# Diagnosis on Computational Grids for Detecting Intelligent Cheating Nodes

Felipe Martins, Rossana M. Andrade
GREat – UFC
Fortaleza,CE – Brazil

Aldri L. dos Santos
NR2 – UFPR
Curitiba,PR – Brazil

Bruno Schulze, José N. de Souza
LNCC
Petrópolis,RJ – Brazil

## Abstract

*Applications on reputation-based computational grids are prone to attacks from malicious nodes. These nodes may not only corrupt the result from processed jobs, but also intentionally acquire a good reputation so as to obtain privileges to damage other nodes. In order to detect and isolate intelligent cheating nodes from a P2P grid computing, this work proposes a system-level diagnosis model, using a strategy based on voting and honeypots. The model is evaluated by means of scenarios that take into account different percentages of malicious and cheating nodes. Achieved results show the model's robustness and efficiency, once all cheating nodes can be detected, with an accuracy of 99.4% of jobs being correctly processed.*

## 1. Introduction

Computational grids are characterized by the collaborative work among environment devices, where neither the involved hardware platforms nor the geographical location play an important role. Roughly speaking, processing units in a grid donate their computational resources in order to provide together greater computational power. However, the success of these environments is based on the requirement that all units (nodes) are inclined to positively contribute, offering their resources in a right way. So, the grid devices must correctly compute the jobs assigned to them, returning valid results.

Nevertheless, in Peer-to-Peer computational grid there is no real assurance that this requirement is achieved, because there may be nodes acting maliciously, providing corrupted results and compromising the efficiency of the grids. The motivation for such a harmful action may be financial, for fame or simply "evilness" [1]. Unfortunately, a few works in literature have driven efforts to deal with this problem of malicious behavior. With the aim to avoid malicious nodes inside these environments, we presented earlier a distributed and hierarchical system-level diagnosis model based on information on the reputation of the nodes [2]. To detect nodes that return arbitrary and corrupted results, the model considers that the role of a node is attributed according to its reliability. Three roles are proposed: the *Executor nodes* are less reliable and only provide the resources; the *Tester nodes* are responsible for providing resources and testing the Executor nodes; and the *Ultra-Reliable nodes*, the most realiable ones, provide the resources, test and manage a group of Executor and Tester nodes, deciding, among other activities, whether a given node is malicious or not.

Although the results presented in this previous work have shown the strength and scalability of the model, it does not deal with *intelligent cheating nodes*. In a grid, cheating nodes provide their resources and contribute to the grid for a while to obtain a good reputation until they pass to return arbitrary or manipulated results. The presence of this kind of node can be even more disastrous. Nodes acting maliciously with really high reputation may compromise not only the execution of ordinary jobs and the grid applications, but also the diagnosis process, thereby damaging the other grid nodes.

In order to detect this rational behavior, we present here an approach that addresses the presence of intelligent cheating nodes in P2P-based grid plataforms. To do that, this work extends the model presented in [2], adding new features such as the use of nodes (honeypots) whose behavior is previously known. If the decision of an Ultra-Reliable node (UR) on the honeypot is not in accordance with the expected behavior, that UR will be considered malicious and subsequently excluded from the environment. Besides, as UR nodes can also provide their resources, it is also necessary to guarantee that these nodes will not return invalid results. To handle that, UR nodes periodically must test one another. If an UR node is believed suspicious by another UR node, a few UR nodes gather together to determine whether the accused node is to be excluded or not.

This way, the current System-Level Diagnosis Model detects and isolates all malicious intelligent cheating nodes interested in getting a high reputation to damage the applications under execution and the grid as a whole. To evaluate the model, simulations were run in two scenarios (with and without the proposed model), addressing different numbers of malicious nodes. For the experiment, such metrics as processing cost, degree of detection and accuracy were aimed at. We also introduce a graphical interface that allows a visualization of how the nodes behave in the system.

This paper is organized as follows: Section 2 introduces the taxonomy and the consequences of malicious behavior in computational grids and briefly presents the fundamentals of system-level diagnosis. Section 3 shows an overview of the diagnosis model previously proposed in [2]. Section 4 introduces the extension of this model, its features and mechanisms. The experiments and achieved results as well as the developed graphical interface are discussed in Section 5. Section 6 brings some related works. Lastly, the conclusion and future works are presented in Section 7.

