Certification of Large Distributed Computations with Task Dependencies in Hostile Environments*

Axel W. Krings, Jean-Louis Roch and Samir Jafar
Laboratoire ID-IMAG
(CNRS-INPG-INRIA-UJF – UMR 5132)
38330 Montbonnot Saint-Martin, France
{axel.krings, jean-louis.roch, samir.jafar}@imag.fr

Abstract

This research addresses certification of large distributed applications executing in hostile environments, where tasks or the results they produce may have been corrupted due to benign or malicious act. We extend recent results addressing applications with task dependencies and introduce new probabilistic certification algorithms that establish whether the computations have been massively attacked. The probabilistic approach does not make any assumptions about task behavior as the result of an attack and certification errors are only due to unlucky random choices. Bounds associated with certification are provided for general graphs and for out-trees found in medical image analysis applications of the French Ragtime *project.*

1. Introduction

Global computing systems, e.g. GRID and Peer-to-peer environments, gather thousands of resources for computationally intensive applications. They use middleware such as the Open Grid Service Architecture (OGSA) [2] to provide strong authentication, secure communications [3] and resource management. However, the computational nodes operate in an unbounded environment and are subjected to a wide range of attacks [1]. Whereas the global computations are expected to tolerate certain low rates of faults [4, 10], one should consider the possibility of *massive attacks* resulting in an error rate larger than can be tolerated by the application. Such massive attacks are especially of concern due to the potential of common-mode faults, might they be the result of virus or Trojan attacks or orchestrated attacks against widespread vulnerabilities of specific operating systems.

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

Most research related to protecting large computations against massive attacks has been in the restrictive context of independent tasks. Concepts like voting, spot-checking, blacklisting or credibility-based fault-tolerance have been applied to detect or minimize the influence of attacks [10]. However, there are limitations of such approaches that can be exploited by intelligent adversaries and there is no guarantee that any given faulty execution will be detected as faulty.

An approach using on statistical testing was shown in [4], where a majority of nodes are assumed honest while those nodes compromised by an attack will always falsify their results. Certification was based on randomly selected independent tasks for re-execution on reliable nodes. The approach was extended to consider any parallel computation with dependent tasks, but dependencies were only used for correction [6]. Maximizing the expected number of corrected results under consideration of task dependencies was presented in [5]. However, the problem was shown to be NP-hard and could be exploited by an attacker.

Making no assumptions on the attack and the distribution of errors in the context of a general parallel computation with dependencies, [7] presented certification inspired by probabilistic algorithms. Specifically, given the results of a global computation with task dependencies, it was certified that the computation had not been subjected to a massive attack. We improve on these results and present lower cost probabilistic algorithms that make random choices and return whether an execution is correct or has been massively attacked. Since certification is probabilistic, its output may be wrong. However, 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.

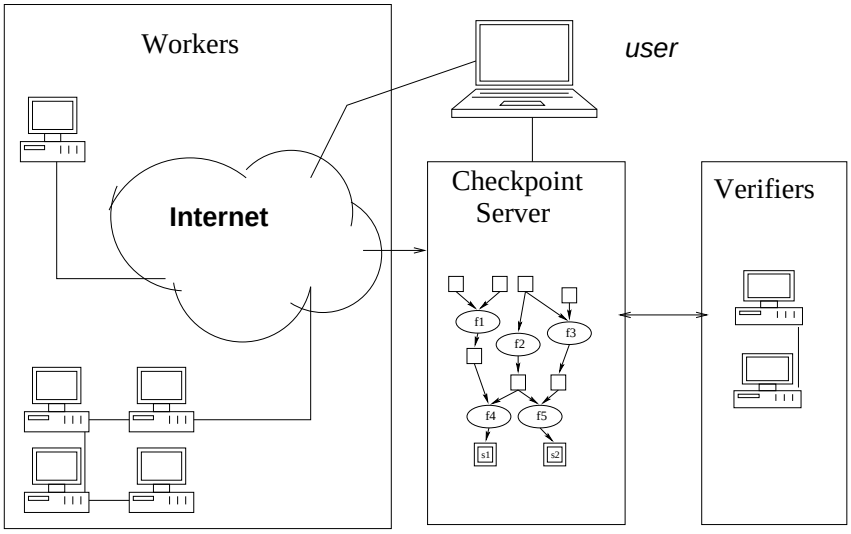

Figure 1. Global computing platform, including workers, checkpoint server and verifiers.

