Improving Fault Tolerance for Extreme Scale Systems

Eduardo Berrocal
Illinois Institute of Technology
Chicago, Illinois, USA
Email: eberroca@iit.edu

Zhiling Lan
Illinois Institute of Technology
Chicago, Illinois, USA
Email: lan@iit.edu

***Abstract*—Mean Time Between Failures (MTFB), now calculated in days or hours, is expected to drop to minutes on exascale machines. The advancement of resilience technologies greatly depends on a deeper understanding of faults arising from hardware and software components. This understanding has the potential to help us build better fault tolerance technologies. For instance, it has been proved that combining checkpointing and failure prediction leads to longer checkpoint intervals, which in turn leads to fewer total checkpoints. In our work, we presented a new density-based approach for failure prediction based on the Void Search (VS) algorithm, and evaluated the algorithm using real environmental logs from the Mira Blue Gene/Q supercomputer at Argonne National Laboratory. As we move to exascale, other problems will also arise as transistor size and energy consumption of future systems must be significantly reduced, steps that might dramatically impact the soft error rate (SER). When soft errors are not detected and corrected properly, either by hardware or software mechanisms, they have the potential to corrupt applications' memory state. In our previous work we leveraged the fact that datasets produced by HPC applications (i.e., the applications' state at a particular point in time) have characteristics that reflect the properties of the underlying physical phenomena that those applications attempt to model. These characteristics can be used effectively to design a general corruption detection scheme with relatively low overhead. Both of these problems are critical for the HPC community and a main focus of my Ph.D. research.**

I. Introduction

The astounding advances in transistor technology, which drove the automatic increases in computational power following the famous Moore's Law since the 1960s (i.e., doubling CPU speed every 18 months or so), came to an end by the mid 2000s due mainly to the problems related to heat disipation when operating at high clock frequencies. Until then, system designers as well as software developers could rely on this "free lunch" to make their applications more and more powerful over time without re-thinking their underlying algorithms.

When this "free lunch" came to an end, however, the computing community was forced to shift from the sequential paradigm, no longer viable in the long run, to the parallel and distributed one in order to keep scaling their systems and applications. Of course, this paradigm is not free of problems either, specially when systems need to scale to hundreds of thousands – or even millions – of components to handle a world with ever-increasing demands for more data and computation.

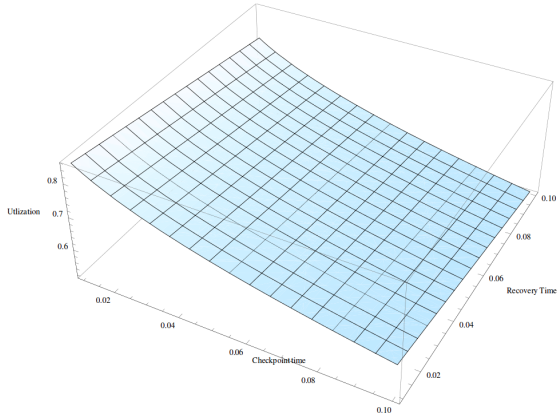

Fig. 1: System utilization as a function of checkpoint and recovery time (from [1]).

One of the areas where parallel and distributed processing was first adopted, in the form of high performance computing (HPC) systems, was scientific computing. Science applications always required, and will continue to require, ever-larger machines to solve problems with higher accuracy. Future HPC systems promise to provide the power needed to tackle new and revolutionary science problems, but they are also raising new challenges. For example, resilience in HPC systems at exascale, where supercomputers are projected to have hundreds of thousands of nodes and millions of cores, will drop mean time between failures (MTBF) from days and hours in current petascale machines, to just minutes [1], [2]. Moreover, typical fault tolerance (FT) approaches – such as classic global Checkpoint/Restart (C/R) – will be rendered useless as it will be impossible to checkpoint the whole applications' memory state in an amount of time small enough to keep a high utilization rate.

As an illustration of this problem, consider Figure 1. The effective utilization of a system is calculated as a function of the checkpoint time and the recovery time, both relative to the MTBF [1]. The checkpoint interval is calculated using the well known Young's Equation $\tau_{opt} = \sqrt{2C(\text{MTBF})}$ [3], where C is the checkpoint time. Supposing we want to achieve

a utilization rate of more than 80%, checkpoint time needs to be kept at 1%-2% of MTBF and recovery time at 2%-5% of MTBF [1]. Assuming a MTBF of 30 minutes at exascale, we would need to have a checkpoint time of at most 20 seconds, and a recovery time of at most 1 minute. With the exponential grow that applications will experience at exascale, it is highly unlikely that global checkpoints will be able to be completed in such a limited time-frame.

