Detecting Silent Data Corruption through Data Dynamic Monitoring for Scientific Applications

Leonardo Bautista Gomez and Franck Cappello

Argonne National Laboratory

Abstract

Parallel programming has become one of the best ways to express scientific models that simulate a wide range of natural phenomena. These complex parallel codes are deployed and executed on large-scale parallel computers, making them important tools for scientific discovery. As supercomputers get faster and larger, the increasing number of components is leading to higher failure rates. In particular, the miniaturization of electronic components is expected to lead to a dramatic rise in soft errors and data corruption. Moreover, soft errors can corrupt data silently and generate large inaccuracies or wrong results at the end of the computation. In this paper we propose a novel technique to detect silent data corruption based on data monitoring. Using this technique, an application can learn the *normal* dynamics of its datasets, allowing it to quickly spot anomalies. We evaluate our technique with synthetic benchmarks and we show that our technique can detect up to 50% of injected errors while incurring only negligible overhead.

Categories and Subject Descriptors C4 [*Performance of systems*]: Fault tolerance

Keywords Fault Tolerance, Supercomputers, Silent Data Corruption, Soft Errors, Bit Flips, Data Entropy.

1. Introduction

Computing-intensive scientific applications need post-petascale machines to achieve results in a reasonable amount of time. Unfortunately, the increasing number of components in such large machines leads to a decreasing mean time between failures (MTBF) for extreme-scale systems. In addition to hardware failures, soft errors can cause one or multiple bits to spontaneously flip to the opposite state. This bit flipping might be due to multiple reasons such as alpha particles from package decay, cosmic rays, or thermal neutrons. Although techniques such as error correcting codes (ECCs) have been proposed and implemented to tackle soft errors, the reality is that a significant number of bit flips still manage to pass undetected. Moreover, the constant need to reduce component size and voltage, limits the use of soft-error mitigation techniques, dramatically increasing the SER in the coming years [1, 3]. Also, some of those bit flips can introduce errors silently in the data, generating wrong results; this is called silent data corruption (SDC). To guarantee accurate and correct results for scientific applications in the presence of SDC is one of the hardest challenges of extreme-scale computing. SDC, by definition, is not detected in the lower levels of the hardware/software stack. In contrast, higher-level software could leverage properties of the data dynamics or analyze data patterns in order to detect outliers that could suggest the presence of corruption. In this article, we propose a novel fault tolerance technique that leverages data entropy characteristics of scientific datasets to determine whether corruption has occurred. Experimental evaluation of our data dynamic monitoring technique on synthetic benchmarks show that our detector is able to reduce the number of SDC seen by the application, by up to 50% and can reach an accuracy of over 98% while incurring less than 1% of overhead to the application.

PPoPP '14, February 15–19, 2014, Orlando, Florida, USA.

ACM 978-1-4503-2656-8/14/02…$15.00.

http://dx.doi.org/10.1145/nnnnnnn.nnnnnnn

2. Ultra-light SDC detector

An ultralight SDC detector should impose negligible overhead to the scientific application, even while executed at high frequency. In addition, the detector should have the highest achievable accuracy. The accuracy of a SDC detector is quantified by using two measures: its *recall* and its *precision*. Our universe obeys a set of physical laws that dictate the behavior of matter and energy. These laws impose some limits (e.g., nothing travels faster than light) that define what is physically possible and meaningful and what is not. These physical limits are inherited by the computational codes that simulate those phenomena, and they can be used to detect abnormalities. For instance, a weather simulation holding a temperature field with values over 500 kelvin is a clear sign of data corruption, because such temperatures are unlikely to be observed on the surface of our planet. The presence of these physical limits in the computational codes executed on supercomputers provides us with our first insight into detecting data corruption. Unfortunately, when the interval of possible values is too large, corrupted data is likely to be inside this interval, decreasing the efficiency of such a strategy.

Our second insight into detecting outliers in large HPC datasets is closely related to the notion of space. We notice that in a large number of physical phenomena, points that are close in space exhibit similar behavior. In the weather example, although the temperature on the Earth's surface can vary widely between the equator line and the poles, any two close points on the planet should have relatively close temperatures, as well as close pressures, wind speeds, and the like. The reason is that points close in space are subject to the same external factors and forces. This observation can be translated as a sort of *local regularity* of HPC datasets. This is supported by the fact that most parallel applications in HPC simulate physical phenomena by decomposing a space domain in millions of cells, on which multiple equations are applied, often involving data from their neighbor cells (e.g., stencil applications). Therefore, when the

difference between neighbor elements in a HPC dataset is unusually high, it could be due to data corruption.