## 2 Background

### 2.1 Malicious Behavior in Grids

Distributed computing environments, especially computational grids, can be formed by nodes that are somehow motivated to damage the application's performance, leading to different kinds of faults, such as the byzantine ones [3]. In this scope, malicious behavior in grids is classified into three categories:

- *Inactive nodes*: nodes that intentionally behave as *inactive*, either by omitting information, refusing to forward packages or even stopping providing their resources when asked for;
- *Selfish nodes*: nodes that refuse any kind of access to their resources at the same time that they consume resources from other nodes. This free-rider behavior conflicts with the collaborative principle of the grids as this kind of node only favors its own interests;
- *Cheating nodes*: nodes that provide their resources, processe the expected jobs, but return arbitrary or manipulated results. The presence of these nodes in a grid can damage the application.

Cheating nodes may be also subdivided into *fool*, *common* and *intelligent* nodes [4]. Fool nodes always return arbitrary results. Common nodes return arbitrary results with a certain fixed probability. Intelligent nodes, on the other hand, act normally for some time in order to obtain a good reputation up to the extent when they deliberately start to return arbitrary results with a fixed probability. As they present a sensible behavior, these nodes are the hardest to deal with.

In reputation-based grids, the damage from intelligent cheating nodes can bring unwished consequences, because once they obtain a high level of reliability, they can subvert all the system and provide false pieces of information about the other nodes which may lead idoneous nodes to be included in blacklists.

### 2.2 System-Level Diagnosis

System-level diagnosis has been commonly used as a strategy to tolerate faults in distributed systems. In this case, periodical tests are performed on the elements of a system. The set of results of such tests (called a *syndrome*) allows the identification of which nodes are presumed to be faulty.

In the literature, the diagnosis models proposed in [5]and [6] are noteworthy. The latter, known as MM (Maeng and Malek), assumes that a certain Tester node sends to two other distinct nodes the same test task. Following, the Tester node collects and compares the results produced by the tested nodes. Then it sends the comparison to a central node, responsible for diagnosing the tested nodes. If the two results are the same, the central node considers both the tested nodes as non-faulty. But if the results are different, the central node, based only on this information, will not be able to identify which (if not both) of the nodes is faulty.

In the model presented in [2], a syndrome is obtained through *test-jobs*. The following section describes the discussed diagnosis model in more details.

## 3 Grid-Applied Diagnosis Model Overview

In the context of computational grids, a diagnosis model based on comparison and reputation has been previously proposed [2]. The model provides a security strategy to detect and minimize the presence of malicious nodes interested in corrupting the results of their jobs. To do that, the model defines the role of a grid node as *Executor*, *Tester* or *Ultra-Reliable*, according to its current reputation. Nodes with a high reputation are considered as better resource providers and tend to have higher status. Malicious nodes that are detected manipulating results have lower reputation and thus tend to be excluded from the grid.

According to the model, the reputation of each node is inferred through test-jobs, which are nothing else than common jobs whose results are previously known by the node that created it. This way, it is possible to know the behavior of the target node to be tested (Executor). If an Executor node provides an answer different from what is expected, the Tester node reports this Executor to its Ultra-Reliable node (UR) as a malicious node.

Figure 1 illustrates the diagnosis strategy of this model. Two nodes in the system, A and B, acting as Testers, periodically send two different test-jobs to a node C, acting as an Executor (a). Node C then executes the test-jobs and sends the results back to the respective Tester nodes (b). Next, nodes A and B send their perceptions on node C to node D, acting as an UR node (c). Taking into account these analysis of perception, node D issues a diagnosis about node C.

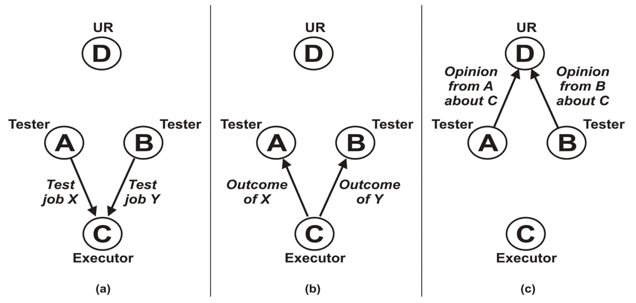

**Figure 1. Diagnosis strategy for the detection of malicious nodes [2]**

Thus, if two Tester nodes A e B agree upon the status (malicious or idoneous) of the Executor node C, the Ultra-Reliable node D assumes these perceptions to be valid and, depending on the results, condemns or not node C. But if the Tester nodes A e B diverge about the status of node C, then D analyzes the historical behavior of the Testers in order to check and see if there is a repeating pattern (for example, if one of them rejected all the tests that it applied, or if one of them rejected only the tests applied to a given Executor node, etc). Based on this behavior pattern analysis, an UR node may decide if any of the nodes (Tester or Executor) acted in a malicious way.

