using the value of
      desired number of processors $P$ to the local job
      dispatcher;
3: end if
4: if a job is assigned to the site then
5:    Receive the broadcast value of optimal strategy $\hat{s}$
      according to (12) from the local job dispatcher;
6:    for $\mathrm{round} = 1, 2$ (i.e., a total of $2\tau$ units of time) do
7:       if $\mathrm{Random} < \hat{s}$ then
8:          Declare that this machine takes up the job;
            {Random is a random number between 0 and 1
            generated by player $i$}
9:          Execute the job;
10:      end if
11:   end for
12: end if
```

5 Simulation Setup

This section describes the experimental setup in our performance evaluation of the three strategies: *Optimal*, *Nash*, and *Random*. The Optimal strategy is based on Algorithms 1 and 2. The Nash strategy is also based on the same algorithms but with the s_i values computed according to (11) instead of (12). The Random strategy models the situation where all the players are completely uncoordinated and use heterogeneous s_i values randomly generated from a uniform distribution in the range $[0, 2s_{\mathrm{Nash}}]$.

A hierarchical semiselfish Grid infrastructure is simulated using a discrete event-driven simulator. At the intersite level, m cooperative Grid sites are simulated. At the intrasite level, a variable number of selfish players are simulated for each site, as governed by n, which is the mean number of players.

Job arrivals are modeled by a random Poisson distribution. Jobs are submitted to a centralized job scheduler. Each site reports their required processing times, in a "truthful" manner [36], to the centralized scheduler. The well-known Min-Min scheduling heuristic [8] is used for the intersite scheduling. Specifically, for each job, the Grid site that gives the earliest *Expected Time to Completion* (ETC) is identified first. Then, the job that has the minimum earliest ETC is selected and then assigned to the identified Grid site.

According to the two-round policy, a job may be rejected repeatedly without being executed even after multiple scheduling batches. Thus, we incorporate a policy in our simulator that enforces a selected Grid site to execute a job which has been rejected three times.

5.1 NAS Trace Workload

We use three months of accounting records for the 128-node iPSC/860 located in the *Numerical Aerodynamic Simulation* (NAS) Systems Division at NASA Ames Research Center [29]. This trace contains 92 days (7,948,800 seconds) data, gathered in 1993. There are 16,000 jobs in the whole trace. For testing the job execution performance under a high-throughput Grid environment, the 92 days trace data is

TABLE 3
Simulation Parameters

Parameter	Value
Number of jobs K	NAS: 16,000; PSA: 10,000
Number of sites m	NAS: 12; PSA: 10, 15, 20 (default), 30, 40
Job arrival rate	NAS: specified by the trace; PSA: 1 job every 100 seconds
Job load level	NAS: fixed; PSA: 20 levels (0–300,000)
Site architectures	NAS: 8×8 nodes and 4×16 nodes; PSA: 10 levels
Mean number of players n	4, 6, 8 (default), 10, 12
Selfish penalty factor α	0.1, 0.2, 0.3 (default), 0.4, 0.5
Reputation Index (RI)	initial: 0.5; weighting factors: $\beta_1 = 1, \beta_2 = 0.5, \gamma = 1$

proportionally squeezed to 46 days. We map the 128 nodes to 12 Grid sites; 4 of the sites each contain 16 nodes, and the other 8 sites each contain 8 nodes. Our simulations are based on the arrival time, job size, and runtime data provided by the trace. This trace was sanitized to remove the user-specified information and preprocessed to correct for system downtime [29].

5.2 PSA Workload

The *parameter sweep application* (PSA) model has emerged as a "killer application model" for composing high-throughput computing applications for processing on global Grids [12]. The parameter sweep application is defined as a set of independent sequential jobs (i.e., no job precedence). The independent jobs operate on different data sets. A range of scenarios and parameters to be explored are applied to the program input values to generate different data sets. The execution model essentially involves processing K independent jobs (each with the same task specification, but a different data set) on M distributed sites, where K is typically much larger than M.

5.3 Simulation Parameters

Table 3 lists the key simulation parameters. We report our results for various parameters. For each parameter, the default value and their varying range are provided. The default values are used for all experiments unless otherwise specified.

In our experiments, the initial values of RI at all sites are set to 0.5. The RI at each site is then updated for every batch process. Using the following equation, the new RI value of site j is calculated from the old value plus some new input value gathered from that batch.

$$\mathrm{RI}_j^{\mathrm{new}} = \mathrm{RI}_j^{\mathrm{old}} + \left(\beta_1 \frac{W_j^1}{W_{\mathrm{total}}} + \beta_2 \frac{W_j^2}{W_{\mathrm{total}}} - \gamma \frac{W_j^r}{W_{\mathrm{total}}}\right), \qquad (13)$$

where W_{total} is the total workload processed by all sites in a batch, W_j^1, W_j^2, and W_j^r represent the workload accepted in the first round, accepted in the second round, and rejected eventually by site j, respectively. The corresponding weighting factors are β_1, β_2, and γ, respectively, which are all positive real numbers. In our study, they are set to 1, 0.5, and 1. Based on the RI updating rule above (i.e., (13)), those sites that accept more jobs (in particular, accept jobs at the first round) will increase their reputation quickly, and those sites that reject more jobs will have their RIs declining rapidly.

6 Performance Evaluation Results

In this section, we present our simulation results over the NAS workload for the three strategies: *Random*, *Nash*, and *Optimal*. We evaluate the overall system performance using the following metrics:

- *Makespan*: the largest finish time among all the jobs,
- *Turnaround Time*: the average time spent by a job in the system,
- *Slowdown Ratio*: the ratio of the average turnaround time to the average waiting time of all jobs,
- *Reputation Index*: defined in Section 4,
- *Utilization*: the fraction of resources used by remote jobs, and
- *Job Rejection Rate*: the percentage of jobs rejected by a site.

6.1 Results over NAS Workload

We first consider the results over the NAS workload. As indicated in Fig. 3, the first observation is that the Optimal strategy consistently outperforms the Random and Nash strategies by a considerable margin. For example, the Optimal strategy's performance is 5 percent to 8 percent better than the other two strategies. Furthermore, the job rejection rate of Optimal is only around 2 percent to 3 percent but those of Random and Nash are around 40 percent on average. Here, notice that the utilization results shown in Fig. 3e are for a typical Grid site in the simulation platform (denoted by S_1).

Thus, we have an important conclusion: It is not necessarily bad for the machines to behave selfishly provided that they all use the same optimal mixed strategy values s_i computed by our analysis. Another interesting conclusion is that the Nash equilibrium strategy is quite poor—almost the same as the Random strategy. Indeed, although there is no incentive for each player to deviate from the Nash equilibrium, the resulting equilibrium performance is not much different from that of a totally uncoordinated job execution scenario.

As to the effects of α, we can see that, as a larger α is used (i.e., heavier penalty), the makespan, turnaround time, slowdown, and utilization increase, while the job rejection rate decreases. This can be explicated by the fact that, as the penalty is heavier, more time is needed to compensate for the refusal of job execution. Accordingly, there is less incentive for the machines to behave selfishly.

