Non-Cooperative Grids: Game-Theoretic Modeling and Strategy Optimization

Yu-Kwong Kwok, *Senior Member, IEEE*, ShanShan Song, *Student Member, IEEE*, and Kai Hwang, *Fellow, IEEE*
University of Southern California, Los Angeles, CA 90089, USA *

December 1, 2004

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 non-cooperativeness 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 intra-site 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 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 valuable reference for designing appropriate strategies in a practical Grid.

Keywords—Grid computing, non-cooperative games, virtual organizations, selfish behaviors, online scheduling, Nash equilibrium, optimal strategies, performance evaluation, NAS workload, parameter sweep application (PSA).

---

*Manuscript submitted to the IEEE TPDS. This paper is significantly extended from the conference version submitted to the IEEE/ACM CCGrid 2005. This research was supported by an NSF ITR Research Grant under contract number ACI-0325409. Corresponding author: Kai Hwang, Email: kaihwang@usc.edu, Tel: 213-740-4470, Fax: 213-740-4418.

1 Introduction

The lofty goal of Grid computing is to leverage on the interconnection of a large number of geographically distributed machines to solve computational problems faster at a gigantic scale [5]. 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 in the non-cooperation among states of a large country, or the non-cooperation among departments in a large organization.

Thus, it is an important research problem to model the Grid and its constituents by taking into account the potential non-cooperativeness at various levels. 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 [8, 12, 13, 39] (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 inter-site level, and the intra-site 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 intra-site job execution game, the intra-site bidding game, and the inter-site bidding game. In this paper, we focus on the intra-site job execution game in this paper and the other games, and most importantly, the interplay among the three games, are presented elsewhere [19].

2. **Mathematical Analysis of the Intra-Site Job Execution Game:** We first propose a novel but real-

istic 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 that the machines are selfish.

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 [6], 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 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 intra-site 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.

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 [3, 15, 23, 28, 30, 41]. More notable examples include TCP congestion control [1, 2], routing [34, 35], bandwidth pricing [9, 40], contents delivery [13], file sharing [31], wireless caching [41],etc.

There have also been some recent results on game theoretic job allocation and scheduling reported in the literature [26]. Regev and Nisan [32] 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 [3, 32]. 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 practicable in a real-life situation where there are tremendous number of machines involved. Similar approaches were also proposed by other researchers [7, 11, 12, 38].

Wolski *et al.* [39] proposed a model called G-Commerce in which computational resources among different Grid sites are traded in a barter manner. The efficient of two different market conditions—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.* [14] 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) [24], 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 [20] pioneered the consideration of the computation cost involved in determining the valuations which 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 [26, 27] formally defined the job allocation in a distributed system using a truthful mechanism framework. 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 and Chronopoulos [16] recently designed a load balancing system based on the Vickrey-Clarke-Groves (VCG) mechanism [17, 29] 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 Lagrangian solution based on the Kuhn-Tucker conditions, Grosu and Chronopoulos derived a set of conditions that are to be enforced algorithmically by each individual computers in a distributed manner. A deficiency in their work is that the physical meaning of the payment functions is unclear and in an open Grid environment, load balancing is not the major concern.

Volper *et al.* [37] 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 [19].

3 A Hierarchical Semi-Selfish Grid Model

As mentioned earlier 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 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 Figure 1(a), 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 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.

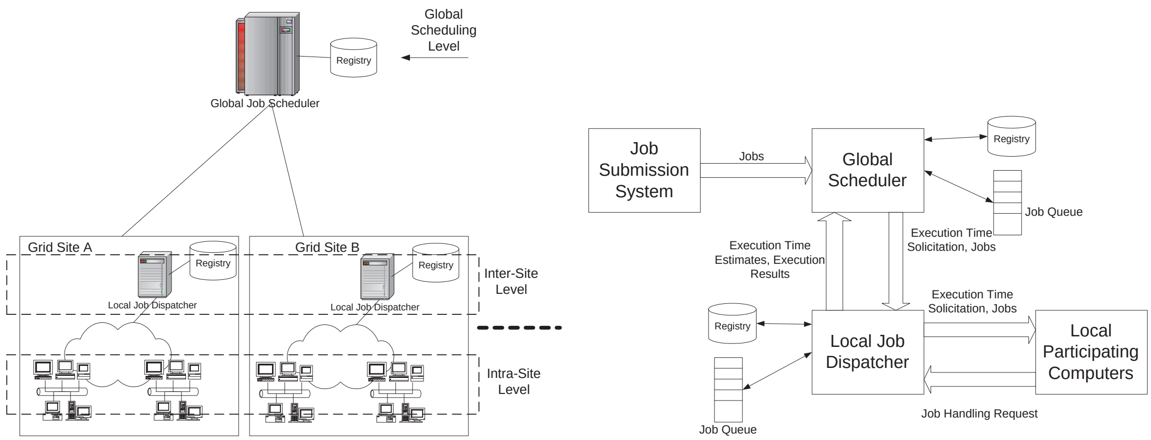
For instance, the computer science department of a university might own a large cluster of PCs, the electrical engineering department might possess another, and the physics department might manage a massively parallel supercomputer. Yet all these computing resources participate in the global Grid community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

With the hierarchical structure shown in Figure 1(a), there are also two levels of job scheduling and dispatching, depicted in Figure 1(b). 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 [4]). Equipped with a global Grid processing resources registry (again could be based on the UDDI protocol), the global scheduler performs job allocation, according to a certain scheduling algorithm.

Most importantly, at the inter-site 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

[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.

of scheduling at the global Grid level can be efficiently handled. Furthermore, again this scheduling model conforms well to the administrative structure of the Grid community in the sense that the global scheduler probably should not "micro-manage" 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.



(a) The hierarchical structure of an open Grid

(b) Control flow in open Grid job scheduling

Figure 1: System model of an open Grid computing platform.

