Social Control Mechanism to coordinate an Unrreliable Agent Society

H. Haidarian Shahri and M. R. Meybodi

Faculty of Computer Engineering and Information Technology
Amirkabir University of Technology (Tehran Polytechnic), Tehran, Iran
hhaidarian@aut.ac.ir

Abstract. In multiagent systems a common problem is how to assign tasks to other agents. It is very desirable to be able to guarantee the error rate of a solution in a multiagent system's society. In this paper, a novel approach for this problem has been introduced by devising social control mechanisms and simulating their mathematical model. It is also shown that the multiagent system is able to achieve any desired and predetermined threshold of correctness for the final solution, regardless of the performance of selfish and unreliable agents in the society or any stipulation about their honesty, which is extremely critical and problematic in the design of open and real world multiagent systems, today.

1. Introduction

In today's complex and distributed systems, the fitting solution for development of a practical application is to effectively utilize a flexible multiagent system (MAS), in which a large number of *self-interested* autonomous agents with different design objectives take advantage of other agents' skills, to solve a problem. There are many issues involved in the engineering of an agent society and integration of separately designed agents, so that they can work together in an unreliable and distributed environment.

One of the important points that must be considered in a large agent society is, how to choose between a huge number of options available in the allocation of tasks to agents. This is of particular interest in an environment were the agents are totally unknown and there is no central agent or general source of information like a directory for querying, to find out about the *reliability* and *characteristics* of agents in the system. To date, this case has not been addressed thoroughly because in most of the real world applications, the agents in a MAS had been designed and implemented by the same group of people. For example in the production of a multiagent system for discovering an unknown planet, the designers are aware of the capabilities and the reliability and honesty of agents, which are working in the system. However, with the growth of geographically distributed agents, communicating through WANs, like many ecommerce applications, there is a very urgent demand for building a mutual *trust* between agents in a society.

A variety of fault tolerance mechanisms have previously been suggested for reducing hardware and software errors in legacy computing systems and specially distributed systems [4], but usually, they are not quiet applicable in the agent context because of their overhead and inefficiency, and also do not have any considerations for a society. So, we prefer to propose the more descriptive term of "social control mechanisms" instead. In this paper, three different social control mechanisms have been devised for assuring correctness of a solution in a multiagent system. The task allocating agent is totally *unaware* of the society that he is living in. It assigns the tasks to other agents and gathers the results. It will gradually learn about the honesty of his partners. Nevertheless, he can *guarantee* any level of accuracy for the final solution, which consists of accumulating intermediate results from untrustable and self-interested agents in the environment.

In the next section, the related work on ensuring correctness is presented. Section three provides the mathematical model used for creating an agent society. Section four is a detailed analysis of three different social control mechanisms, which the last one provides a general and flexible framework for modeling agent behavior. Simulation and evaluation of these mechanisms are described in section five. The last part is the conclusion and some directions for future work are also suggested in section six.

2. Related Work

Indeed the conceptual and practical tools needed for building dependable agent systems, resilient to errors and other unexpected situations, are still at an early stage of research. Better methods are needed to develop multiagent systems that can guarantee correctness, reliability, and robustness in an agent society. This fact has given rise to the arrangement of specialized agent conferences with the theme of agent societies, e.g. [1], [2].

Turner and Jennings have very recently started a project on agents for mobile communication environments and are currently exploring integrity and correctness issues [3]. Using formal transformation systems for multiagent system synthesis is one way, to meet this growing need [8]. Defining boundaries in the behavior space by leveraging the goal hierarchy has also been used for validating complex agent behavior [9]. [6] discusses the development of an agent system having a computational model with correctness criteria. Instead of hardwiring robustness and fault-tolerant behavior into agent plans, notions of correctness are embedded at the semantic level. Verification can then be undertaken at the desired level of abstraction. [5] describes a design for the integration of AgentScape, a multi-agent system support environment, and DARX, a framework for providing fault tolerance in large scale agent systems. Although, these works are related to fault tolerance, they have all used different approaches for achieving it, and do not capture the spirit of a real multiagent community, i.e. they do not try to model the agent society.

