A First Approach on Pareto's Honeypots

Stéfano D. K. Mór, Nicolas B. Maillard
Federal University of Rio Grande do Sul, Porto Alegre/RS, Brazil,
{sdkmor, nicolas}@inf.ufrgs.br

Abstract

On peer-to-peer and grid environments one potential problem is malicious attacks. One method to defend this kind of structure is the Monte-Carlo probabilistic certification, that indicates when massive attacks (i.e., attacks on a large number of processing nodes) occurred with a finite, desired error rate. This paper extends the ideas proposed by Krings (2005), Roch (2007) and Chayeh (2009) , which consider the probability of an attack occurring to be the same among all participating nodes and the attacker to have previously knowledge of the target task-distribution structure. The idea of a Paretto's Honycomb is introduced; we model the distribution of attack attempts using Pareto's probability distribution, specially the application of the Pareto Principle, where the user elects 20% of the vulnerable tasks to be the most important and, thus, to have 80% of the Monte-Carlo random guesses focused on it.

1. Introduction

Distributed computing systems like Grids and peer-to-peer networks, which are based on sharing computational resources, must have a permanent focus on data integrity. One potential modern threat is the use of large-scale distributed attacks, or *massive* attacks, which aim to falsify the result of a parallel task in a given computational node – or nodes – to produce an advantage in the real world (*e.g.* raising a bank account, gaining a powerful new weapon on an online game, *etc*.).

A paper by Krings et al. [5] considers a distributed system where each task may spawn other tasks during runtime, thus generating dependencies among "parent" and "sons" tasks. There is also a whitepaper by Roch et al. [6] extending these results to divide and conquer algorithms used for calculating a product between a vector and a matrix of numbers. Also, the work by Chayeh et al. [3] approaches this kind of issues, proposing a new algorithm that evades too many program re-executions.

All these papers propose probabilistic algorithms (associated with error rates) to detect forged task results when the number of attacked nodes is large. For this, all of them share the common point of supposing two main factors: (1) that the probability of any task being attacked is the same among all tasks, and (2) that the user has a complete knowledge of which nodes execute which tasks.

The point of this paper is that condition two induces the attacker(s) to *not* fulfil condition one, *i.e.*, to have an uniformly distributed probability of attacking any node; once the user knows where the most important tasks are running, it is more likely that those tasks are the one being forged rather than a random minor task. Our main proposition to cover this scenario is replacing the supposed uniform probability function with classical Pareto's probability function, which is shown to model many real world – in natural, social and economical fields – qualitative distributions.

The remain of this paper is disposed as follows: Section 2 introduces the fundamental concepts and notations. Section 3 introduce the concept of a Pareto's Honeypot and relate it to the evaluated scenario. At last, Section 4 traces some conclusions on the subject and lists the issues still open when this technique is adopted as a defence mechanism.

2. Concepts and notations

Let V be a set composed by two kinds (sets) of tasks, T and D, *i.e.*, $V = T \cup D$. Let T denote the tasks as seen in the traditional context of task scheduling – the smallest program unit of an instance of execution.

Tasks in T are 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, re-execute selected tasks. Communication between workers and verifiers is assumed to be fail-proof.

Let D denote the data tasks. 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 $t \in T$. Data tasks will be referred to as inputs or outputs of t . The size of T, in V, is n.

Before proceeding, we will first establish the notion of program execution and the impact of faults. Let X denote the execution of a workload represented by V with a set I of initial inputs on a set of unreliable resources (workers) – one could imagine it as set of 4-uples; inputs, task, designed worker, and output. It is assumed that V is static. Each task t in X executes with inputs $i(t, X)$ and creates output $o(t, X)$.

We will use $\hat{X}$ to denote a set just like X, but with all its tasks executed only over trusted workers (verifiers). If $X = \hat{X}$, *i.e.* if every task in X uses the same inputs and

computes the same outputs as those in $\hat{X}$, then X is said to be "correct". Conversely, if at least one task in X produced a wrong result and the execution results in $X \neq \hat{X}$, then it 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 X or $\hat{X}$, the following notation is adopted. Note that a "hat" always refers to a reliable resource, input or output. Let $\hat{i}(t, \hat{X})$ denote the input of t in X and $i(t, X)$ the input of t in X. Furthermore, let $o(t, X)$ denote the output of t on the client, $\hat{o}(t, \hat{X})$ the output of t on the verifier based on inputs from $\hat{X}$, and $\hat{o}(t, X)$ the output of t on the verifier based on inputs from X. Note that the notations $\hat{o}(t, X)$ and $\hat{o}(t, \hat{X})$ differ. Both indicate outputs generated on a verifier, but the first assumes $i(t, X)$ and the later $\hat{i}(t, \hat{X})$ as inputs.