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. Firstly, any efficient online job scheduling algorithm can be used. Furthermore, it is an important study about 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. **Intra-Site Job Execution Strategies:** This problem concerns about 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 one of the 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 computer 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 game theoretic strategy so as to maximize its own utility (i.e., execute more local jobs) without rendering the whole site non-operational. We focus on this problem in this paper.

2. **Intra-Site Bidding:** This problem concerns about the determination of the advertised "execution capabilities" 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 execution 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 hand, a conservative approach is to perform "maximization"—taking the maximum value instead.

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

3. **Inter-Site Bidding:** Similar to the intra-Site situation, at the inter-site 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 inter-play of these three games, i.e., how the selfishness of each individual computer affects the intra-site bidding, which, in turn, will impact the inter-site bidding in a complicated manner.

Indeed, different combinations of the above games will result in different Grid structures. For a *semi-selfish* Grid, the intra-site games are non-cooperative while the inter-site game is cooperative. This model fits most nowadays' Grid situation 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 among each other. For a *fully-selfish* Grid, the games are assumed to be non-cooperative 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 on our formulation, analysis, and results on the first problem introduced above. Specifically, to simplify the model, we assume that the inter- and intra-site bidding processes, truthful mechanisms [21] are used. In subsequent papers [19], we will present our results on the untruthful revelation of participating machines within each Grid site and the inter-site auction problem.

4 Semi-Selfish Cooperation Mechanisms and Mixed Strategies

In this section, we present our analytical formulation of the game theoretic framework for the intra-site job execution mechanism. We first describe the job model and execution policies. We then formulate the 2-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 [18], 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 in 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., need to expend some processing resources to monitor the Grid status, or to advertise its capabilities, and so on). 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.

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* [29] 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

However, the Grid site as a whole would like to maximize its *Reputation Index* (RI) which quantifies the contributions of the site. 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 non-cooperative game [25] 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 its selfish strategy and it can choose to either ignore the job or take it up.

We consider *mixed strategies* [24, 29] 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 but Q is a variable component which 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. Let us consider a simple scenario first—only two computers are involved.

4.1 The 2-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 refuses to take up the job earlier. Differentiating U_1 with respect to s_1 gives:

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

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 any one 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 *[24, 29] in the 2-player game, i.e., no player can benefit by unilaterally deviating from this strategy combination.*

Proof: *Theorem is true because* $Q = P$ *by Equation (4) and thus,* U_i *does not depend on the value of* s_i *(for* $i = 1, 2$*) under this symmetric combination. (Q.E.D.)*

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 is a weak one and the solution is degenerated [29, 33]. 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:

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

With some sample numerical values (i.e., $P_{\text{total}} = 256$, $P_o = 4$, $P = 32$, and $\alpha = 0.5$), Figure 2(a) shows the relationships among U_1, s_1, and s_2. We can draw a number of conclusions:

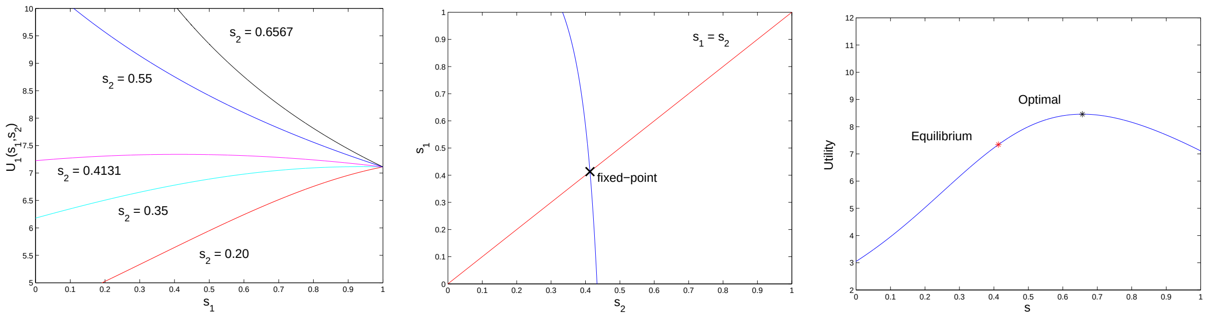
- If M_1 always takes up the job, i.e., $s_1 = 1$, its utility is independent of M_2's strategy s_2;

- 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., $s_1 \in (0, 1)$.



(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

Figure 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.

Furthermore, it can be shown 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, Figure 2(b) shows a plot of Equation (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, again this Nash equilibrium is sub-optimal, as evident by the following analysis. Let us take $s_1 = s_2 = s$ in Equation (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: $s = 0.6567$. Figure 2(c) 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* [29], which is a common characteristic in non-cooperative 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.

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 varies from one job to another. Secondly, as α gets larger, τ becomes a larger fraction of Γ.

Indeed, with $\alpha = 0.1$ (i.e., 10% more processors are needed to finish the job after each round), τ is equal to 9.1% of Γ. On the other hand, with $\alpha = 0.5$ (i.e., 50% more processors are needed to finish the job after each round), τ is equal to 33.3% of Γ. Thus, with $\alpha = 0.5$, after two rounds of waiting, 66.7% of originally useful execution time is wasted and 200% 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 re-scheduled by the global scheduler to a possibly new site in the next batch.

4.3 The n-Player Game

We can easily extend the 2-player game to the n-player scenario, i.e., 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 involved in the n-player game is:*

$$P'^i = P(s_i + (s_i + \alpha)\prod_j^n (1 - s_j) + (1 + 2\alpha)\prod_j^n (1 - s_j)^2) \tag{7}$$

Thus, the symmetric Nash equilibrium strategy is given by:

$$\hat{s} = \frac{2(1+2\alpha)\xi^2 - (1-\alpha)\xi - 1}{2(1+2\alpha)\xi^2 - 2\xi} \tag{8}$$

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

Proof: *Can be easily shown by mathematical induction on n. (Q.E.D.)*

Indeed, the following theorem formalizes the existence of an equilibrium strategy for the n-player intra-site job execution game.