3. Mathematical Model

Consider that the master agent is going to solve a large *problem*. The problem is made up of *tasks* which are executed in *batches*. A new batch begins only after the termination of the previous batch. Each task has a *result*. The final *solution* of the large problem consists of the accumulation of these results. The problem solving agent starts to distribute the tasks of one batch from the pool of tasks, between untrusted agents in the environment, until there are no more tasks in the batch. When the results of tasks in the batch are gathered, the agent starts distributing the next batch of tasks, which can use the results of the previous batch.

Error rate, *e*, is the ratio of the incorrect results accepted to the total number of results. For simplicity, it is assumed that batches are independent and the results of one batch do not affect the next one. The aim is to design a mechanism to ensure a pre-determined non-zero acceptable error rate, *e*, for the solution of the large problem. This rate is very dependent on the type of application in which the multiagent system is used. For example, in a large scale and complex simulation application, different tasks are allocated to the agents. There is no information on how well the agents will solve the tasks; nevertheless, we can guarantee the probability of the final solution of the simulation to be correct. Assume that acceptable error rate for a large problem is e=0.01, i.e. it would only acceptable, if the solution to the problem is wrong in 1% of the cases, and the problem consists of 10 batches with 10 tasks each. If tasks are dependent and a failure in one task causes the whole problem to fail, then the probability of one task to fail must be less than $e/(10*10) = 0.0001$. Although this number might seem small, in section four, it is shown that it can be achieved with a small redundancy increase.

A fraction of malicious agents, *m*, of the total agents' population returns wrong results, without actually doing the assigned tasks or maybe doing it incorrectly for some reason. The master agent which allocates tasks might know *m*, or can assume an upper bound for it. If *m* is higher than the assumed upper bound, then the master does not guarantee any degree of correctness for the final problem. Since there is no assumption about the speed or computational power of other agents, each result could arrive from any of the agents with equal probability. Therefore, without using any error preventative mechanism, the probability of error for the result of a task would be *m*.

The malicious agents are modeled as Bernoulli processes having a probability of *s* (sabotage rate), for producing a bad result, which is constant in time and the same for all malicious agents. It is assumed that the worker agents do not communicate with each other and can not agree on when to give a bad result.

However, when malicious agents return wrong results, they agree on the wrong result to allow voting for malicious agents, i.e. they produce the same but incorrect result for a task. If this assumption does not hold, we would expect more correct results than the guaranteed rate.

The criteria, which are used for comparing the efficiency of different preventative mechanisms, are *redundancy* and *slowdown*, while meeting the pre-determined desired threshold for error rate. Redundancy is the ratio of the total number of tasks assigned, to the original number of tasks. In an analogous manner, slowdown is defined as the ratio of the execution time using the mechanism, to without using it. Although the two criteria are related, if agents can leave the computation or become blacklisted in the middle of a batch, the slowdown would increase but the redundancy would be the same. Social control mechanisms should generally minimize or reduce the final error rate to an acceptable level and at the same time, minimize slowdown and redundancy.

4. Analysis of Social Control Mechanisms

The master agent decomposes a big problem into a pool of tasks. Then it allocates tasks from the pool using eager scheduling in a round robin fashion, and a task could be reallocated to different agents in a *redundant* fashion. Since one task is done by several agents, any agent in the society can leave the computation *without* previous notice, without effecting the whole computation. The slow agents are overtaken by faster ones, because when the required results for one task are collected from faster agents, the task is marked as done and no further computation is required for it.

4.1. Voting

The master agent allocates tasks to different worker agents redundantly. As soon as *v* equal results (votes) from a total of *2v-1* allocations, for the same task are returned, the task is marked as done. When all the tasks in a batch are done, the master accumulates the results and starts distributing the next batch. The redundancy in this method is *v/(1-m)*, where *m* is the fraction of malicious agents. The final error rate in voting is due to wrong results forming a majority which is:

$$e(m,v)=\sum_{i=v}^{2v-1}\binom{2v-1}{i}m^{i}(1-m)^{2v-1-i}\approx cm^{v} \qquad \textbf{(1)}$$

As proven in [11] this error rate shrinks *exponentially* with *v*, as shown in figure 1.

This method performs well, if the fraction of malicious agents, *m*, is low such that they can not easily form a majority. With this assumption, the error rate can be reduced to very low levels with a small increase in redundancy. The weakness of this method is its large overhead, when *m* is large, because *v* has to be higher, to meet the acceptable error threshold. Having a large *m* is a probable situation in an open agent society. More importantly, redundancy can not be reduced to less than two, even when *m* is zero (no malicious agents). Therefore, it can only be used in cases where, we have an honest society and the population is very large, i.e. we have lots of trustable resources.

