GIVS: Integrity Validation for Grid Security

Giuliano Casale and Stefano Zanero*

Dipartimento di Elettronica e Informazione
Politecnico di Milano - via Ponzio 34/5 - 20133 Milano Italy
[casale,zanero]@elet.polimi.it

Abstract. In this paper we address the problem of granting the correctness of Grid computations. We introduce a Grid Integrity Validation Scheme (GIVS) that may reveal the presence of malicious hosts by statistical sampling of computations results. Performance overheads of GIVS strategies are evaluated using statistical models and simulation.

1 Introduction

Computational grids are touted as the next paradigm of computation [1]. Since they provide access to the resources needed for computational intensive applications, a Grid enviroment usually spans a heterogeneous set of machines.

When a Grid extends beyond the systems of a single administrative authority it suddenly becomes a network of potentially untrusted nodes, on which a remote user submits a potentially harmful piece of code together with sensitive data, as already happens in generic peer-to-peer systems [2]. The ensuing problems of security and privacy have not been fully explored in literature yet.

We identify at least four different problems that arise when extending Grid computing beyond a trusted network:

1. Defending each participant from the effects of potentially malicious code executed on the Grid [3, 4]
2. Avoiding unfair parties which tap computational resources without sharing their own [5, 6]
3. The privacy of data submitted for computation should be adequately protected, depending on their sensitivity
4. The effect of the presence of one or more peers that are interested in making the overall computation fail

Our paper will deal in particular with the last problem. We propose a general scheme, named Grid Integrity Validation Scheme (GIVS), based on problem replication and submission of test problems to the Grid hosts.We evaluate the effectiveness and the overhead of the proposed solution both analitically and through simulation.

The paper is organized as follows: in Section 2 we analyze the problem and we introduce our integrity scheme. Section 3 describes a statistical model for evaluating the security level and performance trade-offs of alternative validation scheme within GIVS. Finally, in Section 4 we draw conclusions.

* This work has been partially supported by the Italian FIRB-Perf project

2 The Grid Integrity Validation Scheme

The problem of detecting when remote untrusted machine is running a particular piece of mobile code is difficult to solve in the general case. Checksums and digital signatures can be used to verify the identity of a piece of code we have received, but cannot be used to prove to a remote party that we are actually executing it. Cryptographic protocols would require that the mobile code is endowed with a secret that cannot be accessed by someone controlling the host on which it is running, a constraint that looks impossible to achieve.

A possible solution would be to design the code in such a way that tampering with results in a non-detectable way would require a much longer time than calculating the real solution. Nevertheless, this would require a correct estimate of the real computation times, and thus would implicitly rely on the fact that the malicious user will correctly declare the computational power he has available.

Unless the particular problem we are dealing with has some sort of checksum property, which allows to check the correctness of the results without having to rerun the program, the only feasible approach is to check the correctness of the results by sampling. Specifically, we propose to use some *test problems* to check the correctness of the results. The idea resembles a blind signature protocol [7], in which the submitter prepares N similar documents, the signer opens $(N-1)$ documents checking they are correct, and then blindly signs the last document, with a probability $p = 1/N$ to be cheated. In our case, documents are replaced by computational tasks. We also require to perform our controls by harnessing the computational power of the Grid as much as possible. This is the key idea of our Scheme for Integrity Validation of Grid computation, or GIVS.

In order to generate the test problems required for the integrity validation, we propose a *bootstrap* phase. During the bootstrap, one or more trusted machines compute correct solutions to a subset of test problems, chosen among the available ones according to some criteria (e.g. difficulty, sensitivity to errors, ...). These problems are then used during normal Grid activity to perform security controls, since when a test problem is scheduled to a user, the computated solution is compared to the correct solution to detect whether the user is malicious. Hence, users trying to corrupt the solution of a test problem would be easily identified.

While test problems seem a simple solution to the Grid integrity validation problem, some drawbacks can be identified. First, test problems require a bootstrap phase that slows down the computation startup and waste part of resources. Moreover, increasing the number of test problems could turn into a continuous performance overhead that may become unacceptable.