2. Background

We adopt the definitions and assumptions of [7], which are partially restated. The basis for application execution is the global computing platform [6] shown in Figure 1, consisting of workers, user(s), checkpoint server(s) and verifier(s). We assume that a global computation initiated by the user is represented by a direct acyclic graph G, describing the data-flow between all elementary tasks. This graph is assumed to be known by the user a priori. The global computing platform includes two types of resources called *workers* and *verifiers*. Workers are unreliable resources which compute the tasks in G in an non-secure environment. Verifiers are reliable resources which verify the correctness of selected tasks by re-executing them. Communications between workers and verifiers are only performed through a (possibly distributed) checkpoint server, containing computations submitted by workers [6]. Whereas any attack can occur on a worker or between a worker and the checkpoint server, the checkpoint server and verifiers are considered secure.

Data-flow graph: The application is represented by a macro data-flow graph G. Specifically, $G = (\mathcal{V}, \mathcal{E})$ is a directed graph, where $\mathcal{V}$ is the finite set of vertices v_j and $\mathcal{E}$ is the 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 execution. Let D_k denote a data task, representing inputs and outputs of a task. In Figure 1 tasks and data are represented as circles and squares respectively. 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 tasks T_j. The total number of tasks T_j in G is n.

Executions and the impact of faults: Let E denote the execution of a program, represented by G, on a set of unreliable workers. 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 the initial program inputs or outputs of other tasks T_k, i.e. $o(T_k, E)$.

The number of forged tasks in G is denoted by n_F. 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 $\frac{n_F}{n} \geq q$. We assume that either all tasks execute correctly, i.e. $n_F = 0$, or n_F is large, corresponding to a massive attack. These assumptions will be referred to as the *massive attack hypothesis*.

Let $\hat{E}$ denote the execution of the program on a verifier, i.e. a reliable resource, or set thereof. If under the massive attack hypothesis $E = \hat{E}$, 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".

The execution of a task T can be with reference to a worker or a verifier, and its inputs can be from E or $\hat{E}$. Let $i(T, E)$ denote the input of T in E and $\hat{i}(T, \hat{E})$ the input of T in $\hat{E}$. Furthermore, let $o(T, E)$ denote the output of T on the worker, $\hat{o}(T, E)$ 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 $\hat{i}(T, E)$ and the latter $\hat{i}(T, \hat{E})$ as inputs.

Probabilistic certification: Within the context of this research a probabilistic certification is a probabilistic algorithm that uses randomization in order to state if E is failed or not. Given E, a *Monte Carlo certification* is defined as a randomized algorithm that takes as input an arbitrary ϵ, $0 < \epsilon \leq 1$, and delivers (1) either *CORRECT* or (2) *FAILED*, together with a proof that E has failed. The probabilistic certification is said with error ϵ if the probability of the answer *CORRECT*, when E is actually failed, is less than or equal to ϵ. The objective is to provide a probabilistic Monte Carlo certification against massive attacks and the massive attack hypothesis is only a suitable mechanism to discount low rates of attacks.

3. Certification with dependencies

In [7] two certification algorithms, *Monte Carlo Test* (MCT) and *Extended Monte Carlo Test* (EMCT), suitable for graphs with dependencies were introduced. We will improve on these algorithms and introduce two kinds of certification tests with lower cost. However, we first need to present several definitions.

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 $G^{\leq}(T) = G^{<}(T) \cup \{T\}$.

