Collusion Detection for Distributed Computing

Eugen Staab
Faculty of Sciences,
Technology and Communication
University of Luxembourg
L-1359 Luxembourg
Email: eugen.staab@uni.lu

Thomas Engel
Faculty of Sciences,
Technology and Communication
University of Luxembourg
L-1359 Luxembourg
Email: thomas.engel@uni.lu

***Abstract*—A common technique for result verification in distributed computing is to delegate a computation redundantly and apply majority voting to the results. However, the technique is sensitive to "collusion" where a majority of malicious hosts collectively returns the same incorrect result. In this paper, we propose a mechanism that identifies groups of colluding hosts. The mechanism is based on the fact that colluders can succeed in a vote only when they hold the majority. This information allows us to build clusters of hosts that voted similarly in the past, and so detect collusion.**

***Index Terms*—Distributed Computing, Sabotage Tolerance, Result Verification, Collusion Detection.**

I. Introduction

Problem Context and Motivation: In *distributed computing*, hosts assign computations to other hosts, which are expected to perform the computations and return the results. In certain scenarios such as *desktop grids* [1], an assigning host has no control over the performing hosts, and so it cannot be sure whether a returned result is actually correct or not. Hosts may return incorrect results because they are faulty, because they want to harm the delegating host, or because they want to save resources (e.g. by returning random results). Conventional security mechanisms can assure *data authenticity* and *integrity*. However, these mechanisms cannot ensure the *correctness* of received results. For specific computations, *checksums* can be used to verify results in an efficient way. Unfortunately, no such mechanism is known that works for general computations. Hence, mechanisms are needed to ensure the correctness of the results without explicit verification.

A common principle used to this end is *redundancy*: a computation is redundantly outsourced to several randomly selected hosts; majority voting is applied to the set of returned results to decide in favor of the result that appears most often. This approach is robust against single hosts that return incorrect results. However, it does not address the case where a majority of colluding hosts collectively returns the same incorrect result. For this, mechanisms are required that are able to detect colluding behavior of malicious hosts.

Approach and Contribution: We present an algorithm for collusion detection that exploits the information of how often pairs of hosts are together in the majority/minority in votes, and how often they are not. When colluding hosts win a vote, they are always together in the majority and honest hosts form the minority. Hence, pairs of two colluders or two honest hosts correlate more in their votings than pairs consisting of one malicious and one honest host. In [2], we showed theoretically that this fact allows a line to be drawn between the groups of honest and colluding hosts. In this work, we propose an algorithm that uses graph clustering to discover the division between the groups.

II. Algorithm

The algorithm takes as input a population of hosts N, a set $\mathcal{V}_N$ that contains for each pair of hosts $p_i \in N^2$ two numbers, namely in how many past votes they were together in the same group, a majority or a minority (α_i), and how often they were in opposite groups (β_i), and a parameter P needed for the clustering algorithm. The algorithm proceeds as follows. First,

Algorithm 1 Collusion Detection

1: **procedure** DETECT($N, \mathcal{V}_N, P$)
2: $\mathcal{C} \leftarrow$ ESTIMATE_CORRELATIONS($N, \mathcal{V}_N$)
3: $G \leftarrow$ CONSTRUCT_GRAPH($\mathcal{C}$)
4: $\{C_1, \ldots, C_k\} \leftarrow$ CLUSTER(G, P)
5: $C_{max} \leftarrow \max(C_1, \ldots, C_k)$ ▷ Select largest cluster
6: $S \leftarrow N \setminus C_{max}$ ▷ Take all but largest cluster
7: **return** S ▷ Return IDs of suspects
8: **end procedure**

it estimates for each pair of hosts p_i how much the two hosts correlate as $\frac{\alpha_i+1}{\alpha_i+\beta_i+2}$, which is stored in $\mathcal{C}$ (line 2). In line 3, an undirected weighted graph is constructed, with the hosts being the nodes and the correlation between hosts the edge weights. The graph clustering algorithm partitions the graph G into clusters of hosts that correlate strongly (line 4). Assuming less than 50% colluders in the population of hosts, the largest cluster is supposed to contain the strongly correlating honest hosts, so it is selected (line 5) and subtracted from N (line 6). The remaining part of N is returned containing the suspect hosts (line 7).

References

[1] D. Kondo, M. Taufer, C. L. Brooks, H. Casanova, and A. A. Chien, "Characterizing and evaluating desktop grids: An empirical study," in *Proc. of the Int. Parallel and Distributed Processing Symposium (IPDPS'04)*. IEEE Computer Society, 2004, pp. 26–35.

[2] E. Staab, V. Fusenig, and T. Engel, "Using correlation for collusion detection in grid settings," University of Luxembourg, Tech. Rep. 000657499, July 2008.