

# Confidence Estimation for Exascale Computations

Nageswara S. V. Rao

Oak Ridge National Laboratory, raons@ornl.gov

## INTRODUCTION

Computations running on Exascale systems for hours are expected to experience multiple component failures due to the millions of components, which cannot be guaranteed to be failure-free by the current engineering and manufacturing practices. Indeed such failures are expected by the norm, and all computations on them must be designed with appropriate tolerance mechanisms such as replication, checkpointing, process migration, domain-specific methods, and others. These mechanisms are designed and configured for specific failure levels, for example, triplicate code executions for robustness against single failures. It would be very inefficient to configure these mechanisms for the worst-case failure scenarios; rather, they should be configured with carefully designed failure tolerances. But, due to the random nature of component failures, there are no guarantees that failures will stay below these tolerance limits during the code executions that take hours or days. As a result, it is imperative that these computations be provided with confidence estimates that the failures were indeed within the tolerance limits, which in turn translate to the confidences on code outputs.

Over the past decades, analytical methods and practical systems have been developed to ensure robust computations over failure-prone systems:

- (a) In terms of foundational works, von Neumann [4] studied (in the 50s) the mathematical aspects of achieving reliable computations over systems built using unreliable components. Perhaps due to the subsequent reliability improvements in computing systems, such studies have not been extensively continued.
- (b) In terms of deployed systems, computation in space vehicles (deployed over the past decades) have been enhanced with Software-Implemented Hardware Fault Tolerance (SIHFT) methods to counteract the transient faults in computers due to radiation exposure [1] in space environments.

Despite the commonalities with these works that address ‘smaller’ systems, the challenges of achieving robust computations on exascale computing systems are unprecedented, due to their sheer complexity and scale. At the same time, they also present promising opportunities for the mathematical design and analysis methods. We outline a method that inserts diagnosis tracer codes into application codes to follow the execution paths, detect faults along the way, and generate statistical confidence estimates for the outputs [3]. We describe a set of mathematical problems and potential solution approaches to generate these confidence estimates to ensure that codes are executed under the tolerance limits. While system-level profiles can also be used for such estimation, the application traces are more focussed in following the application execution.

## ROBUST CODE EXECUTIONS

Let  $\mathcal{S}_i, i = 0, 1, \dots$ , denote the random state of the exascale system at time step  $i$ . Typically, this is a large dimensional vector where its components represent the status and other properties of components of the exascale system such as processor core, communication link or memory element. Let  $\mathcal{S}_{\mathcal{I}}$  denote the idealized system with no errors so that if  $\mathcal{S}_i$  is error-free, we have  $\mathcal{S}_{\mathcal{I}} \otimes \mathcal{S}_i = 0$ , where  $\otimes$  represents a difference operation. Consider that an algorithm  $\mathcal{A}$  executed on system in state  $\mathcal{S}_i$  with input  $I$  produces output  $\mathcal{S}_i \odot \mathcal{A}(I)$ , which is random due to the randomness of  $\mathcal{S}_i$ . The ideal fault-free output of the algorithm is  $\mathcal{S}_{\mathcal{I}} \odot \mathcal{A}(I)$  corresponding to input  $I$ , which could be random if the underlying algorithm incorporates Monte Carlo or other random variables. Let  $\mathcal{A}_{\mathcal{R}}$  denote a *robust version* of  $\mathcal{A}$  customized to be executed on an exascale system, for example replicated version of the same code or code augmented with different checkpoints. When executed on a system at state  $\mathcal{S}_i$ , the output error is

$$\mathcal{E}(\mathcal{S}_i, \mathcal{A}_{\mathcal{R}}, I) = (\mathcal{S}_{\mathcal{I}} \odot \mathcal{A}(I)) \oplus (\mathcal{S}_i \odot \mathcal{A}_{\mathcal{R}}(I)),$$

where the difference operator  $\oplus$  is appropriately chosen; for example, it could be the Euclidean distance for codes that produce deterministic real vector output, and a statistical 0-vector test for Monte Carlo simulations. This error may incorporate multiple sources of randomness: (i) distribution of input  $I$ , denoted by  $\mathbf{P}_I$ ; (ii) distribution of the output of algorithm  $\mathcal{A}_{\mathcal{R}}$  if it contains random variables, denoted by  $\mathbf{P}_{\mathcal{A}_{\mathcal{R}}(I)}$ , and (iii) distribution of the system state  $\mathcal{S}_i$  due to the randomness of various failures of components, denoted by  $\mathbf{P}_{\mathcal{S}_i}$ . The expected error of the algorithm  $\mathcal{A}_{\mathcal{R}}$  executed on the exascale system in state  $\mathcal{S}_i$  is given by

$$\begin{aligned} \bar{\mathcal{E}}_I(\mathcal{S}_i, \mathcal{A}_{\mathcal{R}}) &= \int \mathcal{E}(\mathcal{S}_i, \mathcal{A}_{\mathcal{R}}, I) d\mathbf{P}_{\mathcal{A}_{\mathcal{R}}(I), I} \\ &= \int_I \int [(\mathcal{S}_{\mathcal{I}} \odot \mathcal{A}(I)) \oplus (\mathcal{S}_i \odot \mathcal{A}_{\mathcal{R}}(I))] d\mathbf{P}_{\mathcal{S}_i \odot \mathcal{A}_{\mathcal{R}}(I) | I} d\mathbf{P}_I. \end{aligned}$$