Certification based on task re-execution on verifiers presents challenges when considering task dependencies. This is due to limitations of re-execution in the presence of fault-propagation. We will utilize the concept of *initiators* in order to address the problem of fault-propagation.

Let F denote the set of all falsified tasks for a given G with n tasks. An initiator is a task $T_i \in F$ which has no predecessors in F, i.e. it has not falsified predecessors. The *initiator set* $\mathcal{I}(F)$ is defined as the set of all initiators, i.e.

$$\mathcal{I}(F) = \{T_i \in F : F \cap G^{<}(T_i) = \emptyset\}.$$

Let n_I denote the total number of initiators, i.e. $n_I = |\mathcal{I}(F)|$. It is obvious that the actual tasks in F and $\mathcal{I}(F)$ are not known, since otherwise certification would be trivial.

Next, 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

$$\gamma_V(k) = min|G^{\leq}(V) \cap \mathcal{I}(F)|$$

over all $F \subseteq G$, s.t. $|F| \geq k$ and $G^{\leq}(V) \cap \mathcal{I}(F) \neq \emptyset$. Thus $\gamma_G(n_F)$, for $n_F = n_q$, is the smallest n_I possible, i.e. the n_I associated with a pathological attack scenario.

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

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

The minimal initiator ratio is helpful in determining bounds on the probabilities associated with selecting initiators in predecessor sets.

3.1. Relationships between quantities

For an arbitrary subset of tasks V in G, the relationship between $\gamma_V(k)$, $\gamma_G(k)$, and n_I with respect to $k = n_F$ or $k = n_q$ is not obvious. By definition, we have $q \leq \frac{n_F}{n}$ and thus $n_q \leq n_F$. Also, by definition $n_I \leq n_F$. The minimum number of initiators is defined with respect to the pathological attack scenario, whereas n_I and n_F are the actual number of initiators and falsified tasks respectively. Thus, we always have

$$\gamma_V(n_F) \leq \gamma_G(n_F) \leq n_I \leq n_F.$$

However, where does n_q fit into this inequality? The only certain relationship is $n_q \leq n_F$. On the other hand, with respect to n_q we can always say that

$$\gamma_V(n_q) \leq \gamma_G(n_q) \leq n_q \leq n_F.$$

However, where does n_I fit into this inequality? The only certain relationship is $\gamma_G(n_q) \leq n_I \leq n_F$. On the other hand, with respect to $n_q \leq n_F$ the different quantities can be directly compared, i.e. $\gamma_V(n_q) \leq \gamma_V(n_F)$ and $\gamma_G(n_q) \leq \gamma_G(n_F)$. This implies that $\Gamma_V(n_q) \leq \Gamma_V(n_F)$ and $\Gamma_G(n_q) \leq \Gamma_G(n_F)$.

3.2. Verifying fractions of $G^{\leq}(T)$

We first consider an algorithm that re-executes a fraction α, $0 < \alpha \leq 1$, of tasks in predecessor graphs. It differs from Algorithm $EMCT(E)$ presented in [7] only in Step 2, where the entire $G^{\leq}(T)$ rather than a fraction thereof was re-executed.

Algorithm $EMCT_\alpha(E)$

1. Uniformly choose one task T in G.
2. Uniformly select $n_\alpha = \lceil \alpha |G^{\leq}(T)| \rceil$ tasks in $G^{\leq}(T)$ and let this set be denoted by A. If for any $T_j \in A$, that has not been verified yet, re-execution on a verifier results in $\hat{o}(T_j, E) \neq o(T_j, E)$ then return FAILED.
3. Return CORRECT.