Theorem 3 *There exists an equilibrium strategy for the n-player intra-site 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 , are continuous on $\Pi_{1 \le j \le n}\{s_j\} \subset \Re^n$. Furthermore, the best-reply mapping is single-valued. Thus, we can conclude that there exists an equilibrium strategy for the n-player game [29]. (Q.E.D.)*

Although Theorem 3 does not rule out the possibility of more than one equilibrium, we observe, from simulations, that the equilibrium strategy (Equation (8)) appears to be unique.

On the other hand, the optimal strategy 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:*

$$1 - n(s+\alpha)(1-s)^{n-1} + (1-s)^n - 2n(1+2\alpha)(1-s)^{2n-1} = 0 \tag{9}$$

Proof: *Can be easily shown by mathematical induction on n. (Q.E.D.)*

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.

Table 2: Some Sample s_i Values of the Nash and Optimal Strategies.

α	$n=2$		$n=4$		$n=6$		$n=8$		$n=12$	
	Nash	Optimal	Nash	Optimal	Nash	Optimal	Nash	Optimal	Nash	Optimal
0.1	0.1377	0.4826	0.0488	0.3139	0.0297	0.2427	0.0213	0.2008	0.0136	0.1522
0.2	0.2353	0.5445	0.0875	0.3462	0.0537	0.2659	0.0387	0.2194	0.0248	0.1660
0.3	0.3087	0.5908	0.1192	0.3725	0.0738	0.2852	0.0535	0.2349	0.0344	0.1775
0.4	0.3663	0.6272	0.1461	0.3946	0.0910	0.3016	0.0661	0.2482	0.0427	0.1875
0.5	0.4131	0.6567	0.1692	0.4136	0.1061	0.3158	0.0773	0.2598	0.0500	0.1961

4.4 Algorithms for Intra-Site 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

[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.

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.

If there is a job assigned to the Grid site, the local job dispatcher will then coordinate the intra-site 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 re-schedule 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 Grid site. Corresponding to Algorithm 1, each participating machine uses the Algorithm 2 to play the intra-site game.

Algorithm 1 Local Job Dispatcher

if global scheduler solicits "execution time estimates" for a job, J_k **then**
  Solicit execution time estimates from all participating computers, $T_i(J_k)$;
  Return max$\{T_i(J_k)\}$ to the global scheduler;
end if
if a job is assigned to the site **then**
  Check the currently active number of participating computers, n;
  Broadcast the value of optimal strategy $\hat{s}$ according to Equation (9) to all the participating computers;
  for round = 1, 2 (i.e., a total of 2τ units of time) **do**
    if a computer M_i takes up the job (ties are broken randomly) **then**
      Send the job to M_i;
      Declare that the job is unavailable;
    end if
  end for
  if no computer takes up the job **then**
    Declare that the job fails;
  end if
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 Equation (8) instead of Equation (9).

Algorithm 2 Participating Machine (Player i)

if local job dispatcher solicits "execution time estimates" for a job, J_k **then**
    Return the local estimate of $T_i(J_k)$ using the value of desired number of processors P to the local job dispatcher;
end if
if a job is assigned to the site **then**
    Receive the broadcast value of optimal strategy $\hat{s}$ according to Equation (9) from the local job dispatcher;
    for round = 1, 2 (i.e., a total of 2τ units of time) **do**
        if Random $< \hat{s}$ **then**
            Declare that this machine takes up the job;
            Execute the job;
        end if
    end for
end if

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, $2s_{\text{Nash}}$].

A hierarchical semi-selfish Grid infrastructure is simulated using a discrete event-driven simulator. At the inter-site level, m cooperative Grid sites are simulated. At the intra-site 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 [29], to the centralized scheduler. The well known Min-Min scheduling heuristic [6] is used for the inter-site 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 [22]. This trace contains 92 days (7,948,800 seconds) data, gathered in year 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 proportionally squeezed to 46 days. We map the 128 nodes to 12 Grid sites-four of the sites each contain 16 nodes, and the other eight 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 pre-processed to correct for system downtime [22].

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 [10]. The parameter sweep application is defined as a set of independent sequential jobs (i.e., no job precedence). The independent jobs operate on different datasets. 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 dataset) 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.

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: $\beta_1 = 1$, $\beta_2 = 0.5$, $\gamma = 1$

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.

$$\mathrm{RI}_j^{\text{new}} = \mathrm{RI}_j^{\text{old}} + (\beta_1 \frac{W_j^1}{W_{\text{total}}} + \beta_2 \frac{W_j^2}{W_{\text{total}}} - \gamma \frac{W_j^r}{W_{\text{total}}}) \tag{10}$$

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., Equation (10)), 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

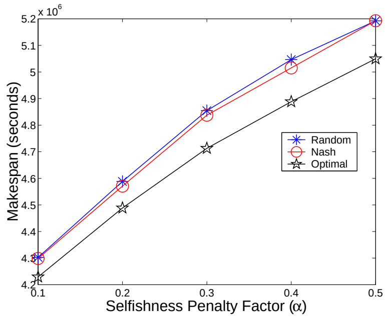
In this section, we present our simulation results over the NAS workload for the three strategies: *Random*, *Nash*, and *Optimal*. We evaluate the overall system performance using the following metrics:

- *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.

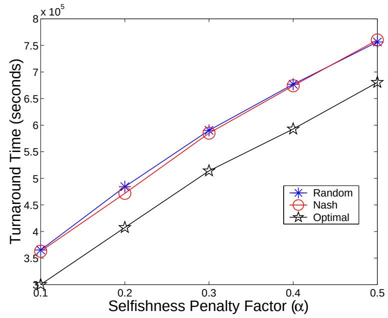
6.1 Results over NAS Workload

We first consider the results over the NAS workload. As indicated in Figure 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 to 8% better than the other two strategies. Furthermore, the job rejection rate of Optimal is only around 2 to 3% but those of Random and Nash are around 40% on average. Here, notice that the utilization results shown in Figure 3(e) is for a typical Grid site in the simulation platform (denoted by S_1).