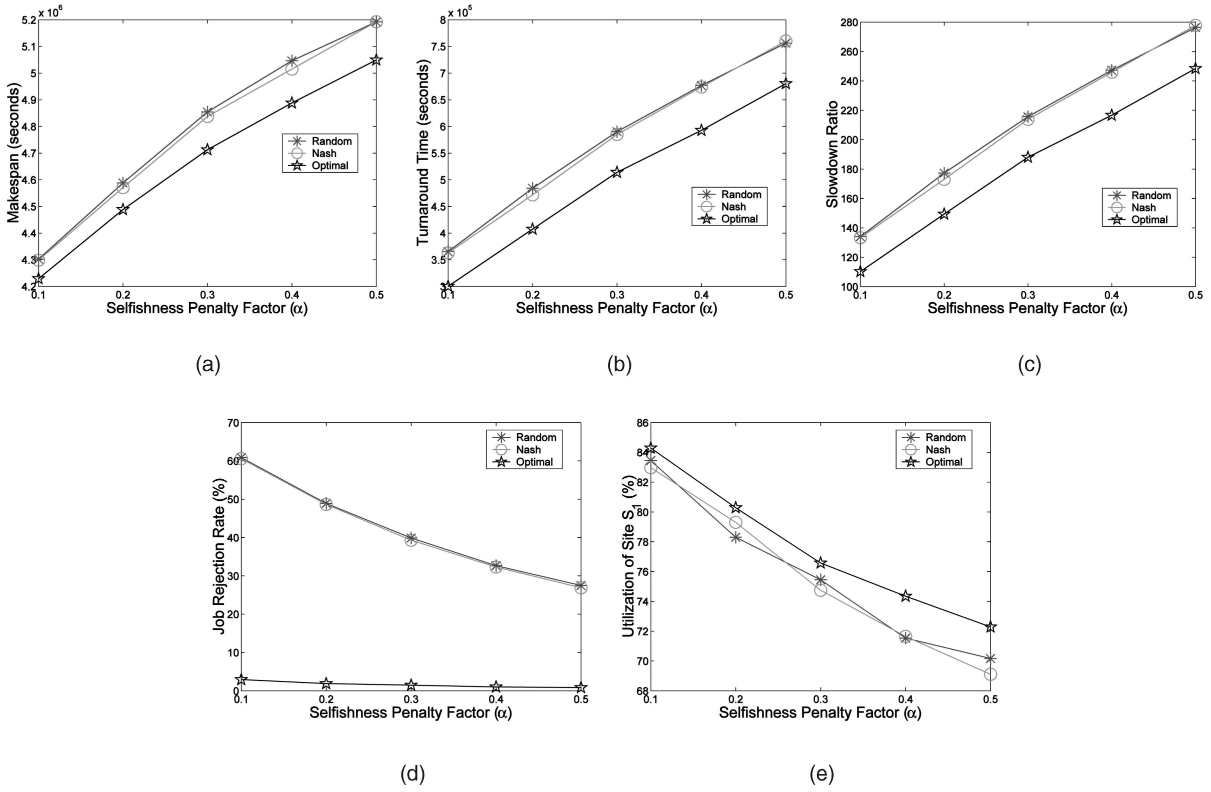


Fig. 3. The NAS performance using the three game strategies (Random, Nash, and Optimal) under various values of the selfishness penalty factor α. (a) Makespan. (b) Turnaround time. (c) Slowdown ratio. (d) Job rejection rate. (e) Utilization.

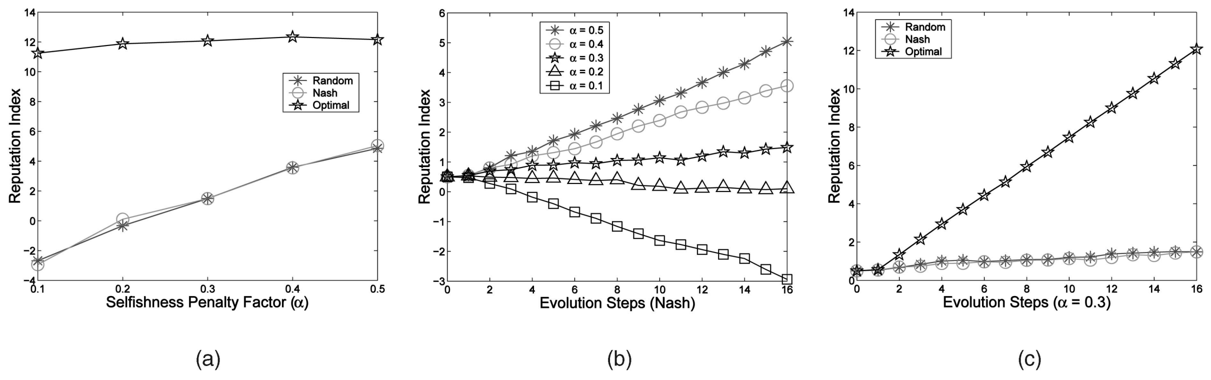


Fig. 4. The variation of the Reputation Index (RI) for the NAS workload using the three game strategies (Random, Nash, and Optimal) under various values of the selfishness penalty factor α. (a) RI versus α. (b) RI evolution of the Nash strategy. (c) RI evolution.

For the RI, we can see from Fig. 4a that the Optimal strategy is robust to the variation of α, while a larger α leads to a higher RI in the Random and Nash strategies. On the other hand, as the RI evolution illustrated in Figs. 4b and 4c shows, a larger selfishness penalty factor is needed for the Nash strategy but the Optimal strategy generates a linearly increasing RI.

Fig. 5 shows the performance results with a varying mean number of players in each site. Again, we can see that the performance of the Optimal strategy is consistently better than the Random and Nash strategies and is quite robust. We can also see that, for the Random and Nash strategies, the effects of a larger number of players are similar to those of a larger selfishness penalty factor. For instance, the job rejection rate of Optimal stays rather constant at around 2 percent, while those of Random and Nash are as high as around 35 percent on average. For the utilization, we can see that, as more players are in a Grid site, the utilization of each machine becomes lower due to more spread-out load sharing.