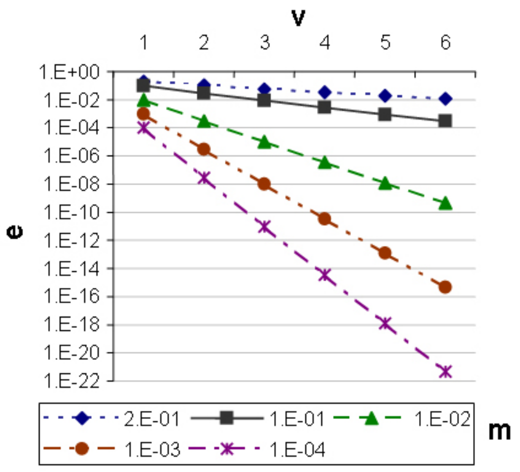


Fig. 1. Error rate of voting for different values of *v* and *m*

4.2. Verification

In the method that we call verification, instead of asking for the result of a task from at least two agents, we might sometimes allocate a task with a known result to an agent to verify its reliability. However, the agents are not aware of when they are going to be tested, with a task that the result of it is already known to the master. It is like scaring students into taking an unplanned quiz, while actually, not always planning to do so. This way the teacher ensures that the students are working correctly, without incurring too much slowdown in class progress. If a malicious agent replies with a wrong result to a known task, its results would be *backtracked* to the beginning of the batch. The agent could also be *blacklisted* and no more tasks would be assigned to it. In this case, if the probability of verifying an arbitrary agent is *p*, redundancy will be *1/(1-p)*, which has a lower bound of one instead of two for the previous method.

By utilizing *blacklisting*, the errors could only be caused by agents, which survive until the end of the batch, and the error rate is:

$$e(p,n,m,s) = \frac{sm(1-ps)^n}{(1-m)+m(1-ps)^n} \qquad (2)$$

where *p* is the probability of verifying an agent, *n* is the number of tasks assigned to an agent from a batch, *m* is the malicious fraction of an agent population, and *s* is the sabotage rate of an agent. See figure 2, which demonstrates the error rate for various values of *s*. By approximation [7], the above function has an upper bound of:

$$e^*(p,m,n,s) = \frac{m}{(1-m)} \cdot \frac{1}{pne} \qquad (3)$$

where *e* is the base of the natural logarithm, and the maximum is at:

$$s^* = \min(1, \frac{1}{m(n+1)}) . \qquad (4)$$

If the agent can change its sabotage rate (*s*) and set it to the optimal value, the error rate would decrease *linearly* with the length of the batch, i.e. *n*, and in time. Hence, it is advantageous for the master to make the batches longer. Nevertheless, that is in the worst case only and the malicious agent does not have the required information to set *s* in such a way.

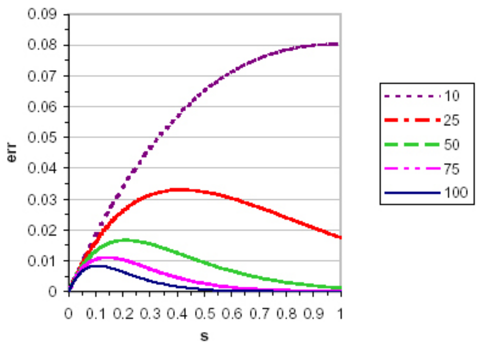


Fig. 2. Verification with blacklisting: theoretical error rate (*e*) at *p* = 0.1 and *m* = 20% for various values of *s* and *n*