2.1. Probabilistic Certification

Given an execution X of V, consider probabilistic certification based on a probabilistic algorithm that uses randomization in order to state if X has failed or not. A Monte Carlo certification is defined as a randomized algorithm that takes an arbitrary , $0 < \epsilon \leq 1$, as input and delivers (1) either CORRECT or (2) FAILED, together with a proof that X has failed.

The probabilistic certification is said to be with error ϵ if the probability of the answer CORRECT, when X 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 V on a verifier, comparing results to those obtained in X. If the results differ, X has failed. Otherwise, X may be correct or failed. However, if X has failed, a probabilistic certification with error ensures that the probability of non-detection of failure (based on randomly selecting tasks in V for re-execution) is less than or equal to ϵ.

In the sequel we denote the number of forged tasks at V 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 $n_q = \lceil q \times n \rceil \leq n_F$ tasks. X is said to be "attacked with ratio q" and $n_F \geq q$. It should be noted that q is assumed relatively large.

We will only consider the case where all tasks in V are independent. (For more information on dependent tasks see Roch et al. [6]) 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 X contains forged tasks.

Algorithm MCT

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

Since all tasks in T are independent we always have $i(t, X) = \hat{i}(t, \hat{X})$. If Algorithm MCT selects a forged task, then one knows with certainty that the execution X has failed. However, if MCT returns CORRECT, then one can only make conclusions based on the probabilities of randomly selecting a falsified or non-falsified task.

As demonstrated by Krings et al. [5], for any error rate ϵ, the minimum number N of random re-executions needed to achieve probabilistic certification ϵ is

$$N \geq \left\lceil \frac{\log \epsilon}{\log(1-q)} \right\rceil$$

3. Pareto's Honeypots

A *honeypot* is a classical trap for catching an oblivious attacker. Generally speaking, it is a special machine/data chunk that seems to be important and easily targeted, but is designed to capture the attacker or to spare the entire system from the massive attack. We use this word in the sense that we *expect* the attacker to focus on certain tasks. These tasks are statistically selected and are more likely to be chosen by a weighted version of Algorithm MCT.

As previously stated, Pareto's Honeypots are first determined statistically. Over a certain number of massive attack attempts (it does not matter whether they are successful or not), a statistical analyser tags normal tasks according to the number of times they were detected to be forged. So, the most forged tasks receive rank 0, the second most forged tasks receive rank 1, and so on.

After this statistical analysis, it makes little sense to uniformelly choose the tasks to be re-executed; its more effective for the attacker to target important tasks and for the defender to be pooling those more important tasks. Nevertheless, switching to a certain static task pooling distribution is too restrictive, since attack load is suitably different from system to system.

What we do, instead, is to build, first, a Pareto's Diagram. Here are the four main steps at doing so:

1. Consider the total number of tasks δ.
2. Assemble a frequency histogram H of type o task *vs.* how many times it was forged.
3. Taking the histogram from step (1), assemble a monotonic function ϕ that act as follows: $\phi(H, i)$ returns the sum of how many times the i-est most forged tasks were forged. *E.g.*, $\phi(H, 2)$ returns the sum of the number of times that task 0 (the most forged one), task 1 (the second most forged), and task 2 (the third most forged task) were forged.
4. Taking the histogram from step (1), assemble a function θ that act as follows: $\theta(H, i)$ returns a histogram

$H' \subseteq H$ containing only the i-est most forged tasks and its reports.

The name "Pareto's Diagram" refers to the fact that H and ϕ are usually plotted at the same graphic.

The interesting property (for us) of a Pareto's Diagram is the fact that it (non-strictly) follows Paretto's Principle:

"*20% of the values are reported 80% of time.*"

Pareto's Principle is even more relevant than this because it is *recursive*; *i.e.*, 20% of the initial 20% of the values are reported 80% of the original 80% of time, *ad infinitum*. Pareto's Principle is non-strict because the percentage do not need to be 20/80 – resulting at a nice sum of 100. In fact, both values are *not bounded* by its sum; they measure different things and are not required to sum up to 100 or any other value.

In our case, the expected values – following the proposed notation and Pareto's Principle – are:

$$\phi(\theta(H, 0)) \approx \frac{\delta}{5}$$

and when generalizing to a recursive case it would be

$$\phi(\theta^k(H, 0)) \approx \frac{\delta}{5^k}$$

where k is the recursion level and θ^k is the k-est recursive call of function θ over initial input $(H, 0)$.

An interesting aspect of the fact above is that the use of both numbers 0 and 5 is *not required* to effectively model the defence scenario. Both could be replaced, respectively, by i – meaning the defence grain size (*i.e.*, the $i - th$ level of *pooling concentration*) – and by C, a constant that better fits the attack distribution among the number of tasks. As we will see after, C could be easily modelled using the concept of a *power law*.