If the algorithm returns CORRECT it implies that n_α tasks in $G^{\leq}(T)$ have been verified. Now the following lemma can be stated:

Lemma 1 *Let T be a task randomly chosen by $EMCT_\alpha(E)$. Then the probability of error, e_α, when $EMCT_\alpha(E)$ returns CORRECT is given by*

$$e_\alpha \leq \begin{cases} (1 - q\alpha\Gamma_T(n_q)) & \text{for} \quad 0 < \alpha \leq 1 - \Gamma_T(n_q) \\ (1-q) & otherwise. \end{cases}$$

Proof: Assume that E has been falsified, but $EMCT_\alpha(E)$ returns CORRECT. Let p denote the probability that the result of T is correct. The result of T is either correct or it is incorrect, implying that there is at least one initiator in $G^{\leq}(T)$. The probability that $EMCT_\alpha(E)$ returns CORRECT is composed of (1) the probability p that the result of T was indeed correct and (2) the probability that the result of T is incorrect but none of the n_α randomly chosen tasks in A turned out to be initiators. Thus

$$e_\alpha \leq p + (1-p)Prob(no\ initiator\ in\ A). \qquad (1)$$

Depending on the value of α two cases exist:

1. First, there is a probability that A contains no initator when $n_\alpha \leq |G^{\leq}(T)| - \gamma_T(n_q)$ and thus $\alpha \leq 1 - \frac{\gamma_T(n_q)}{|G^{\leq}(T)|} = 1 - \Gamma_T(n_q)$.
2. Second, A contains at least one initator when $n_\alpha > |G^{\leq}(T)| - \gamma_T(n_q)$ and thus $Prob(no\ initiator\ in\ A) = 0$. This is the case when $\alpha > 1 - \Gamma_T(n_q)$. We say that α is *saturated*.

We first consider Equation 1 for $\alpha \leq 1 - \Gamma_T(n_q)$. For a given attack scenario, the minimum number of tasks that have incorrect results is n_F. This is the case when there is no error propagation to any non-faulty task. Thus $p \leq 1 - \frac{n_F}{n}$ and substituting p in Equation 1 we get

$$\begin{aligned} e_\alpha &\leq p + (1-p)(1 - \alpha\Gamma_T(n_F)) \\ &\leq 1 - \frac{n_F}{n}\alpha\Gamma_T(n_F) \\ &\leq 1 - \frac{n_q}{n}\alpha\Gamma_T(n_F) \\ &\leq 1 - q\alpha\Gamma_T(n_F) \\ &\leq 1 - q\alpha\Gamma_T(n_q). \end{aligned}$$

Next, we consider Equation 1 for $\alpha > 1 - \Gamma_T(n_q)$, i.e. α is saturated. Now

$$\begin{aligned} e_\alpha &\leq p + (1-p)Prob(no\ initiator\ in\ A) = p \\ &\leq 1 - \frac{n_F}{n} \leq 1 - \frac{n_q}{n} \leq 1 - q. \quad \square \end{aligned}$$

Given Lemma 1 we can now relate ϵ, α, q and N.

Theorem 1 *Let E be an execution with dependencies that is either correct or massively attacked with ratio q. Given ϵ and $0 < \alpha \leq 1$, N independent invocations of Algorithm $EMCT_\alpha(E)$ provide a certification with error probability*

$$\epsilon \leq \begin{cases} (1 - q\alpha\Gamma_G(n_q))^N & for\ 0 < \alpha \leq 1 - \Gamma_T(n_q) \\ (1-q)^N & otherwise. \end{cases}$$

Proof: To prove the first case, we know from Lemma 1 that a single invocation of Algorithm $EMCT_\alpha(E)$ can produce a maximal error of $1 - q\alpha\Gamma_T(n_q)$. This term is largest when $\Gamma_T(n_q)$ is minimal and it depends on the T selected. Therefore $\Gamma_T(n_q)$, and thus the actual number of independent invocations, can only be determined at run-time when the specific T is known. However, in the worst case, for any $T \in G$ we have $\Gamma_T(n_q) \geq \Gamma_G(n_q)$ and thus $1 - \Gamma_T(n_q) \leq 1 - \Gamma_G(n_q)$. Then from $\epsilon \leq (1 - q\alpha\Gamma_G(n_q))^N$ we get the bound

$$N \geq \frac{\log \epsilon}{log(1 - q\alpha\Gamma_G(n_q))}.$$

The proof of the second case follows immediately from the fact that each invocation contributes a maximal error of $1 - q$ shown in Lemma 1. $\square$