It is not always possible to enforce *blacklisting*, since the malicious agent might hide or forge its identity and IP in a network. In this situation, the error rate is:

$$e(p,n,m,s) = m.\frac{1}{n}.s.\text{Average length of "last run"} \qquad \mathbf{(5)}$$

$$= \frac{ms}{n}\left[n(1-ps)^n + \sum_{i=0}^{n-1} i(1-ps)^i.ps\right]$$

Assuming that enforcing the agent to stay, until the end of the computation is not done, and if the agent stays in the computation for *l* tasks (*l*<*n*) and rejoins immediately under a new identity when caught, then by approximation, the error rate has an upper bound according to the following relation:

$$e^*(p,l,m,s) < \frac{m}{pl} \qquad \mathbf{(6)}$$

which is much worse than the case of when using blacklisting. See figure 3, for an illustration of the error rate. The error rate decreases *linearly* with the length of run (*l*) that, the agent participates in the computation. Here, error rate is significantly higher than with blacklisting. This is caused by unreliable agents, contributing to the results and leaving the computation early, before getting caught. Therefore they increase the error rate and it would be desirable to somehow increase *l*. Since malicious agents can return again in the same batch, the situation would become worsened. To prevent the return of malicious agents, sign-in delays could be enforced. If the sign-in process is delayed until the next batch, then the malicious agents do not gain any benefit from leaving the computation early.

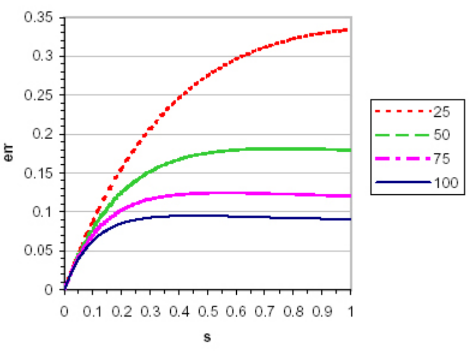


Fig. 3. Verification without blacklisting: theoretical error rate (e) at p = 0.1 and m = 20% for various values of s and n

4.3. Honesty

Now, the notion that comes to mind is to use voting on top of verification to exponentially reduce the already linearly reduced error rate and to achieve better result correctness for the same redundancy. This section culminates with an effective framework for *combining* any social control mechanism similar to the two, mentioned above, or any other possible future mechanism. Hence, this method is open to *variations*.

A parameter that, we call honesty is assigned to every agent in the society, in analogy to our own *human society*. If the result of a task is only accepted when the conditional probability of that result being correct is higher than a threshold *p*, then the probability of accepting a correct result, averaged over all tasks is at least *p*. This principle is critical in ensuring the correctness. Meaning that, by accepting results from agents which are honest enough, the correctness of the final *solution* can be ensured, for any *desired non-zero error rate*. The honesty of an agent is determined by monitoring his behavior and testing him. Obviously the newly arriving agents are unreliable and have a low degree of honesty. Honesty might also depend on the agent society and the master's prediction about the fraction of malicious agents in the population.

There are four types of objects in the system: *agent*, *result* coming from an agent, *result group* and *task*. Figure 4 is a visualization of this method, showing the four types of objects, the round-robin allocation of tasks from a pool to agents, and how the probability of correctness is computed as later explained. The honesty of an agent determines the probability of correctness for the result, passed by him. The master might receive different results from agents in the society, and a result group consists of all *matching* results for the same task. Each task could have several result groups. The probability of correctness for the results in a result group, determine the conditional probability of correctness for the result group for a particular task, as explained in equation 8. The probability of correctness of the *best* result group is assigned to its task. The task will be accepted and marked as done, only when it has reached the desired probability of correctness for the task, which is equal to the determined threshold for the final solution.

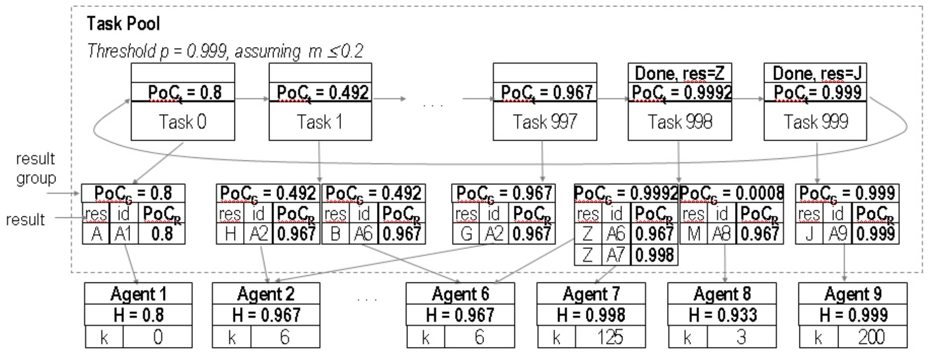


Fig. 4. Honesty framework and assignment of a pool of tasks to agents

While the master is assigning tasks from a batch, the probability of correctness for a result group in the system increases, if the agent passes verification tests, or a matching result arrives from another agent (vote), or both. By time, the probability of correctness for a result group will reach the desired threshold. The task related to that result group will be marked as done and it will not be reassigned by the master. This method is very efficient, because if the result is arriving from an honest agent, then it will not require voting and it reduces redundancy.