Other problems will also arise as transistor size and energy consumption of future systems must be significantly reduced, steps that might dramatically impact the soft error rate (SER) according to recent studies [4], [5]. When soft errors are not detected and corrected properly, either by hardware or software mechanisms, they have the potential to corrupt applications' memory state. These type of errors are commonly known as silent data corruption (SDC), and are extremely damaging because they can make applications silently (without the user knowing it) produce wrong results.

Both of these problems are critical for the HPC community and a main focus of my Ph.D. research. In the following section, I will outline the current state-of-the-art regarding solutions to these two problems, including my own past research.

II. Errors in HPC

A. Hard Error Prediction

Numerous work has been devoted to the problem of reducing the large overheads imposed by the classic global C/R model. For example, [6] and [7] look into data compression as a solution to scale classic C/R into the future. Although their results are promising, compression does not break itself apart from the traditional C/R architecture, so it is unclear yet for how long it will be viable at extreme scales. More promising is multi-level (hierarchical) checkpoint, where local – and very fast – checkpoints to the local storage (such as SSD disks or RAM disks), or to neighboring nodes' storage, can be combined with global checkpoints so as to reduce total global checkpoint frquency [8], [9]. Furthermore, recoveries can also be done very fast by reading back local checkpoints from local storage in the case of a soft error, or from neighboring nodes' storage in the case of a hard error (e.g., the failed node went down and can't be booted up again). The global checkpoint is used only in cases where local checkpoints are impossible to be recovered due to multiple failures.

These works can be categorized as *reactive*; the don't attempt to understand failures occurring in the system, but rather to minimize their harmful effects by reducing the amount of work that needs to be redone after a failure. Another type of methods aim to decrease checkpoint cost by predicting when and where failures are going to take place in advance. For example, if we know in advance where a failure is going to take place, we can just checkpoint that portion of the system instead of the whole memory state. In addition, checkpoint frequency can be substantially reduced if we can predict a large portion of the total number of failures [10], [11].

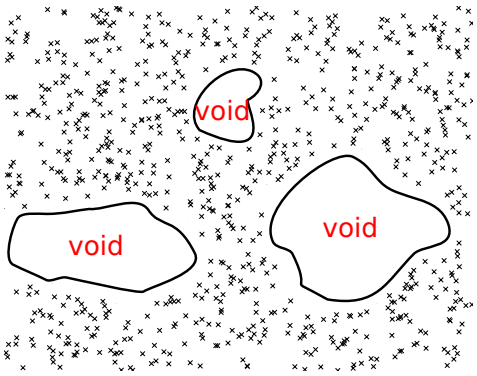

Fig. 2: Three voids in a given set of randomly generated points.

Traditionally, only the Reliability, Availability and Serviceability (RAS) logs produced by different types of systems has been used for failure prediction [12], [13], [14], [15]. RAS logs, however, are simple and limited since they are designed to be read and understood by humans (e.g., system administrators). These logs are composed of messages generated by a centralized monitor process, which queries the different components at a given frequency and may rise a warning if some value is above a given threshold. For example, the time to complete an I/O operation is too long of CPU temperature is too high. Or, it may rise an error if a component is not functioning correctly or not working altogether. By looking at one value at a time, however, RAS logs may miss important patterns hidden in the data.

Environmental logs, on the other hand, are numerical values directly read from the hardware sensors spread all over the system such as fan speed, CPU temperature, input voltage, and so on. Since every new generation of supercomputing systems come with better hardware sensors and profiling capabilities, it is of great importance to understand environmental data especially with respect to resilience. In our previous work [16], we proposed a failure prediction algorithm – based on the Void Search (VS) family of algorithms – that works with environmental logs instead of RAS logs.

Void Search (VS) algorithms are a family of density-based methods aimed at finding regions of empty – or low density – space for a given set of data points. These empty regions are known as voids, and in astrophysics they are essential for studying galaxy formation and the structure of the cosmic web [17]. For us, voids are essentially patterns of feature space for existing data regions. This contrasts sharply with traditional algorithms which search for patterns of feature space for existing data points. Voids can be of great interest in the case where one of the data classes in a two-class learning problem is hard to characterize. Figure 2 shows an example of three voids in a two dimensional space.

