Understanding and Improving the Trust in Results of Numerical Simulations and Scientific Data Analytics

Franck Cappello(✉), Rinku Gupta, Sheng Di, Emil Constantinescu, Thomas Peterka, and Stefan M. Wild

Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439, USA
{cappello,rgupta,sdi1,emconsta,tpeterka,wild}@mcs.anl.gov

Abstract. With ever-increasing execution scale of parallel scientific simulations, potential unnoticed corruptions to scientific data during simulation make users more suspicious about the correctness of floating-point calculations than ever before. In this paper, we analyze the issue of the trust in results of numerical simulations and scientific data analytics. We first classify the corruptions into two categories, nonsystematic corruption and systematic corruption, and also discuss their origins. Then, we provide a formal definition of the trust in simulation and analytical results across multiple areas. We also discuss what kind of result accuracy would be expected from user's perspective and how to build trust by existing techniques. We finally identify the current gap and discuss two potential research directions based on existing techniques. We believe that this paper will be interesting to the researchers who are working on the detection of potential unnoticed corruptions of scientific simulation and data analytics, in that not only does it provide a clear definition and classification of corruption as well as an in-depth survey on corruption sources, but we also discuss potential research directions/topics based on existing detection techniques.

Keywords: Trust · Numerical simulation · Data analytics

1 Introduction

Today and future scientific simulations or data analytics are facing a huge risk with potential unnoticed corruptions because of ever-increasing execution scale and more and more complicated system architecture. Multiple examples of hardware bugs in floating-point units and software bugs in application stack make users more suspicious about the correctness of floating-point calculations. In 2004 the AMD Opteron had an instruction bug that could result in succeeding instructions being skipped or an incorrect address size or data size being used [4]. Other bugs were reported in the Opteron processor in 2012 and 2014 [3].

In this paper, we investigate several key aspects of the trust that a user can give to the results of numerical simulations and scientific data analytics. The notion of trust is related to the integrity of numerical simulations and data analytics applications and not on whether the application actually completes.

To simplify the presentation without loss of generality, we consider that trust in results can be lost (or the results' integrity impaired) because of any form of corruption happening during the execution of the numerical simulation or the data analytics application. In general, the sources of such corruption are threefold: errors, bugs, and attacks. Current applications are already using techniques to deal with different types of corruption, but these techniques are not all-encompassing. The current level of trust that a user has in the results is at least partially founded on ignorance of this issue or the hope that no undetected corruptions will occur during the execution.

So far, there have been a lot of research studying the trust/reliability issue, such as detection of silent data corruptions (SDC) in numerical simulations (to be detailed in Sect. 6). However, there are no specific surveys to categorize, formalize the research from the perspective of technical background and summarize the corresponding solutions comprehensively. The work in this paper aims to fill this gap.

In this paper, we look at (1) exploring the sources of trust loss; (2) reviewing the definitions of trust in several areas; (3) providing numerous cases of result alteration, some of them leading to catastrophic failures; (4) examining the current notion of trust in numerical simulation and scientific data analytics; (5) providing a gap analysis; and (6) suggesting two important research directions and their respective research topics. We also, specifically, suggest recommendations for developing a more scientifically grounded notion of trust in aforementioned applications. We first formulate the problem and show that it goes beyond previous questions regarding the quality of results such as Verification and Validation (V&V), uncertainty quantification, and data assimilation. We then explore the complexity of this difficult problem, and we sketch complementary general approaches to address it.

The product of simulation or of data analytic executions is the final element of a potentially long chain of transformations, where each stage has the potential to introduce harmful corruptions. These corruptions may produce simulation results that deviate from the user-expected accuracy without notifying the user of this deviation. There are many potential sources of corruption before and during the execution; consequently, in this paper we do not focus on the protection of the end result after the execution (latter is covered in the paper [10], through the notion of provenance and trustable communications and storage).

