Result Verification and Trust-based Scheduling in Open Peer-to-Peer Cycle Sharing Systems

Shanyu Zhao and Virginia Lo
Computer and Information Science Department
University of Oregon
Eugene, Oregon 97403
szhao,lo@cs.uoregon.edu

Abstract—**Systems that seek to harvest idle cycles available throughout the Internet are vulnerable to hosts that fraudently accept computational tasks and then maliciously return arbitrary results. Current strategies employed by popular cooperative computing systems, such as SETI@Home, rely heavily on task replication to check results. However, result verification through replication suffers from two potential shortcomings: (1) susceptibility to collusion in which a group of malicious hosts conspire to return the same bad results and (2) high fixed overhead incurred by running redundant copies of the task.**

In this paper, we first propose a scheme called Quiz to combat collusion. The basic idea of Quiz is to insert indistinguishable quiz tasks with verifiable results known to the client within a package containing several normal tasks. The client can then accept or reject the normal task results based on the correctness of quiz results. Our second contribution is the promotion of trust-based task scheduling. By coupling a reputation system with the basic verification schemes - Replication and Quiz - a client can potentially avoid malicious hosts and also reduce the overhead of verification. Furthermore, by adjusting the degree of result verification according to the trust value of a particular host, a client can tailor the system to achieve the desired level of accuracy.

Our mathematical analysis and simulation results show that Quiz greatly outperforms Replication in terms of accuracy and overhead under collusion assumptions. In non-collusion scenarios Replication is a better solution although Quiz also performs well. Reputation systems effectively improve the performance of both Quiz and Replication in a variety of cheating scenarios.

Keywords: Result Verification, Grid Computing, Trust, Reputation System, Peer-to-Peer

I. Introduction

In the converging realms of peer-to-peer networks and grid computing, peer-to-peer cycle sharing systems seek to harness idle computational cycles throughout the Internet, providing tremendous computing power for a range of scientific applications. The highly successful cooperative computing project SETI@Home [1] has more than 4.5 million users contributing their computer's idle cycles (mostly home PCs) and utilizes one thousand years of CPU time every day. This is equivalent to 15 Teraflops of compute power, exceeding the 12-Teraflops achievement of IBM ASCI White [2].

Our research project CCOF (Cluster Computing On the Fly) [3] supports the formation of cycle sharing communities by aggregating machines on the edge of the Internet into a shared resource pool, providing on-demand cycles to a wide range of client applications. CCOF goes beyond the current client-server-based cycle sharing systems, such as SETI@home and the Stanford Folding project [4], assuming a pure peer-to-peer cycle sharing model, in which each peer is potentially a host or a client (or both). Four classes of applications have been addressed in CCOF: infinite workpile, workpile with deadlines, tree-based search, and point-of-presence application. Other research projects investigating the peer-to-peer cycle sharing paradigm include [5][6][7][8]

In contrast to institution-based resource sharing systems, such as Globus [9] and Condor [5], peer-to-peer cycle sharing systems operate in an open, untrusted, and insecure environment. Under a system in which hosts volunteer their cycles, a malicious or irresponsible participant could potentially subvert the computing community by accepting computational tasks (or *Work Units* in SETI@Home, or *Bag-of-Task* in the OurGrid project [6]) from other peers and return arbitrary or imprecise results. For example, it has been reported that the SETI@Home project suffered from cheating by some volunteers who faked the number of work units completed in order to gain higher ranking on the website list of top donators [10]. Molnar also described a problem in which unauthorized patches to make the SETI code run faster actually returned incorrect results [11]! It is quite conceivable that in the future when "selling spare cycles" becomes a common business practice, this cheating problem will wreak serious havoc.

Current strategies against this result cheating problem can be roughly divided into two categories: generic solutions and specialized solutions. Generic verification schemes can be utilized by a wide range of applications while specialized solutions are tailored to specific types of computational tasks. For example, encrypted functions [12] and ringer schemes [13] have been proposed to prevent cheating in strictly defined environments. To the best of our knowledge, result verification through replication is the only generic solution.

Replication sends out copies of the computational task to multiple hosts, compares all the results, and relies on majority agreement to determine the correct result. However, this intuitive and straightforward solution has two drawbacks: (1) It works on the assumption that there is no collusion among malicious hosts. In the absence of collusion, the chance that two malicious hosts will return the same bad result is negligible, making the replication method virtually 100% accurate. However, a group of colluding hosts can easily communicate and return the same bad results by using, for example, a DHT (Distributed Hash Table) to store the hash value of tasks and the corresponding bad result to be returned. The wide distribution of unauthorized patches mentioned earlier also produced consistent bad results across multiple hosts. (2) Replication has a fixed high overhead: for each computational task, multiple redundant tasks are computed. This duplication of effort may not be sustainable in a limited cycle sharing environment.

Our research investigates the effectiveness and efficiency

Research supported in part by NSF ANI 9977524.

of result verification schemes in an open untrusted cycle sharing environment. Through mathematical analysis and simulation, we evaluate the performance of replication-based result verification, and compare it to a new scheme we have developed called Quiz whose goal is to combat collusion. In order to further enhance the efficacy of Reputation and Quiz, we propose the notion of *trust-based scheduling*, in which the basic verification scheme is coupled with a reputation system to help clients select reliable hosts and to reduce the overhead of verification. We also introduce the idea of *accuracy-on-demand* for cycle sharing systems: by tuning the result verification parameters appropriately, a client can tailor the system to achieve a desired level of accuracy at the cost of greater (or less) overhead.

Our analysis and simulation results reveal that:

- Quiz greatly outperforms Replication under the assumption of collusion. In the absence of collusion, Quiz lags a little behind Replication.
- The use of a reputation system helps accuracy converge to 1 over time whereas it significantly reduces the overhead imposed by the Replication or Quiz scheme.
- Adding blacklists into reputation systems improves the efficacy of Quiz but hurts Replication.

II. Result Verification: Replication VS. Quiz

Before giving our mathematical analysis of the performance of Replication vs. Quiz, we describe the cheating models assumed and give a more detailed description of the operation of the Replication and Quiz result verification schemes.

A. Cheating Models

The cheating models we presume for this study combine those proposed in previous work [13] [14]. We define three types of cheaters and two cheating scenarios.

The three basic types of cheaters are:

- **Type I: Foolish malicious hosts** that always return wrong results.
- **Type II: Ordinary malicious hosts** that return wrong results with a fixed probability. Type I is an extreme example of Type II.
- **Type III: Smart malicious hosts** that will perform well for a period of time to accumulate a good reputation before finally returning bad results with a fixed probability.

The other dimensions of our cheating models address collusion and the lifetime of cheaters.