To assign the initial honesty value to agents, if verification is not used, the honesty can be set to *H=1-m* (*m*: the fraction of malicious agents). By using the verification method, the more tests the agent passes, the better his honesty would be. With blacklisting, the only bad results are from agents which survive the verification test. The probability of receiving a bad result from an agent, after surviving *k* verification tests is (similar to equation 2):

P (result from Agent is bad | Agent survived *k* verification tests) (**7**)

$$= \frac{sm(1-s)^k}{(1-m)+m(1-s)^k}$$

$$< \frac{m}{(1-m)} \cdot \frac{1}{ke}$$

where *m* is the fraction of malicious agents and *e* is the base of the natural logarithm. Therefore, with blacklisting, the strict lower bound for honesty (probability of receiving a good result) would be *H=1-(m/(1-m)ke)*. This can be used for computing the honesty of an agent.

Without blacklisting, the probability of a result being bad ould be higher than the previous case. It would be *m.s* (*s*: sabotage rate). But *s* is not known and has to be estimated. $\hat{s} = 1/k$ seems to be a good estimate for *s*, considering that a malicious agent would be detected after *k* verifications. Hence, honesty of an agent will be set to *H=1-m/k*. When using verification, with or without blacklisting, if *k* is equal to zero, honesty should be set to *1-m*, instead of using the previous formulae.

The probability of correctness for a result is equal to the honesty of the agent which produces it. If there are *g* result groups, the probability of correctness (*PoC*) for a result group G_a, where $1 < a < g$, is:

$$PoC(G_a) = \frac{P(G_a\ good).P(all\ others\ bad)}{P(get\ g\ groups,\ where\ each\ G_a\ has\ m_a\ members)} \quad (\mathbf{8})$$

$$= \frac{P(G_a good).\prod_{i \neq a} P(G_i bad)}{\prod_{j=1}^{g} P(G_i bad) + \sum_{j=1}^{g} \left[P(G_i good) \prod_{i \neq j} P(G_i bad) \right]}$$

Here, $P(G_a good)$ is the probability of all the results in G_a being good, computed as $\prod_{i=1}^{m_a} H(R_{ai})$ for all results R_{ai} in the group G_a . Correspondingly, $P(G_a bad)$ is the probability of all the results in G_a being bad, computed as $\prod_{i=1}^{m_a} (1 - H(R_{ai}))$.

When applying the honesty mechanism, both voting and verification can be combined to reduce the error rate to any desired threshold. Voting does not increase the honesty of an agent, but increases the *PoC* of a result group according to equation 8. Verification increases the honesty of the agents. As shown later, by using the mechanism and combining the two methods, extremely high *accuracies* can be reached with an acceptable amount of redundancy and slowdown. It also guarantees a limit on error by watching the conditional probabilities.

For verification, the assumption was that the correct result for the computation was known or one of the few trusted agents must do the task. This limits the probability of verification (*p*) and reduces the performance and speed of increase of *PoC* for a result. Honesty alleviates the problem by using voting *for* verification, i.e. whenever one of the task's result groups reaches the threshold (such that the task is considered as done), *k* which is the number of verifications passed, is incremented for the agents which contributed to that result group. The agents contributing to the loser result groups are considered as caught and their results are backtracked.

The other advantage of honesty is that by using honesty without blacklisting and *H=1-m/k*, there is no benefit for a malicious agent to forge its identity or leaving the computation early and returning again. This also fixes the problem of blacklisting. Therefore using honesty provides a very flexible framework for assuring the correctness of a solution without any previous stipulation on the reliability of agents in a society.