An alternative solution, consists in replicating a set of problems of unknown solution throughout the Grid, that we call *replicated problems* or just *replicas*. The integrity validation is performed in this case by comparing the different results provided by the untrusted hosts to the same replicated problem. If a conflict is detected, a trusted machine is asked to compute the correct result, so that the malicious hosts can be identified. Compared to the test problems approach, we are now accepting a trade-off between the performance overheads

imposed by the integrity validation and the degree to which we can rely on our own security controls.

Notice that if a subset of M machines are cooperating to cheat, if they receive any test problem o replica in common, they can detect it and cheat. Moreover, if any of the results previously computed by any of the M machines is given to another machine as a test problem, it can be detected. The risk connected to this situation are evaluated in the next sections.

3 Performance-security trade-offs

3.1 Notation and basic definitions

Let us begin to model a Grid composed of H hosts, with $M \leq H$ malicious nodes that are cooperatively trying to disrupt the computation. In normal conditions we expect $M << H$.

Let T be a period of activity of the Grid after the bootstrap phase. We discretize T in the K intervals $T_1, \ldots, T_K$, thus $\sum T_k = T$. In each period T_k, every host h receives a collection $\mathbf{P_h^k} = (p_1^k, \ldots, p_P^k)$ of P_h problems[1]. We assume the solution of each problem p_i takes the same computational effort on all machines of the Grid. We can classify the problems $p_m \in \mathbf{P_h}$ in several classes: we denote test problems with q_m, replicas with r_m, and generic problems, which don't have any role in security controls, are denoted with g_m. Thus, for instance, the set $\mathbf{P_h} = (c_1, \ldots, c_{C_h}, r_1, \ldots, r_{R_h}, g_1, \ldots, g_{G_h})$ is composed of C_h test problems, R_h replicas and G_h generic problems with $P_h = C_h + R_h + G_h$.

In the rest of this section we explore the two different strategies for GIVS described in Section 2: in the first one we use only test problems, while in the second all test problems are replaced with replicas. Our aim is to study, for each strategy, the performance overhead OH introduced by the security controls, and the probability p_{CF} of accepting a corrupted final value as a correct result.

3.2 First strategy: test problems

We now consider that each host receives a constant number of problems P_h, among which G_h are generic problems, while $C_h = P_h - G_h$ are test problems. Then, no problem replicas are present in the sets $\mathbf{P_h}$ ($R_h = 0$). We denote with $\mathbf{q_h}$ the subset of test problems of $\mathbf{P_h}$ and with C_h the cardinality of $\mathbf{q_h}$. All test problems $q_m \in \mathbf{q_h}$ are distinct and extracted randomly from a set of C elements.

Under these assumptions, the performance overhead OH of this scheme is given by $OH = H \cdot C_h$ since the solution of the test problems does not give any useful result for the applications running on the Grid.

We observe that the probability that two machines share the same control set is negligible. In fact, given that the probability of receiving a particular control

[1] For the clarity of the notation, we will omit in the rest of the paper the indices k when we do not need to refer to different time slices.

set is the inverse of the number of possible distinct sets $\mathbf{q_h}$, we have

$$p(\mathbf{q_{h_k}}) = \binom{C}{C_h}^{-1} = \frac{C_h!\,(C-C_h)!}{C_h!}.$$

Now, observing that two random extractions $\mathbf{q_1}$ and $\mathbf{q_2}$ of the control sets are independent, the probability of two hosts h_1 and h_2 of receiving the same control set $\mathbf{q_{h_k}}$ is

$$p(\mathbf{q_{h_1}} \equiv \mathbf{q_{h_2}}) = p(\mathbf{q_{h_k}})^2 = \left(\frac{C_h!\,(C-C_h)!}{C!}\right)^2.$$

In non-trivial cases this probability is extremely low. As an example, with $C = 5$ and $C_h = 3$, it narrows down to a modest 0.25%. As C grows, this probability tends quickly to zero.

We now consider the probability $p(q_m \in \mathbf{q_h})$ that a particular problem q_m belongs to a control set $\mathbf{q_h}$. This probability is modeled using an hypergeometric distribution[2], since it reduces to the probability of extracting a winning ball from a set of C, in C_h extractions without replacement. Then we have

$$p(q_m \in \mathbf{q_h}) = hypergeom(C, 1, C_h, 1) = \frac{(C-1)!}{C!}\frac{C_h!(C-C_h)!}{(C_h-1)!(C-C_h)!} = \frac{C_h}{C}$$

It is interesting to note that such probability is the same as if we would have dropped the no-replacement condition in the extraction, and so it suggest that, from a statistical point of view, the condition of uniqueness of the test problems in the same $\mathbf{q_h}$ can be dropped.