- **Colluding vs. Non-colluding Scenario**. Colluding cheaters return the same wrong result if they are chosen to compute the same task. Non-colluding cheaters are assumed to each return a unique (wrong) result.
- **Dynamic vs. Static Scenario**. Under a dynamic model, the malicious host will periodically leave and rejoin the community with a new ID, in order to erase its notorious history. In the static model, malicious hosts remain in the system under the same ID and thus can be neutralized using Black Lists.

B. Replication: A Voting Principle

Under Replication, a client node selects one task at a time from its task input queue, and given *replication factor* k, chooses $k + 1$ distinct hosts to ship the same task to. When all the chosen hosts have finished the task and returned the results, the client compares the results and if more than h of the hosts agree on the same result, it accepts that result as the correct one. If there is no dominant result, the client will discard all the results and reschedule that task later.

C. Quiz: A Sampling Principle

The basic idea of Quiz is to insert indistinguishable quiz tasks with verifiable results known to the client within a package containing several normal tasks. The client can then accept or reject the normal task results based on the correctness of quiz results. The Quiz scheme assumes that it is impossible for a host to discriminate quizzes from normal tasks and that the cost to verify quiz results is trivial.

More precisely, under Quiz, given package size s and quiz ratio r, the ratio of quizzes to tasks, a client fetches t tasks from its task queue, and mixes them randomly with m quizzes, where $t + m = s$. The client then ships the whole package, either serially or in one transmission, to a chosen host for execution. Upon completion of all tasks and quizzes in that package, the client checks the results for the hidden quizzes. The client accepts all tasks in a package only if every quiz was correctly computed; otherwise, the client will discard all tasks in that package, and reschedule the tasks later.

Note that in reality the concept of package in Quiz could be implemented by a result buffer. The client ships tasks serially with mixed quizzes but collects results with a buffer. One challenging problem of Quiz is that it requires an efficient method to generate verifiable quizzes. The most straightforward approach is for the client to run one of the tasks itself so that it knows the correct results. We consider this an open problem and will investigate it in the future.

D. Differences between Replication and Quiz

Replication can be regarded as a "horizontal" voting while Quiz can be seen as "vertical" sampling. Each has scenarios for which it is clearly superior with respect to accuracy or overhead alone. Under no collusion, Replication is 100% accurate while with Type I cheaters Quiz can rely on a single quiz for detection. Quiz does not suffer from collusion since only one host is involved. In other scenarios, however, the accuracy and overhead tradeoffs are complex. We will give a detailed probability and statistics based analysis in section III. If we take a scheduling view of Quiz and Replication, we see that Quiz has unfavorable *turnaround time* (the time to complete computation and data transfer for a task or a batch of tasks[15]). Quiz's turnaround time depends on the package size. In a system with an abundance of hosts, Replication's turnaround time is exactly one round, except when rescheduling is called for. In a resource-limited system, Replication may have to serialize its voting, thus increasing turnaround time.

III. Mathematical Analysis

We analyze the behvior of Replication and Quiz under the assumptions of Type II malicious hosts, those that cheat with fixed probability b. (Type I is covered by the case $b = 1.0$.)

We consider a large pool of hosts in which p fraction are malicious. Thus, $p*b$, denoted by q, expresses the probability of attaining a bad result when randomly choosing a host.

We evaluate these schemes using two metrics: *accuracy* and *overhead* as defined in Table I. For overhead we only address the computational overhead, because the network communication time for CPU-intensive workpile applications is negligible compared to task execution times, which is normally measured in hours or tens of hours [15].

Symbol	Explanation
p	fraction of malicious nodes in the system
b	the probability that a malicious node returns a bad result
q	p*b, the probability of attaining a bad result when randomly choosing a host
k	(for Replication) replication factor, a client send one task and k replicas to (k+1) hosts
r	(for Quiz)quiz ratio, the number of quizzes divided by the number of normal tasks
s	(for Quiz)package size, the total number of normal tasks + quizzes sent to a host
m	(for Quiz)the number of quizzes in a package, equal to $\frac{rs}{r+1}$
Accuracy	$\frac{\#\text{ of tasks with correct results}}{\#\text{ of tasks accepted by the client as correct}}$
Overhead	$\frac{\#\text{ of extra copies of tasks or quizzes executed}}{\#\text{ of tasks accepted by the client as correct}}$

TABLE I
NOTATIONS FOR ANALYZING REPLICATION AND QUIZ

Replication in No Collusion Scenario. If there is no collusion, the chance that two malicious hosts return the same bad result is negligible, and thus the accuracy of replication should be 1. Now let O_{RN} represent the overhead for Replication with No Collusion. The calculation of overhead can be broken down into two parts. The probability of getting a majority consensus on the first round of task scheduling, denoted by α, is $\sum_{i=0}^{k/2} q^i(1-q)^{k+1-i}C_i^{k+1}$, and overhead in this situation is just k. If no majority is achieved in the first round, a second attempt to schedule the same task is performed. The overhead in this situation is $k+1+O_{RN}$. By coupling the two situations, we have an equation $O_{RN} = \alpha * k + (1-\alpha)(k+1+O_{RN})$, from which we can get:

$$O_{RN} = \frac{k+1}{\sum\limits_{i=0}^{k/2} q^i(1-q)^{k+1-i}C_i^{k+1}} - 1 \tag{1}$$

Replication in Type II Collusion Scenario. Assuming that all the malicous hosts selected by the client return the same result, the probability that good results dominate is $\sum_{i=0}^{k/2} q^i(1-q)^{k+1-i}C_i^{k+1}$. The probability that bad results take the majority is $\sum_{i=0}^{k/2}(1-q)^i q^{k+1-i}C_i^{k+1}$. When there is a tie (only when k is an odd number), meaning the number of good results equals the number of bad result, the task must be rescheduled. Thus, the accuracy for Replication with Collusion, denoted by A_{RC}, and the overhead for Replication with Collusion, denoted by O_{RC}, is as follows:

$$A_{RC} = \frac{\sum\limits_{i=0}^{k/2} q^i(1-q)^{k+1-i}C_i^{k+1}}{\sum\limits_{i=0}^{k/2} q^i(1-q)^{k+1-i}C_i^{k+1} + \sum\limits_{i=0}^{k/2}(1-q)^i q^{k+1-i}C_i^{k+1}} \tag{2}$$

$$O_{RC} = \frac{k}{\sum\limits_{i=0}^{k/2} q^i(1-q)^{k+1-i}C_i^{k+1} + \sum\limits_{i=0}^{k/2}(1-q)^i q^{k+1-i}C_i^{k+1}} \tag{3}$$

