

Online Verification via External Algorithmic Observer

Franck Cappello, Argonne National Laboratory: cappello@anl.gov

Once a successful intrusion has happened at a supercomputing center, an insider threat can alter the integrity of numerical simulations and data analytics executions in at least two ways: (1) by corrupting the product (data) of the execution (Figure 1) and (2) by affecting the execution in such a way that the execution does not complete (making it crash or hang), provoking the equivalent of a denial of service. This paper focuses on attacks producing corruptions that stay unnoticed by classic security and resilience techniques.¹ We consider that attacks may lead to two main classes of corruptions: nonsystematic and systematic. An attack that targets executions individually may generate nonsystematic corruptions, affecting an execution in a unique way; that is, the probability of repetition of the exact same corruption in another execution is very low. Attacks that consistently affect executions (same code, same input parameters) the same way lead to systematic corruptions. Executions do not need to be identical to produce the same corruptions. Different executions may execute the same part of the code or the same instruction that will cause the same corruption.

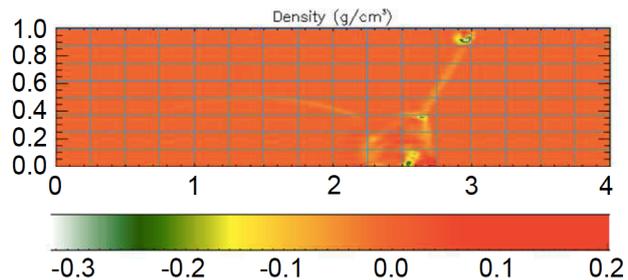


Figure 1: Propagation at iteration 200 of a corruption in density field injected at coordinate [2, 0.0] at iteration 50 in an unsteady planar shock simulation.

Limits of Existing Techniques

Nonsystematic corruptions are easily detected by classic techniques such as replication. Ensemble computations also cover nonsystematic corruptions, since statistical analysis of ensemble results may detect or absorb the corruptions. However, replication and ensemble computations both suffer similar limitations: they can be expensive, and thus not all executions can afford them. Systematic corruptions are even more difficult to detect since comparing multiple identical executions will not help. Algorithm-based fault tolerance (ABFT) techniques provide a useful approach to detect both types of corruptions. However, ABFT applies only to a limited set of numerical kernels, and it covers only data protected by the ABFT scheme. It also requires code changes and may not detect corruptions affecting the ABFT calculation itself.

N-version programming [1], proposed almost three decades ago, can detect systematic corruptions. This approach has some similarity with alternates in recovery blocks [2]: results of the execution of multiple different versions responding to the same specification are compared in order to detect potential corruptions. The higher the diversity of the versions (from hardware to application), the higher is the chance of detecting corruptions. This approach does not seem systematically applicable in our domain, however, because of the cost of developing multiple versions of all levels of the stacks, from the hardware to the application.

Proposed Solution: External Algorithmic Observer

Since the problem spans all layers of the stack, from the hardware to the application, we believe that a holistic approach, covering all potential sources of corruptions, has a better chance of succeeding. We propose an **online verification approach** using an external algorithmic observer that will detect nonsystematic as well as systematic corruptions of application data. During the execution, the transformations applied by the hardware and software stack to the state data are verified against trusted models run by the observer. This direction is close to n-

¹ All corruptions leading to the execution hanging or crashing or to results obviously wrong (easily detected by the end user) are beyond the scope of this white paper.

version programming but uses verification algorithms much simpler than the execution stack. The external algorithmic observer approach is similar to the simplex architecture technique for critical systems [3]. The main idea is that the external algorithmic observer (Figure 2) checks that the observed execution respects constraints set by the application developer. Specifically, the external algorithmic observer executes a model of the data transformation performed by the application. Few research results of this approach have been published in the HPC and scientific data analytics domain. The model could implement a simpler version of the application [4], or it could be derived from observed properties of the data transformation as in [5], learned by using some machine learning algorithms. The critical point is that the application and the external model should be diverse enough that they would not be affected by systematic corruptions the same way. In principle this approach allows a large spectrum of model complexities (compute and memory complexities) that could go up to the complexity of the application and its software stack.