2 Corruption Classification and Origins

In this section, we focus on corruptions that stay unnoticed. The corruptions for which significant research efforts are needed in the context of trust are those that corrupt the results in a harmful way but are not detected by hardware, software, or the users. We consider two main classes of corruptions: nonsystematic and systematic.

Nonsystematic corruptions are those affecting an execution in a unique way; that is, the probability of repetition of the exact same corruption in another execution is very low. A harmful corruption is manifested as an alteration of one of more data elements. Origins of such corruptions may be radiations (cosmic ray, alpha particles from package decay), bugs in some paths of nondeterministic executions, attacks targeting executions individually, and other potential sources.

Systematic corruptions (including conception/model errors and epistemic uncertainties) affect multiple executions of the same code, with the same input parameters, in the same way. The harmful corruption also is manifested as an alteration of one of more data elements. Executions do not need to be identical to produce the same corruptions because of possible uncertainty of the input/execution data or so. Origins of these corruptions are twofold: (1) bugs or defects (hardware or software) that are exercised the same way by executions (different executions will execute a same code region or the same instruction that will cause the same corruption) and (2) attacks that will consistently affect executions the same way.

A question that usually arises is that is **the trust in numerical results a real problem?** We argue that trust is a serious and insufficiently recognized problem. For a list of software bugs that impacted users in domains such as space exploration and telecommunications, see [1].

Two serious issues could be raised because of such unnoticed corruptions.

1. A large number of executions may have been corrupted before the discovery; bad decisions may have been taken [2]; and it might be difficult after-the-fact to check whether executions have actually been corrupted or not, without heavy checking (e.g., re-executing the simulations entirely)
2. Even if silent corruptions do not lead to accidents, they may lead to significant productivity losses.

3 Definition of Trust in Multiple Areas (Computer Science, Sociology, Economy)

All types of corruptions mentioned in this paper are considered as part of the general dependability problem as formulated in the paper [12]: "the ability to deliver service that can justifiably be trusted". Table 1 shows the relation between dependability, survivability, and trustworthiness, as mentioned in the paper [12]: the three concepts essentially cover equivalent goals and threats.

A survey of definitions related to dependability and trustworthiness is presented in the paper [13]. In that survey, trust depends on many elements: safety, correctness, reliability, availability, confidentiality/privacy, performance, certification, and security.

Multiple definitions of trust [6] are relative to other contexts: social sciences, psychology, philosophy, and economics. The definitions that may help address

Table 1. Relation between dependability, survivability, and trustworthiness

Concept	Dependability	Survivability	Trustworthiness
Goal	(1) ability to deliver service that can justifiably be trusted; (2) ability of a system to avoid failures that are more frequent or more severe than is acceptable to the user(s)	capability of a system to fulfill its mission in a timely manner	assurance that a system will perform as expected
Threats present	(1) development faults (e.g., software flaws, hardware errata, malicious logic); (2) physical faults (e.g., production defects, physical deterioration); (3) interaction faults (e.g., physical interference, input mistakes, attacks, including viruses, worms, and intrusions)	(1) attacks (e.g., intrusions, probes, denials of service); (2) failures (internally generated events due to, e.g., software design errors, hardware degradation, human errors, corrupted data); (3) accidents (externally generated events such as natural disasters)	(1) hostile attacks (from hackers and insiders); (2) environmental disruptions (accidental disruptions, either manmade or natural); (3) human and operator errors (e.g., software flaws, mistakes by human operators)
Reference	this paper	"Survivable network systems" (Ellison el al. 1999)	"Trust in cyberspace" (Schneider 1999)

the trust problem in our context are the following: "One party (trustor) is willing to rely on the actions of another party (trustee)" and "The trustor is uncertain about the outcome of the other's actions; they can only develop and evaluate expectations".

Although there exist several metrics for trust [7] and approaches to building trust, there is no consensus on or norm for which metrics should be used in which case. In numerical simulation and scientific data analytics, there is a lack of trust metrics that could be used to quantitatively compute and express the trustworthiness of the execution results.