The overall quality of computation of the algorithm  $\mathcal{A}_{\mathcal{R}}$  executed on the exascale system is given by

$$\begin{aligned} \bar{\mathcal{E}}(\mathcal{A}_{\mathcal{R}}) &= \int \mathcal{E}(\mathcal{S}_i, \mathcal{A}_{\mathcal{R}}, I) d\mathbf{P}_{\mathcal{S}_i, \mathcal{A}_{\mathcal{R}}(I), I} \\ &= \int_{\mathcal{S}_i} \left[ \int \mathcal{E}(\mathcal{S}_i, \mathcal{A}_{\mathcal{R}}, I) d\mathbf{P}_{\mathcal{A}_{\mathcal{R}}(I), I | \mathcal{S}_i} \right] d\mathbf{P}_{\mathcal{S}_i} \\ &= \int_{\mathcal{S}_i} \left[ \int [(\mathcal{S}_{\mathcal{I}} \odot \mathcal{A}(I)) \oplus (\mathcal{S}_i \odot \mathcal{A}_{\mathcal{R}}(I))] \right. \\ &\quad \left. d\mathbf{P}_{\mathcal{S}_i \odot \mathcal{A}_{\mathcal{R}}(I), I | \mathcal{S}_i} \right] d\mathbf{P}_{\mathcal{S}_i} \end{aligned}$$

which indicates that this error can be reduced by customizing  $\mathcal{A}_{\mathcal{R}}$  to exploit the state  $\mathcal{S}_i$ . A customized code to the

state estimate  $\hat{S}_i$  is denoted by  $\mathcal{A}_{\hat{S}_i, \mathcal{R}}$ , and its objective could be to ensure  $\mathcal{E}(S_i, \mathcal{A}_{\hat{S}_i, \mathcal{R}}, I) \leq \mathcal{E}(S_i, \mathcal{A}_{\mathcal{R}}, I)$ , or its weaker versions given by  $\bar{\mathcal{E}}_I(S_i, \mathcal{A}_{\hat{S}_i, \mathcal{R}}) \leq \bar{\mathcal{E}}_I(S_i, \mathcal{A}_{\mathcal{R}})$  and  $\bar{\mathcal{E}}(\mathcal{A}_{\hat{S}_i, \mathcal{R}}) \leq \bar{\mathcal{E}}(\mathcal{A}_{\mathcal{R}})$ . The code customization may be carried out in different ways, including, executing it only on parts of the machine with high reliability estimates, replicating it as in N-version programming, checkpointing, using application specific methods, or a combination. This optimization requires an accurate estimate  $\hat{S}_i$  of the state  $S_i$  that would be valid during the execution of  $\mathcal{A}_{\hat{S}_i, \mathcal{R}}$ , which is derived from the output of tracer codes. Let the detection modules in tracer codes be executed  $R_p$  times per second, and  $P_{1/R_p}$  denote the probability of a node failure during  $1/R_p$  seconds interval. Let  $C_N(\alpha, N_p)$  denote the confidence that node failure probability is less than  $\alpha$  after  $N_p$  detection steps. If  $\hat{f}$  is the fraction of  $N_p$  steps that detected failures, we have

$$C_N(\alpha, N_p) = \mathbf{P} \left\{ \left| P_{1/R_p} - \hat{f} \right| < \beta \right\} > 1 - 2^{-1} [1 - (1 - \beta)^{N_p}]^{2N_p}$$

based on Hoeffdings inequality under statistical independence of component failures. These estimates can be used to compute the expected number of components failures during the computation, and the detection information can be used to estimate the portion of  $\hat{S}_i$  that overlap with application execution path.

We note that  $\mathcal{A}_{\mathcal{R}}$  might itself be designed to tolerate certain types of failures, and  $\mathcal{A}_{\hat{S}_i, \mathcal{R}}$  represents a finer level optimization based on the current state estimate  $\hat{S}_i$ . Also,  $\mathcal{A}_{\mathcal{R}}$  could itself be sufficiently robust, and methods are needed to verify such a property, for example, statistical methods that combine outputs from multiple code runs and system failure profiles. Computations on exascale systems must incorporate the component failures as an integral part, for example, by taking into account the random state estimate  $\hat{S}_i$ . The state vector can be used to partition the computing units of the system into different zones based on the levels of robustness and connections between the components. This task involves two challenges: (i) stochastic optimization methods for decomposing the set of computing units  $\mathcal{C}_i$  of the system based on state estimate  $\hat{S}_i$ , and (ii) optimally mapping the computational modules of  $\mathcal{A}_{\mathcal{R}}$  or  $\mathcal{A}_{\hat{S}_i, \mathcal{R}}$  onto  $\mathcal{C}_i$  based on the robustness zones, while ensuring that module dependencies are suitably supported. The underlying mathematical problems can be formulated using an assignment  $\mathcal{M}_{\hat{S}_i} : \mathcal{A}_{\mathcal{R}} \mapsto \mathcal{C}_i$  designed to capture strategies such as restricting code executions only to certain zones with sufficient robustness levels, replicating  $\mathcal{A}_{\mathcal{R}}$  in zones in numbers inversely proportion to their robustness levels, and other finer module-to-unit mappings.

