A Probabilistic Approach for Task and Result Certification of Large-scale Distributed Applications in Hostile Environments*

Axel Krings, Jean-Louis Roch, Samir Jafar, and Sébastien Varrette

Laboratoire ID-IMAG (CNRS-INPG-INRIA-UJF – UMR 5132), Grenoble, France
{axel.krings,jean-louis.roch,samir.jafar,sebastien.varrette}@imag.fr

Abstract. This paper presents a new approach for certifying the correctness of program executions in hostile environments, where tasks or their results have been corrupted due to benign or malicious act. Extending previous results in the restricted context of independent tasks, we introduce a *probabilistic certification* that establishes whether the results of computations are correct. This probabilistic approach does not make any assumptions about the attack and certification errors are only due to unlucky random choices. Bounds associated with certification are provided for general graphs and for tasks with out-tree dependencies found in a medical image analysis application that motivated the research.

1 Introduction

Large scale global computing systems like the GRID and Peer-to-peer computing platforms gather thousands of resources for computing parallel applications, utilizing middleware infrastructures such as the Open Grid Service Architecture (OGSA) [2] to provide strong authentication, secure communications [11], and resource management. In this unbounded environment one should consider possible malicious act that may result in *massive attacks* against the whole global computation. This is supported by an exponentially increasing number of reported incidents [1], e.g. CERT/CC recorded close to 140,000 incidents in 2003.

Usually, global computations are expected to tolerate certain rates of faults [5, 9], e.g. small number of isolated intrusions. However, in order to ensure correctness of the computed results, one should detect if the global computation has been the victim of a massive attack resulting in an error rate larger than can be tolerated by the application.

The problem of protecting a computation against massive attacks has been mainly addressed for independent tasks. The analysis of *voting*, *spot-checking* and *credibility-based fault tolerance* is presented in [9]. An approach based on re-execution of tasks on reliable nodes is considered in [5], assuming that the majority of workers are honest while workers compromised by an attack will

* This work has been supported by CNRS ACI Grid-DOCG and the Region Rhône-Alpes (Ragtime project).

always falsify their results. Under the same assumption, task dependencies are considered in [6], however, dependencies are used only for correction. Faults in systems with task dependencies are addressed in [4] where tasks are determined to execute on reliable or non-reliable nodes in order to maximize the expected number of correct results. Whereas the approach considers the critical issue of fault propagation, it is deterministic and therefore could be exploited by an intelligent adversary.

In order to eliminate any assumption on the attack and the distribution of errors in the context of a general parallel computation with dependencies, we propose to adopt a view directly inspired by probabilistic algorithms. Specifically, given the results of a global computation with task dependencies, we attempt to detect if the execution contains faulty results. Probabilistic algorithms are presented that make random choices and determine whether the execution is correct or faulty. Since the detection is probabilistic, its output may be wrong. However, contrary to previous approaches, the probability of certification error is not related to the application, i.e. the global computation, but only to the unlucky random choices associated with task selection for verification.

This work is motivated by medical applications [7] studied in the context of the French research project *Ragtime* [8] where certain highly computational manipulations of medical 3D/4D images, distributed over their production sites, allow for an acceptable fraction of error. The probabilistic certification algorithms presented can detect if these computations have been subjected to a massive attack with a so-called *attack ratio* greater than or equal to $q < 1$, with no other assumption about the attack. We show that pathological cases exist that minimize the probability of detection. The bound on the error is not related to q, but to the minimum number of so-called *initiator* tasks.

2 Definitions and assumptions

Applications are executed on the global computing platform presented in [6]. A *user* initiates a computation, represented by a directed acyclic graph G, which is then executed on (a potentially large number of) unreliable *workers*. In order to verify the correctness of the results of the execution, *verifiers*, implemented by reliable resources which know graph G, re-execute selected tasks. Communication between workers and verifiers is through a checkpoint server containing computations submitted by workers [6]. Whereas any attack can occur on the worker or between the worker and the checkpoint server, the checkpoint server and verifiers are considered secure.