4 Building Trust in Application Results

The trust in the results of numerical simulation and data analytics execution is related to two main notions: correctness of computation and integrity of the execution stack. However, neither of them could be proven formally for nontrivial execution scenarios. To address this issue, users have to develop a process to build trust in their execution results.

The simple pattern of building trust generally involves the following process. The users first start with the smallest-scale, simplest problem that can be reasonably modeled, and compare the output with expectations. The simulation is then repeatedly scaled up in complexity and size (both simulation size and system size), while repeating the comparisons of output with expectations. Any odd or unexpected behavior is scrutinized and assumed to be an error until demonstrated otherwise.

4.1 Expected Result Accuracy

Expected result accuracy is application dependent. Some applications are sensitive to the details of calculation; for example, they can even act as tests of the randomness of the pseudo-random number generator used. Other applications model systems following a trajectory to an attractor state and small perturbations to that trajectory have no impact on the final outcome. During the execution, accuracy is affected by round-off errors; such errors accumulate, and the expected accuracy at the end of the execution is much lower than the machine precision. Typical expected accuracies at the end of the execution are 10^{-6} for the HACC cosmology code executions and 10^{-8} for Nek5000 computational fluid dynamics executions.

At its most fundamental, expected result accuracy can be defined as follows: If the corruption of the data does not result in any measurable changes to any physically meaningful statistics of the simulation between a run that contained the corruption and a run that does not, then the user's expectation of accuracy has been satisfied. This definition suggests that research should focus on detecting corruptions that make the end results diverge from the expected user accuracy, as did by the adaptive impact-driven SDC detector [18].

4.2 How Existing Techniques Help Building Trust

Verification and validation form the basis for building trust in codes and the models underlying them. We follow the convention of [16], whereby validation determines the faithfulness of the mathematical/computational models to the real world and verification determines the faithfulness of the code to the mathematical/numerical models. While solution verification techniques quantify the accuracy at which algorithms solve the model, code verification techniques certify that a code is a truthful implementation of the algorithms themselves. Following best practices (e.g., unit and regression testing) and standards for software design is a common, although incomplete, attempt toward verification.

Another common software development technique for building trust is to incorporate physical, mathematical, and numerical knowledge alongside a computation in order to flag potential errors. Examples in the course of a computation can include ensuring that mass or other quantities are conserved, that two linear basis vectors remain orthogonal, and that an accumulated remainder term lies below a round-off bound.

Uncertainty quantification is an umbrella term for several activities involved in improving the trust in the simulations and data in the hope of accounting for all sources of uncertainty involved in the simulation of real-world/physical quantities. Several techniques are used to improve the trust in the numerical model, data, and simulation productivity under random effects. For example, gridded or complete data sets are constructed from sparse data by solving inverse problems. Simulations are corrected (or guided) by using data through a process referred to as data assimilation. Complex mathematical models and models that are used to represent real processes that are not well understood typically use parameterizations. Parameterizations are surrogate models that depend on a set of parameters that do not necessarily have a physical meaning. These parameters are usually calibrated by solving a parameter estimation problem. Although UQ techniques are often segregated along domain science and scientific community lines, they support a common mathematical formulation and are often used in tandem or in a manner that is not always transparent.

5 Gap Analysis

Many techniques are already applied from the hardware to the application in order to detect corruptions. These techniques do not cover all potential sources of corruptions, however, and large gaps put execution results at risk.

Harmful nonsystematic corruptions (undetected corruptions that corrupt execution results in a non-noticeable way) can be detected by classic approaches such as replication or algorithm-based fault tolerance (ABFT). Replication is too expensive in our domain to be applied on all executions, however, and ABFT covers only the data protected by the ABFT scheme: other application data are not protected. Ensemble computations also offer a way to deal with nonsystematic corruptions, since statistical analysis of the ensemble results may detect or absorb the corruptions.