#### CONFIDENCE MEASURES FOR COMPUTATIONS

The confidence measures for outputs of  $\mathcal{A}_{\mathcal{R}}$  can be generated using an estimation algorithm  $\mathcal{C}$  such that  $\mathcal{C}(\hat{S}_i, S_i \odot \mathcal{A}_{\mathcal{R}}(I)) \in [0, 1]$  denotes the likelihood that the output of  $\mathcal{A}_{\mathcal{R}}$  using input  $I$  corresponds to fault-free execution, namely,

$$\mathcal{C}(\hat{S}_i, S_i \odot \mathcal{A}_{\mathcal{R}}(I)) = \mathbf{P} \{ (S_i \odot \mathcal{A}_{\mathcal{R}}(I)) = (S_I \odot \mathcal{A}(I)) \},$$

which is a random variable due to the distribution of input, system state  $S_i$  and estimation error in  $\hat{S}_i$ . The overall expected

confidence is given by

$$\bar{\mathcal{C}}(\hat{S}_i, \mathcal{A}_{\mathcal{R}}) = \int \mathcal{C}(\hat{S}_i, S_i \odot \mathcal{A}_{\mathcal{R}}(I)) d\mathbf{P}_{S_i \odot \mathcal{A}_{\mathcal{R}}(I), I}.$$

Based on  $l$  executions of  $\mathcal{A}_{\mathcal{R}}$  with different inputs  $I_1, I_2, \dots, I_l$  with known outputs we can provide confidence measures as follows. We compute the empirical estimate

$$\hat{\mathcal{C}}(\hat{S}_i, \mathcal{A}_{\mathcal{R}}) = \frac{1}{l} \sum_{i=1}^l \mathcal{C}(\hat{S}_i, S_i \odot \mathcal{A}_{\mathcal{R}}(I_i))$$

which can be shown to be close to  $\bar{\mathcal{C}}(\hat{S}_i, \mathcal{A}_{\mathcal{R}})$  when the underlying function  $\mathcal{C}(\cdot)$  is chosen from a class of functions with finite scale-sensitive dimension. Extensions to this method would be needed to exploit and account for the underlying distributions and statistical correlations.

When the outputs of ideal code execution are not known, one can utilize the outputs from replicated codes to generate approximations for  $\mathcal{C}(\hat{S}_i, S_i \odot \mathcal{A}_{\mathcal{R}}(I_i))$ , to be used in the confidence measures for computations. Rigorous methods are needed to generate such approximations based on code outputs and the current  $\hat{S}_i$ . For example, when the failure profiles are close to 0-function, the agreements between the replicated code outputs constitute a good indicator for the confidence in the code output. While a test for such an agreement between deterministic outputs can be based on simple differences, codes that incorporate Monte Carlo computations or random strategies present challenges, particularly by requiring suitable statistical 0-tests.

We briefly describe a scenario of computations that employ random strategies, wherein code replications used for fault tolerance can be leveraged to derive better outputs (as a byproduct). For applications such as stochastic search, different outputs of replicated codes could be due to the randomness in the search process and neither of them could be incorrect. In such cases, different ‘‘correct’’ outputs can be combined to provide an improved solution using the information fusion methods [2]. This approach requires statistical tests to distinguish between the correct and erroneous code outputs, which can be achieved by examining the confidence measures. The methods that would identify and eliminate the ‘‘erroneous’’ outputs can be formulated as statistical tests, and ones that fuse accurate outputs can be formulated as generic information fusion problems; existing solutions to both classes of problems lay the foundations for enhancements needed for exascale computations.

#### REFERENCES

- [1] P. P. Shirvani et al. Software-implemented hardware fault tolerance experiments: Cots in space. In *Proc. International Conference on Dependable Systems and Networks*, pages 56–57. 2000.
- [2] N. S. V. Rao. Measurement-based statistical fusion methods for distributed sensor networks. In S. S. Iyengar and R. R. Brooks, editors, *Distributed Sensor Networks*. Chapman and Hall/CRC Publishers, 2011. 2nd Edition.
- [3] N. S. V. Rao. Chaotic-identity maps for robustness estimation of exascale computations. In *2nd Workshop on Fault-Tolerance for HPC at Extreme Scale (FTXS 2012)*, 2012.
- [4] J. von Neumann. Probabilistic logics and the synthesis of reliable organisms from unreliable components. In C. E. Shannon and J. McCarthy, editors, *Automata Studies*. Princeton University Press, 1956.