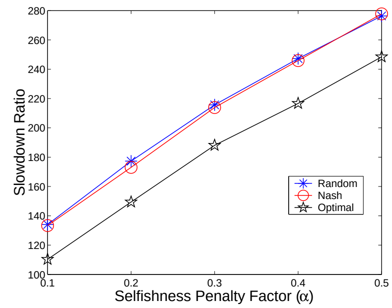
Thus, we have an important conclusion: it is not necessarily bad for the machines to behave selfishly provided that they all use the same optimal mixed strategy values s_i computed by our analysis. Another interesting conclusion is that the Nash equilibrium strategy is quite poor—almost the same as the 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.



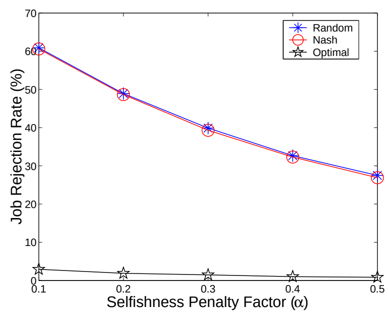
(a) Makespan



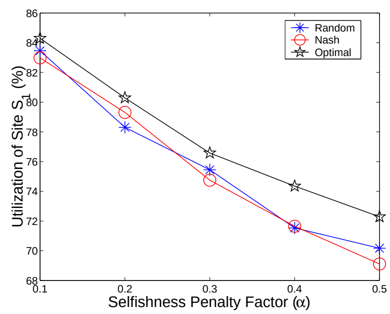
(b) Turnaround time



(c) Slowdown ratio



(d) Job rejection rate



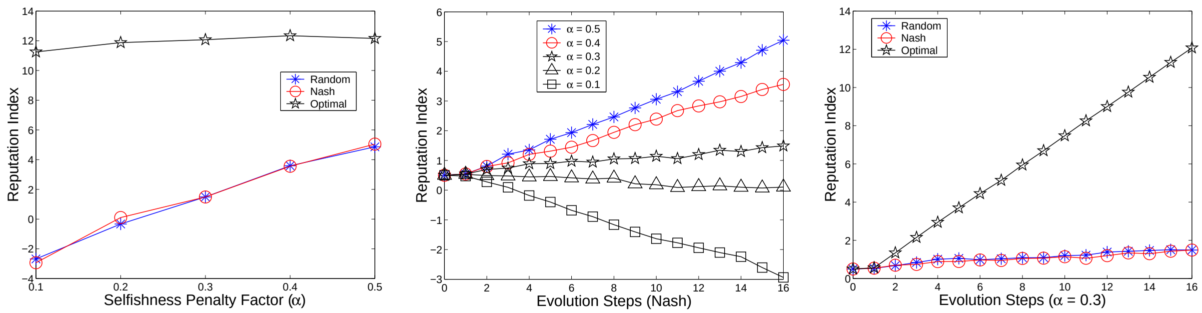
(e) Utilization

Figure 3: The NAS performance using the three game strategies (Random, Nash, and Optimal) under various values of the selfishness penalty factor α.

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.

For the RI, we can see from Figure 4(a) 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 Figures 4(b) and 4(c) shows, a larger selfishness penalty factor is needed for the Nash strategy but the Optimal strategy generates a linearly increasing RI.

Figure 5 shows the performance results with varying mean number of players in each site. Again we can see that the performance of 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% while those of Random and Nash are as high as around 35% 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 a more spread out of load sharing.



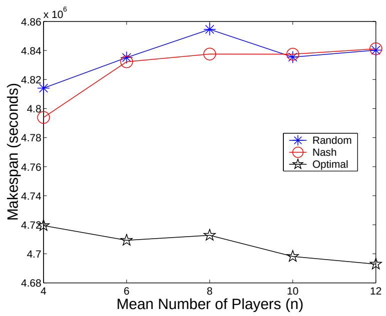
(a) RI versus α (b) RI evolution of the Nash strategy (c) RI evolution

Figure 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.

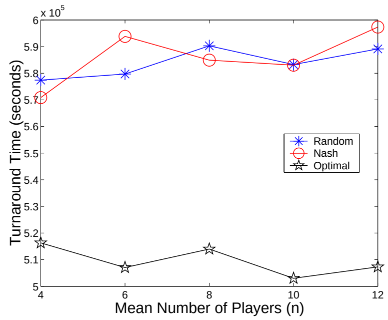
Figure 6(a) shows the variation of the RI with increasing mean number of players. We can see that the Optimal strategy is again quite robust. On the other hand, the Random and Nash strategies show a slight downward trend, indicating that as there are more players, a job is slightly more likely to be rejected, leading to a decreasing RI. This observation is reinforced by the results shown in Figure 6(b) which illustrates 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 Figure 6(c), the Optimal strategy continuously increases the RI as the system evolves.

6.2 Results over PSA Workload

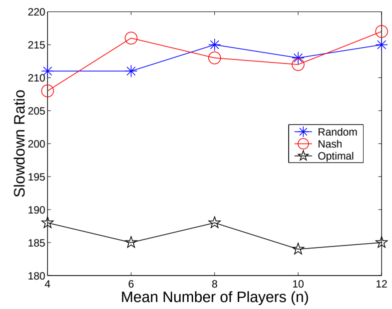
Now let us consider the results over PSA workload. Figure 7 shows the results for the PSA workload with various values of selfishness penalty factor α. Compared with the results for the NAS workload discussed above, the relative performance differences of the three strategies for the PSA workload are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