Fig. 6a shows the variation of the RI with an increasing mean number of players. We can see that the Optimal strategy is again quite robust. On the other hand, the Random and Nash strategies show a slight downward trend, indicating that, as there are more players, a job is slightly more likely to be rejected, leading to a decreasing RI. This observation is reinforced by the results shown in Fig. 6b which illustrate that, as the system evolves, a smaller number of players can lead to a higher RI. Indeed, as Table 2 indicates, a smaller number of players gives a higher s_i value (i.e., higher degree of cooperation) in both the Nash and Optimal strategies. Finally, as depicted in

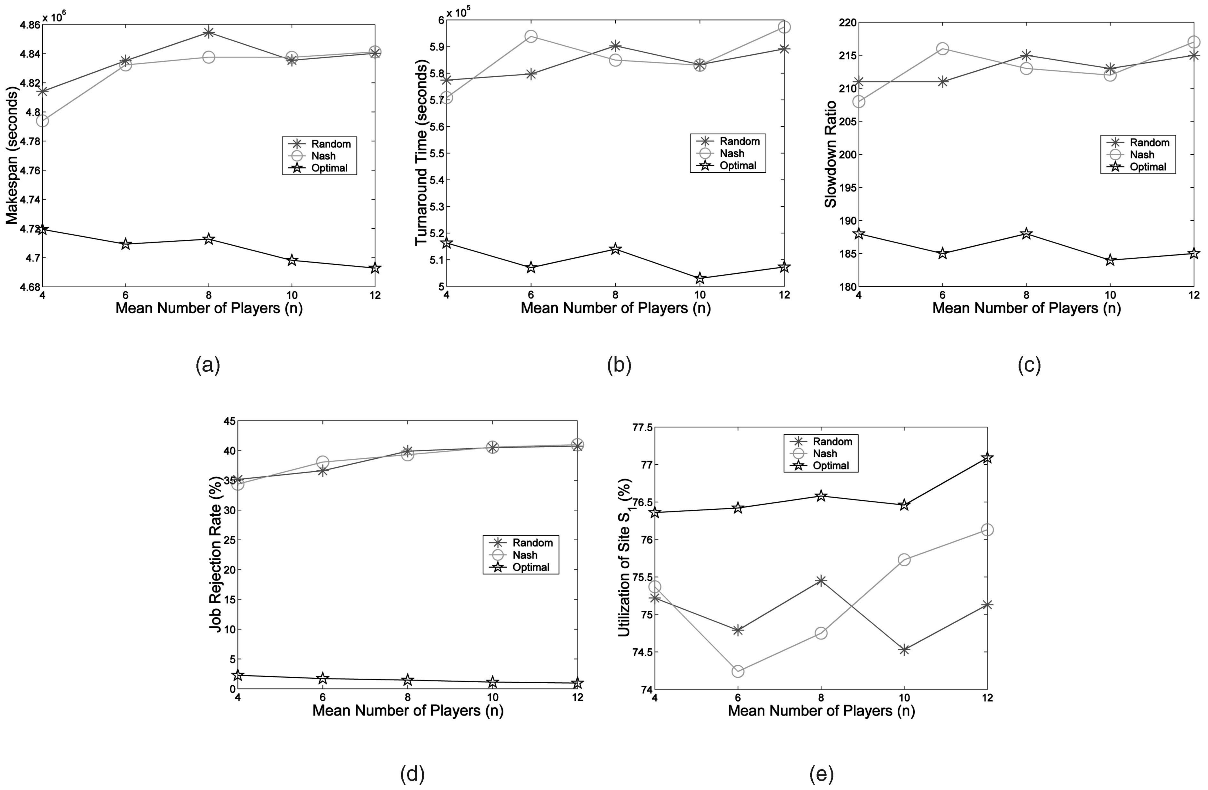


Fig. 5. The NAS performance using the three game strategies (Random, Nash, and Optimal) under various values of the mean number of players n in a Grid site. (a) Makespan. (b) Turnaround time. (c) Slowdown ratio. (d) Job rejection rate. (e) Utilization.

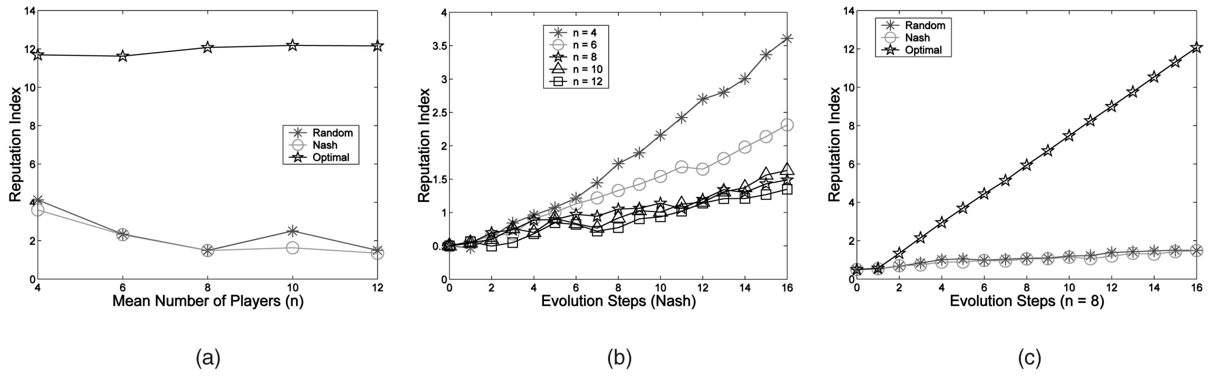


Fig. 6. The variation of the Reputation Index (RI) for the NAS workload using the three game strategies (Random, Nash, and Optimal) under various values of the mean number of players n in a Grid site. (a) RI versus n. (b) RI evolution of the Nash strategy. (c) RI evolution.

Fig. 6c, the Optimal strategy continuously increases the RI as the system evolves.

6.2 Results over PSA Workload

Now, let us consider the results over PSA workload. Fig. 7 shows the results for the PSA workload with various values of selfishness penalty factor α. Compared with the results for the NAS workload discussed above, the relative performance differences of the three strategies for the PSA workload are similar as for the NAS workload. For instance, the job rejection rate of Optimal is also around 2 percent to 3 percent while those of Random and Nash are as profound as around 40 percent on average.

Fig. 8 contains the results for the PSA workload with various values of the selfishness penalty factor. Again the trends are similar to those for the NAS workload. However, as the number of jobs in the PSA workload is smaller than those in the NAS workload (i.e., 10,000 versus 16,000), the values of RI are smaller. Nevertheless, the relative performance differences are similar.

Fig. 9 shows the results for the PSA workload with various mean numbers of players in a Grid site. The Optimal strategy also performs consistently better than the Random and Nash strategies. An interesting observation is that, for the utilization performance, while Random slightly outperforms Nash for the NAS workload, here, for the PSA workload, they perform quite close to each other.