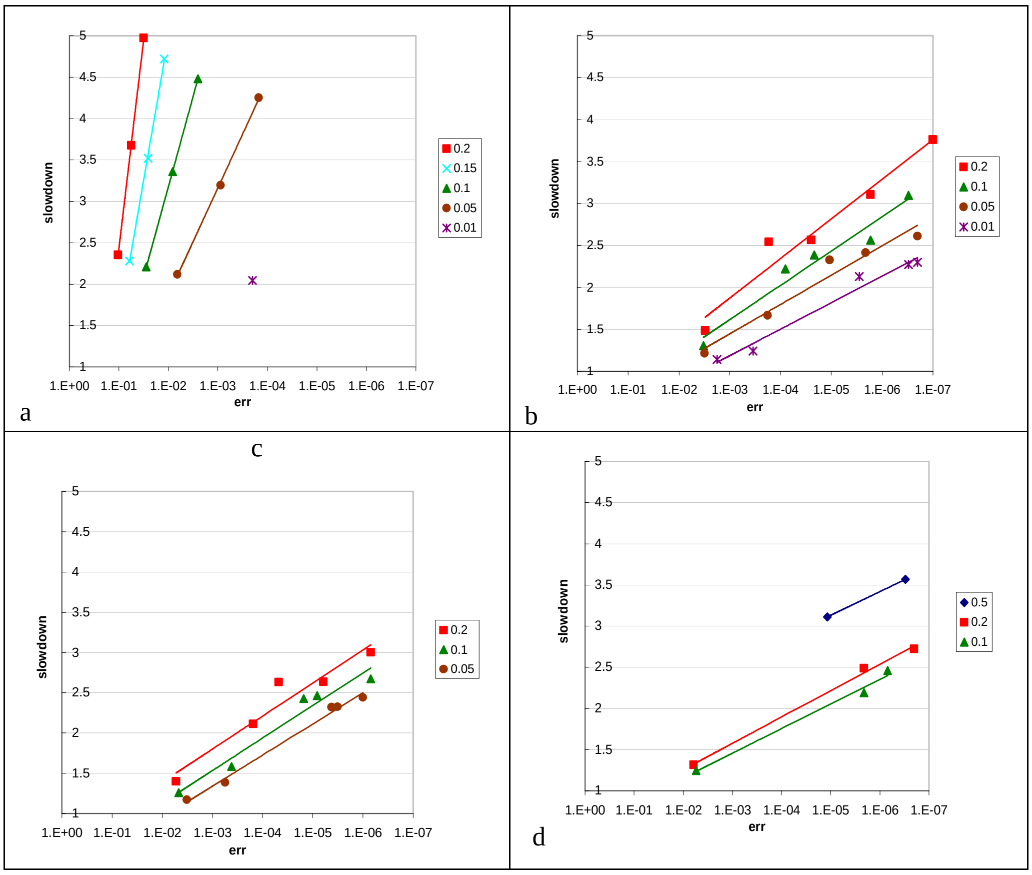
5. Simulation and Evaluation

To implement the above mathematical model, the Java programming language was used. Although in our implementation agents and other objects in the model were all simulated on one computer, this language can flexibly meet the requirements of writing a distributed agent application working over a network like real software agents. This way the agents can communicate with each other and work towards solving a big problem, even though they do not trust each other in terms of reliability and correctness of the returned results.

The main program breaks down the problem into tasks which are grouped into batches. The tasks of a batch are assigned to agents in a round-robin fashion. When a result is returned, the honesty of the agent is given to the probability of correctness of the result. As different results arrive for one task, different result groups are gradually formed, and the *PoC* of each result group is computed. When one of the result groups reaches the acceptable threshold, it is accepted immediately and the task is done. This threshold is equal to the probability of correctness of the final solution which has been guaranteed by the multiagent system. In the setup of the simulation 10 batches of 10000 tasks each, were allocated to 200 agents, which randomly contained a malicious fraction *m*, in the agent population. The experiment was repeated for different values of *m*.

Different social control mechanisms were used in combination with various *m* values and the incurred slowdowns, for several predetermined final error rates, were calculated. The results are summarized in table 1, below. In table 1a, when m is larger the slope of the line is steeper, which demonstrates the weak performance of voting in a society with many malicious agents. Here, slowdown is always above 2 as well. Note that in table 1c, at *m*=0.2 an error rate of 1×10^{-6} is achieved using honesty with voting and verification with blacklisting with a slowdown of 3, while in table 1a the same error rate would be achieved with a slowdown of more than 30. Table 1d even exhibits a better performance than 1c, which has blacklisting. In table 1d, honesty with voting for verification without blacklisting achieves an error rate of 1×10^{-6} , which shows 100,000 times better performance than voting alone in table 1a, for the same slowdown of 2.5.