Quiz - A Simple Model. Quiz is not affected by collusion since only one host is involved. The calculation of the probability that a client accepts bad results under Quiz, denoted by F_Q, could be broken down to three parts. First, when a good host is selected(with probability $1-p$), the client will never get bad results; second, when a malicious host is selected and gives correct results to quizzes(with combined probability $p*(1-b)^m$), the probability of accepting bad results is b; third, when a malicious host is selected but fails on quizzes(with combined probability $p*(1-(1-b)^m)$), all the tasks should be rescheduled. Thus the probability that the client is cheated in this situation is F_Q. Therefore we have the equation $F_Q = p*(1-b)^m*b + p*(1-(1-b)^m)*F_Q$. By resolving this equation we can calculate the accuracy for Quiz in the presence of Type II cheaters as given in Equation (4). The calculation of overhead can also be broken down to two situations, like we have done in calculating O_{RN}. The probability that a package will be accepted in one round of scheduling is $(1-p)+p(1-b)^m$. The total overhead is given in equation (5).

$$A_Q = \frac{1-p+p(1-b)^{m+1}}{1-p+p(1-b)^m} \tag{4}$$

$$O_Q = \frac{r+1}{(1-p)+p(1-b)^m} - 1 \tag{5}$$

Quiz - A General Model. In reality every malicious host may have a different probability of cheating. Let random variable β represent this probability value, and β falls into a distribution whose density function is $f(x), x \in [0,1]$. For every possible value(denoted by x) of random variable β, the corresponding accuracy could be calculated as: $p(1-x)^m(1-x) + p(1-(1-x)^m)A_Q$. By counting every situation pertaining to possible values of β, we have:

$$\begin{aligned} A_Q = \;& 1-p+p\int_0^1 f(x)(1-x)^{m+1}\mathrm{d}x + \\ & pA_Q\int_0^1 f(x)[1-(1-x)^m]\mathrm{d}x \end{aligned}$$

By resolving this equation, we can get the relation between A_Q, m and the distribution of β, as shown in equation (6). Following the similar process we can calculate the overhead given in equation (7). This general model will be utilized to analyze and implement a technique called *Accuracy on Demand* in section IV.

$$A_Q = \frac{p\int_0^1 f(x)(1-x)^{m+1}\mathrm{d}x + 1 - p}{p\int_0^1 f(x)(1-x)^m\mathrm{d}x + 1 - p} \tag{6}$$

$$O_Q = \frac{r+1}{1-p+p\int_0^1 f(x)(1-x)^m \mathrm{d}x} - 1 \qquad (7)$$

The next figures visually illustrate these equations developed for Type II cheaters for Replication and Quiz(we only show simple model here). For all the graphs, we set *k*=*r* (replication factor equal to quiz factor), yielding comparable overhead for both schemes. In these graphs we set the parameter b, probability of returning a bad result to 0.5. This is a fair choice for comparison of the two schemes since at the extremes of $b = 0.0$ and $b = 1.0$, Quiz will have 100% accuracy while Replication will have the worst performance at $b = 1.0$.

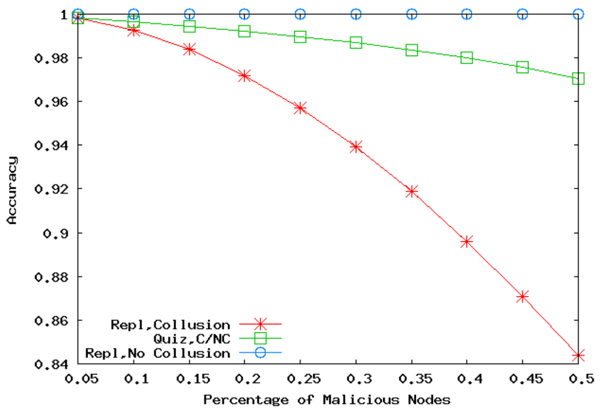

Fig. 1. Accuracy vs. Percentage of Malicious Hosts, k=r=2

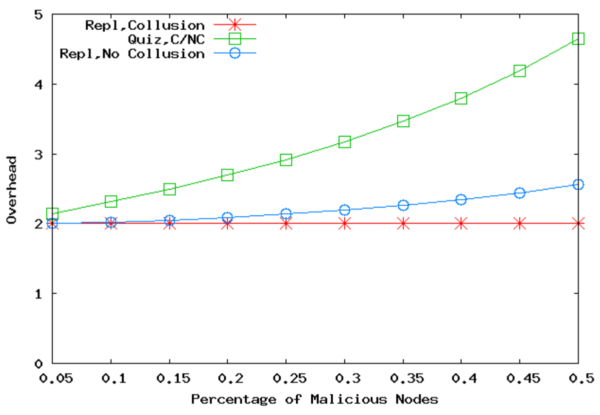

Fig. 2. Overhead vs. Percentage of Malicious Hosts, k=r=2

As depicted by Figure 1, under collusion when the fraction of malicious hosts increases, the accuracy of Replication drops off dramatically, whereas the accuracy of Quiz declines gradually. The accuracy of Quiz lies in between that of Replication with collusion and Replication with no collusion. Figure 2 shows the corresponding overhead which increases as expected with increasing percentage of cheaters. Intuitively, increasing the quiz ratio or replication factor increases accuracy but at the cost of higher overhead. This is verified by Figure 3, where both Quiz and Replication with collusion slowly gain higher accuracy as the quiz ratio or replication factor is increased. It is worth noticing that when quiz ratio or replication factor is smaller than 8, which is a practically

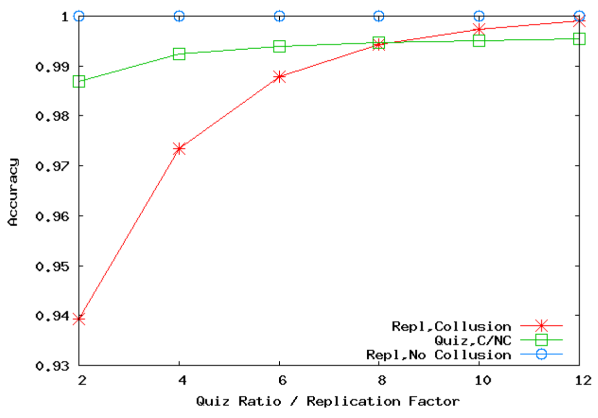

Fig. 3. Accuracy vs. Replication/Quiz Factor, 30% of Type II, k=r=2

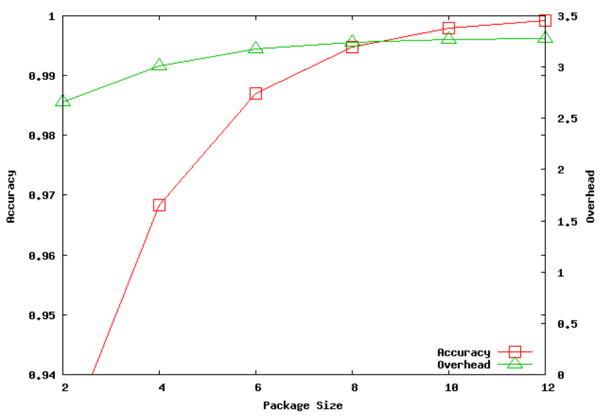

Fig. 4. Accuracy & Overhead vs. Package Size in Quiz, r=2

large value, the accuracy of Quiz is much better than Replication under the collusion assumption. Figure 4 shows the tradeoff between accuracy and overhead, with respect to increasing package size. As package size increases accuracy increase reapidly while overhead grows gradually. However, in reality, we have to consider the prolonged *turnaround time* as a side-effect of larger package size.