Fig. 10 contains the RI variation results for the PSA workload with different mean number of players. We can see that as more players are involved, the RI values become smaller. As discussed above, this is likely due to the fact

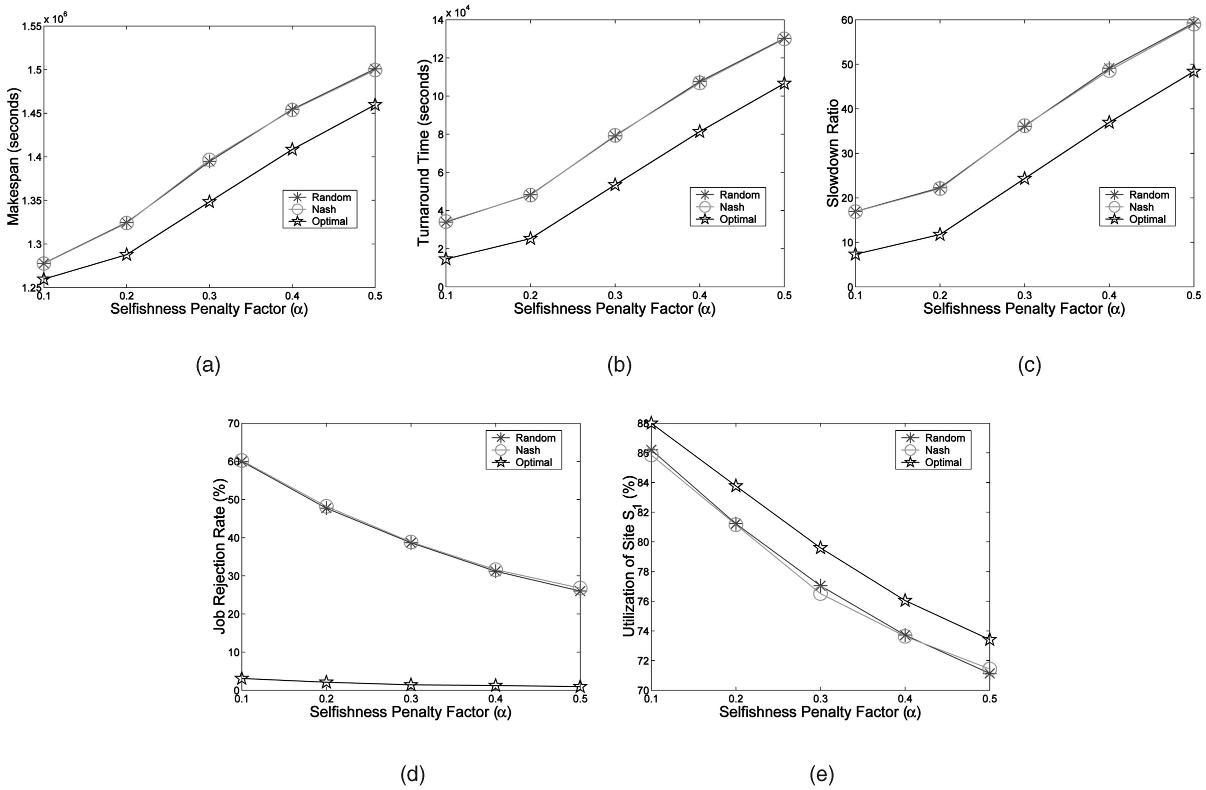


Fig. 7. The PSA performance using the three game strategies (Random, Nash, and Optimal) under various values of the selfishness penalty factor α. (a) Makespan. (b) Turnaround time. (c) Slowdown ratio. (d) Job rejection rate. (e) Utilization.

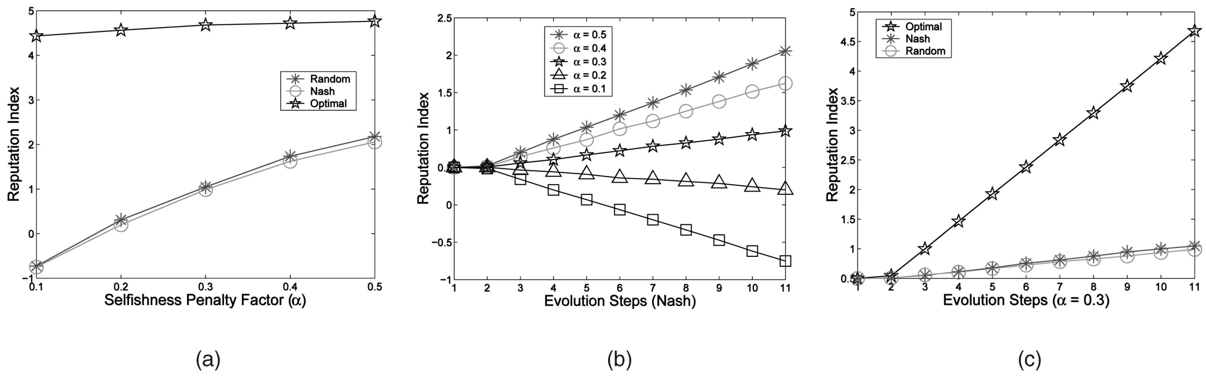


Fig. 8. The variation of the Reputation Index (RI) for the PSA workload using the three game strategies (Random, Nash, and Optimal) under various values of the selfishness penalty factor α. (a) RI versus α. (b) RI evolution of the Nash strategy. (c) RI evolution.

that as more players are involved the s_i values get smaller (as indicated in Table 2).

Our last set of simulation results is for testing the scalability effect of the Grid site. Fig. 11 shows the job execution performance results with varying number of Grid sites. As expected, since the workload size is fixed (i.e., there is a fixed number of jobs in both the PSA workload), the makespan, turnaround time, utilization, and slowdown ratio all decrease with increasing Grid site because the workload is shared by a larger number of machines. Interestingly, the job rejection rate results are rather independent of the Grid size due to the fact that the intra-site game is played for each assigned job only, indicating that the selfish behaviors of machines are independently distributed probabilistically for each job.

Finally, Fig. 12 shows the RI variations with different Grid sizes. Again, we can see that the RI value decreases as the Grid size increases because the jobs are shared by more sites so that each site gets less opportunity to increase its RI. Overall, the Optimal strategy is the best in all cases.

6.3 Comparison of Different Integrated Intersite and Intrasite Scheduling Schemes

Although the focus of this paper is on the intrasite job execution game, we believe that it is also interesting to study the interplay between different intersite job allocation methods integrated with the optimal intrasite strategy. In this final set of results, we consider the following three intersite job allocation policies:

- Min-Min algorithm. This is the algorithm that we use for all the results presented so far.