Harmful systematic corruptions are not detected by replication because replication detects errors by comparing identical (or comparable) executions. Since the systematic corruptions will affect replicated executions the same way, the comparison of executions will not detect any corruption. Ensemble computations will suffer the same limitation and will not be able to detect or absorb such corruptions.

One approach to detect systematic corruptions, called n-version programming [11], was proposed three decades ago. In this approach, which has some similarity with the notion of alternates in recovery blocks [21], the results of the executions 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 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. Moreover, it has been demonstrated experimentally that different versions may suffer the same bugs (and lead to the same corruptions) [19].

Formal validation and verification often presuppose the availability of a correct reference solution that can be used to assess model accuracy and code correctness. Although codes can be designed to capture these subsystems as special cases, the potential for increased trust is rarely deemed to outweigh the resulting efficiency loss; and this gap widens at scale. As highlighted in the paper [16], problem classes for which formal V&V methods exist (e.g., quantifying the numerical error in the solution of linear elliptic PDEs) seldom overlap with the complex simulations performed for DOE.

Uncertainty quantification considers that the hardware and the software stack produce correct results. Uncertainty quantification is almost entirely focused on addressing randomness introduced through the mathematical model. In general, all algorithms assume that the hardware/software stack produces asymptotically correct, if not exact, results. In the presence of numerical errors or spurious software, outcomes can lead to biases in UQ that render the analysis useless or can have a significant detrimental effect on trust.

6 Analysis of Research Directions and Solution

Since the trust problem spans all layers of the stack, from the hardware to the application, and is related to many aspects of numerical simulation and data analytics (modeling, initial conditions, numerical accuracy, parametric settings, etc.), we believe that holistic approaches, considering all potential sources of corruptions, have a better chance of succeeding. Figure 1 presents complementary research directions.

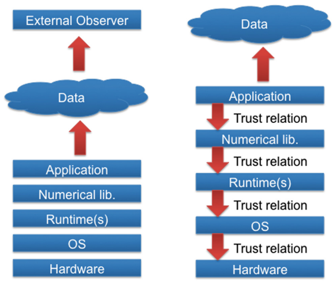

Fig. 1. Complementary research directions to address the trust problem

The first direction performs on-line verification by using an external algorithmic observer that does not trust the execution stack. During the execution, the transformations applied by the hardware and software stack to the data are verified against trusted models run by the observer. This direction is close to n-version programming but uses verification algorithms much simpler than the execution stack (the external algorithmic observer method assumes that the observer is simpler to code than the full execution stack, hence can be more easily

Table 2. Advantages and drawbacks of two research directions

	External observer	Trust relations
Detection approach	Simulation and observer are checking each other	Checking object results
Detection assumptions	External observer is correct (should be verified, validated)	All verifications and reputation calc. are correct
Detection latency	Short (depends on sampling rate, typically 1 appl. iteration)	Long (actual detection could be long: months)
Timeliness of notification after detection	Short (one iteration to next)	Short (immediate upper layer)
Time to build trust	Low (trust depends on verisimilitude of results not on components)	High (h/w and s/w components need to acquire trust level)
Targeted level of trust	User-expected accuracy	Machine precision (modulo round-off errors)
Dev. time and cost	Low (requires only to develop the observer)	High (affects all layers of the stack)
Tolerance	High (corruptions of the appl. data lower than user-expected accuracy are tolerated)	Low (any corruption at object level is suspicious since the consequence on appl. data is unknown)

verified). The second direction establishes trust relations between levels of the execution stack. Establishing these trust relations may involve thorough verification of each level, reputation mechanisms, and layer-level on-line verification. Table 2 shows the advantages and drawbacks of the two directions.