3.3. Verifying fixed numbers of tasks

Rather than verifying a fraction of tasks as discussed previously we now fix the number of tasks in $G^{\leq}(T)$ to be tested. We limit our consideration to the optimal case of unity, i.e. only one task in $G^{\leq}(T)$ is verified.

Algorithm $EMCT^1(E)$

1. Uniformly choose one task T in G.
2. Uniformly select a single T_j in $G^{\leq}(T)$. If re-execution of T_j on a verifier results in $\hat{o}(T_j, E) \neq o(T_j, E)$ then return FAILED.
3. Return CORRECT.

Now the following lemma, which is similar to Lemma 1, can be stated.

Lemma 2 *Let T be a task randomly chosen by $EMCT^1(E)$ and let $V = G^{\leq}(T)$. Then the probability of error, e_1, when $EMCT^1(E)$ returns CORRECT is given by*

$$e_1 \leq 1 - \frac{n_F}{n}\Gamma_T(n_F) \leq 1 - q\Gamma_T(n_q)$$

Proof: Let T_j be the task selected from $G^{\leq}(T)$ in step 2 of $EMCT^1(E)$. Assume that E has been falsified, but $EMCT^1(E)$ returns CORRECT. Again, let p be the probability that the result of T is correct. The output of T is either

correct or it is incorrect, implying that there is at least one initiator in $G^{\leq}(T)$. Then the probability that $EMCT^1(E)$ returns CORRECT is composed of (1) the probability p that the result of T was indeed correct and (2) the probability that the result of T is incorrect but T_j was not an initiator. Thus

$$\begin{aligned} e_1 &\leq p + (1-p)Prob(T_j \; is \; not \; an \; initiator) \\ &= p + (1-p)(1 - \Gamma_T(n_F)). \end{aligned}$$

For a given attack scenario, the minimum number of tasks that have incorrect results is n_F. Thus $p \leq 1 - \frac{n_F}{n}$ and substituting p in the previous equation we get

$$\begin{aligned} e_1 &\leq 1 - \frac{n_F}{n}\Gamma_T(n_F) \\ &\leq 1 - \frac{n_q}{n}\Gamma_T(n_F) \\ &\leq 1 - \frac{n_q}{n}\Gamma_T(n_q) \quad \square \end{aligned}$$

We can now relate ϵ, q and N.

Theorem 2 *Let E be an execution with dependencies that is either correct or massively attacked with ratio q. Given ϵ then N independent invocations of Algorithm $EMCT^1(E)$ provide a certification with error probability*

$$\epsilon \leq (1 - q\Gamma_G(n_q))^N.$$

The proof is similar to the one in Theorem 1.

4. Results

In order to show the strength of the $EMCT_\alpha(E)$ and $EMCT^1(E)$ algorithms we compare them with the $MCT(E)$ and $EMCT(E)$ algorithm of [7] in Table 1.

For each algorithm the number of *effective initiators* is indicated. We use the term "effective initiator" to emphasize that this is the number of initiators as perceived by the algorithm. Whereas the real number of initiators is given only for $MCT(E)$, the number for the others is reflecting the probability of stumbling onto an initiator. For example, since $EMCT(E)$ re-executes the entire predecessor subgraph, a falsified task in $G^{\leq}(T)$ is *always* found, if it exists. The largest number of effective initiators that can be achieved is n_q. This is the case when G has no dependencies and thus every falsified tasks is an initiator [4, 6, 7].