Our choice of Vilfredo Pareto's principle was not random, as seen next. The principle is an instance of a probability distribution of a continuous random variable named Pareto's Distribution. It appears for the first time at his seminar 1906's work, *Manuale di Economia Politica*. Pareto's distribution is a probabilistic power law given by

$$\Pr\{A \geq a\} = \begin{cases} \left(\frac{a_m}{a}\right)^{\alpha} & \text{for } a \geq a_m, \\ 1 & \text{for } a < a_m \end{cases}$$

where a_m is the (positive) minimum possible value of X, and α is a positive parameter, the *Paretto Index*. Both are the distribution's parameters. Pareto's principle is the instance where $\alpha = \log_4 5$.

(When comparing to our previous notation, it is easy to notice that the parameter k is, in fact, α, and that C corresponds to a.)

Pareto's Distribution is widely used at many research fields, including computer science, where it was found to model recurring patterns, specially at networks [7] [2]. Nevertheless, our choice was not made by its large spectrum of possible applications. Pareto's distribution belongs to a family of probability distribution functions called *power laws*, which some recent papers [8] [1] [4] show to be the recurrent distribution of attacks on both peer-to-peer and grids implemented over efficient networks (which are also modelled by power laws). So, the considerations on this paper could be trivially extended to other power laws through a simple parameter change, varying it according to best fit for a real-world case.

The main advantage we focus is that a simple histogram would not allow the defender to vary the *granularity* of the pooling distribution – noted by k – on performing a weighted version of the MCT algorithm. If a first statiscal histogram does not fit attacks distribution, another one would have to be made over a (potentially) long period of time. Through Pareto's Distribution, however, we can always adjust our pooling by increasing or decreasing the value of k. It also fits better real-world scenarios than the pure random MCT algorithm, although it is not asymptotically better.

4. Conclusion

This is a preliminary paper, showing the main ideas that could lead to a full academic work. Although it provides some useful notations and ideas, it is incomplete in the following ways:

- It does not provide *accurate* mathematical modelling of a Paretto Honeypot, int the sense that no parameter range is given and proven to fit this peer-to-peer/grid scenario.
- It lacks benchmarks and comparsions to real-world attack scenarios. Specially, after parameters are estimated, it would still lack of stochastical fitness tests – *e.g.*, χ^2 (Qui-square).
- A modified MCT algorithm taking into account a Pareto's Diagram yet remains to be formally introduced and proved to be correct.

The last fault is the main question that remains open for future works. Specially, it should be verified whether a simple weighted pooling is sufficient to implement a good Pareto-weighted MCT algorithm, *i.e.*, if testing the 20% more attacked tasks with 80% of the random guesses – and *vice-versa* – is sufficient to conform with the desired error-catching rate. This is not simple, because, as previously stated, the values 20/80 are not fixed and, thus, MCT must be able to efficiently handle *any* granularity of attacks specified by the defender.

Beware, however, that this paper introduces a relevant discussion and early notation about modelling the distribution of massive attack attempts against large parallel/concurrent systems. A malicious user's knowledge of the priority of the tasks is used *pro bonno* and a future Pareto-weighted MCT algorithm is likely to perform better at a real-world scenario. Also, one could try to (dis)prove that it still retains its asymptotical efficiency when random distributed attacks are considered.

References

[1] M. Altunay, S. Leyffer, J. T. Linderoth, and Z. Xie. Optimal response to attacks on the open science grid. *Comput. Netw.*, 55:61–73, January 2011.

[2] P. Barford and M. Crovella. Generating representative web workloads for network and server performance evaluation. *SIGMETRICS Perform. Eval. Rev.*, 26:151–160, June 1998.

[3] R. Chayeh, C. Cerin, and M. Jemni. A probabilistic fault-tolerant recovery mechanism for task and result certification of large-scale distributed applications. In *Proceedings of the 4th International Conference on Advances in Grid and Pervasive Computing*, GPC '09, pages 471–482, Berlin, Heidelberg, 2009. Springer-Verlag.

[4] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. *SIGCOMM Comput. Commun. Rev.*, 29:251–262, August 1999.

[5] A. W. Krings, J.-L. Roch, S. Jafar, and S. Varrette. A probabilistic approach for task and result certification of large-scale distributed applications in hostile environments. In *EGC*, pages 323–333, 2005.

[6] J.-L. Roch and S. Varrette. Probabilistic certification of divide & conquer algorithms on global computing platforms: application to fault-tolerant exact matrix-vector product. In *PASCO*, pages 88–92, 2007.

[7] B. Schroeder, S. Damouras, and P. Gill. Understanding latent sector errors and how to protect against them. *Trans. Storage*, 6:9:1–9:23, September 2010.

[8] W. Yu, S. Chellappan, X. Wang, and D. Xuan. Peer-to-peer system-based active worm attacks: Modeling, analysis and defense. *Comput. Commun.*, 31:4005–4017, November 2008.