6.1 External Algorithmic Observer

The external observer approach is similar to the simplex architecture technique for critical systems [22]. It is also similar in principle to a direction developed for cyber security at the UIUC/Information Trust Institute where the predictable/expected behavior of a system is defined and used for detecting anomalies [9]. The main idea is that the external observer checks that the observed execution respects constraints set by the developer of the application/user.

In our context, the external algorithmic observer executes a model of the data transformation performed by the application. There are several related techniques proposed recently. Di et al. [17] proposed a silent data corruption detection method with error-feedback control and even-sampling for HPC applications. It is designed particularly for iterative scientific simulations with multiple time steps/snapshots generated. The detection performs the data prediction mainly along the time series dimension for each data point in each snapshot. Based on that work, they further proposed an improved solution [18] by taking into account only significant corruptions regarding their impact on the final execution results. They also proposed an adaptive solution allowing each process/rank to select the best-fit prediction method based on its dynamic local dataset, significantly improving the detection ability and lowering the memory cost meanwhile. In absolute terms, Experiments with about 20 benchmarks indicate that it can detect 80–99.99% of influential SDCs with the false positive rate reduced to 0–1% in most cases. In addition, Berrocal et al. [15] explored partial replication to improve lightweight silent data corruption detection for HPC Applications.

Alternatively, the detection model could be derived from observed properties of the data transformation [17], learned using some machine leaning algorithms, or could implement a simpler version of the model used in the application [14, 24]. The critical point is that the application and the external model should be diverse enough that they would not be affected by systematic corruptions in the same way. In principle this approach allows a very large spectrum of model complexities (compute and memory complexities) that could go up to the complexity of the application plus the stack running the application. Since we cannot afford such complexity in our domain, however, the research should focus on models of a much lower complexity.

Low-complexity models implement trade-offs between complexity, accuracy, and other properties. For example, the model proposed by Benson et al. [14] relaxes numerical stability assuming that (1) the model can be restarted at each step from the verified results of application at the previous step and (2) corruptions happening in one step are detected in the same step. In Di et al.'s work [17], the model computes only local predictions for the next simulation step, from the application results at the current step (one step prediction), 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.

Because the model is purposely simpler than the simulation, the data produced by the model diverges slightly from the one of the application. Therefore, the detection cannot be based on perfect comparison. A tolerance margin should be considered that controls the detection accuracy that conditions the number of false positives (detection of a corruptions that did not happen) and false negatives (nondetection of corruptions that actually happened). Other metrics include overhead in execution time and overhead in memory occupation (the model needs memory space for its execution). The tolerance margin should avoid false detection due to the natural divergence between the model and the application. It also should be lower than the user-expected accuracy in order to ensure that corruptions exceeding the user-expected accuracy will be detected.

An important advantage of this approach is that by being much simpler than the simulation stack, the software implementing the model is also easier to verify and to protect. For example, the multiversion programming approach is not applicable to the simulation stack but it is applicable to the software implementing the model. Several implementations of the same model or several different models could be executed and compared. Because the software implementing the model has a low compute complexity, in principle, it could be executed on a more secure environment, such as a secure processor. This allows increasing the trust in the model itself.

6.2 Trust Relations

The direction based on trust relations is more mature in the sense that a large body of research has been devoted to this topic in computer science. The DOE report on Cybersecurity for Scientific Computing Integrity [10] provides considerable coverage of the issues and approaches related to this direction. This section complements the report by providing additional analysis and references.

To simplify the presentation, we call an “object” any piece (or layer) of software of hardware that needs to be trusted. The trust relation direction supposes at least (1) a way to certify that each used object is actually the object it is supposed to be, (2) a method to evaluate a level of trust for each object involved in the execution, (3) a metric of the level of trust, and (4) a way to protect the trust level acquired by an object.