IV. Trust-based Scheduling

The philosophy behind trust-based scheduling is called "trust but verify", which we believe should be the tenet of future design of applications in open peer-to-peer networks. Trust-based scheduling couples a result verification scheme with a reputation system. By using a reputation system, a client can reduce the chance of selecting malicious hosts, thereby increasing overall accuracy. At the same time, by giving priority to reliable hosts, the overhead will be reduced.

A. *Trust-based Scheduling System Model*

Figure 5 depicts our system model of trust-based scheduling. The Task Scheduler fetches tasks from the task queue, and selects a set of trusted hosts using the Reputation System. The Reputation System contains a Trusted List, a list of candidate host nodes, each with a trust rating, and an optional Black List, a list of host nodes known to be malicious. The Result Verification Module uses its Trust List to select reliable hosts. It then inserts quizzes or disseminates replicas, and verifies the results of quizzes or replicas. The Reputation System is updated based on the consequence of this verification, i.e., the trust values of hosts in the Trust List are updated.

The Result Verification Module uses two functions in its operations. Based on the reputation value of a given host, this module calculates the appropriate quiz ratio or replication factor using a *quiz ratio function* or *replication factor function*. These functions generate quiz ratios (replication factors) that are lower for more highly trusted nodes, reflecting the reduced need to verify the results from trustworthy hosts. The Results Verification Module uses a second function, the *trust function*, to update hosts trust values after verifying the returned results. The functions we use are described in Section V.

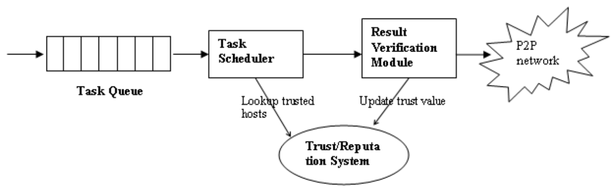

Fig. 5. Trust-based Scheduling Model

B. Reputation System Classification

There are two dimensions involved in the computation of a host's trust value. Formation of *local trust values* is controlled by a *trust function*, which computes the trust value based on direct transactions with the host to be rated. Previous work on reputation systems [14] [16] either uses the cumulative number of successful transactions, or uses the highest satisfaction value in the near past. Formation of *aggregated trust values* uses a number of methods ranging from simple averages to sophisticated mathematical functions.

Reputation systems can be characterized by the *degree* of trust information aggregation:

- Local Sharing. Each peer in the system only uses information based on its own interactions with hosts that have volunteered to do work for that peer. There is no information sharing about trust values with other peers.
- Partial Sharing. Each peer shares information with a subset of all of the peers. The most common type is information sharing with neighboring nodes in the overlay (or physical) network.
- Global Sharing. This presumes a mechanism to gather information from everyone in the peer-to-peer community to calculate a global trust value for a particular host and to share the trust values among all peers.

Intuitively, as the level of cooperation becomes broader from local sharing to global sharing, reputation systems will gain performance, but at the cost of increased network traffic and communication overhead. For cycle sharing systems, a more global reputation system is appropriate because the time comsumed executing a single task is several orders of magnitude higher than the amount of network traffic generated.

Based on this taxonomy, we study five different reputation systems in order to quantify the performance gains achievable from the use of a reputation system for trust-based scheduling.

(1) Local Sharing Strategy:

- **Local Reputation System**. Every peer maintains its own trusted list and black list separately, which are purely based on direct transactions with others in the past. Peers never exchange any trust information with others. Clearly, this implementation is the least expensive in terms of network traffic.

(2) Partial Sharing Strategies:

- **NICE Reputation System** [16]. NICE is similar to local reputation system except that when deciding the reputation value for a previously unrated peer, NICE performs a tree-based search on the trust graph that is logically formed using the trust relationship among peers. By traversing the trust graph, it builds a chain of trust and the new trust value is based on this trust path. This scheme has high overhead in the searching process.
- **Gossip Reputation System**. Peers periodically gossip about host trust values with the most trusted hosts in their own Trust List. The traffic overhead is controlled by adjusting the gossiping frequency and number of gossiping partners.

(3) Global Sharing Strategies:

- **EigenTrust Reputation System** [14]. EigenTrust proposed a way of aggregating trust values reported by every peer in the system to form a normalized global reputation value for each peer. The original EigenTrust scheme required recalculating the reputation value vector each time a single trust value was updated by some peer. This incurs a high overhead in communication and computation.
- **Global Reputation System**. All peers share a single Trust List and optional Black List. Trust values reported by individual peers are aggregated using simple a function (see Simulation section for details).

C. Trust-based Scheduling Algorithms

Traditional reputation systems only provide a service to look up the reputation value for a single peer. We extend these reputation systems to return a list of most trusted *free* hosts. Whenever a request is received, the trusted list is traversed from the most trusted to least trusted hosts to find currently available hosts.

Table II and and Table III give the pseudo codes for Trust-based Replication and Trust-based Quiz. These algorithms contain two parts: *Scheduling* uses the reputation system to choose the hosts with the highest trust rating. *Verification and Updating Reputation System* updates the trust values upon receiving correctly or incorrectly completed tasks as discerned from the replication voting process or the quiz results.

D. Accuracy on Demand

Applications in peer-to-peer cycle sharing system normally have different goals and properties, thus requiring different levels of accuracy. Instead of gambling on the unpredictable accuracy which depends on the behavior of malicious hosts, most clients want to have a desired level of certainty concerning the correctness of their tasks. This is what we called *Accuracy On Demand(AOD)*. *AOD* ensures that the obtained accuracy is above a demanded level, by dynamically adjusting quiz ratio or replication factor according to the reputation value of a given host.

```
Trust-based Replication
1. Scheduling
 while(task queue not empty):
  for each task in task queue:
   a. fetch a most trusted free host(aid by repu sys);
   b. calculate the replication factor $k$, and determine
   the # of extra hosts needed, say $c$;
   c. pick $c$ most trusted free hosts(aid by repu sys),
    if(not enough hosts are allocated) then
     stall for a while;
2. Verification and Updating Reputation System
 for each task scheduled:
  upon receiving the result set from all replicas:
   a. cross check the results,
    if(# of majority results > $(k + 1)/2$) then
     accept the majority result;
    else
     reject all results, reschedule that task later;
   b. update reputation system,
    if(task result is accepted) then
     for each host $H$ who was chosen in replication:
      if(host $H$ gave majority result) then
       increase trust value for $H$;
      else if(host $H$ gave minority result) then
       decrease trust value for $H$;
```