We now turn our attention to p_{CF}. In order to quantify such probability, we first need to estimate the mean number $\bar{s}$ of test problems that each user shares with the other malicious hosts. We can try to model such quantity as the probability $P_M(k)$ that each host of a group of M has exactly k overlaps with the others. For the case $M = 2$, host i has probability $P_2(k) = hypergeom(C, C_h, C_h, k)$, since the winning problems are exactly the C_h that belong to the control set $\mathbf{q_j}$ of host j. Unfortunately, if we try to extend this idea to the case with three or more hosts, we find that we cannot simply add together the pairwise probabilities, because we must account for the overlap between all three sets. This extension is quite difficult to derive analitically with a closed form, and due to its combinatorial nature, it raises some concerns about the computational complexity of a possible iterative description. Moreover, what we actually need is the mean number of overlaps $\bar{s}$, rather than the complete probability distribution, and so a simulation can provide accurate estimates of $\bar{s}$ at a reasonable computational cost and with limited modelling efforts.

In Figure 1 are shown the estimated $\bar{s}$, as a function of the number of malicious hosts M and of the total number of test problems C for the cases $C_h/C = 1/2$ and $C_h/C = 1/4$. All the experiments have been conducted using

[2] We denote with $hypergeom(n+m,\, n,\, N,\, i)$ the probability of having i successful selections from a hypergeometric distribution with n winning balls out of $n+m$ and after N extractions.

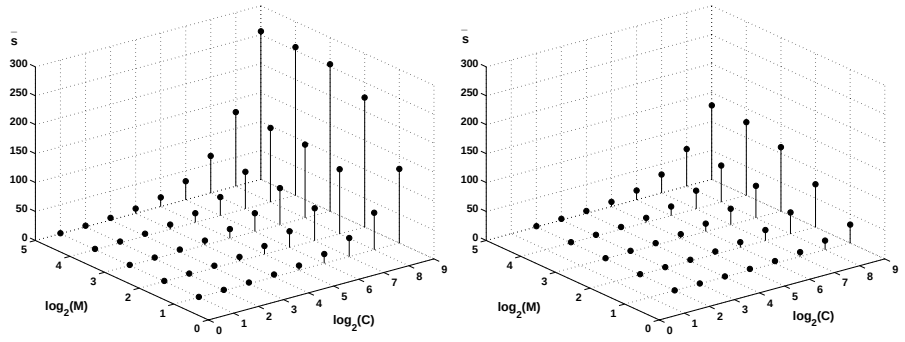

Fig. 1. Expected number of overlappings for different ratios of C_h versus C. In the left figure, $C_h/C = 1/2$. On the right, $C_h/C = 1/4$.

an Independent Replications Estimate of the mean $\bar{s}$ values with 50 replication of a total of over 1500 different samples. Confidence intervals have been computed for a 95% confidence level; however, since such intervals have resulted to be extremely tight to the estimated $\bar{s}$, we have omitted them from all figures.

Assuming now to know the value of $\bar{s}$, we can model the success probability p_{CF} of a malicious user using an hypergeometric distribution. We consider a malicious user h who gives b wrong answers to the problems of $\mathbf{P_h}$ and answers correctly to the $\bar{s}$ test problems that he has in common with other malicious users. In this case its probability of success p_{CF} is the probability of never giving a wrong answer a test problem, i.e.

$$p_{CF} \cong hypergeom(P_h - \bar{s}_+, C_h - \bar{s}_+, b, 0) = \frac{(P_h - C_h)!(P_h - \bar{s}_+ - b)!}{(P - C_h - b)!(P_h - \bar{s}_+)!}$$

where $\bar{s}_+ = ceil(\bar{s})$ is used instead of $\bar{s}$ since the formulas require integer values. Please note that if $\bar{s}$ is integer the previous formula holds with the equality. The previous formula is defined only for $b < C_h - \bar{s}_+$, otherwise we have trivially that the user always corrupts a test problems and so $p_{CF} = 0$.