Our previous work [16] seems to show that the combination of both, environmental logs and a VS-based algorithm, could

have a great potential for failure prediction in extreme scale systems. In fact, current state-of-the-art in failure prediction using RAS logs provides a sensitivity (also called recall) of about 0.5 (i.e., only half of the failures can be predicted effectively) [18], while we could achieve well above 0.9 using environmental logs, without incurring in an excessive number of false alarms. The data used was four months of environmental and RAS logs from the Mira BlueGene/Q Supercomputer at Argonne National Laboratory (ANL).

B. Soft Error Detection

The problem of data corruption for extreme-scale computers has been the target of numerous studies. They can be classified in four groups depending on their level of generality, that is, how easily a technique can be applied to a wide spectrum of HPC applications. They also have different cost in time, space, and energy. An ideal SDC detection technique should be as general as possible, while incurring a minimum cost over the application. These four groups are: hardware-level detection, process replication, algorithm-based fault tolerance, and approximate computing.

Hardware-level detection, such as error-correcting codes (ECCs) in random memory access (RAM) devices, is extremely general because applications do not require any adaptation to benefit from such detectors. Recent studies, however, indicate that ECCs alone cannot correct an important number of DRAM errors [19]. In addition, not all parts of the system are ECC-protected: in particular, logic units and registers inside the processing units are usually not ECC-protected because of the space, time, and energy cost that ECC requires in order to work at low level. Historically, the SER of central processing units was minimized through a technique called *radiation hardening* [20], which consists of increasing the capacitance of circuits nodes in order to increase the critical charge needed to change the logic level. Unfortunately, this technique involves increasing either the size or the energy consumption of the components, which is prohibitively expensive at extreme scale. Thus, a non-negligible ratio of soft errors could pass undetected by the hardware, corrupting the numerical data of HPC applications. This is what we call silent data corruption (SDC).

Process replication has been used for many years to guarantee correctness in critical systems, and its application to HPC systems has been studied. Fiala et al., for example, proposed using double-redundant computation to detect SDC by comparing the messages transmitted between the replicated processes [21]. The authors also suggested using triple redundancy to enable data correction through a voting scheme. This approach is general in that applications need little adaptation to benefit from double and triple redundancy. Unfortunately, double- and triple-redundant computation always imposes large overheads, since the number of hardware resources will be double or triple.

A promising technique against data corruption is algorithm-based fault tolerance (ABFT) [22]. This technique uses extra checksums in linear algebra kernels in order to detect and correct corruptions [23]. However, ABFT is not general, since each algorithm needs to be adapted by hand, and only some linear algebra kernels have been adapted, which is only a subset of the vast spectrum of computational kernels. Furthermore, even applications that employ only ABFT-protected kernels could fail to detect SDCs if the corruption lies *outside* the ABFT-protected regions.

The last group of SDC detection is based on the idea of approximate computing. In this detection method, a computing kernel is paired with a cheaper and less accurate kernel that will produce *close enough* results. Such results can be compared with those generated by the main computational kernel [24]. This detection mechanism shows promising results, but again it is still not general enough, since each application needs to be manually complemented with the required approximate computing kernels. Furthermore, complex applications also need to adapt multiple kernels to offer good coverage.

In our previous work [25], [26] we leveraged the fact that datasets produced by HPC applications (i.e., the applications' state at a particular point in time) have characteristics that reflect the properties of the underlying physical phenomena that those applications attempt to model. These characteristics can be used effectively to design a general SDC detection scheme with relatively low overhead. In particular, we proposed to leverage the spatial and temporal behavior of HPC datasets to predict an interval of *normal* values for the evolution of the datasets, such that any corruption will *push* the corrupted data point outside the expected interval of *normal* values, and it will, therefore, become and *outlier*. We call this approach data-analytic-based (DAB) fault tolerance. As shown in Figure 3, the prediction – done using different linear methods, such as Auto-Regressive (AR) model or Linear-Curve Fitting (LCF) – is dependent upon recent past values of the same data point. Therefore, we can think of this problem as a one-step ahead prediction for a time series.