- Earliest finish time first (EFTF). This algorithm allocates a job to a Grid site based on the site's

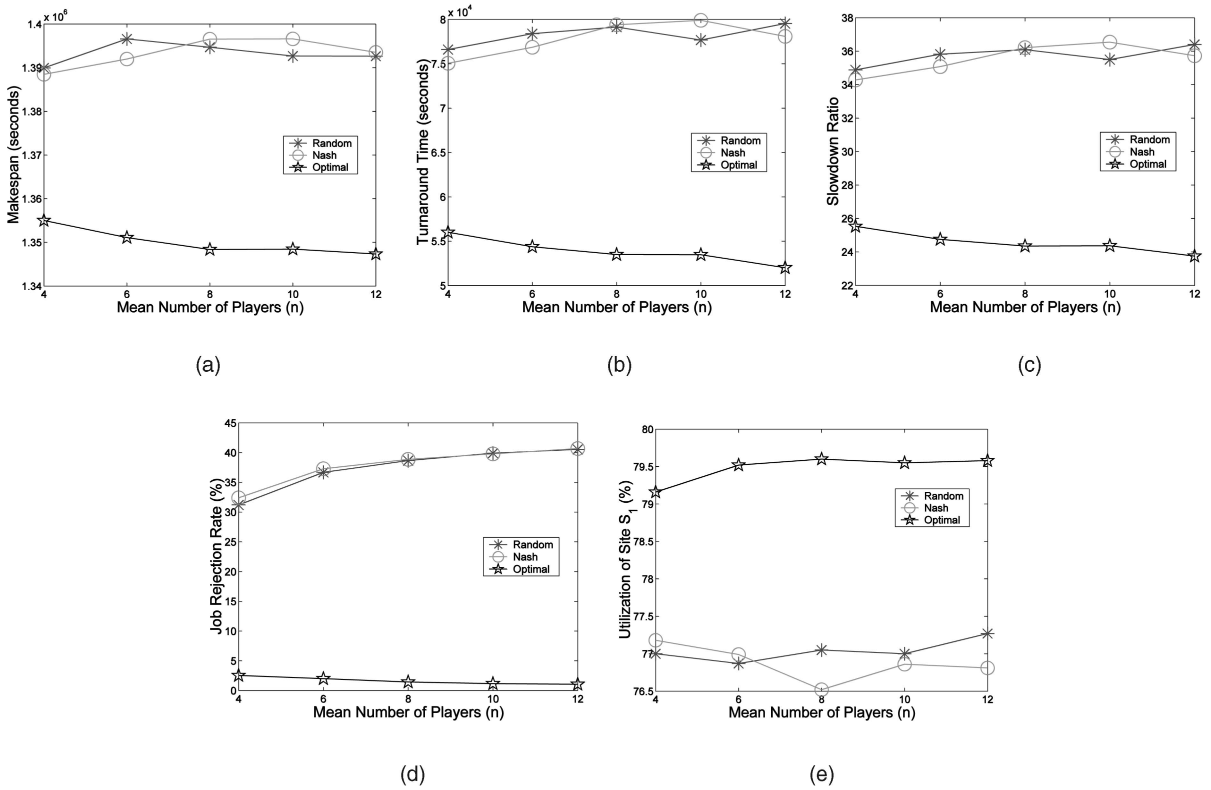


Fig. 9. The PSA performance using the three game strategies (Random, Nash, and Optimal) under various values of the mean number of players n in a Grid site. (a) Makespan. (b) Turnaround time. (c) Slowdown ratio. (d) Job rejection rate. (e) Utilization.

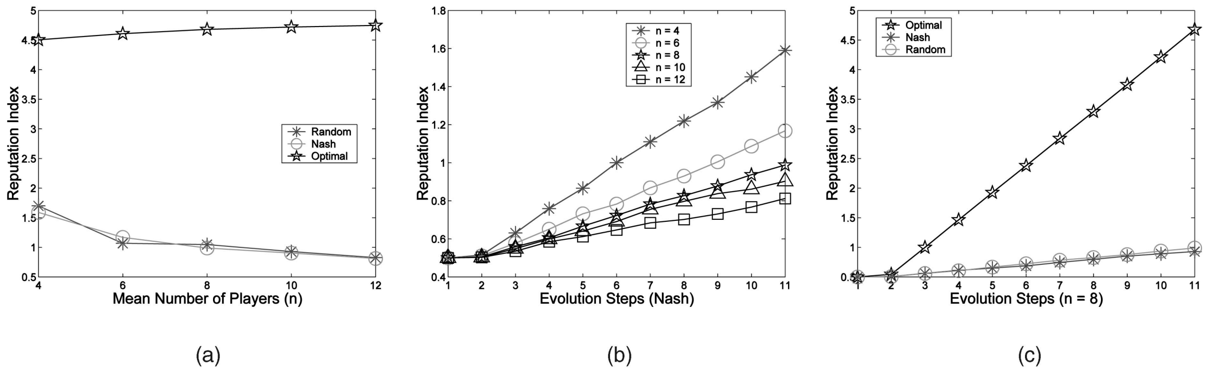


Fig. 10. The variation of the Reputation Index (RI) for the PSA workload using the three game strategies (Random, Nash, and Optimal) under various values of the mean number of players n in a Grid site. (a) RI versus n. (b) RI evolution of the Nash strategy. (c) RI evolution.

advertised finish time of the *last* remote job allocated to it.

- Auction. We use a simple auction strategy here—the intersite job allocation is based on the "second-highest bid" auction. That is, a job is allocated to a Grid site which advertises the second smallest execution time for the job. This is in sharp contrast with the Min-Min algorithm we used earlier in that the Min-Min algorithm considers only the shortest execution times.

We generated the results of applying these three intersite schemes combined with the optimal intrasite strategy for both the NAS and PSA workloads. In order to obtain a more macroscopic view of the relative performance, we also extended the system size by increasing the maximum mean number of players from 12 to 30. Fig. 13 shows the results of the comparison. We can see that, as we scale up the system size, all the combined scheduling methods can reduce the makespan significantly. For the NAS workload, we find that the Min-Min approach is always better than the EFTF algorithm. However, it is interesting to see that the auction approach, while initially outperformed by the Min-Min algorithm for small system sizes, improves quite remarkably for large system sizes. A plausible explanation is that, as we scale up the number of players in each site, the auction process becomes more efficient in the sense that the successful bidder site can usually find a "volunteer" to eventually execute a job on time. The PSA workload shows a similar trend.

The interplay of intersite and intrasite scheduling highly deserves further investigation. Due to space limitations, we will leave this for a future paper.