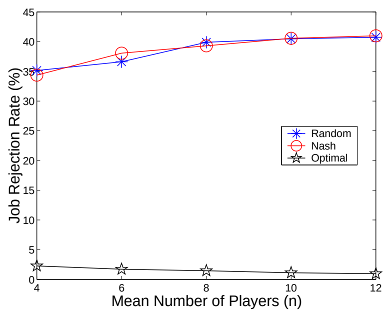
(a) Makespan



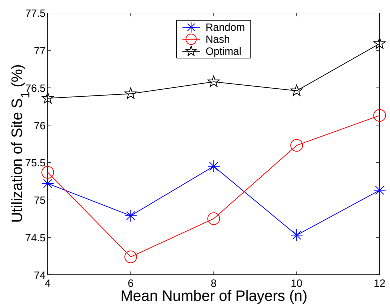
(b) Turnaround time



(c) Slowdown ratio

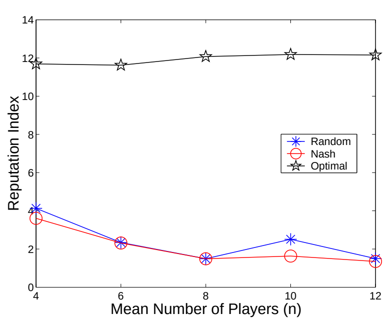


(d) Job rejection rate

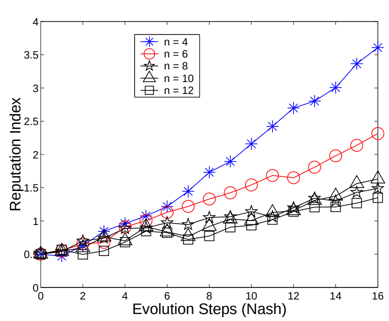


(e) Utilization

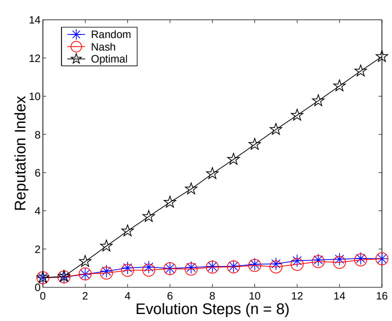
Figure 5: The NAS performance using the three game strategies (Random, Nash, and Optimal) under various values of the mean number of players n in a Grid site.



(a) RI versus n

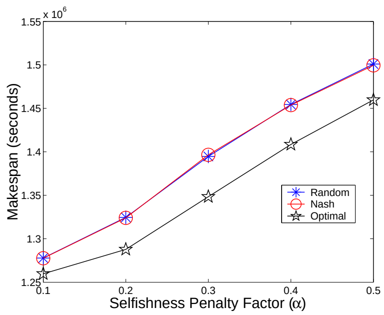


(b) RI evolution of the Nash strategy

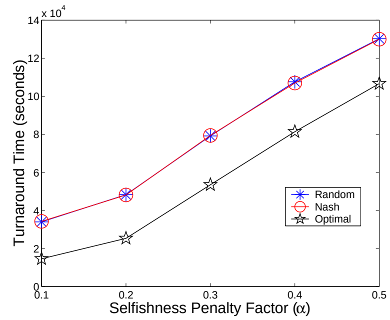


(c) RI evolution

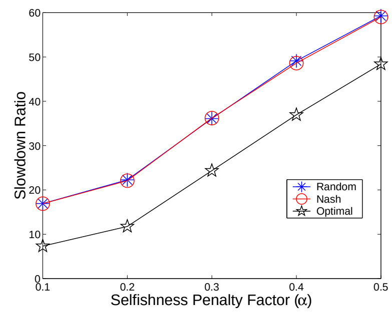
Figure 6: The variation of the Reputation Index (RI) for the NAS workload using the three game strategies (Random, Nash, and Optimal) under various values of the mean number of players n in a Grid site.



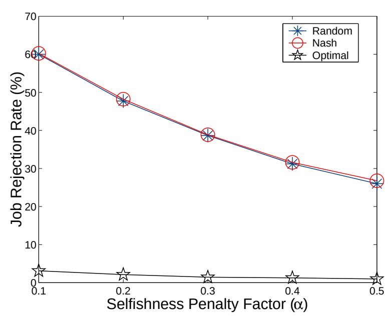
(a) Makespan



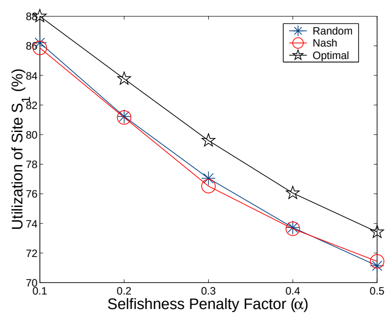
(b) Turnaround time



(c) Slowdown ratio

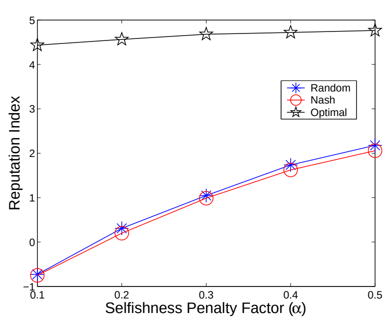


(d) Job rejection rate

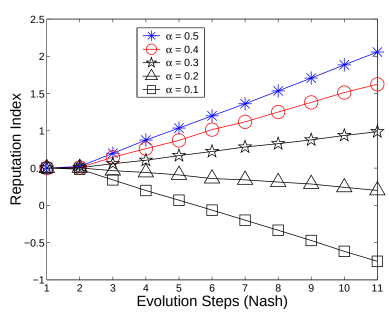


(e) Utilization

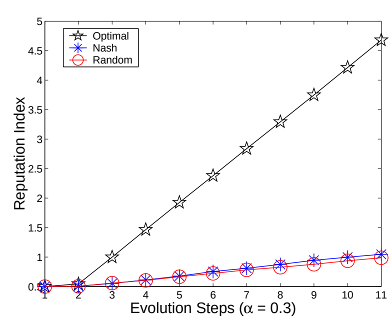
Figure 7: The PSA performance using the three game strategies (Random, Nash, and Optimal) under various values of the selfishness penalty factor α.