Dataflow graph: The data-flow graph referred to above is a directed graph $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is a finite set of vertices v_j and $\mathcal{E}$ is a set of edges e_{jk}, $j \neq k$, representing precedence relations between $v_j, v_k \in \mathcal{V}$. The vertex set consists of two kinds of tasks. Let T_j denote the tasks as seen in the traditional context of task scheduling, i.e. a task is the smallest program unit of an instance of execution. Let D_k denote a data task. Data tasks represent the inputs and

outputs of a task. In the remainder of this paper, when talking about a task, it is implied to be a task T_j. Data tasks will be referred to as inputs or outputs of T_j. The total number of tasks T_j in G is n.

Executions and the impact of faults: We will first establish the notion of program execution and the impact of faults. Let E denote the execution of a workload represented by G with a set $\hat{I}$ of initial inputs on a set of unreliable resources. It is assumed that G is static, i.e. it is fixed. Each task T in E executes with inputs $i(T, E)$ and creates output $o(T, E)$. The inputs of a task T_j are composed of either inputs from $\hat{I}$ or outputs of other tasks T_k, i.e. $o(T_k, E)$.

Let $\hat{E}$ denote the execution of the program on a verifier, i.e. a reliable resource, or set thereof. If $E = \hat{E}$, i.e. if every task in E uses the same inputs and computes the same outputs as those in $\hat{E}$, then E is said to be "correct". Conversely, if $E \neq \hat{E}$, then at least one task in E produced a wrong result and the execution is said to have "failed".

In order to differentiate whether a task execution is considered to be on a client or verifier and whether the inputs and outputs of the execution are those of E or $\hat{E}$, the following notation is adapted. Note that a "hat" always refers to a reliable resource, input or output. Let $i(T, E)$ denote the input of T in E and $\widehat{i}(T, \hat{E})$ the input of T in $\hat{E}$. Furthermore, let $o(T, E)$ denote the output of T on the client, $\hat{o}(T, E)$ the output of T on the verifier based on inputs from E, and $\hat{o}(T, \hat{E})$ the output of T on the verifier based on inputs from $\hat{E}$. Note that the notations $\hat{o}(T, E)$ and $\hat{o}(T, \hat{E})$ differ. Both indicate outputs generated on a verifier, but the first assumes $i(T, E)$ and the latter $\widehat{i}(T, \hat{E})$ as inputs.

Probabilistic certification: We consider probabilistic certification based on a probabilistic algorithm that uses randomization in order to state if E has failed or not. Given an execution E, a Monte Carlo certification is defined as a randomized algorithm that takes an arbitrary ϵ, $0 < \epsilon \leq 1$, as input and delivers (1) either *CORRECT* or (2) *FAILED*, together with a proof that E has failed. The probabilistic certification is said to be with error ϵ if the probability of the answer *CORRECT*, when E has actually failed, is less than or equal to ϵ.

For instance, a Monte Carlo certification may consist of re-executing randomly chosen tasks in G on a verifier, comparing results to those obtained in E. If the results differ E has failed. Otherwise, E may be correct or failed. However, if E has failed, a probabilistic certification with error ϵ ensures that the probability of non-detection of failure (based on randomly selecting tasks in G for re-execution) is less than or equal to ϵ.

Monte Carlo certification against massive attacks: In the sequel we denote the number of forged tasks in G by n_F. We are considering the two scenarios where either all tasks execute correctly, i.e. $n_F = 0$, or n_F is large, corresponding to a massive attack. A *massive attack* with *attack ratio* q consists of falsifying the execution of at least $n_q = \lceil qn \rceil \leq n_F$ tasks. E is said to be "attacked with ratio q" and $\frac{n_F}{n} \geq q$. It should be noted that q is assumed relatively large, see Section 5, resulting from massive attacks such as caused by a virus or Trojan.

The objective is to provide a probabilistic Monte Carlo certification against such massive attacks. Note that detection of small attacks, e.g. single intrusions, is not the scope of this work. As indicated in Section 1, global computations are expected to tolerate certain fault rates.