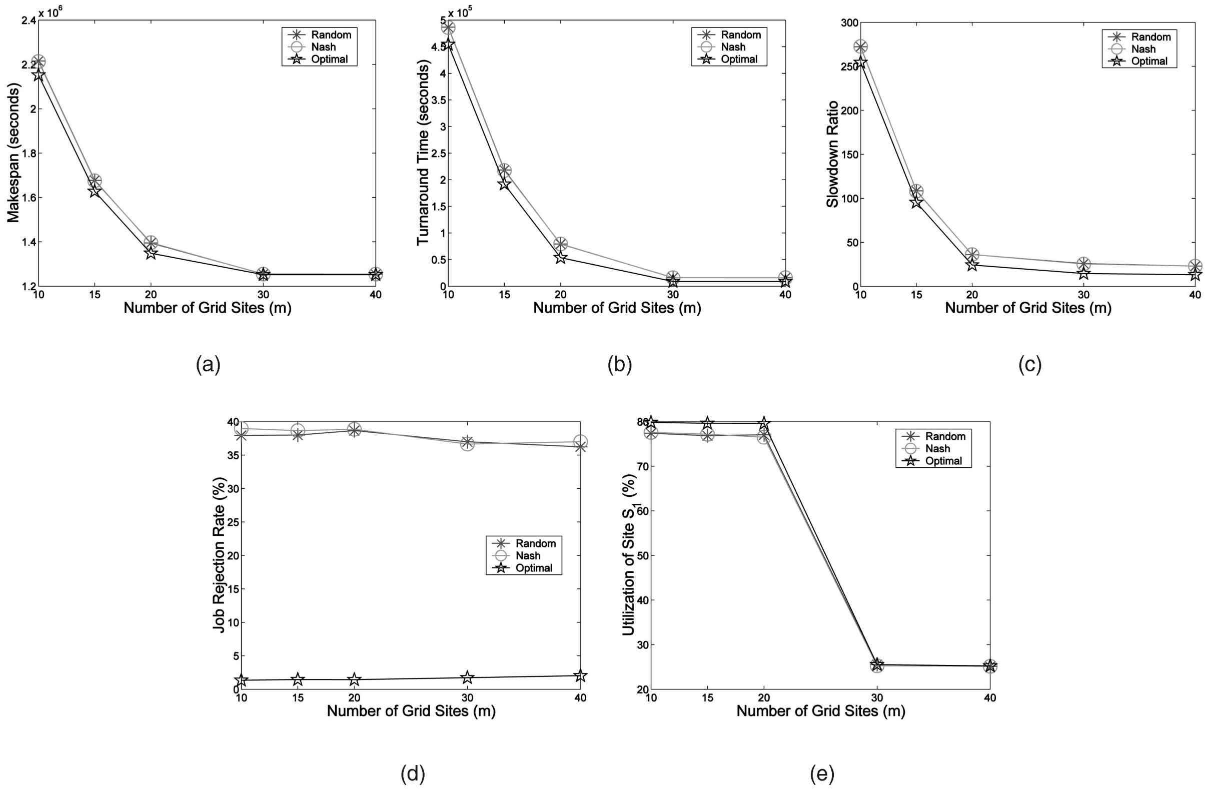


Fig. 11. The PSA performance using the three game strategies (Random, Nash, and Optimal) under various values of the number of sites m in the Grid. (a) Makespan. (b) Turnaround time. (c) Slowdown ratio. (d) Job rejection rate. (e) Utilization.

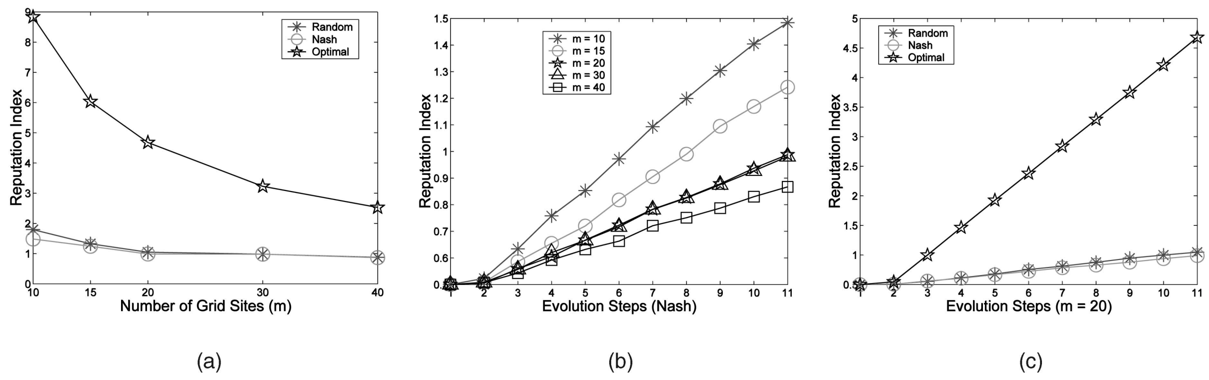


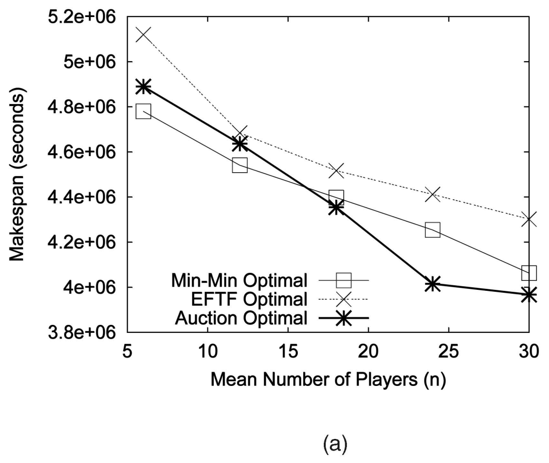
Fig. 12. The variation of the Reputation Index (RI) for the PSA workload using the three game strategies (Random, Nash, and Optimal) under various values of the number of sites m in the Grid. (a) RI versus m. (b) RI evolution of the Nash strategy. (c) RI evolution.

7 Conclusions and Future Work

We have presented a novel and general hierarchical Grid computing model by taking machine selfishness into account. Our model matches well with the real-life administrative structure of a practical Grid computing platform which is open and owned by a large number of autonomous management units organized at the intersite and intrasite levels. Using this hierarchical model, we can formulate three different game theoretic frameworks for studying the behaviors of the Grid machines: intrasite execution game, intrasite execution time bidding game, and intersite bidding game.

In this paper, we focus on the first framework to present a detailed mathematical analysis of the selfish behavior of individual machines within one single Grid site. Nash equilibrium and optimal strategies are analytically derived. We have also carried out detailed simulations to study the overall system performance under a wide range of parameters. We have reached two important conclusions: First, it is not necessarily bad for the machines to behave selfishly, provided that they all use the same optimal mixed strategy values s_i computed by our analysis. Second, the Nash equilibrium strategy is quite poor—almost the same as the Random strategy. Indeed, although there is no incentive for each player to deviate from the Nash equilibrium, the resulting equilibrium performance is not much different from that of a totally uncoordinated job execution scenario.

Our study presented in this paper has several limitations. For one thing, our approximate analysis is focused on symmetric mixed strategies. It would be interesting to analyze situations where heterogeneous strategies and deterministic actions are used by the machines. Moreover, the simulation study would provide more insights if we integrate all three levels of scheduling together with



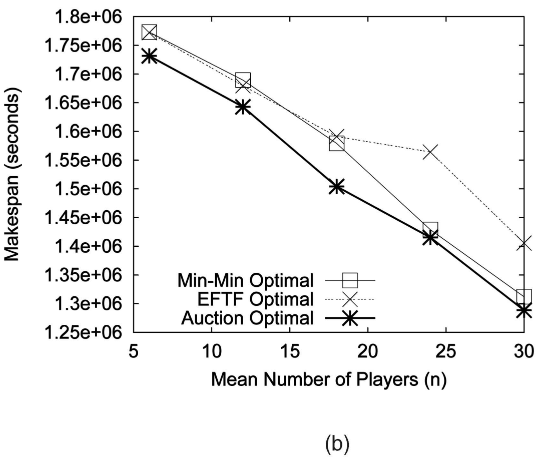


Fig. 13. Comparison of makespan performance generated by three different integrated inter-site and intra-site scheduling schemes. (a) NAS workload. (b) PSA workload.