(a) RI versus α



(b) RI evolution of the Nash strategy

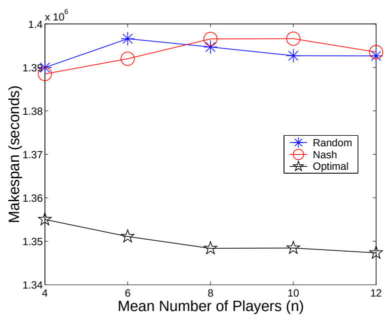


(c) RI evolution

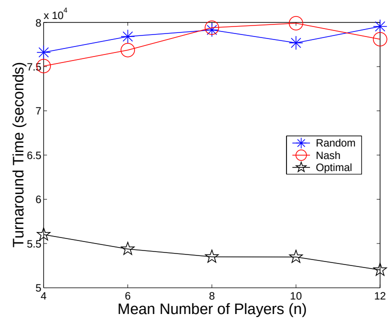
Figure 8: The variation of the Reputation Index (RI) for the PSA workload using the three game strategies (Random, Nash, and Optimal) under various values of the selfishness penalty factor α.

Figure 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 vs. 16,000), the values of RI are smaller. Nevertheless, the relative performance differences are similar.

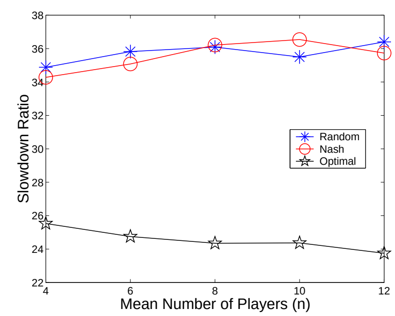
Figure 9 shows the results for the PSA workload with various mean numbers of players in a Grid site. The Optimal strategy also performs consistently better than the Random and Nash strategies. An interesting observation is that for the utilization performance, while Random slightly outperforms Nash for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



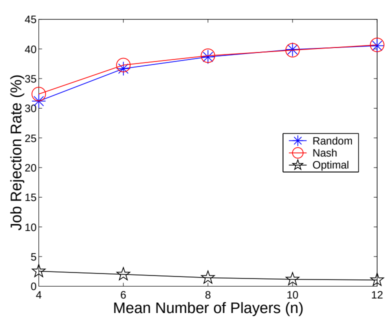
(a) Makespan



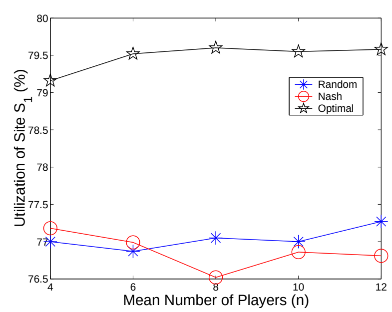
(b) Turnaround time



(c) Slowdown ratio



(d) Job rejection rate

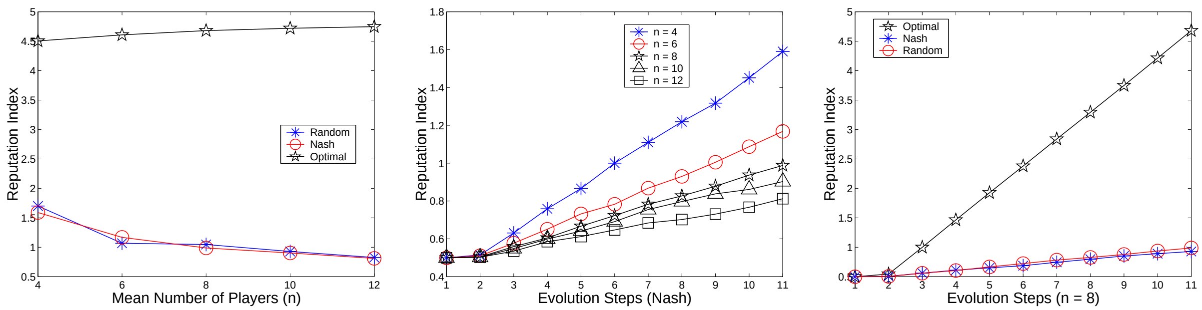


(e) Utilization

Figure 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.

Figure 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 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. Figure 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.



(a) RI versus n (b) RI evolution of the Nash strategy (c) RI evolution

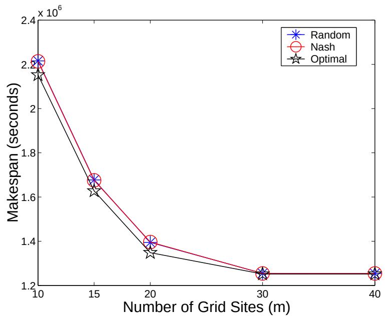
Figure 10: The variation of the Reputation Index (RI) for the PSA workload using the three game strategies (Random, Nash, and Optimal) under various values of the mean number of players n in a Grid site.

Finally, Figure 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.

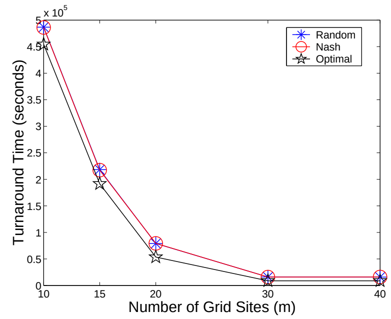
7 Conclusions

We have presented a novel and general hierarchical Grid computing model by taking machine selfishness into account. Our model matches well with 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 inter-site and intra-site levels. Using this model, we can formulate three different game theoretic frameworks for studying the behaviors of the Grid machines: intra-site execution game, intra-site execution time bidding game, and inter-site bidding game.