3 Certification of independent tasks

We first consider the case where all tasks in G are independent. In this case, certification of tasks is equivalent to certification of results. The following Monte-Carlo Test (MCT), based on task re-execution on a verifier, will be used to detect if execution E contains forged tasks.

Algorithm MTC

1. Uniformly choose one task T in G. The input and output of T in E are $i(T, E)$ and $o(T, E)$ respectively.
2. Re-execute T on a verifier, using inputs from E, i.e. $i(T, E)$, to get output $\hat{o}(T, E)$. If $o(T, E) \neq \hat{o}(T, E)$ return FAILED;
3. Return CORRECT.

Since all tasks in G are independent[1] we always have $i(T, E) = \widehat{i}(T, \hat{E})$. If Algorithm MTC selects a forged task, then one knows with certainty that the execution E has failed. However, if MTC returns CORRECT, then one can only make conclusions based on the probabilities of randomly selecting a falsified or non-falsified task. The following lemma addresses these probabilities.

Lemma 1. *Let E be an execution with n independent tasks, n_F of which have been forged. The probability that MTC returns FAILED is $\frac{n_F}{n}$ and the probability that it returns CORRECT is $1 - \frac{n_F}{n} \leq 1 - q$.*

Proof. The probability that MTC chooses a forged task is $\frac{n_F}{n}$. Then the probability that MTC returns CORRECT is $\frac{n-n_F}{n} = 1 - \frac{n_F}{n} \leq 1 - q$. □

The theorem below gives a lower bound on the number of tasks to be re-executed in order to achieve a specific ϵ.

Theorem 1. *Let E be an execution with only independent tasks and assume that E is either correct or massively attacked with ratio q. For a given ϵ, the number of independent executions of algorithm MTC necessary to achieve a certification of E with probability of error less than or equal to ϵ is $N \geq \lceil \frac{\log \epsilon}{\log(1-q)} \rceil$.*

Proof. Consider N executions of Algorithm MTC. If during any of the N executions MTC selects a forged task, the execution has failed. Therefore, assume that only non-forged tasks are selected. According to Lemma 1 the probability of MTC selecting a non-forged task is $\frac{n-n_F}{n} \leq 1-q$. Then N independent applications of MTC lead to a Monte-Carlo certification with a probability of error bound by $\epsilon \leq (1-q)^N$. For a given ϵ, it is thus sufficient to select $N \geq \lceil \frac{\log \epsilon}{\log(1-q)} \rceil$ tasks. □

[1] In the case of task dependencies this assumption about the inputs does not hold anymore, as will be addressed later.

4 Certification in the presence of task dependencies

In the previous section there is no difference between certification of tasks and their respective results. If one allows for dependencies among tasks the certification of the results of tasks is more difficult. The problem lies in the way a reliable resource has to determine the validity of results. Any measure of validity of a task's result based on the comparison to the results obtained by re-executing the same task on a reliable resource, depends on the validity of the inputs the reliable resource uses for re-execution. Just the fact that the outputs of a task execution and its re-execution on a reliable resource produce identical results does not say much about the validity of that result, since in the assumed deterministic computing environment the same faulty input will produce identical faulty output. Thus, in the presence of dependencies, $o(T, E) = \hat{o}(T, E)$ only indicates that the results are the same, but not that they are correct. It should be noted that correctness would imply that $o(T, E) = \hat{o}(T, \hat{E})$.

4.1 Faulty tasks and the concept of initiators

The randomized testing used in Section 3 is only valid for result certification of independent tasks. If we were to apply the same reasoning in the presence of dependencies, certification based on repeated application of Algorithm MTC would only certify results if $o(T, E) \neq \hat{o}(T, E)$ for each falsified T selected by MTC. However, this assumption is too restrictive since it would assume that a re-execution with some (perhaps incorrect) input values would always expose[2] the forgery. This weak assumption could be easily exploited by an attacker.