intelligent algorithms used at each level. Furthermore, our simulation results are based on a small scale Grid due to various constraints in our testbed. An important future research work would be to perform more extensive simulations to study the performance of the integrated scheduling system under large-scale Grids.

There are some other interesting avenues of further research. First, it is important to devise an algorithm (possibly game theoretic) for the site manager to determine an accurate representative value of job execution time based on the objective of successfully bidding the job without overcommitting its resources. Second, the job deadlines in our performance study are governed by the underlying application (e.g., NAS or PSA) and are not under the control of an individual site. However, if the deadline of a job can be manipulated by a site, it would be interesting to design a game theoretic deadline setting strategy in order to maximize the successful remote execution probability. Third, linking the selfish factors with appropriate trust negotiation models for Grid computing is still a wide open research problem. We are currently working on integrating the three different games into a unified framework, which is then incorporated into our previously proposed trust binding methodology [43].

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers (in particular, Reviewers 2 and 3) for their constructive suggestions to improve the quality and presentation of this paper. Thanks are also due to Professor Xiaodong Zhang for his useful suggestions on the final version of the manuscript. Finally, the authors thank the members of the University of Southern California GridSec research team for their inspiring comments about this research. This paper is significantly extended from the conference version submitted to the IEEE/ACM CCCGrid 2005. This research was supported by an NSF ITR Research Grant under contract number ACI-0325409.

REFERENCES

[1] A. Akella, S. Seshan, R. Karp, and S. Shenker, “Selfish Behavior and Stability of the Internet: A Game-Theoretic Analysis of TCP,” *Proc. ACM SIGCOMM ’02,* pp. 117-130, Aug. 2002.

[2] E. Altman, T. Basar, and R. Srikant, “Nash Equilibria for Combined Flow Control and Routing in Networks: Asymptotic Behavior for a Large Number of Users,” *IEEE Trans. Automatic Control,* vol. 47, no. 6, pp. 917-930, June 2002.

[3] E. Altman, T. Basar, T. Jimenez, and N. Shimkin, “Routing into Two Parallel Links: Game-Theoretic Distributed Algorithms,” *J. Parallel and Distributed Computing,* vol. 61, no. 9, pp. 1367-1381, Sept. 2001.

[4] *Handbook of Game Theory with Economic Applications,* vol. 2, R. Aumann and S. Hart, eds. Elsevier, 1994.

[5] D.K. Barry, *Web Services and Service-Oriented Architectures.* Morgan Kaufmann, 2003.

[6] *Grid Computing: Making the Global Infrastructure a Reality,* F. Berman, G. Fox, and T. Hey, eds. John Wiley & Sons, 2003.

[7] T. Boulogne, E. Altman, H. Kameda, and O. Pourtallier, “Mixed Equilibrium (ME) for Multiclass Routing Games,” *IEEE Trans. Automatic Control,* vol. 47, no. 6, pp. 903-916, June 2002.

[8] T.D. Braun et al., “A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems,” *J. Parallel and Distributed Computing,* vol. 61, no. 6, pp. 810-837, June 2001.

[9] J. Bredin, R.T. Maheswaran, C. Imer, T. Basar, D. Kotz, and D. Rus, “A Game-Theoretic Formulation of Multi-Agent Resource Allocation,” *Proc. ACM Int’l Conf. Autonomous Agents (Agents ’00),* pp. 349-356, June 2000.

[10] R. Buyya, “Economic-Based Distributed Resource Management and Scheduling for Grid Computing,” PhD thesis, Monash Univ., Apr. 2002.

[11] X.-R. Cao, H.-X. Shen, R. Milito, and P. Wirth, “Internet Pricing with a Game Theoretical Approach: Concepts and Examples,” *IEEE/ACM Trans. Networking,* vol. 10, no. 2, pp. 208-216, Apr. 2002.

[12] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman, “Heuristics for Scheduling Parameter Sweep Applications in Grid Environments,” *Proc. Heterogeneous Computing Workshop (HCW ’00),* pp. 349-363, May 2000.

[13] K.-M. Chao, R. Anane, J.-H. Chen, and R. Gatward, “Negotiating Agents in a Market-Oriented Grid,” *Proc. Int’l Symp. Cluster Computing and the Grid (CCGrid ’02),* pp. 436-437, May 2002.

[14] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke, “Resource Management Architecture for Metacomputing Systems,” *Proc. Fourth Int’l Workshop Job Scheduling Strategies for Parallel Processing,* pp. 62-82, Mar. 1998.

[15] A. Czumaj and A. Ronen, “On the Expected Payment of Mechanisms for Task Allocation,” *Proc. ACM Symp. Principles of Distributed Computing (PODC ’04),* pp. 98-106, July 2004.

[16] Ö. Ercetin and L. Tassiulas, “Market-Based Resource Allocation for Content Delivery in the Internet,” *IEEE Trans. Computers,* vol. 52, no. 12, pp. 1573-1585, Dec. 2003.

[17] I. Foster, C. Kesselman, and S. Tueke, “The Anatomy of the Grid: Enabling Scalable Virtual Organizations,” *Int’l J. Supercomputing Applications,* vol. 15, no. 3, pp. 200-222, 2001.

[18] I. Foster and C. Kesselman, *The Grid: Blueprint for a New Computing Infrastructure.* Morgan Kaufman, 1999.

[19] P. Ghosh, N. Roy, S.K. Das, and K. Basu, "A Game Theory Based Pricing Strategy for Job Allocation in Mobile Grids," *Proc. Int'l Parallel and Distributed Processing Symp. (IPDPS '04),* pp. 82-91, Apr. 2004.
[20] J. Gray, "Distributed Computing Economics," Technical Report MSR-TR-2003-24, Microsoft Research, Mar. 2003.
[21] D. Grosu and A.T. Chronopoulos, "Algorithmic Mechanism Design for Load Balancing in Distributed Systems," *IEEE Trans. Systems, Man, and Cybernetics,* vol. 34, no. 1, pp. 77-84, Feb. 2004.
[22] D. Grosu and A.T. Chronopoulos, "Noncooperative Load Balancing in Distributed Systems," *J. Parallel and Distributed Computing,* vol. 65, no. 9, pp. 1022-1034, Sept. 2005.
[23] D. Grosu, A.T. Chronopoulos, and M.Y. Leung, "Load Balancing in Distributed Systems: An Approach Using Cooperative Games," *Proc. 16th IEEE Int'l Parallel and Distributed Processing Symp.,* pp. 501-510, Apr. 2002.
[24] T. Groves, "Incentive in Teams," *Econometrica,* vol. 41, no. 4, pp. 617-631, July 1973.
[25] L.V. Kalé, S. Kumar, and J. DeSouza, "A Malleable-Job System for Time-Shared Parallel Machines," *Proc. Int'l Symp. Cluster Computing and the Grid (CCGrid '02),* pp. 230-237, May 2002.