To reach a high prediction accuracy, one must keep a set of recent time steps of the data in memory (evident window). The more data one keeps, however, the bigger the overhead. Nevertheless, we can keep this overhead under tolerable levels. If we consider, for example, that a particular predictor needs to keep the recent data with three time steps for each data point, the total *state memory* is usually just a fraction of the total memory used by the application. This is because applications may need extra buffers for different computational kernels (e.g., FFTs). Total memory overhead, then, can always be kept under 100% using 3-4 time steps, which is better than double-redundant computation. Furthermore, our techniques do not incur any network overhead, which is doubled (or tripled) in process replication.

We perform a comprehensive evaluation using all our predictors and detectors with a number of popular HPC applications (two computational fluid dynamic (CFD) kernels and an N-body cosmology application, all developed at ANL), and we show that our detectors can guaranteee over 90% of SDC coverage on real application runs with very high precision.

In previous work [27], we also worked on a new adaptive

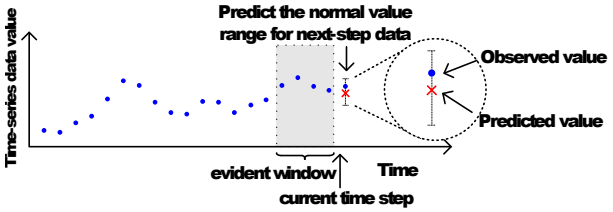

Fig. 3: Runtime One-Step value Prediction Model for SDC Detection (from [26]).

SDC detection approach that combines the merits of replication and DAB. More specifically, we have observed that not all processes of some MPI applications experience the same level of data variability at exactly the same time; hence, one can smartly choose and replicate only those processes for which lightweight data-analytic detectors would perform poorly.

In addition, evaluating detectors solely on overall single-bit precision and recall may not be enough to understand how well applications are actually protected. Instead, we calculate the probability that a corruption will pass unnoticed by a particular detector. An interesting observation is that generally, the fewer bits that can get "flipped" in a system, the harder it is to detect corruptions using software mechanisms. The key idea is that protecting the data of simulations at this level is not so much protecting against particular bit-flips as it is protecting against numerical deviations from the original data.

We evaluated our approach using two applications dealing with explosions from the FLASH code package [28], which are excellent candidates for testing partial replication. Our results show that our adaptive approach is able to protect the MPI applications analyzed (99.999% detection recall) replicating only 43-51% of all the processes with a maximun total overhead of only 52-56% (compared with 110% for pure duplication).

References

[1] "Addressing failures in exascale computing," *ANL report*, March 2013.

[2] B. Schroeder and G. Gibson, "Understanding failure in petascale computers," *Journal of Physics Conference Series: SciDAC*, vol. 78, p. 012022, June 2007.

[3] J. W. Young, "A first order approximation to the optimum checkpoint interval," *Communications of the ACM*, vol. 17, no. 9, September 1974.

[4] S. Di, D. Kondo, and W. Cirne, "Host load prediction in a google compute cloud with a bayesian model," in *SC'12*, 2012, pp. 21:1–21:11.

[5] E. Normand, "Single event upset at ground level," *IEEE Transactions on Nuclear Science*, vol. 43, no. 6, pp. 2742–2750, 1996.

[6] D. Ibtesham, D. Arnold, P. G. Bridges, K. B. Ferreira, and R. Brightwell, "On the viability of compression for reducing the overheads of checkpoint/restart-based fault tolerance," in *41st International Conference on Parallel Processing*, 2012, pp. 148–157.

[7] T. Z. Islam, K. Mohror, S. Bagchi, A. Moody, B. D. Supinski, and R. Eigenmann, "Mcrengine: A scalable checkpointing system using data-aware aggregation and compression," in *High Performance Computing, Networking, Storage and Analysis (SC)*, 2012.

[8] L. Bautista-Gomez, D. Komatitsch, N. Maruyama, S. Tsuboi, F. Cappello, and S. Matsuoka, "Fti: high performance fault tolerance interface for hybrid systems," in *High Performance Computing, Networking, Storage and Analysis (SC)*, 2011, pp. 1–12.

[9] A. Moody, G. Bronevetsky, K. Mohror, and B. R. D. Supinski, "Design, modeling, and evaluation of a scalable multi-level checkpointing system," in *High Performance Computing, Networking, Storage and Analysis (SC)*, 2010, pp. 1–11.

[10] G. Aupy, Y. Robert, F. Vivien, and D. Zaidouni, "Checkpointing algorithms and fault prediction," *Technical Report, INRIA*, no. RR-8237, February 2013.