Suppose Algorithm MTC is used. If $o(T, E) \neq \hat{o}(T, E)$ then E has failed. However, $o(T, E) = \hat{o}(T, E)$ indicates a correct output only if the inputs are correct, i.e. $\widehat{i}(T, \hat{E})$. This implies that T has no forged predecessors. In the following discussion, falsified tasks which have no falsified predecessors will be called *initiators*. The probabilities associated with randomly selecting initiators will be the basis for result certification. It should be noted that it is difficult to speculate on the capabilities of detecting incorrect results of falsified tasks that are not initiators. Pathological attacks may be derived where the output of one falsified task may be custom tailored to produce results for other falsified tasks that do not differ from their re-executions (with the forged inputs) on reliable resources.

4.2 Certification

Result certification is directly related to the probability of the certification algorithm selecting initiators. Let n_I denote the number of initiators in G. Note that the determination of n_I depends on the graph and which nodes have been falsified. The following lemma and theorem, modified from Lemma 1 and Theorem 1, can be stated.

[2] It should be noted that the task inputs define a rather limited "test vector" for the task. The quality of test vectors with respect to fault coverage has been extensively studied in the context of the *Test Vector Generation Problem* [3].

Lemma 2. *Let E be an execution with n tasks with dependencies. Furthermore, let n_F and n_I be the number of forged tasks and initiators respectively, $n_I \leq n_F$. The probability that MTC returns FAILED is at least $\frac{n_I}{n}$ and the probability that it returns CORRECT is less than or equal to $1 - \frac{n_I}{n}$.*

Proof. The probability that MTC selects an initiator, and thus returns value FAILED, is $\frac{n_I}{n}$. Then the probability that MTC returns CORRECT is less than or equal to $\frac{n-n_I}{n} = 1 - \frac{n_I}{n}$. □

Lemma 3. *Let E be an execution of tasks with dependencies and assume that E is either correct or massively attacked with ratio q. For a given ϵ, the number of independent executions of algorithm MTC necessary to achieve a certification of E with probability of error less than or equal to ϵ is $N \geq \lceil \frac{\log \epsilon}{\log(1-\frac{n_I}{n})} \rceil$.*

Proof. Consider N executions of Algorithm MTC. If during any of the N executions MTC selects an initiator, the execution has failed. Therefore, assume that only non-initiator tasks are selected. According to Lemma 2 the probability of MTC selecting a non-initiator task is $\frac{n-n_I}{n} = 1 - \frac{n_I}{n}$. Then N independent applications of MTC lead to a Monte-Carlo certification with a probability of error bound by $\epsilon \leq (1 - \frac{n_I}{n})^N$. For a given ϵ, it is thus sufficient to select $N \geq \lceil \frac{\log \epsilon}{\log(1-\frac{n_I}{n})} \rceil$ tasks. □

Unlike the case of Theorem 1, this result is more restrictive since the value of n_I depends on G under consideration of the worst-case attack scenario and it is likely to be small. In the remainder of the paper we study the implications of worse case attacks, where the attacker knows or can estimate the structure of the graph. We will prove that even in pathological cases a lower bound on the number of initiators n_I can be defined, considering G and attack ratio q.

Let $G^<(T)$ denote the sub-graph induced by all predecessors of a task T or a set of tasks V, i.e. $G^<(V)$. Furthermore, let $G^\leq(T) = G^<(T) \cup \{T\}$. The graphs of successors are denoted similarly, i.e. $G^>(T)$ and $G^\geq(T)$. We now formally define the set of initiators.

For a given G with n tasks let F denote the set of all falsified tasks. The *initiator set* $\mathcal{I}(F)$ is defined as the set of all $T_i \in F$ which have no predecessors in F, i.e. $\mathcal{I}(F) = \{T_i \in F : F \cap G^<(T_i) = \emptyset\}$. It is obvious that the actual tasks in sets F and $\mathcal{I}(F)$ are not known, since otherwise certification would be trivial.

Since re-execution of a task with incorrect inputs may still result in $o(T, E) = \hat{o}(T, E)$ one has to consider the limitations induced by the inputs.