Table 1. (a) voting alone, (b) honesty with verification without blacklisting, (c) honesty with voting and verification with blacklisting and (d) honesty with voting for verification without blacklisting (for different m: malicious fraction).



6. Conclusion and Future Work

In an open and geographically distributed multiagent system, malicious agents might exist. There is no way of guaranteeing that they actually do the tasks that are allocated to them, since the agents have made no stipulation regarding this issue. They are usually autonomous, self-interested and selfish, or even worse, could have spiteful intensions. It is very critical to utilize social control mechanisms to make the error rate tractable. The contributions of this paper are: setting up the assumption for producing an open multiagent system, modeling the society of agents without any stipulation on agent reliability, presenting an analysis of mechanisms for assurance of solution correctness in such a society, comparing their performance by simulation and introducing the generic and effective honesty framework for modeling a society.

The first mechanism introduced is voting. After all, the more agents return the same result, the more likely it is to be true, assuming that the good agents form a majority. The assumption limits the use of this mechanism and it is highly redundant and inefficient. The second mechanism is verification. The agents are sometimes tested to see that, they are working properly. This way redundancy is mitigated and in effect, agents are scared off into doing the right thing. The human society acts in an analogous fashion.

Finally the honesty mechanism is proposed to create the required infrastructure in a society, for agents to act legitimately. The honesty attribute is not stipulated in advance, nor is it a characteristic of the agent. Instead it is computed gradually, based on the agent's own behavior in the society. It represents the faith of others towards the agent. The generic approach of solving this problem culminated to, acquiring a better understanding of the agents' interactions in an inaccessible environment. Best results were achieved by honesty with voting for verification.

These mechanisms are essential and some of their applicability is for software agents living in a competitive society like the web environment; agent simulation applications [10] for research, related to sociology, political science, economics,
business, and ecology; and creating a framework for negotiation and collaboration among agents. Considering the many possible real world applications that can harvest these mechanisms and the comparative mathematical and simulation analysis that is presented here, the next logical step and future direction for research is, to deploy the mechanisms in a real application similar to the ones mentioned above.

References

1. Agent 2003 Conference on Challenges in Social Simulation, Chicago, Illinois, USA, October 3-4, (2003)
2. Fourth International Workshop on Engineering Societies in the Agents World, Imperial College, London, UK, October 29-31, (2003)
3. Jennings: http://www.ecs.soton.ac.uk/~nrj/mobilevce.html
4. Lynch, N. A., Distributed Algorithms. Morgan Kauffman Publishers, Inc., (1996)
5. Overeinder, B., Brazier, F. and Marin, O., Fault Tolerance in Scalable Agent Support Systems: Integrating DARX in the AgentScape Framework, 3rd International Symposium on Cluster Computing and the Grid, Tokyo, Japan, May 12 - 15, (2003)
6. Ramamohanarao, K. and Bailey, J., Transaction Oriented Computational Models for Multi-Agent Systems, 13th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'01), Dallas, Texas, USA, November 07 - 09, (2001)
7. Sarmenta, L.F.G., Volunteer Computing, Ph.D. thesis, Dept. of Electrical Engineering and Computer Science, MIT, Cambridge, MA, Dec., (2000)
8. Sparkman, C.H., DeLoach, S.A., and Self, A.L., Automated Derivation of Complex Agent Architectures from Analysis Specifications, Second International Workshop On Agent-Oriented Software Engineering (AOSE-2001) Montreal, Canada - May 29th (2001)
9. Wallace, S., Validating Agent Behavior, 23rd Soar Workshop, Ann Arbor, Michigan, USA, June 23-27, (2003)
10. Winoto, P. "A Multi-Agent Based Simulation of the Market for Offenses." AAAI-02Workshop on Multi-Agent Modeling and Simulation of Economic Systems (MAMSES-02), Edmonton, Canada, July 29, (2002)
11. Zuev, Y.A., The Estimation of Efficiency of Voting Procedures, in Theory of Probability and its Applications, Vol. 42, No. 1, March, (1997) http://www.siam.org/journals/tvp/42-1/97594.html