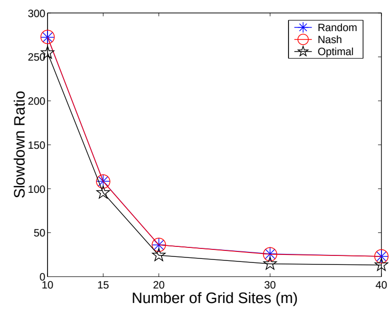
In this paper, we focus on the first framework to present a detailed mathematical analysis of the selfish behavior of individual machines within a Grid site. Nash equilibrium and optimal strategies are analytically



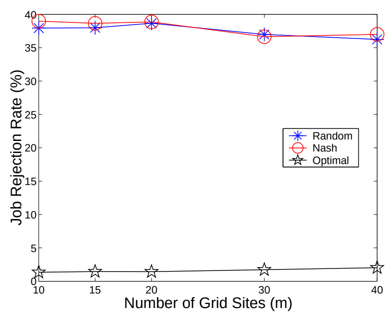
(a) Makespan



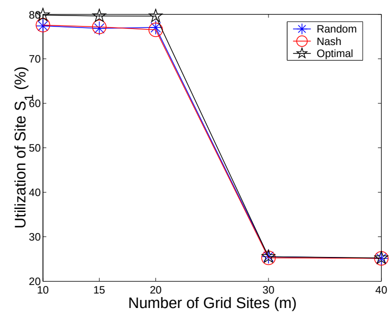
(b) Turnaround time



(c) Slowdown ratio

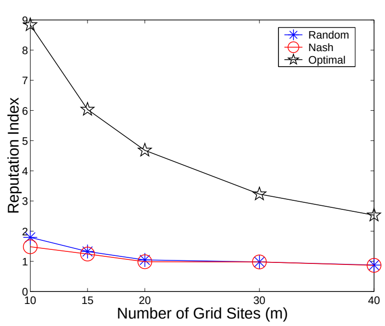


(d) Job rejection rate

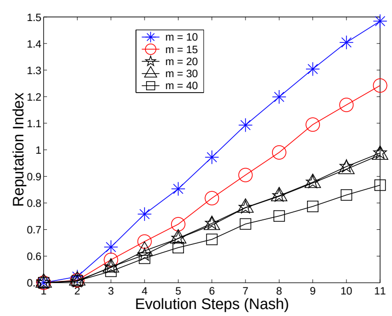


(e) Utilization

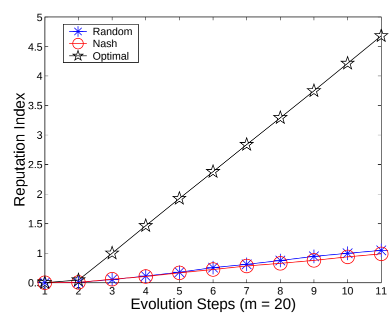
Figure 11: The PSA performance using the three game strategies (Random, Nash, and Optimal) under various values of the number of sites m in the Grid.



(a) RI versus m



(b) RI evolution of the Nash strategy



(c) RI evolution

Figure 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.

derived. Using the analytical results obtained, we have performed extensive simulations to study the overall system performance under a wide range of parameters. We have reached two important conclusions. Firstly, 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. Secondly, 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.

The game-theoretic model and the optimal strategies suggested in this paper are designed to rescue selfish Grids from becoming useless in real-life applications. A broader impact of this work is the restoring of faith of cooperative distributed computing in open Grids organized by virtual organizations. The performance results under Nash equilibrium and optimal strategies over the NAS and PSA workloads offer the first set of quantitative data towards the design and evaluation of practical computational, information, data, and business Grids. How to link the selfish factors in these Grids 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 [36].

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 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, June 2002, pp. 917–930.

[3] R. Aumann and S. Hart (Eds.), *Handbook of Game Theory with Economic Applications*, Volume 2, Elsevier, North Holland, 1994.

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

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

[6] T. D. Braun, H. J. Siegel *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, 2001, pp. 810–837.

[7] 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 Agents 2000*.

[8] R. Buyya, *Economic-based Distributed Resource Management and Scheduling for Grid Computing*, Ph.D. Thesis, Monash University, Melbourne, Australia, Apr. 2002.

[9] 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, Apr. 2002, pp. 208–216.

[10] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman, “Heuristics for Scheduling Parameter Sweep Applications in Grid Environments,” *Proc. HCW 2000*.

[11] K.-M. Chao, R. Anane, J.-H. Chen, and R. Gatward, “Negotiating Agents in a Market-Oriented Grid,” *Proc. CCGrid 2002*.

[12] A. Czumaj and A. Ronen, “On the Expected Payment of Mechanisms for Task Allocation,” *Proc. ACM PODC 2004*.

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

[14] P. Ghosh, N. Roy, S. K. Das, and K. Basu, “A Game Theory Based Pricing Strategy for Job Allocation in Mobile Grids,” *Proc. IPDPS 2004*.

[15] J. Gray, “Distributed Computing Economics,” Technical Report MSR-TR-2003-24, Microsoft Research, Mar. 2003.

[16] 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, Feb. 2004, pp. 77–84.

[17] T. Groves, “Incentive in Teams,” *Econometrica*, vol. 41, no. 4, July 1973, pp. 617–631.

[18] L. V. Kalé, S. Kumar, and J. DeSouza, “A Malleable-Job System for Time-Shared Parallel Machines,” *Proc. CCGrid 2002*.

[19] Y.-K. Kwok, S. Song, and K. Hwang, “Modeling and Strategies of Mixed Multi-Level Games in Selfish Grid Computing,” Technical Report, Internet and Grid Computing Laboratory, USC, Oct. 2004.

[20] K. Larson and T. Sandholm, “Miscomputing Ratio: Social Cost of Selfish Computing,” *Proc. AAMAS 2003*, pp. 273–280.