Lemma 4. *Given the set of all falsified tasks F and an arbitrary T in G, if the outputs of T are not correct, then it must be that $G^\leq(T) \cap \mathcal{I}(F) \neq \emptyset$.*

The proof follows directly from the fact that, if the output of T is not correct, then either T is faulty, i.e. $T \in F$, or there must be at least one forged task in predecessor set $G^<(T)$. Determining that the output of T is incorrect may require to verify few or many tasks and is largely dictated by the size of $G^\leq(T)$.

The following definitions will aid in capturing the difficulties associated with determining whether results are incorrect, considering that a pathological attacker will minimize the probability of finding forged tasks.

First, the minimum number of initiators with respect to given subgraph of G is defined. Let V be a set of tasks in G and let $k \leq n_F$ be the number of falsified tasks assumed. Define $\gamma_V(k)$ as the *minimum number of initiators* with respect to V and k such that $\gamma_V(k) = min|G^{\leq}(V) \cap \mathcal{I}(F)|$ for $|F| \geq k$ and of all $G^{\leq}(V) \cap \mathcal{I}(F) \neq \emptyset$. Recall that $n_q \leq n_F$ is the smallest number of falsified tasks as the result of an attack with ratio q. Then $\gamma_G(n_q)$ is the smallest number of initiators possible, e.g. the number associated with a pathological attack scenario. With respect to Lemma 2 the probability that MTC returns correct can thus be written as $1 - \frac{\gamma_G(n_q)}{n}$.

Next, we will define the *minimal initiator ratio* $\Gamma_V(k)$ as

$$\Gamma_V(k) = \frac{\gamma_V(k)}{|G^{\leq}(V)|}. \tag{1}$$

The minimum initiator ratio is helpful in determining bounds on probabilities of selecting initiators in predecessor sets. This is of interest when the structure of G will allow for adjustments of probabilities of finding initiators based on the specific task considered by an algorithm like MTC. With respect to G this allows the number of verifications in Lemma 3 to be expressed as $N \geq \lceil \frac{\log \epsilon}{\log(1-\Gamma_G(n_q))} \rceil$.

4.3 The impact of graph G

Knowing the graph, an attacker may attempt to minimize the visibility of even a massive attack with ratio q. From an attacker's point of view, it is advantageous to "hide" falsified nodes in the successor graphs of certain tasks in order to achieve the pathological minimum number of initiators $\gamma_G(n_F)$. This allows to generate a general bound on $\gamma_G(n_F)$ based on the the size of successor graphs.

Lemma 5. *Given height h (the length of the critical path) and maximum out-degree d of a graph G, the minimum number of initiators is*

$$\gamma_G(n_F) = \lceil \frac{n_F}{\left(\frac{1-d^h}{1-d}\right)} \rceil. \tag{2}$$

Proof. Given h and d and any task T_i in G the maximal size of the successor graph $G^{\geq}(T_i)$ is bound by $|G^{\geq}(T_i)| \leq 1 + d + d^2 + \ldots + d^{h-1} = \frac{1-d^h}{1-d}$. Thus, a single initiator T_i can "hide" at most $\frac{1-d^h}{1-d}$ of the n_F falsified tasks in F. This would, by definition, make all tasks in $G^{>}(T_i)$ non-initiators. If each initiator can "hide" a maximum of $\frac{1-d^h}{1-d}$ tasks in F, the minimum number of such initiators is $\lceil \frac{n_F}{\left(\frac{1-d^h}{1-d}\right)} \rceil$. Note that this term is has the smallest value for $n_F = n_q$. □

For specific graphs this general worst-case scenario may be overly conservative. The following Extended Monte-Carlo Test (EMCT) will allow graphs with

relatively small predecessor subgraphs to overcome the restrictions imposed by $\gamma_G(n_q)$. Note that it is similar to Algorithm MTC except that it contains previsions to verify all predecessors for the task T selected for verification. Thus, it effectively verifies $G^{\leq}(T)$.