The qualitative behavior of p_{CF} is shown in Figure 2, where the probability is plotted against the ratio of undetected test problems ($C_h - \bar{s}_+$) to the number of problems P_h. The figure is plotted with $\bar{s} = 0$, since this term simply shifts the origin of the x-axis of a corresponding quantity (e.g. with $\bar{s} = 10$ p_{CF} is 1 for $x = C_h/P_h = 10$). Several observations can be drawn from the shape of the p_{CF}. First, as intuitive, if a host tries to cheat on a single problem, its probability of success decreases linearly with the number of test problems. This has a very important consequence on every integrity validation schema: if the correctness of the global computation requires all subtasks to be computed correctly, a severe performance overhead is required to grant to grant the correctness of the results. Then, according to the model, a Grid provider would require a strong additional cost to grant the security of such applications. However, in many cases of interest

a malicious user would need to affect a large number of tasks in to affect the global solution. For example, affecting a multiobjective optimization of a set of functions over a numerical domain would probably require the simultaneous corruption of the results on many different subdomains for each of the objective functions. In the case of multiple corruptions ($b >> 1$), the probability p_{CF} drops quickly as the number of test problems grows. This suggests, if possible, to split the functional domain in the highest possible number of regions, although a trade-off with the involved communication overhead should be also taken into account [8]. However, even in this case, a large waste of computational resources is required to grant low values of p_{CF}. Therefore, using replicas could be a suitable way to handle these problems.

3.3 Second strategy: validation using replicated problems

We are now going to extend the model developed for the first strategy to the case where replicated problems replace test problems. Problem sets have now the general form $\mathbf{P_h} = (r_1, \ldots, r_{R_h}, g_1, \ldots, g_{G_h}) = \mathbf{r_h} \cup \mathbf{g_h}$, and the replicas $r_m \in \mathbf{r_h}$ are drawn from a set of U problems of unknown solution, in such a way that ρ replicas of each problem are scheduled on the Grid. As explained before, the main advantage here is that, as a side effect of the security controls, the solution of U problems is computed, and so less computational resources are wasted for security purposes. In fact, the minimum performance overhead when ρ replicas of each of the U problems are sent on the Grid is $OH^- = (\rho - 1)U$ that accounts for the fact that replicas also yield the solution of the unknown problems. However, we must handle the case where m problems have been recognized as potentially corrupted, and so the performance overhead becomes $OH = (\rho - 1)U + 2m$ since we lack m solutions and these must be recomputed by trusted machines. However, in normal conditions we expect $OH \cong OH^-$.

As observed for the first strategy, finding an analytical expression for the average number of overlappings $\bar{s}$ is quite difficult. Furthermore, the model of the second strategy should include additional constraints and problems that are difficult to be handled by means of a statistical model. In fact, if we want to ensure that exactly ρ replicas of each of the U problems are computed on the H hosts, during the assignment of the replicas we must satisfy the constraint $C_h H = \rho U$.

Moreover, we observe that the constraint excludes the possibility of using a completely random assignment of the replicas: let us consider, for example, Table 1, where is shown a possible random assignment of $U = 10$ problems with a replication factor $\rho = 2$ (i.e. $\rho U = 20$). In this case we still need to assign two replicated problems to host h_5, but only the replicas of problems 2 and 9 are still unassigned. But this assignment is unacceptable, because it would break the condition that a host must receive distinct test problems. In this case, a random strategy would lock searching for an impossible assignment of the replicas.

A simple assignment scheme that avoids locking is the following: we assign randomly, until possible, the replicas. When a lock condition is detected, we

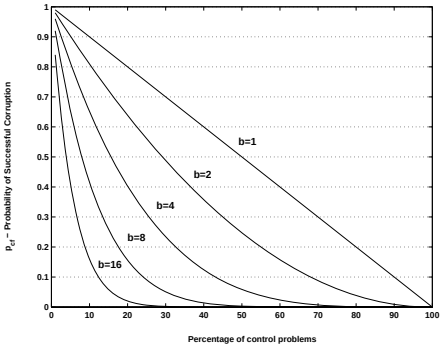

Fig. 2. Probability p_{CF} of accepting a corrupted result as function of C_h/P_h (here is $\bar{s}_+ = 0$) and of the number of simultaneous wrong answers b