Although this approach seems to be very simple and robust for detecting fool and common cheating nodes, it is not capable of detecting malicious nodes that act rationally with the purpose of achieving high reputation and, consequently, UR status. If any malicious node becomes an UR, the system is jeopardized, since such nodes can arbitrarily decide the destiny of other nodes with lower status. Considering the possibility of collusion, a malicious node with high reliability will be able even to promote its accomplices, ending up in the contamination of the whole environment.

## 4 Voting and Honeypots-based Diagnosis Model

The algorithm employed allows the identification of malicious behavior of resource provider nodes considered highly reliable. Such UR nodes are tested periodically by other UR nodes. So, if an UR node has doubts about another UR node due to its wrong test-job result, the former will invoke a group of UR nodes to decide on whether the suspected UR node is malicious or not.

The size of the diagnosis group - given by a function *f(n)* described later - varies according to the total number of UR nodes in the environment. A diagnosis group is composed of those UR nodes with the highest reputation. Besides, neither the suspected UR node nor the complaining UR node are selected to be members of the group. Once the size and the members of the diagnosis group are chosen, the complainant asks them to diagnose the suspected node.

The creation of the diagnosis group is illustrated in Figure 2. It presents six UR nodes, each one responsible for a logical cluster made up of a set of eight nodes (Executors and Testers). In a certain moment, node A sends a test-job to node E (a). Node A suspects that node E responded an arbitrary result to the test and considers it to be a possible malicious node. Then node A invokes a diagnosis group which is composed of the UR nodes with the highest reputation. In this case, as the system has six nodes, the group will have three components, according to Function 1, described later. In the example, nodes B, D and F are chosen and node E is the suspected node (b).

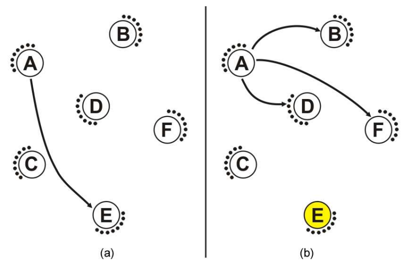

**Figure 2. Invocation of the Diagnosis group**

Once the diagnosis group is arranged, each UR member of the group checks the suspected UR node to judge its behavior. Then, each UR member creates a different test-job and sends it to the suspected UR node. Clearly, the UR members know in advance the result of the submitted test-jobs. The suspected UR node processes the test-jobs and sends the results back. Each UR member compares the responses against the expected result. The perceptions from all the members about the suspected UR node are next broadcasted to the members of the group only. At the end, each UR member in the group knows the opinion of the others and, using a majority voting scheme, the group diagnoses the behavior of the suspected node.

If the suspected UR node is considered guilty (malicious), the UR node with the highest reputation in the group is chosen as the coordinator, which is responsible for excluding the malicious UR node and redistributing its Tester and Executor nodes among other UR nodes in the system. In order to guarantee the efficiency of the model, it is assumed that the number of reliable UR nodes is higher than the number of cheating UR nodes. Figure 3 illustrates the decision-making process of the UR nodes B, D and F on the suspicious UR node E.

The members of the group test node E with their test-jobs (a). Based on a comparison between the results returned by node E and the expected results, each member can evaluate the suspected node E. Then, they exchange their perceptions on node E (b). If the majority believes the suspected

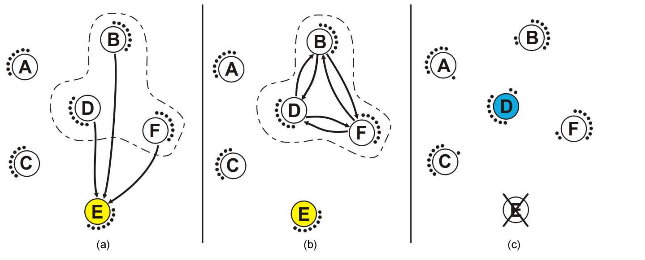

**Figure 3. Decision-taking process**

UR node to be malicious, the member with the highest reputation in the group - in this case, node D, called the coordinator - is responsible for excluding node E from the grid and including it into a blacklist. Next, the coordinator must redistribute the Executor and Tester nodes that were managed by node E among the other UR nodes in the system (c). This cluster reconfiguration phase as well as reputation calculation remains as defined in the original approach [2].