Algorithm EMTC

1. Uniformly chose one task T in G.
2. Re-execute all T_j in $G^{\leq}(T)$, which have not been verified yet, with input $i(T_j, E)$ on a reliable resource and return FAILED if for any T_j we have $\hat{o}(T_j, E) \neq o(T_j, E)$.
3. Return CORRECT.

In the context of our application domain in which G consists mainly of out-trees, Algorithm $EMTC$ exhibits its strength, e.g. the worst number of re-executions in Step 2 is less than height h.

Theorem 2. *For a single execution of Algorithms EMTC the probability of error is $e_E \leq 1 - q$. The average cost in terms of verification, i.e. the expected number of verifications, is*

$$C = \frac{\sum_{T_i \in G} |G^{\leq}(T_i)|}{n}. \tag{3}$$

Proof. A pathological attacker who knows that uniform random task selection is used and that all predecessor tasks are verified can minimize detection by falsifying tasks in such a way as to minimize error propagation, thereby minimizing the total number of tasks affected by falsifications. In other words, in the worst case n_q falsified tasks in G are distributed so that the number of T whose $G^{\leq}(T)$ contain falsified tasks is minimized. This can be achieved in any scenario which attacks the n_q tasks T_i with the smallest successor graph $G^{\geq}(i)$, e.g. first attack only leaf tasks, then tasks at the second level etc. until n_q tasks have been attacked. Finally, the error e_E is 1 minus the probability of $G^{\leq}(T)$ containing a faulty task. In the worst case described above this leads to $e_E \leq 1 - \frac{n_q}{n} \leq 1 - q$.

The average number of verifications is simply the average number of tasks in the predecessor graph verified in $EMTC$ Step 2. Note that once T is selected, the cost can be specified exactly as $|G^{\leq}(T)|$. □

5 Results

Table 1 shows results with respect to a single invocation of the algorithm specified for pathological cases associated with general graphs and out-trees (as indicated). The number of effective initiators is the number of initiators as perceived by the algorithm[3]. The probability of error is a direct result of the number of effective

[3] The term "effective" initiator is used to emphasize that in Algorithm $EMTC$ step 2 any falsification in $G^{\leq}(T)$ is guaranteed to result in detection.

Algorithm	MTC	$EMTC$
Number of effective initiators	$\lceil \frac{n_q}{\left(\frac{1-d^h}{1-d}\right)} \rceil$	n_q
Probability of error	$1 - \frac{\lceil \frac{n_q}{\left(\frac{1-d^h}{1-d}\right)} \rceil}{n}$	$1-q$
Verification cost: general G	1	$O(n)$
Verification cost: G is out-tree	1	$h - log_d(n_v)$
Ave. # effective initiators, G is out-tree	$\lceil \frac{n_q}{\left(\frac{1-(h+2)d^{h+1}+(h+1)d^{h+2}}{(1-d)(1-d^{h+1})}\right)} \rceil$	n_q

Table 1. Results for general graph and forest of out-trees

initiators. The cost of verification reflects the number of tasks verified for each invocation of the algorithm. When G is a forest of out-trees the cost of verification changes based on the number of tasks verified, n_v, since only non-verified tasks are re-executed. If attacks are random, the average number of effective initiators depends on the average size of successor graphs.

Fig. 1. Impact of q on N

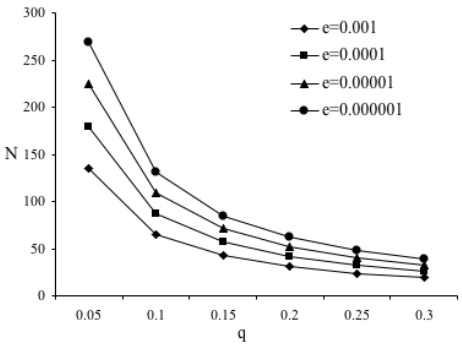
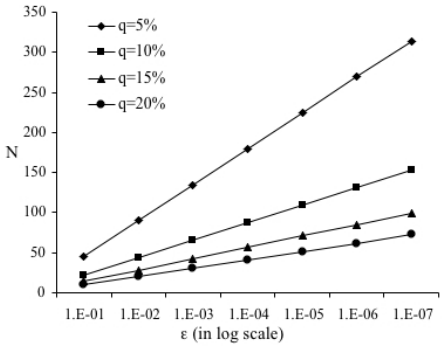