[21] D. Lehmann, L. I. O’Callaghan, and Y. Shoham, “Truth Revelation in Approximately Efficient Combinatorial Auctions,” *Journal of the ACM*, vol. 49, no. 5, Sept. 2002, pp. 577-602.

[22] V. Lo and J. Mache, “Job Scheduling for Prime Time vs. Non-Prime Time,” *Proc. Cluster Computing 2002*.

[23] R. Mahajan, M. Rodrig, D. Wetherall, and J. Zahorjan, “Experiences Applying Game Theory to System Design,” *Proc. ACM SIGCOMM 2004*.

[24] J. F. Nash, Jr., “The Bargaining Problem,” *Econometrica*, vol. 18, no. 2, Apr. 1950, pp. 155–162.

[25] J. F. Nash, Jr., “Non-Cooperative Games,” *The Annals of Mathematics*, Second Series, vol. 54, no. 2, Sept. 1951, pp. 286–295.

[26] N. Nisan, “Algorithms for Selfish Agents: Mechansim Design for Distributed Computation,” *Proc. 16th Symposium on Theoretical Aspects of Computer Science* (Lecture Notes in Computer Science, vol. 1563), pp. 1–17, 1999.

[27] N. Nisan and A. Ronen, “Algorithmic Mechanism Design,” *Proc. ACM STOC 1999*, pp. 129–140.

[28] S. Olafsson, “Games on Networks,” *Proceedings of the IEEE*, vol. 85, no. 10, Oct. 1997, pp. 1556–1562.

[29] M. J. Osborne and A. Rubinstein, *A Course in Game Theory*, MIT Press, 1994.

[30] C. H. Papadimitriou, “Algorithms, Games, and the Internet,” *Proc. ACM STOC 2001*.

[31] K. Ranganathan, M. Ripeanu, A. Sarin, and I. Foster, “Incentive Mechanisms for Large Collaborative Resource Sharing,” *Proc. CCGrid 2004*.

[32] O. Regev and N. Nisan, “The POPCORN Market—An Online Market for Computational Resources,” *Proc. ACM ICE 1998*.

[33] J. B. Rosen, “Existence and Uniqueness of Equilibrium Points for Concave N-Person Games,” *Econometrica*, vol. 33, no. 3, July 1965, pp. 520–534.

[34] T. RoughGarden and E. Tardos, “How Bad is Selfish Routing?,” *Journal of the ACM*, vol. 49, no. 2, Mar. 2002, pp. 236–259.

[35] R. Sami, *Distributed Algorithmic Mechanism Design*, Ph.D. Thesis, Yale University, Dec. 2003.

[36] S. Song and K. Hwang, “Security Binding for Trusted Resource Allocation in Computational Grids,” *IEEE Trans. Parallel and Distributed Systems*, under revision, November 2004.

[37] D. E. Volper, J. C. Oh, and M. Jung, “GameMosix: Game-Theoretic Middleware for CPU Sharing in Untrusted P2P Environment,” *Proc. PDCS 2004*.

[38] M. P. Wellman, J. K. Mackie-Mason, D. M. Reeves, and S. Swaminathan, “Exploring Bidding Strategies for Market-Based Scheduling,” *Proc. ACM EC 2003*.

[39] R. Wolski, J. S. Plank, T. Bryan, and J. Brevik, “G-Commerce: Market Formulations Controlling Resource Allocation on the Computational Grid,” *Proc. IPDPS 2001*.

[40] 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, Oct. 2000, pp. 667–678.

[41] M. K. H. Yeung, *On Channel Adaptive Wireless Cache Invalidation and Game Theoretic Power Aware Wireless Data Access*, M.Phil. Thesis, The University of Hong Kong, Aug. 2004.

Authors’ Biographical Sketches

Yu-Kwong Kwok received the B.S. degree in Computer Engineering from the University of Hong Kong, Hong Kong, in 1991, and the M.Phil. and Ph.D. degrees in Computer Science from the Hong Kong University of Science and Technology (HKUST), Hong Kong, in 1994 and 1997, respectively. He is an Associate Professor in the Department of Electrical and Electronic Engineering, University of Hong Kong. Dr. Kwok is currently on leave from the University of Hong Kong and is a Visiting Associate Professor at the University of Southern California. His research interests include Grid computing, mobile computing, wireless communications, network protocols, and distributed computing algorithms. Dr. Kwok is a member of the Association for Computing Machinery (ACM), the IEEE Computer Society, and the IEEE Communications Society. He is a Senior Member of the IEEE. He can be reached at *ykwok@hku.hk*.

ShanShan Song received her B.S. degree in Computer Science from Special Class for Gifted Young in University of Science and Technology of China in 2001. She is currently pursuing the Ph.D. degree in the Department of Computer Science at University of Southern California. Her research interest lies primarily in the area of trust management and resource allocation for computational Grids. She can be reached at *shanshan.song@usc.edu*.

Kai Hwang is a Professor and Director of Internet and Grid Computing Laboratory at the University of Southern California. He received the Ph.D. from the University of California, Berkeley. An IEEE Fellow, he specializes in computer architecture, parallel processing, Internet and wireless security, and distributed computing systems. He has authored or coauthored 7 scientific books and 180 journal/conference papers in these areas. Hwang is the founding Editor-in-Chief of the *Journal of Parallel and Distributed Computing*. Currently, he is also an Associate Editor of the IEEE Transactions on Parallel and Distributed Systems. He has performed advisory and consulting work for IBM Fishkill, Intel SSD, MIT Lincoln Lab., ETL in Japan, and GMD in Germany. Presently, he leads the NSF-supported ITR GridSec project at USC. The GridSec group develops security-binding techniques for trusted job scheduling in Grid computing, distributed IDS and pushback of DDoS attacks, fuzzy-logic trust models and selfish Grid Computing models, and self-defense software systems for protecting network-centric computing resources. Professor Hwang can be reached at *kaihwang@usc.edu* or through the URL: http://GridSec.usc.edu/Hwang.html.