Considering points (1) and (4), the Trust Computing Group [5] has produced the Trusted Platform Module (TPM) specification [8], which is an ISO/IEC international standard. This specification details embedded crypto capability that supports user, application, and machine authentication. More than 500 million PCs have shipped with TPM. One application of TPM is the verification of the integrity of the platform to ensure no unauthorized changes have occurred in the BIOS, disk master boot record, boot sector, operating system, and application software. We believe that points (1) and (4) can leverage this well-established technology to reduce the risk of attack-induced corruptions. However, TPM does not protect against sophisticated attacks, and some TPM circuits showed vulnerability [23,26].

Regarding point (2), the evaluation of the trust level of an object could rely on extensive verification and validation of that object by a combination of formal verification when applicable and empirical methods (checking against known results, checking results against actual measurements). In principle the external observer approach can be applied for each object. However, modeling the data transformation of some functions in order to perform effective and efficient detection may require a model complexity close to that of the function.

Regarding point (3), the trust metrics could have multiple dimensions (such as time since first trusted, time since last verification, number of independent verifications, or number of validations). The trust metrics would help compute a trust level for the whole execution (a function of the trust of each object involved in the execution). Thus, a user could explore different combinations of objects for a given overall trust level. Conversely, the user could explore different combinations of objects and their impact on the overall trust score. Researchers in security and networking domains [20,25] have already investigated this problem: they represent objects in a graph where edges are trust relations and the trust evaluation is modeled as a path problem on a directed graph.

All these precautions will not avoid corruptions from a highly trusted object, however, because verification and validation cannot test exhaustively the behavior of all objects. This fact motivates research in the context of trust relations beyond reputation or research, in order to develop new reputation techniques.

7 Conclusion

In this paper, we analyze the research issue of the trust in numerical simulation results and scientific data analytics and identify possible research directions based on existing state-of-the-art solutions. A classic assumption that users make when running numerical simulations and data analytics is that floating-point computations are correct. Unfortunately, multiple examples of hardware bugs in floating-point units and software bugs in application stack make users more suspicious about the correctness of floating-point calculations. A significant issue for hardware bugs is that the time until the detection and the time between the detection of the issue and the repair could be very long. At the application level, fixing bugs that lead to corruptions in a version of a software does not mean that the number of corruptions would be lower accordingly. Parameterization defects leading to wrong results could be considered as a form of user-level corruptions. There are two possible research directions/solutions about the detection of corruptions: (1) performing on-line verification by using an external algorithmic observer that does not trust the execution stack; (2) establishing trust relations between levels of the execution stack.

Acknowledgments.. This material was based upon work supported by the U.S. Department of Energy,Office of Science, Advanced Scientific Computing Research Program, under Contract DE-AC02-06CH11357.

References

1. http://en.wikipedia.org/wiki/List_of_software_bugs . Accessed 08 May 2017
2. Disasters in bad numerical computing. http://www.iro.umontreal.ca/~mignotte/IFT2425/Disasters.html. Accessed 08 May 2017
3. Opteron bugs. https://access.redhat.com/solutions/918043. Accessed 08 May 2017
4. Opteron REP bug. http://www.theinquirer.net/inquirer/news/1042490/amd-opteron-bug-cause-incorrect-results. Accessed 08 May 2017
5. Trust Computing Group. http://www.trustedcomputinggroup.org/. Accessed 08 May 2017
6. Trust in Social Sciences. http://en.wikipedia.org/wiki/Trust_(social_sciences). Accessed 08 May 2017
7. Trust Metrics. http://en.wikipedia.org/wiki/Trust_metric. Accessed 08 May 2017
8. Trusted Platform Module (TPM) Specification. http://www.trustedcomputinggroup.org/resources/tpm_main_specification. Accessed 08 May 2017
9. TWC: Small: behavior-based zero-day intrusion detection for real-time cyber-physical systems. https://www.collectiveip.com/grants/NSF:1423334. Accessed 08 May 2017