[11] M. S. Bouguerra, A. Gainaru, F. Cappello, L. B. Gomez, N. Maruyama, and S. Matsuoka, "Improving the computing efficiency of hpc systems using a combination of proactive and preventive checkpointing," in *International Parallel and Distributed Processing Symposium (IPDPS)*, 2013.

[12] Z. Lan, J. Gu, Z. Zheng, R. Thakur, and S. Coghlan, "A study of dynamic meta-learning for failure prediction in large-scale systems," *Journal of Parallel and Distributed Computing (JPDC)*, 2010.

[13] R. K. Sahoo, A. J. Oliner, I. Rish, M. Gupta, J. E. Moreira, and S. Ma, "Critical event prediction for proactive management in large-scale computer clusters," in *International conference on Knowledge discovery and data mining*, 2003, pp. 426–435.

[14] Z. Zheng, Z. Lan, R. Gupta, S. Coghlan, and P. Beckman, "A practical failure prediction with location and lead time for blue gene/p," in *Proc. of the 1st Workshop on Fault-Tolerance for HPC at Extreme Scale (FTXS) in conjunction with DSN*, 2010.

[15] Y. Liang, Y. Zhang, H. Xiong, and R. Shaoo, "Failure prediction in ibm bluegene/l event logs," in *7th International Conference on Data Mining*, 2007.

[16] E. Berrocal, L. Yu, S. Wallace, M. E. Papka, and Z. Lan, "Exploring void search for fault detection on extreme scale systems," in *IEEE Cluster (Best Paper Award)*, 2014.

[17] M. C. Neyrinck, "Zobov: a parameter-free void-finding algorithm," *Monthly Notices of the Royal Astronomical Society*, vol. 386, pp. 2101–2109, 2007.

[18] A. Gainaru, F. Cappello, M. Snir, and W. Kramer, "Fault prediction under the microscope: A closer look into hpc systems," in *High Performance Computing, Networking, Storage and Analysis (SC)*, 2012.

[19] A. A. Hwang, I. A. Stefanovici, and B. Schroeder, "Cosmic rays don't strike twice: Understanding the nature of dram errors and the implications for system design," in *ASPLOS'XVII*, 2012, pp. 111–122. [Online]. Available: http://doi.acm.org/10.1145/2150976.2150989

[20] S. Krishnamohan and N. R. Mahapatra, "Analysis and design of soft-error hardened latches," in *GLSVLSI'05*, 2005, pp. 328–331. [Online]. Available: http://doi.acm.org/10.1145/1057661.1057740

[21] D. Fiala, F. Mueller, C. Engelmann, R. Riesen, K. Ferreira, and R. Brightwell, "Detection and correction of silent data corruption for large-scale high-performance computing," in *SC'12*, 2012, pp. 78:1–78:12.

[22] K.-H. Huang and J. A. Abraham, "Algorithm-based fault tolerance for matrix operations," *IEEE Transactions on Computers (TC)*, vol. 33, no. 6, pp. 518–528, Jun. 1984. [Online]. Available: http://dx.doi.org/10.1109/TC.1984.1676475

[23] Z. Chen, "Online-abft: An online algorithm based fault tolerance scheme for soft error detection in iterative methods," in *PPoPP'13*, 2013, pp. 167–176. [Online]. Available: http://doi.acm.org/10.1145/2442516.2442533

[24] A. R. Benson, S. Schmit, and R. Schreiber, "Silent error detection in numerical time-stepping schemes," *International Journal of High Performance Computing Applications*, pp. 1–20, 2014.

[25] E. Berrocal, L. Bautista-Gomez, S. Di, Z. Lan, and F. Cappello, "Lightweight silent data corruption detection based on runtime data analysis for hpc applications," in *HPDC'15 (short paper)*, 2015.

[26] S. Di, E. Berrocal, and F. Cappello, "An efficient silent data corruption detection method with error-feedback control and even sampling for hpc applications," ser. CCGRID, 2015.

[27] E. Berrocal, L. Bautista-Gomez, S. Di, Z. Lan, and F. Cappello, "Exploring partial replication to improve lightweight silent data corruption detection for hpc applications," in *EuroPar'16*, 2016.

[28] B. Fryxell, K. Olson, P. Ricker, F. Timmes, M. Zingale, D. Lamb, P. MacNeice, R. Rosner, J. Truran, and H. Tufo, "Flash: an adaptive mesh hydrodynamics code fro modeling astrophysical thermonuclear flashes," *ApJS*, vol. 131, no. 273, 2000.