TABLE II
TRUST-BASED REPLICATION ALGORITHM

```
Trust-based Quiz
1. Scheduling
 while(task queue not empty):
   a. fetch a most trusted free host $H$(aid by repu sys);
   b. calculate the quiz factor $r$, and determine
    the # of tasks to schedule, say $t$,
    and the # of quizzes to be inserted, say $i$;
   c. pick $t$ tasks from task queue, mixed with $i$ quizzes
2. Verification and Updating Reputation System
 for each package scheduled:
  upon receiving all results from the host $H$:
   a. check the results for quizzes in that package,
    if(all results for quizzes are correct) then
     accept all the results in that package;
    else
     reject all results, reschedule those tasks later;
   b. update reputation system,
    if(task result is accepted) then
     increase trust value for $H$;
    else
     decrease trust value for $H$;
```

TABLE III
TRUST-BASED QUIZ ALGORITHM

In trust-based scheduling, the accuracy a client will gain depends on two factors. First, the reputation system helps to select a host with trust value v. This ensures some level of accuracy. Second, the verification scheme itself also can statistically guarantee a certain level of accuracy based on the quiz ratio r or replication factor k. Therefore, given a particular host with trust value v, the result verification module in *AOD* will automatically select the appropriate quiz ratio or replication factor to meet the desired level of accuracy.

We use Quiz to illustrate the calculation of the proper quiz ratio m' to achieve a demanded level of accuracy A. Using Equation (6), we can solve for *m*, the total number of quizzes needed to achieve an accuracy value A. When selecting a host that has previously correctly computed *v* quizzes, we can compute $m' = m - v$, the additional number of quizzes needed to achieve accuracy A. We can keep track of v by using the linear trust function described in [14] in which the trust function increments a node's trust value *v* by one for each correctly computed quiz.

Note: the percentage of malicious node p and the distribution of β in Equation (6) is unknown for the clients. We need to conduct real system measurements or invent automatic detection techniques for p and β distribution in the future. For now, we just assume a rather dangerous scenario where $p = 0.5$ and β falls into uniform distribution between 0 and 1 (means $f(x) = 1$). Take these assumptions, we can have the relation between m' and A, as:

$$m' = \sqrt{\frac{1}{1-A}} - v - 2 \qquad (8)$$

Equation (8) demonstrates that given the trust value of a host, if we want to achieve an accuracy A, how many quizzes should be contained in one package. Notice that when the required number of quizzes exceeds the configured package size, a client will just send a whole package of quizzes, to avoid the risk of lower accuracy than the desired level.

V. SIMULATIONS

A suite of simulations were conducted in order to verify the effectiveness of the trust-based scheduling model and compare the performance of Replication and Quiz in a variety of scenarios. Our simulator evaluates the two result verification schemes combined with the five reputation systems described earlier. Our experiments simulate the macro view of cycle sharing systems, including the task generation model, task execution by peers, fraudulence of task results, etc., but does not model low level activities such actual computation or network traffic.

A. Simulation Models

A.1 Cycle Sharing Model

We adopt a dedicated cycle sharing model in our simulation, in which every peer has no private computing burden and devotes all its cycles to client applications. In reality, a peer in a cycle sharing system might only donate its computing resources when it is in idle state, e.g. in screensaver mode.

In our simulations, tasks have the same length in terms of run time. We model a homogeneous system in which every peer has the same computing power. Each peer performs both as a client who generates tasks and as a host who computes tasks. A host can only conduct one task at a time and if it accepts *n* tasks, it keeps busy for *n* rounds. We simulate a system with a total number of 1000 nodes and some of them are configured as Type I, Type II or Type III malicious nodes as described in section II.

The topology of the peer-to-peer cycle sharing overlay network is not a critical factor in our simulation, because we don't measure traffic overhead. Thus, we assume an underlying routing infrastructure which connects every pair of peers. Whenever a node wishes to contact other unknown nodes, it randomly chooses nodes from the overlay network. This random peer selection occurs when a node recruits new hosts to its trusted list, e.g. during bootstrap period or upon exhausting its trusted list.

A.2 Task Generation Model

We use synthetic task generation models based on classic probability distributions. Table IV lists two synthetic task generation models used in our simulation: *Syn1* and *Syn2*.

Syn1 uses a normal distribution for the number of tasks generated per round and an exponential distribution for inter-arrivals of task generation events. *Syn2* is based on our analysis of a real trace from the Condor [5] load sharing system. We found that the total number of tasks generated by one client during a long period (e.g. 72 hours) falls into an exponential distribution. This means that many peers only generate very few tasks while few peers produce large numbers of tasks. *Syn2* models this skewed task generation pattern. Our simulation results didn't show significant difference between these two workload models. Thus, we only show results from Syn1.

The expected value μ of the exponential distribution of task inter-arrival periods is a simulation parameter. In section D we will modify this value in order to change system load.

	# of tasks	**task runtime**	**inter-arrivals**
Syn1	Normal $\mu = 20$	1 round	Exponential $\mu = 50rounds$
Syn2	Normal $\mu \sim$ Exponential	1 round	Exponential $\mu = 50rounds$

TABLE IV
TWO SYNTHETIC MODELS OF TASK GENERATION

A.3 Reputation System Models and Trust Functions

The trust values used in our simulation lie in the interval [0,1], where 0 means most untrustworthy and 1 means most trustworthy. Initially, all peers have the default trust value of 0. When an unknown peer is recruited to a host's trust list, the default trust value for that peer is also 0.

The trust function defines how a client should adjust the local trust value for a host after checking the quiz or replication results. Previous work [14] proposed a linearly increasing function after each successful transaction. We modeled three trust functions as follows:

Linear Increase Sudden Death (LISD). Increase the trust value linearly when successfully verifying a quiz or replication result, and clear the trust value to 0 upon a failed quiz or false replication.

Additive Increase Multiplicative Decrease (AIMD.) Increase the trust value linearly upon success, and decrease the trust value by half upon a failure. It is observed in the simulation that this function performs better for replication under the assumption of collusion.

Blacklisting. Increase as in LISD or AIMD. After a verification failure, put the bad host into a blacklist.

A.4 Quiz Ratio and Replication Factor Functions

A linear function serves as both the quiz ratio and replication factor function. When the trust value is less than a certain value, this function gives a linearly decreased quiz ratio or replication factor as the trust value increases. After the trust value exceeds the threshold, a constant value of quiz ratio or replication factor will remain as a strategy fighting the Type III malicious behavior.