10. ASCR Cybersecurity for Scientific Computing Integrity, February 2015. http://www.osti.gov/scitech/servlets/purl/1223021
11. Avizienis, A.: The N-version approach to fault-tolerant software. IEEE Trans. Softw. Eng. **11**(12), 1491–1501 (1985)
12. Avižienis, A., Laprie, J.-C., Randell, B.: Dependability and its threats: a taxonomy. In: Jacquart, R. (ed.) Building the Information Society. IIFIP, vol. 156, pp. 91–120. Springer, Boston (2004). https://doi.org/10.1007/978-1-4020-8157-6_13
13. Becker, S., Hasselbring, W., Paul, A., Boskovic, M., Koziolek, H., Ploski, J., Dhama, A., Lipskoch, H., Rohr, M., Winteler, D., Giesecke, S., Meyer, R., Swaminathan, M., Happe, J., Muhle, M., Warns, T.: Trustworthy software systems: a discussion of basic concepts and terminology. SIGSOFT Softw. Eng. Notes **31**(6), 1–18 (2006)
14. Benson, A.R., Schmit, S., Schreiber, R.: Silent error detection in numerical time-stepping schemes. Int. J. High Perform. Comput. Appl. **29**(4), 403–421 (2015)
15. Berrocal, E., Bautista-Gomez, L., Di, S., Lan, Z., Cappello, F.: Exploring partial replication to improve lightweight silent data corruption detection for HPC applications. In: Dutot, P.-F., Trystram, D. (eds.) Euro-Par 2016. LNCS, vol. 9833, pp. 419–430. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-43659-3_31
16. National Research Council: Assessing the Reliability of Complex Models: Mathematical and Statistical Foundations of Verification, Validation, and Uncertainty Quantification. The National Academies Press, Washington, D.C. (2012). https://www.nap.edu/catalog/13395/assessing-the-reliability-of-complex-models-mathematical-and-statistical-foundations
17. Di, S., Berrocal, E., Cappello, F.: An efficient silent data corruption detection method with error-feedback control and even sampling for HPC applications. In: 2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, pp. 271–280, May 2015
18. Di, S., Cappello, F.: Adaptive impact-driven detection of silent data corruption for HPC applications. IEEE Trans. Parallel Distrib. Syst. **27**(10), 2809–2823 (2016). https://doi.org/10.1109/TPDS.2016.2517639
19. Knight, J.C., Leveson, N.G.: An experimental evaluation of the assumption of independence in multiversion programming. IEEE Trans. Softw. Eng. **12**(1), 96–109 (1986)
20. Levien, R., Aiken, A.: Attack-resistant trust metrics for public key certification. In: Proceedings of the 7th Conference on USENIX Security Symposium, SSYM 1998, vol. 7, pp. 18–18. USENIX Association, Berkeley (1998)
21. Randell, B., Xu, J.: The evolution of the recovery block concept. In: Software Fault Tolerance, pp. 1–22. Wiley (1994)
22. Sha, L.: Using simplicity to control complexity. IEEE Softw. **18**(4), 20–28 (2001)
23. Sparks, E.R.: A security assessment of trusted platform modules. Technical report TR2007-597, Dartmouth College, Computer Science, Hanover, NH, June 2007
24. Subasi, O., Di, S., Bautista-Gomez, L., Balaprakash, P., Unsal, O., Labarta, J., Cristal, A., Cappello, F.: Spatial support vector regression to detect silent errors in the exascale era. In: 2016 16th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), pp. 413–424 May 2016
25. Theodorakopoulos, G., Baras, J.S.: On trust models and trust evaluation metrics for ad hoc networks. IEEE J. Sel. A. Commun. **24**(2), 318–328 (2006)
26. Türpe, S., Poller, A., Steffan, J., Stotz, J.-P., Trukenmüller, J.: Attacking the BitLocker boot process. In: Chen, L., Mitchell, C.J., Martin, A. (eds.) Trust 2009. LNCS, vol. 5471, pp. 183–196. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-00587-9_12