### 4.1 Diagnosis Group Size

The size of the diagnosis group must be carefully chosen. Ideally, all UR nodes (with the exception of the suspected and the complaining nodes) should participate in the group. However, the number of exchanged messages during the decision-making phase could become extremely high, making the solution non-scalable. In order to restrict the number of nodes in the diagnosis group, without compromising the scalability of the system nor the efficiency of the whole process, the algorithm makes use of a function based on the total amount of UR nodes in the grid, as described below:

$$f : Z_+^* - \{1, 2\} \rightarrow Z_+^* \quad (1)$$

$$f(n) = \begin{cases} 1 & \text{, if } n \leq 4 \\ \lfloor log_2 n \rfloor + (\lfloor log_2 n \rfloor \quad mod \quad 2) + 1 & \text{, if } n > 4 \end{cases}$$

where *n* represents the total amount of UR nodes.

Because of the use of a logarithmic function, the size of the diagnosis group suffers little variation regardless of the number of UR nodes in the environment, which avoids a high overhead when the members of the group must later exchange messages. As the value that represents the group size has to be an integer, the result found is truncated by using the *floor* function. Besides that, the size must be an odd number in order to prevent a tie after the votes are counted.

Although a diagnosis group is composed only of the nodes with the highest reputations, it is possible to have malicious nodes inside the group. However, even these nodes remain under continuous monitoring by the others UR nodes, playing as honeypots, as described in the next section. This way, it is possible to detect and eliminate them from the environment, as shown in Section 5.1.

### 4.2 Honeypots

Originally, honeypots are computing resources whose goal is to be attacked and compromised by others in order to understand the attackers' strategy. This approach, largely found in security systems of traditional networks, can also be applied to grid environments, with the purpose of collecting information on malicious nodes.

Thenceforth, the system's administrator can insert emulated nodes into the grid. Each honeypot can play as idoneous ou falsely malicious in order to find if the UR node responsible for testing it is inferring the correct diagnosis. This way, the system will know if an UR node is a disguised malicious node. If so, it will be excluded from the grid and inserted into a blacklist.

## 5 Experiments

To validate the proposed diagnosis model, experiments were run in the GridSim grid simulator [7]. The scenarios assumed an environment initially of 200 Executor nodes managed by only one UR node. During the simulation, the Executor nodes may improve their reputation, thereby attaining a higher status. In the first scenario, the scheme to detect intelligent cheating nodes is not applied, so the nodes that gain the UR status can freely act maliciously. In the second scenario, the decision-making algorithm and the honeypots were active as shown next. We also presented a graphical interface built with the purpose of allowing the set-up of these scenarios and the visualization of the generated logs.

### 5.1 Scenarios Description

In both scenarios, nodes that may act maliciously were modeled with a probability of 25% chance of returning a non expected result. Also, new nodes can join the environment after a test round is concluded. Every time a new node joins the grid, a method is called to randomly define if the new node will have a malicious behavior, with a probability of 50%. Nevertheless, the maximum number of nodes in the environment at any given moment is limited to 200.

Moreover, in both scenarios the total amount of UR nodes was limited to 10. Different percentages of cheating UR nodes were experienced, namely 10, 20 and 40%. The number of malicious Executor nodes (1/6, 1/3 and 2/3 of the total number, respectively) as well as the process of generating jobs involves the same procedures described in [6]. Besides, the scenarios were simulated with 8 test rounds carried out in different periods (at each 6, 12 and 24 hours).

The experiments took into account metrics such as the amount of detected malicious nodes (degree of detection), the processing cost (the ratio between the number of test-jobs and the total number of jobs) and the accuracy (the ratio between the number of jobs correctly processed and the total number of jobs).

## 5.2 Results

The variation in the quantity of detected malicious nodes when the percentage of cheating UR nodes is varied in an environment with 2/3 of its nodes compromised and tests rounds at every 12 hours is presented in Figure 4. The figure shows that the level of nodes detected is proportional to the number of malicious nodes in the grid.