[26] Y.-K. Kwok, S. Song, and K. Hwang, "Selfish Grid Computing: Game-Theoretic Modeling and NAS Performance Results," *Proc. Int'l Symp. Cluster Computing and the Grid (CCGrid '05),* vol. 2, pp. 1143-1150, May 2005.
[27] K. Larson and T. Sandholm, "Miscomputing Ratio: Social Cost of Selfish Computing," *Proc. Int'l Conf. Autonomous Agents and Multi-Agent Systems (AAMAS '03),* pp. 273-280, 2003.
[28] D. Lehmann, L.I. O'Callaghan, and Y. Shoham, "Truth Revelation in Approximately Efficient Combinatorial Auctions," *J. ACM,* vol. 49, no. 5, pp. 577-602, Sept. 2002.
[29] V. Lo and J. Mache, "Job Scheduling for Prime Time vs. Non-Prime Time," *Proc. Conf. Cluster Computing (Cluster '02),* pp. 488-493, Sept. 2002.
[30] R. Mahajan, M. Rodrig, D. Wetherall, and J. Zahorjan, "Experiences Applying Game Theory to System Design," *Proc. ACM SIGCOMM '04,* pp. 183-190, Sept. 2004.
[31] J.F. Nash Jr., "The Bargaining Problem," *Econometrica,* vol. 18, no. 2, pp. 155-162, Apr. 1950.
[32] J.F. Nash Jr., "Non-Cooperative Games," *Annals of Math.,* second series, vol. 54, no. 2, pp. 286-295, Sept. 1951.
[33] N. Nisan, "Algorithms for Selfish Agents: Mechansim Design for Distributed Computation," *Proc. 16th Symp. Theoretical Aspects of Computer Science,* pp. 1-17, 1999.
[34] N. Nisan and A. Ronen, "Algorithmic Mechanism Design," *Proc. ACM Symp. Theory of Computing,* pp. 129-140, 1999.
[35] S. Olafsson, "Games on Networks," *Proc. IEEE,* vol. 85, no. 10, pp. 1556-1562, Oct. 1997.
[36] M.J. Osborne and A. Rubinstein, *A Course in Game Theory.* MIT Press, 1994.
[37] C.H. Papadimitriou, "Algorithms, Games, and the Internet," *Proc. ACM Symp. Theories of Computing (STOC '01),* pp. 749-753, July 2001.
[38] K. Ranganathan, M. Ripeanu, A. Sarin, and I. Foster, "Incentive Mechanisms for Large Collaborative Resource Sharing," *Proc. Int'l Symp. Cluster Computing and the Grid (CCGrid '04),* pp. 1-8, Apr. 2004.
[39] O. Regev and N. Nisan, "The POPCORN Market—An Online Market for Computational Resources," *Proc. ACM Int'l Conf. Information and Computation Economies (ICE '98),* pp. 148-157, Oct. 1998.
[40] J.B. Rosen, "Existence and Uniqueness of Equilibrium Points for Concave N-Person Games," *Econometrica,* vol. 33, no. 3, pp. 520-534, July 1965.
[41] T. Roughgarden and E. Tardos, "How Bad Is Selfish Routing?" *J. ACM,* vol. 49, no. 2, pp. 236-259, Mar. 2002.
[42] R. Sami, "Distributed Algorithmic Mechanism Design," PhD thesis, Yale Univ., Dec. 2003.
[43] S. Song, K. Hwang, and Y.-K. Kwok, "Trusted Grid Computing with Security Binding and Trust Integration," *J. Grid Computing,* vol. 3, nos. 1-2, pp. 53-73, June 2005.
[44] D.E. Volper, J.C. Oh, and M. Jung, "GameMosix: Game-Theoretic Middleware for CPU Sharing in Untrusted P2P Environment," *Proc. Conf. Parallel and Distributed Computing and Systems (SPDCS),* 2004.
[45] M.P. Wellman, J.K. Mackie-Mason, D.M. Reeves, and S. Swaminathan, "Exploring Bidding Strategies for Market-Based Scheduling," *Proc. ACM Conf. Electronic Commerce (EC '03),* pp. 115-124, June 2003.
[46] R. Wolski, J.S. Plank, T. Bryan, and J. Brevik, "G-Commerce: Market Formulations Controlling Resource Allocation on the Computational Grid," *Proc. Int'l Parallel and Distributed Processing Symp. (IPDPS '01),* Apr. 2001.
[47] H. Yaïche, R.R. Mazumdar, and C. Rosenberg, "A Game Theoretic Framework for Bandwidth Allocation and Pricing in Broadband Networks," *IEEE/ACM Trans. Networking,* vol. 8, no. 5, pp. 667-678, Oct. 2000.
[48] M.K.H. Yeung, "On Channel Adaptive Wireless Cache Invalidation and Game Theoretic Power Aware Wireless Data Access," MPhil thesis, Univ. of Hong Kong, Aug. 2004.

Yu-Kwong Kwok received the BSc degree in computer engineering from the University of Hong Kong in 1991, the MPhil and PhD degrees in computer science from the Hong Kong University of Science and Technology (HKUST) in 1994 and 1997, respectively. He is an associate professor in the Department of Electrical and Electronic Engineering at the University of Hong Kong (HKU). Before joining HKU in August 1998, he was a visiting scholar for one year in the parallel processing laboratory at the School of Electrical and Computer Engineering at Purdue University. He recently served as a visiting associate professor at the Department of Electrical Engineering–Systems at the University of Southern California from August 2004 to July 2005, on his sabbatical leave from HKU. His research interests include distributed computing systems, wireless networking, and mobile computing. He is a senior member of the IEEE. He is also a member of the ACM, the IEEE Computer Society, and the IEEE Communications Society. He received the Outstanding Young Researcher Award from HKU in November 2004.

Kai Hwang received the PhD degree from the University of California, Berkeley in 1972. He is a professor of electrical engineering and computer science and director of the Internet and Grid Research Laboratory at the University of Southern California. An IEEE fellow, he specializes in computer architecture, parallel processing, Internet and wireless security, Grid and cluster computing, and distributed computing systems. Dr. Hwang is the founding editor-in-chief of the *Journal of Parallel and Distributed Computing* published by Elsevier. He is also on the editorial boards of the *IEEE Transactions on Parallel and Distributed Systems* and of the *International Journal of High-Performance Computing and Networking*. He has published more than 200 scientific papers and 7 books. His latest two books, *Scalable Parallel Computing* and *Advanced Computer Architecture* are being adopted worldwide and translated into four languages. Currently, he leads a US National Science Foundation-supported GridSec project in developing security-binding techniques and distributed defense systems against worms and DDoS attacks for trusted Grid, P2P, and Internet computing. Contact him at kaihwang@usc.edu or visit the project Web site, http://GridSec.usc.edu/Hwang.html, for details.

ShanShan Song received the PhD degree in computer science from the University of Southern California in 2005. She is a member of the technical staff at Oracle Corporation. She specializes in P2P networks, network security, parallel and distributed computing, and database systems. She is a member of the IEEE and the IEEE Computer Society.