The *convergence*, i.e. the number N of invocations of the specific algorithms necessary to certify an execution for a given ϵ depends on the probability of error contributed by a single invocation of the algorithm. For each algorithm, this probability is shown in the second row of Table 1. However, it depends on the graph and the specific task T selected by the algorithm for those algorithms affected by $\Gamma_T(n_q)$, i.e. for $EMCT_\alpha(E)$ with $0 < \alpha \leq 1 - \Gamma_T(n_q)$ (see Lemma 1) and $EMCT^1(E)$.

Next we want to indicate the convergence, i.e. N, for all algorithms in general. Let q_e denote the *effective attack ratio* associated with an algorithm. Then for a single invocation of an algorithm the contribution to the certification error is $1 - q_e$. If ϵ_e denotes the current cumulative error of a certification, then after another single invocation of an algorithm we have $\epsilon_e = \epsilon_e(1 - q_e)$.

The number of invocations necessary to achieve a Monte Carlo certification for a given ϵ can be determined in two ways, (1) a priori and (2) at run-time.

1. In the first case it is the result of a fixed number of invocations determined a priori. However, this approach is more pessimistic, since it cannot take advantage of $\Gamma_T(n_q)$ but rather has to use $\Gamma_G(n_q)$. The a priori convergence can be computed as $N \geq \frac{\log(\epsilon)}{\log(1-q_e)}$ (see Theorem 1) and is shown in the table. The effective attack ratios (q_e a priori) associated with each algorithm is shown as well.

2. In the second case it is the result of a run-time dependent process that starts with $\epsilon_e = 1 - q_e$ for the first invocation and continues invoking the algorithm, computing $\epsilon_e = \epsilon_e(1 - q_e)$ each time, until $\epsilon_e \leq \epsilon$. Since the number of invocations is run-time dependent, it cannot be explicitly stated. However, the a priori convergence shown in Table 1 constitutes an upper bound. The effective attack ratio (q_e run-time) associated with each invocation is also shown.

The convergence indicates the number of invocations of the respective algorithms to achieve certification with ϵ. Whereas it is a good measure of how fast the certification achieves the desired ϵ, it does not indicate the cost associated with each invocation.

The *verification cost* refers to the number of tasks to be verified in one invocation of the algorithm. The exact verification costs are shown first. It should be noted that $EMCT_\alpha(E)$ reduces the verification cost of $EMCT(E)$ by factor α. As was shown in Theorem 1, for certain values of α this comes at no penalty with respect to convergence. The same can be said about algorithms $MCT(E)$ and $EMCT^1(E)$, where for the same cost of unity the latter algorithm shows faster convergence.

It is interesting to note the difference in the performance of $EMCT_\alpha(E)$ (with unsaturated α) and $EMCT^1(E)$ that clearly favors the latter. In fact, $EMCT^1(E)$ has a lower verification cost of unity *and* faster convergence, since it is not burdened by the term α in the denominator.

This certification research was motivated by medical applications [8] studied in the context of the French research project Ragtime [9], which are heavily based on out-trees, for which the maximum verification cost is shown last.

	$MCT(E)$ [7]	$EMCT(E)$ [7]	$EMCT_\alpha(E)$	$EMCT^1(E)$
# of effective initiators	$\lceil \frac{n_q}{\left(\frac{1-d^h}{1-d}\right)} \rceil$	n_q	$n_q \alpha \Gamma_T(n_q)$ or n_q	$n_q \Gamma_T(n_q)$
Probability of error	$1 - \frac{\lceil \frac{n_q}{\left(\frac{1-d^h}{1-d}\right)} \rceil}{n}$	$1-q$	$1 - q\alpha\Gamma_T(n_q)$ or $1-q$	$1 - q\Gamma_T(n_q)$
A priori convergence	$\frac{\log \epsilon}{\log(1 - \frac{\lceil \frac{n_q}{\left(\frac{1-d^h}{1-d}\right)} \rceil}{n})}$	$\frac{\log \epsilon}{\log(1-q)}$	$\frac{\log \epsilon}{\log(1-q\alpha\Gamma_G(n_q))}$ or $\frac{\log \epsilon}{\log(1-q)}$	$\frac{\log \epsilon}{\log(1-q\Gamma_G(n_q))}$
q_e a priori	$\frac{\lceil \frac{n_q}{\left(\frac{1-d^h}{1-d}\right)} \rceil}{n}$	q	$q\alpha\Gamma_G(n_q)$ or q	$q\Gamma_G(n_q)$
q_e run-time	$\frac{\lceil \frac{n_q}{\left(\frac{1-d^h}{1-d}\right)} \rceil}{n}$	q	$q\alpha\Gamma_T(n_q)$ or q	$q\Gamma_T(n_q)$
Verification cost (exact)	1	$\lvert G^{\leq}(T) \rvert$	$\lceil \alpha \lvert G^{\leq}(T) \rvert \rceil$	1
Max. cost (out-tree)	1	h	αh	1