Host	Replicated problems	
h_1	1 5 7	10
h_2	3 7 8	10
h_3	3 4 6	8
h_4	1 4 5	6
h_5	2 9 **?**	**?**
Unassigned replicated problems		
Avail	2 9	

Table 1. An example of bad random assignment: any further assignment would alert h_5 that he is subject to a security control.

change some of the previous assignment in order to solve correctly the assignment. For instance, in Table 1 we could easily avoid locking assigning replicas 2 and 9 to h_4 and moving 5 and 6 to h_5. It is clear that such heuristics is quite complex to be modeled analitically, while using simulation we can easily estimate the involved overlappings $\bar{s}$. In Figure 3 we show in logarithmic scale some numerical results from a simulation with $H = 48$ and $\rho = 2$: we can identify an exponential growth of the overlapping as the number of malicious users M and the unknown problems U grow. This turns to be a linear growth in linear scale, and so we conjecture an approximation $\bar{s} \cong f(M, U)$, where $f(\cdot, \cdot)$ is a linear function.

Given that $\bar{s}$ are known, the analysis of the probability p_{CF} for the second strategy results in the same formula derived for the first strategy, where the quantities C_h and C are replaced respectively by R_h and U. We also observe that, in most cases, the average overlappings $\bar{s}$ for the second strategy are generally lower that those of the first one, but this is affected by the presence of $H - M$ non-malicious hosts. The simulation of such users is a requirement for modeling the second strategy, since otherwise we would have trivially that $\bar{s} = R_h$. Such hosts, instead, are not considered in the simulations of the first strategy.

It is also interesting to note that U is in general bigger than C even with the optimal replication factor $\rho = 2$. This observation suggests that choosing between the two strategies should be done after a critical comparison of both the performance overheads OH and the p_{CF} for the two presented schemes, according to the specific characteristics of the Grid and its workload.

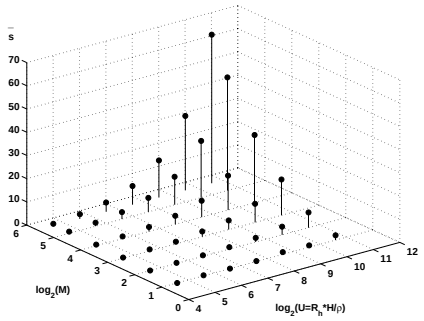

Fig. 3. Expected number of overlaps for the second strategy, with $H = 48$ and $\rho = 2$.

4 Conclusions and future works

In this paper we have introduced GIVS, a Grid Integrity Validation Scheme, based on the idea of using test problems in order to detect the presence of malicious users. We have defined two different strategies for generating and distributing these problems, studying performance overhead and success probabilities of each using statistical models and simulation. We have shown that the cost for ensuring integrity is far from negligible. Thus, future extensions of our work include optimization techniques such as the introduction of trust and reputation mechanisms.

References

1. Foster, I., Kesselman, C.: The grid: blueprint for a new computing infrastructure. Morgan Kaufmann Publishers Inc. (1999)
2. Oram, A., ed.: Peer-to-Peer: Harnessing the Power of Disruptive Technologies. O'Reilly & Associates, Inc. (2001)
3. Wahbe, R., Lucco, S., Anderson, T.E., Graham, S.L.: Efficient software-based fault isolation. In: Proc. of the 14th ACM Symp. on O.S. princ., ACM Press (1993) 203–216
4. Necula, G.C.: Proof-carrying code. In: Proc. of the 24th ACM SIGPLAN-SIGACT Symp. on Principles of programming languages, ACM Press (1997) 106–119
5. Damiani, E., di Vimercati, D.C., Paraboschi, S., Samarati, P., Violante, F.: A reputation-based approach for choosing reliable resources in peer-to-peer networks. In: Proc. of the 9th ACM CCS, ACM Press (2002) 207–216
6. Aberer, K., Despotovic, Z.: Managing trust in a peer-2-peer information system. In: Proc. of the 10th Int'l Conf. on Inf. and Knowledge Manag., ACM Press (2001) 310–317
7. Chaum, D.: Blind signatures for untraceable payments. In: Advances in Cryptology - Crypto '82, Springer-Verlag (1983) 199–203
8. Muttoni, L., Casale, G., Granata, F., Zanero, S.: Optimal number of nodes for computations in a grid environment. In: 12th EuroMicro Conf. PDP04. (2004)