A.5 Metrics

We use the same metrics used in our mathematical analysis of Replication and Quiz scheme in Section III. The following lists the definitions used in our simulations:

$$Accuracy = \frac{\#\ of\ tasks\ with\ correct\ results}{\#\ of\ tasks\ accepted\ by\ the\ clients}$$

$$Overhead = \frac{\#\ of\ quizzes\ or\ replicas + \#\ of\ rejected\ tasks}{\#\ of\ tasks\ accepted\ by\ the\ clients}$$

B. Simulation Results

B.1 Contribution of Reputation Systems

In order to show the contribution of various reputation systems, we experimented with the five reputation systems discussed in section IV: Local, NICE, Gossip, EigenTrust and Global, listed from the least information sharing(local) to the highest degree of knowledge sharing(global).

In the following graphs, we show the evolution of accuracy and overhead over time. The X-axis represents the total number of finished tasks throughout the system. At the end of the simulation, on average 500 tasks are executed for each client. The Y-axis gives the average accuracy and overhead.

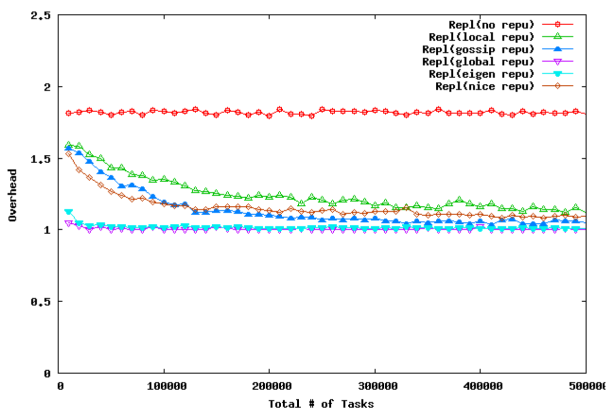

Fig. 6. Overhead reduced over time, for Replication with reputation systems, replication factor is 1, 30% Type II malicious nodes.

Figure 6 shows the impact of adding a reputation system to Replication under non-colluding scenario. We clearly see

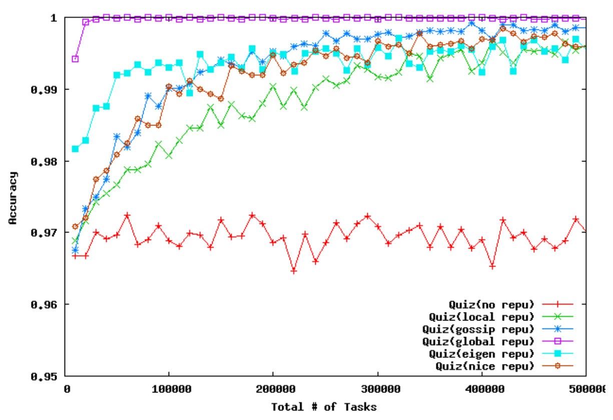

Fig. 7. Accuracy converges to 1 over time, for Quiz with reputation systems, quiz ratio in [0.05,1], 30% Type II malicious nodes.

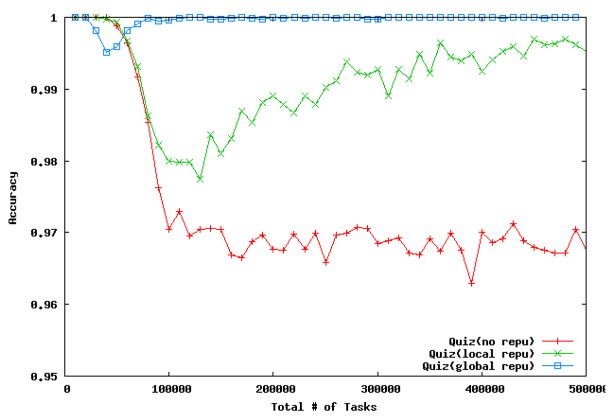

Fig. 9. Accuracy recovers over time, for Quiz with reputation systems, quiz ratio in [0.05,1], 30% Type III malicious nodes.

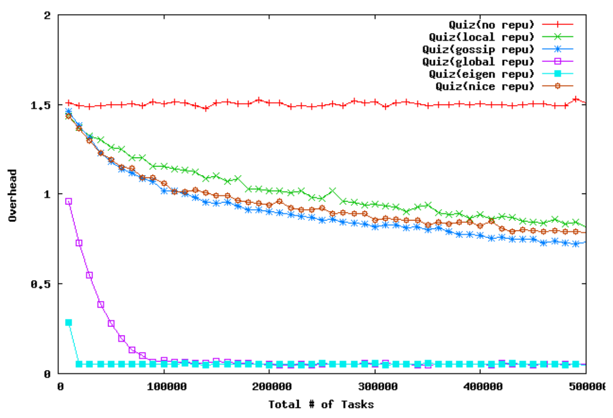

Fig. 8. Overhead reduced over time, for Quiz with reputation systems, quiz ratio in [0.05,1], 30% Type II malicious nodes.

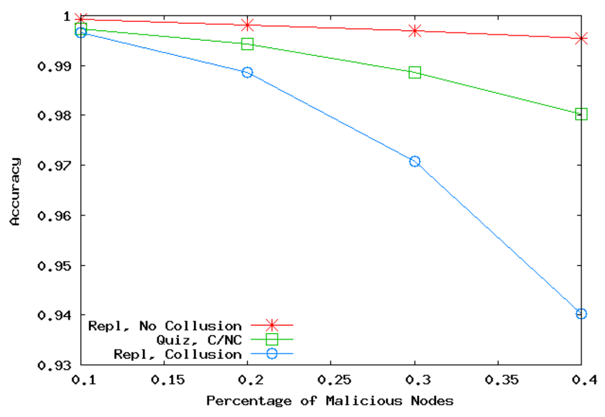

Fig. 10. Accuracy on different percentage of colluding hosts, calculated among the first 500000 tasks submitted to the system from the beginning

the declining overhead over time in the replication scheme, as the reputation system becomes mature and guides the client to select the trustworthy host to replicate tasks, reducing the chance of rescheduling. The two global reputation systems brings the overhead close to the ideal value of 1.0. Note the consistently high overhead of replication without reputation system.

Figures 7 and 8 shows the improvement of accuracy and overhead over time in the trust-based Quiz scheme. As can be seen, the accuracy obtained through trust-based Quiz is gradually increased whereas the overhead is firmly reduced. Note that the accuracy converges to 100% while the corresponding overhead converges to a minimum level. It is conceivable that for a long lived system, the accuracy can potentially reach 100% without causing excessive overhead. Also note that this strong performance is achieved with very small values of quiz ratio.