In addition, the variation on the total of malicious nodes detected using the proposed scheme is irrelevant with just 10% of intelligent cheating UR nodes. This happens because there are much more malicious Executor nodes in the grid than intelligent cheating UR nodes. However, the heuristics for verifying UR nodes can enhance the efficacy of the system, since more malicious nodes are detected and inserted in the blacklist. Thus, the bigger the presence of cheating UR nodes in the grid, the stronger the system is, if compared to an environment without any verification. While the scheme without verification detects 242 malicious nodes, with 40% of the UR nodes cheating, the use of the verification technique increases this number to 264 malicious nodes, a gain of approximately 10% in the degree of detection.

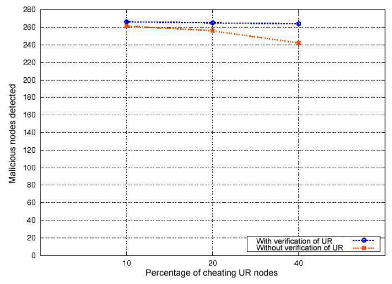

**Figure 4. Detection level with 2/3 of malicious nodes and tests at every 12 hours**

The system performance is reduced as the tests are less often. But, it we emphasize that *all cheating UR nodes are detected*, no matter the quantity of malicious nodes in the environment nor the frequency of the tests. Further, for all situations, the accuracy obtained through the use of the proposed strategy is significantly higher when compared to an environment without any verification. For example, according to Figure 5, when there are 1/6 of malicious nodes, an accuracy of 99.4% is obtained with 10% of cheating UR nodes being tested in the environment at every 6 hours. Without the verification and without honeypots, the accuracy plunges to 97.2% for the same quantity of cheating UR nodes acting freely. Comparatively, the accuracy obtained with the strategy is even better when 40% of the UR nodes present a cheating behavior. While without verification the accuracy is 89.7%, with the proposed scheme the accuracy soars to 98.7%, representing a 9% gain. In the worst case (with 2/3 of malicious Executor nodes, 40% of cheating UR nodes and tests rounds at every 24 hours), the accuracy plunges to 93%. Even in this situation, when more than half of the grid is jeopardized, the system efficiency with the proposed strategy is approximately 8.5% higher than an environment without any sort of verification.

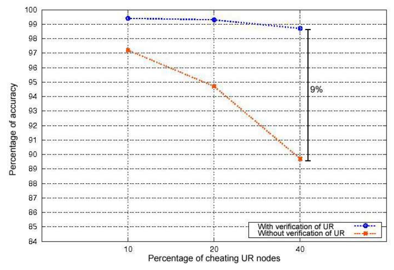

**Figure 5. Accuracy with 1/6 of malicious nodes and tests at every 6 or 12 hours**

As for the same quantity of malicious Executor nodes the accuracy obtained is almost the same when the tests are taken at every 6 and 12 hours, it can be concluded that applying tests more frequently does not bring significant benefits in the current model. As a matter of fact, with tests at each 12 hours, it is possible to get a similar percentage of jobs processed correctly, at the same time that the cost is reduced. Figure 6 shows the processing cost with and without the proposed scheme for the verification of UR nodes.

Pursuant to the Figure 6, when applying tests at every 12 hours, the cost rises from 12.3% to 17% with the UR verification. In this case, it is necessary to know whether this trade-off is acceptable for the application being executed. Nevertheless, the results show that the processing cost does not depend on the amount of malicious Executor nodes or cheating UR nodes in the grid. It varies exclusively according to the testing frequency. This feature gives the model a good scalability.

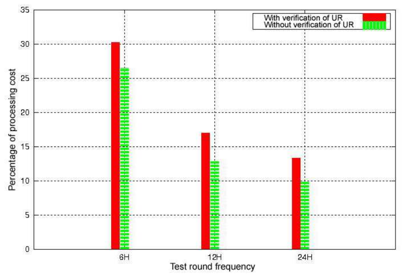

**Figure 6. Processing cost**

## 5.3 Graphical Interface

GridSim offers the ability to create logs that store information about each simulation run. These logs can be better understood if represented as graphics. However, the extraction and analysis of the information stored in the logs is a costly process. To follow the dynamics of the simulations in the scenarios previously described and to make it easier to interpret the results, a graphical interface was implemented.

The developed environment not only improves the productivity of creating and setting scenarios, but also generates a new output log file with pieces of information described in natural language. This feature allows the user to analyze the results more immediately than if he or she had to manually decipher the traditional log format.