Fig. 2. Impact of ϵ on N



The impact of ϵ and q on the number of invocations of MTC or $EMTC$ is shown next. The results shown in the Figures 1 and 2 are identical for certification of independent tasks with MTC and dependent tasks with $EMTC$. For different values of ϵ Figure 1 shows the impact of attack ratio q on N. It should be noted that N decreases fast with increasing values of q. Conversely, it shows that probabilistic detection is unsuitable for very small q. Next, Figure 2 shows the impact of ϵ on N for fixed attack ratios q. Note that N only grows logarithmically in ϵ. This is very desirable, as it allows for certification with high degrees of certainty, i.e. very small ϵ.

6 Conclusion

This paper discussed certification of large distributed applications executing in hostile environments, where tasks or data may be manipulated by attacks. Unlike previous work based on independent tasks, we considered fault propagation occurring in applications with dependent tasks. In addition we used a probabilistic approach with no assumptions about fault behavior. Two probabilistic algorithms were introduced that selected a small number of tasks to be re-executed on a reliable resource, indicating correct execution with a probability of error based on probabilities associated with task selection. Task re-execution was based on utilizing inputs available through macro data-flow checkpointing. By introducing the concept of initiators, the fault-detection problem associated with fault propagation were overcome. The cost for this were additional verifications noticeable in Algorithm $EMTC$. However, in the context of real-world applications, represented by out-trees, this translated to only minor overhead.

References

1. CERT/CC Statistics 1988-2004, CERT Coordination Center, http://www.cert.org/stats/cert_stats.html
2. Foster, I., Kesselman, C., Nick, J. and Tuecke, S., *Grid Services for Distributed System Integration*, IEEE Computer, No. 6, Vol. 35 (2002) 37-46
3. Fujiwara Hideo, *Logic Testing and Design for Testability*, MIT Press, 1985
4. Gao, L., and Malewicz, G., *Internet Computing of Tasks with Dependencies using Unreliable Workers*, 8th International Conference on Principles of Distributed Systems (OPODIS'04), Dec., 15-17, 2004 (to appear)
5. Germain, C., and Playez, N., *Result Checking in Global Computing Systems*, Proceedings of the 17th Annual ACM International Conference on Supercomputing (ICS 03), San Francisco, California, 23-26 June, (2003) 218-227
6. Jafar S., Varrette S., and Roch J.-L., *Using Data-Flow Analysis for Resilence and Result Checking in Peer to Peer Computations*, Proceedings of the 15th International Workshop on Database and Expert Systems Applications (DEXA 2004), Zaragoza, Espagne, 30^{th} Aug. - 3^{rd} Sep., (2004) 512-516
7. Montagnat, J., Breton, V., and Magnin, I., *Partitioning medical image databases for content-based queries on grid*, Methods of Information in Medicine, Special Issue HealthGrid04, 2004, (to appear)
8. Ragtime: Grille pour le Traitement d'Informations Médicales, Région Rhône-Alpes http://liris.univ-lyon2.fr/∼miguet/ragtime/
9. Sarmenta, Luis F.G., *Sabotage-Tolerance Mechanisms for Volunteer Computing Systems*, Future Generation Computer Systems, Vol. 18, Issue 4 (2002) 561-572
10. Wasserman, H., and M. Blum, Software reliability via run-time result-checking, Journal of the ACM, Vol. 44, No. 6 (1997) 826-849
11. Von Welch, et.al., *Security for Grid Services*, 12^{th} Intl. Symposium on High Performance Distributed Computing (HPDC-12), 22-24 June, Seattle, WA, (2003) 48–57