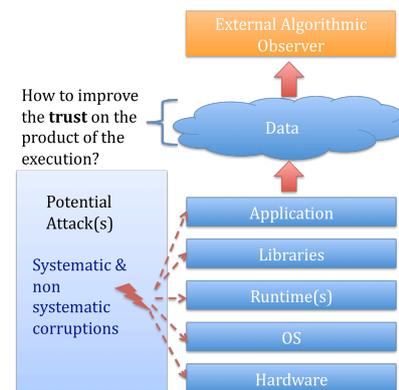


Figure 2: Principle of the external algorithmic observer.

Low-complexity models implement tradeoffs between complexity, accuracy, and other properties. For example, the model used in [4] relaxes numerical stability. In [5], the model computes only local predictions for the immediate next simulation step, leveraging the spatiotemporal continuity present in many applications simulating physics phenomena. This model does not compute solutions of the equations governing the simulation; rather, it verifies that the simulation respects a particular physics property between steps. By being much simpler than the simulation stack, the software implementing the model is also easier to verify and to protect. While multi-version programming is not applicable to the simulation stack, it is applicable to the software implementing the model. Several implementations of the same model or several different models could be executed and compared with the application. Because the software implementing the model has a low compute complexity, it could be executed on a more secure environment, such as a protected virtual machine, a secure OS domain, or a secure processor.

Early Results

The external algorithmic observer approach has been successfully applied to several key DOE applications for the detection of corruptions [5, 6], where it can detect up to 80%-99% of harmful corruptions (Figure 3), with only few false alarms (5% of the iterations of a dynamic simulation involving multiple time steps) and minimal memory overhead (1% < O < 20%) and computation overhead (<6%). The approach presented in [5, 6] uses simple models that perform next-step value prediction for the monitored state data. It monitors the state variables of the simulation and models their trajectories in time. The trajectory predicted by the model for each variable is compared with the one produced by the simulation. If the two trajectories differ by more than a tolerance range, an anomaly is raised. Advanced techniques such as error feedback control, spatial sampling, and impact-driven tolerance setting are needed to improve the detection performance and reduce the overheads.

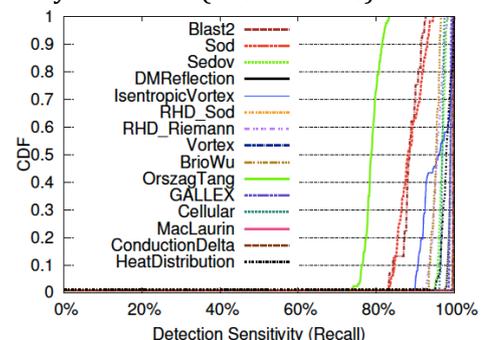


Figure 3: Detection sensitivity of the external algorithmic observer.

The external algorithmic observer is a promising approach to improve supercomputing trustworthiness. The community is just at the beginning of its exploration. More research is needed to improve its performance, to cover a larger diversity of applications, and to integrate this approach in a secure environment and combine it with other cyber-security techniques.

- [1] A. Avizienis. The N-version approach to fault-tolerant software. *IEEE Trans. Software Eng.* 11,12, pp. 1491-1501, 1985.
- [2] B. Randell, J. Xu. The evolution of the recovery block concept. In *Software Fault Tolerance*, pp. 1-22, John Wiley & Sons, 1994.
- [3] Sha, Lui, Using simplicity to control complexity. *IEEE Software* 18,4, pp. 20-28, 2001.
- [4] A.R. Benson, S. Schmit, R. Schreiber. Silent error detection in numerical time-stepping schemes. To appear in *International Journal of High Performance Computing Applications*, 2014.
- [5] E. Berrocal, L. Bautista-Gomez, S. Di, Z. Lan, F. Cappello, Lightweight silent data corruption detection based on runtime data analysis for HPC applications, short paper, ACM HPDC 2015
- [6] S. Di, E. Berrocal, F. Cappello, An efficient silent data corruption detection method with error-feedback control and even sampling for HPC applications, IEEE CCGRID 2015