Besides the readability, the interface offers the ability to animate the information of the log file, by showing the events in real time during the simulation. So it is possible to accompany how the grid works, with information about the tests and details about clusters and nodes in the environment, such as status, reputation, lifetime in the grid, if it is an idoneous or malicious node, among others. The source codes of the model and the graphical interface are available at `<http://www.great.ufc.br/˜felipe/DGmodel.html>`.

## 6 Related Works

Research on processing integrity in computational grids is found in [4] and [8]. Such approaches propose strategies based on spot-checking, whose job results are previously known by the node responsible for the tests. But, all these solutions are essentially centralized, which causes a high cost for the manager node responsible for the tests, considering that every test-job is processed by a unique manager, differently from our distributed and hierarchical approach.

In P2P systems, as in [9] and [10], a node can evaluate the reliability of another node based on its reputation, which is inferred through its interactions with the rest of the network. A common issue to all these models is the fact that they use information provided by other nodes. So, a node can, intentionally and maliciously, condemn another node, unfairly damaging its reputation. If this behavior spreads out to the entire environment, the efficiency of such models becomes compromised. In [10], this can even be more critical, because only negative feedbacks are considered.

With relation to the above approaches, the diagnosis model here presented handles the existence of rational nodes that obtain a high reputation in order to deliberately damage other nodes in the grid. This way, the model is able to exclude nodes that corrupt the jobs results that they process, as well as to avoid the presence of intelligent cheating nodes interested in disrupting the application under execution and the grid itself.

## 7 Conclusion and Future Work

The use of system-level diagnosis emerge as an efficient solution against results manipulation attacks in P2P computational grids, since it allows to eliminate nodes that present a malicious behavior. However, considering that cheating nodes act rationally and can therefore obtain a good reputation in order to damage the grid, it is necessary to incorporate new detection techniques to make the system more robust and efficient.

The proposed model combines the use of honeypots with a voting scheme in a grid environment, allowing the identification of *all* cheating nodes, independently of the number of existing malicious nodes, as observed in the experiments. The results also shows a significant processing accuracy improvement, since 99.4% of the jobs are correctly processed. Hence, combining these techniques makes it possible to increase the processing quality in grid computing environments. As a future work, a more extensive evaluation of the metrics can be done as a means to reduce the processing cost. Besides, this diagnosis model will be implemented and incorporated to a real grid plataform, such as OurGrid.

## References

[1] L. Catuogno, P. Faruolo, U. F. Petrillo, and I. Visconti. Reliable accounting in grid economic transactions. In *Grid and Cooperative Computing - GCC Workshops*, pages 514–521, 2004.

[2] F. Martins, M. Maia, R. M. de Castro Andrade, A. Luiz dos Santos, and J. Neuman de Souza. A grid computing diagnosis model for tolerating manipulation attacks. In *International Transactions on Systems Science and Applications*, volume 2, pages 135–146, Erfurt, Germany, September 2006.

[3] M. Hollick. On the effect of node misbehavior in ad hoc networks. In *Proceedings of IEEE International Conference on Communications, ICC'04*, volume 6, pages 3759–3763, 2004.

[4] S. Zhao, V. Lo, and C. GauthierDickey. Result verification and trust-based scheduling in peer-to-peer grids. In *P2P '05: Proceedings of the Fifth IEEE International Conference on Peer-to-Peer Computing (P2P'05)*, pages 31–38, Washington, DC, USA, 2005. IEEE Computer Society.

[5] F. Preparata, G. Metze, and R. Chien. On the connection assignment problem of diagnosable systems. *IEEE Transactions on Electronic Computers*, 16:848–854, 1968.

[6] J. Maeng and M. Malek. A comparison connection assignment for self-diagnosis of multiprocessor systems. In *Digest 11th International Symposium Fault Tolerant Computing*, pages 173–175, 1981.

[7] R. Buyya and M. Murshed. Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. In *Journal of Concurrency and Computation: Practice and Experience (CCPE)*, 2002.

[8] L. F. G. Sarmenta. Sabotage-tolerance mechanisms for volunteer computing systems. In *CCGRID '01*, page 337, Washington, DC, USA, 2001. IEEE Computer Society.

[9] R. Sherwood, S. Lee, and B. Bhattacharjee. Cooperative peer groups in nice. *Computer Networks*, 50(4):523–544, 2006.

[10] K. Aberer and Z. Despotovic. Managing trust in a peer-2-peer information system. In *CIKM '01: Proceedings of the 10 International Conference on Information and Knowledge Management*, pages 310–317, New York, NY, USA, 2001. ACM Press.