To see how Type III malicious hosts influence trust-based scheduling, see Figure 9. The dip in the graph corresponds to the moment when the cheater switches from good to bad behavior. Note that Quiz with global reputation system recovers and quickly converges to 1; Quiz with local reputation system recover much more slowly; without a reputation system, Quiz cannot recover from this switch. This figure shows that a minimum quiz ratio or replication factor must be maintained to fight for Type III malicious hosts.

B.2 Replication VS Quiz

In this section, we investigate how a reputation system affects the relative ranking of performance of Replication and Quiz. Figures 10 and 11 show the accuracy and overhead of the three scenarios: *Replication under No Collusion*, *Quiz*, and *Replication under Collusion*. The addition of reputation systems to the basic verification schemes doesn't change the *relative* performance of Quiz and Replication (see Figures 2 and 3 from the theoretical analysis). But it clearly improves both accuracy and overhead in all the three scenarios. Another point worth noticing is that the relative improvement of Quiz overhead is prominent compared to the replication scheme. This is because Quiz is more efficient in feeding verification outcomes to reputation systems. Quiz can definitely detect whether a host is cheating for every instance of verification, whereas Replication sometimes just throw tasks away without updating the reputation system because there is no dominating result.

Notice that the range of quiz ratio and replication factor is set to [0.05,1], meaning sometimes a client will not replicate at all and directly accept the task result. That is why *Replication with No Collusion* can not always achieve 100% accuracy.

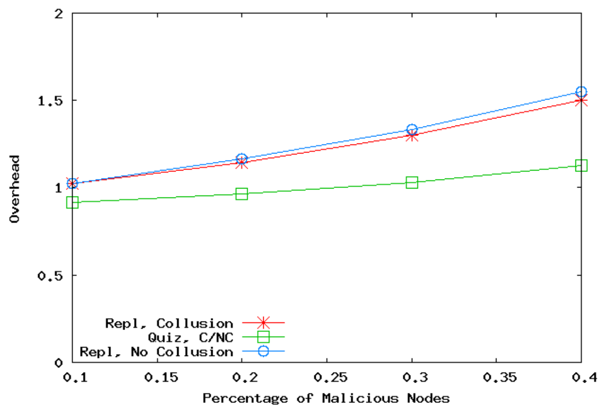

Fig. 11. Overhead on different percentage of colluding hosts, calculated among the first 500000 tasks submitted to the system from the beginning

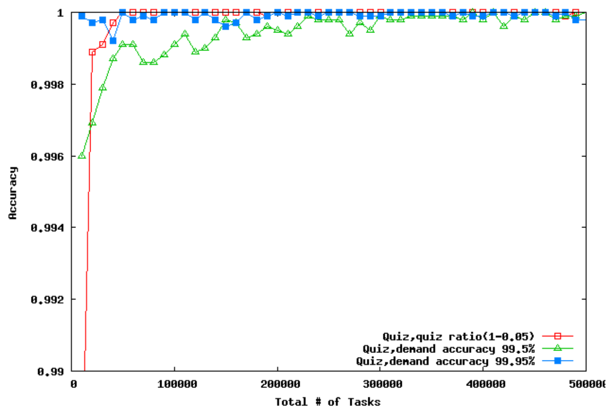

Fig. 13. *Accuracy on Demand* ensures the demanded accuracy, for Quiz with global reputation system, 50% Type II malicious nodes.

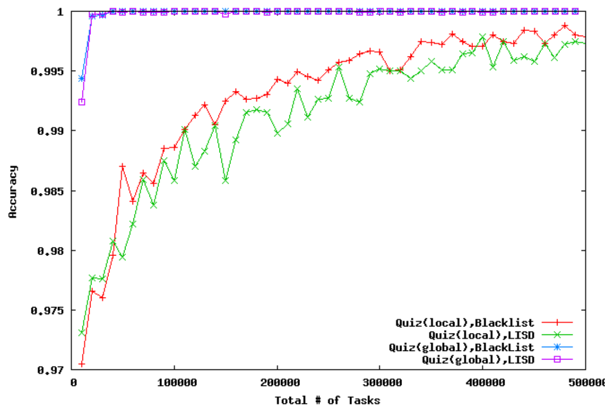

Fig. 12. Accuracy is improved by adding blacklist to Quiz with reputation systems, 30% Type II malicious nodes

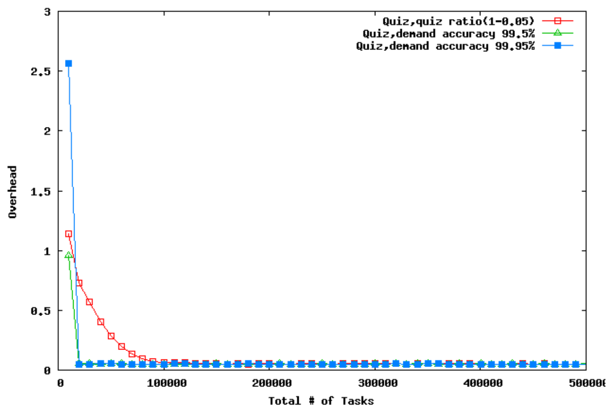

Fig. 14. *Accuracy on Demand* incurs high overhead in the beginning, for quiz with global reputation system, 50% Type II malicious nodes.

B.3 The Impact of Blacklist in Reputation Systems

Since Quiz scheme can be very confident in detecting malicious hosts, a blacklist can be utilized to avoid a client choosing a malicious host in the future, thus further improving the performance of Quiz.

Figure 12 shows the greater accuracy gained by using a blacklist in Quiz. We also observed that overhead decreases, but we don't show the graph due to space limits. However, it is interesting to note that a Black List has negative consequences for performance of Replication since colluding nodes in the majority will cause Replication to put honest nodes on the Blacklist!

B.4 Accuracy On Demand

To exemplify the effectiveness of *Accuracy on Demand(AOD)*, figure 13 shows a comparison of accuracy over time among three configurations: linear quiz ratio function, 99.5% *AOD* and 99.95% *AOD*, under a system with half Type II malicious hosts. We only use Quiz with global reputation system to illustrate that the accuracy of *AOD* is solidly maintained above the demanded level, whereas Quiz with linear quiz ratio function has accuracy as low as 98% at the beginning, which is beyond the scope of the graph.

Figure 14 shows the corresponding overhead. The large overhead costs for *AOD* in the beginning of simulation falls into our expectation. Actually, to guarantee a certain level of accuracy, *AOD* has to issue huge number of quizzes for unfamiliar hosts. This is the situation at the start of our simulation when every host has the reputation value of 0. But in an evolved system where good hosts already have high reputation, *AOD* will send fewer quizzes to trusted hosts, resulting in a lower overhead.

By using accuracy on demand, we argue that trust-based scheduling can provide an arbitrarily high accuracy without the cost of long-term high overhead (like traditional replication scheme does).