Table 1. Results for pathological general case and out-trees

5. Conclusion

This paper expanded the theory of certification of large distributed computations in hostile environments, where tasks or data may be falsified by malicious intruders. Whereas it is assumed that an application can tolerate a certain small number of falsified results, it is necessary to know if a massive attack has taken place, surpassing the application's fault-tolerance. The *massive attack hypothesis* was used as the basis for Monte Carlo certification against massive attacks suitable for the global computing platform. The probabilistic certification is based on task re-execution on reliable resources, using tasks and data available from macro data-flow checkpointing.

Two probabilistic algorithms, $EMCT_\alpha(E)$ and $EMCT^1(E)$, were introduced that improve on the algorithms presented in [7]. They make no assumptions about the behavior of faults and utilize the concept of initiators to address impact of fault-propagation. The algorithms have lower convergence and certification cost compared to previous work.

The issues of certification under consideration of information available at run-time, after the random selections of the algorithms, were discussed. It was shown that a priori convergence constitutes an upper bound to run-time convergence. For each algorithm the verification cost was given for general graphs and for out-trees, which represent the typical application graphs associated with medical applications in the context of the French Ragtime project.

References

[1] CERT/CC Statistics 1988-2004, CERT Coordination Center, http://www.cert.org/stats/cert_stats.html

[2] Foster, I., et.al., *Grid Services for Distributed System Integration*, IEEE Computer, No. 6, Vol. 35, 2002.

[3] Foster, I., et.al., *Security for Grid Services*, Twelfth International Symposium on High Performance Distributed Computing (HPDC-12), 2003.

[4] Germain, C. and Playez, N., *Result Checking in Global Computing Systems*, Proc. 17th Annual ACM International Conference on Supercomputing (ICS 03), June 23-26, 2003

[5] Gao, L. and Malewicz, G., *Internet Computing of Tasks with Dependencies using Unreliable Workers*, 8th International Conference on Principles of Distributed Systems (OPODIS'04), to appear in Springer Verlag LNCS.

[6] Jafar S., et.al., *Using Data-Flow Analysis for Resilence and Result Checking in Peer to Peer Computations*, 15^{th} Intl. Workshop on Database and Expert Systems Applications, 2004, pp. 512-516.

[7] Krings, A.W., et.al.. *A Probabilistic Approach for Task and Result Certification of Large-scale Distributed Applications in Hostile Environments*, EGC-2005, LNCS 3470, 2005.

[8] Montagnat, J. and Breton, V. and Magnin, I., *Partitioning medical image databases for content-based queries on grid*, Methods of Information in Medicine, Special Issue HealthGrid04, to appear.

[9] Ragtime: Grille pour le Traitement d'Informations Médicales, Région Rhône-Alpes http://liris.univ-lyon2.fr/ miguet/ragtime/

[10] Sarmenta, Luis F.G., *Sabotage-Tolerance Mechanisms for Volunteer Computing Systems*, Future Generation Computer Systems, No. 4, Vol. 18, 2002.