VI. Related Work

A. *Result Verification Schemes*

The result verification problems exist in every distributed computing system. However, most of the previous work such as mobile agent system seldom considers the cheating on computation a serious problem, instead, the protection of host machine from malicious mobile code is extremely studied. Sander and Tschudin [12] took an opposite road to seek for a method to protect mobile agent from malicious hosts. Their scheme requires an encrypted function E(f). The client sends two functions f(x) and E(f)(x) to the host. After getting the results, it performs a comparison of P(E(f))(x) and f(x) , and the equality of this two value demonstrates the honest computation. As pointed out in their paper, finding an encrypted function for a general function is very difficult.

Golle and Mironov [13] proposed a ringer scheme in distributed computing for a special type of computation: they assume f(x) is a strictly one-way function, and host should compute f(x) for all x in a domain D. The ringer scheme allows the client to compute several "checkpoint" value $y_i = f(x_i)$, and send all the y_i to the host to expect the value of corresponding x_i (which is known in advance to the client). Du et. al. [17] extends this scheme to achieve uncheatable grid computing. They require a host who compute f(x) in the domain of D save all the intermediate result of f(x) and build a merkle tree to prove the honest computation for every input x. However, this scheme is weak in that the building of the huge merkle tree is costly and it only combats the cheating of incomplete computation.

B. Reputation Systems

Trust or reputation mechanisms [18] [19] are effective in enhancing the reliability of shared resources in the p2p file-sharing systems, in which the quality of the resources is measurable after the resource trading. The basic idea is to appoint every node a trust value based on its history behavior, and save that value properly in the system. Kamvar et al. [14] proposed a scheme that uses a DHT to calculate and store the trust value of each node. Hash functions are used to determine the mother nodes (more than one) of each node, which take care of calculation and storage of the trust value of that node. The main drawbacks of this scheme are the maintenance of the consistency among the mother nodes and the vulnerability of the mother nodes since everyone knows who its mother nodes are.

Singh and Liu [20] designed a more secure scheme which utilized a smart public key algorithm to query trust value anonymously. If a peer A want to know another peer say B's trust value, it floods a query to the p2p network, then only peer B's THA (denotes for trust holding agents) nodes return the digital signed trust value. The bootstrap servers are in charge of the assignment of THA nodes to each peer, along with the public-private key pairs. So the THA nodes are protected since nobody knows the THA nodes of a particular peer. But the message flooding makes this scheme un-scalable.

A more scalable and efficient scheme was proposed by Lee et al. [16], which could infer reputation ranking for a particular node based on a chain of trust. Every node only stores the trust value for those nodes that had direct transaction with it. When a node needs to know a stranger node's reputation value, it only queries its honest friends, and in turn its friends would query their friends. As the distance increases, the weight of the returned trust value decreases. The weak point of this scheme is the assumption that the recommendation of a trusted friend is also trustworthy. This assumption does not hold if a malicious peer always conducts transaction honestly while cheats on reputation inference.

VII. Conclusion and Future Work

We have presented a sampling-based result verification scheme called Quiz as an alternative approach for Replication, especially in systems with colluding cheaters. We also proposed *trust-based scheduling* in open cycle sharing systems by coupling verification schemes with reputation systems. A client in such system can dynamically adjust the level of verification based on a host's reputation, thus ensuring arbitrarily high accuracy. The philosophy behind this paper is called "trust but verify". We believe this tenet should be taken into serious consideration for the future design of solutions in open peer-to-peer networks.

Our future work involves three parts. First, to make Quiz a reality, we need to find efficient and scalable mechanisms for generating quizzes. Second, a further study of *Accuracy on Demand* for a variety of reputation systems and verification schemes should be conducted. Finally, we would like to invent a cooperative scheme to detect or estimate the fraction of malicious nodes and the distribution of malicious nodes' behavior to make *Accuracy on Demand* more precise.

References

[1] "Seti@home: The search for extraterrestrial intelligence project," http://setiathome.berkeley.edu/.

[2] Andy Oram, Ed., *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*, O'Reilly & Associates, Sebastopol,CA,USA, 2001.

[3] Virginia Lo, Daniel Zappala, Dayi Zhou, Yuhong Liu, and Shanyu Zhao, "Cluster computing on the fly: P2p scheduling of idle cycles in the internet," in *IPTPS*, 2004.

[4] "Folding@home distributed computing project," http://www.stanford.edu/ group/ pandegroup/ folding/.

[5] "Condor project," http://www.cs.wisc.edu/condor/.

[6] "Ourgrid project," http://www.ourgrid.org/.

[7] Keir A Fraser, Steven M Hand, Timothy L Harris, Ian M Leslie, and Ian A Pratt, "The xenoserver computing infrastructure," in *Technical Report UCAM-CL-TR-552, Cambridge University, UK*, 2003.

[8] Yun Fu, Jeffrey Chasey, Brent Chunz, Stephen Schwabx, and Amin Vahdat, "Sharp: An architecture for secure resource peering," in *SOSP'03*, 2003.

[9] Ian Foster and Carl Kesselman, Eds., *The Grid:Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, San Francisco,USA, 1999.

[10] Andrew Colley, "Cheats wreak havoc on seti@home: participants," ZDNet Australia, 2002. http://www.zdnet.com.au/ news/ security/ 0,2000061744,20269509,00.htm.

[11] David Molnar, "The seti@home problem," *E-Commerce*, 2000.

[12] T. Sander and C. Tschudin, "Protecting mobile agents against malicious hosts," in *G. Vigna (ed.) Mobile Agents and Security, LNCS*, 1998.

[13] Philippe Golle and Ilya Mironov, "Uncheatable distributed computations," in *Proceeding of RSA Conference, Cryptographer's track*, 2001.

[14] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina, "The eigentrust algorithm for reputation management in p2p networks," in *Proceedings of the Twelfth International World Wide Web Conference*, 2003.

[15] Derrick Kondo, Henri Casanova, Eric Wing, and Francine Berman, "Models and scheduling mechanisms for global computing applications," in *International Parallel and Distributed Processing Symposium*, 2002.

[16] Seungjoon Lee, Rob Sherwood, and Bobby Bhattacharjee, "Cooperative peer groups in nice," in *IEEE Infocomm*, 2003.

[17] Wenliang Du, Jing Jia, Manish Mangal, and Mummoorthy Murugesan, "Uncheatable grid computing," in *ICDCS*, 2004.

[18] Stephen Marsh, *Formalising Trust as a Computational Concept*, Ph.D. thesis, University of Sterling, 1994.

[19] Karl Aberer and Zoran Despotovic, "Managing trust in a peer-2-peer information system," in *ACM CIKM*, 2001.

[20] Aameek Singh and Ling Liu, "Trustme: Anonymous management of trust relationships in decentralized p2p systems," in *IEEE International Conference on Peer-to-Peer Computing (ICP2PC)*, 2003.