Based on these observations, we define a threshold t_d to trigger corruption suspicion. Each dataset d of the application has its own threshold t_d. When the difference between two neighbor elements of a dataset is higher than the threshold t_d, then the detector triggers a corruption suspicion alert. The threshold t_d can be defined as *the historical maximum of the absolute value of the finite difference between neighbor points*. Given this definition, our lightweight detector consists of a data dynamics monitor, DADYMO that scrubs the different datasets at the application level during runtime and that issues an alert in the case of a suspected corruption every time the difference between two neighbor elements is higher than the threshold t_d of the particular dataset. As we can observe, the computational work performed inside DADYMO is limited to only one subtraction and one comparison per element. Such lightweight execution can be performed at high frequency without disturbing the application significantly. An application using DADYMO can quickly detect strange patterns or abnormal variations in the datasets and then trigger more complex data analysis to determine whether a corruption has occurred.

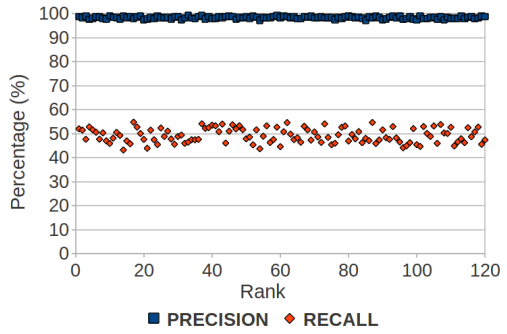

Figure 1. DADYMO Precision and Recall

3. Evaluation

In this section, we evaluate the accuracy of DADYMO with a synthetic benchmark. To evaluate our technique, we developed an error injector capable of making modifications in datasets at a binary level (i.e., bit flips). To verify that our injector works correctly, we overwrote the random bit position with 0 and verified that the corrupted values were just changing sign (i.e., the first bit in floating-point representation is the sign bit). We also performed extra verifications with known values and known bit locations to check that we were injecting errors correctly. We note that our injector assumes an equal probability of corruption among all bits.

3.1 DADYMO Performance and Accuracy

First, we set up an experiment using a synthetic benchmark of a heat distribution simulation executed on over 100 MPI ranks. Each rank holds a temperature dataset with values in the range of [260.0, 360.0] kelvin in single precision and a maximum variation of 1.0 kelvin between neighbor cells. This configuration guarantees that from one value to its neighbor, only a maximum of 16 bits (out of 32) changes. We note that in this experiment the finite difference between neighbor cells varies between 0.0 and 1.0 for all MPI processes. Then, during the execution we inject over a thousand bit flips on each rank for a total of over $170,000$ injected bit flips. During the whole execution, we execute DADYMO at high frequency in order to detect SDC.

We first want to prove that DADYMO is a lightweight detector that imposes no (or low) overhead on the application. Thus, we measure its performance. The performance metric for DADYMO is the amount of data analyzed per second (i.e., the throughput). We measure the throughput of DADYMO for each rank and for each time DADYMO is executed trough the simulation. We observed that all the processes have the same throughput (700 MB/s). We emphasize that 700 MB/s is the DADYMO throughput per MPI process. Thus, in a node with 15 MPI ranks (our system had 16 cores per node, but we used only 15) the total node-level DADYMO throughput is 10.5 GB/s. The size of the dataset for each rank was 256 MBs; thus each DADYMO execution took just a fraction of a second, causing less than 1% overhead.

After proving that DADYMO is a lightweight detector, we evaluate the accuracy of its detections. Thus, we count the number of detected bit flips and for each detection check whether the detected bit flip corresponds to an injection or not, so that we get the number of true detections and false detections. In this way we are able to get the precision and recall for each rank. We plot the results in Figure 2. As we can see, all processes have around 50% of recall and 98% of precision. These results mean that our technique learns quickly about the data *dynamics* and makes few false positives (high precision). We conclude from these results that although DADYMO does not guarantee corruption-free data, it can substantially reduce the SDC perceived by the application.

4. Related work

A large literature exists on soft errors rates [1, 4] and detection and correction techniques. Most of these techniques are implemented at the hardware level. Most of these strategies, however, target soft errors in general and do not focus on the particular case of SDC in parallel scientific applications. For some specific linear algebra algorithms, one of the most promising techniques against SDC is algorithm-based fault tolerance (ABFT). ABFT [2] is a technique that uses extra checksums to correct errors. However, ABFT has been implemented only on linear algebra kernels.

5. Conclusions

In this work we have highlighted some of the properties of HPC datasets, such as low variations for close elements. Based on this property, we proposed to set a threshold t_d for each dataset that indicates the *maximum allowed* finite difference between neighbor elements. We evaluated our proposed scheme with synthetic benchmarks and our results show that our technique can detect over 50% of silent corruptions (prediction recall) with a precision a nearly 100% for a *negligible overhead*.

References

[1] Shekhar Borkar. Designing reliable systems from unreliable components: The challenges of transistor variability and degradation. *IEEE Micro*, 25:10–16, November 2005.

[2] Kuang-Hua Huang and Jacob A. Abraham. Algorithm-based fault tolerance for matrix operations. *Computers, IEEE Transactions on*, 100(6):518–528, 1984.

[3] Dong Li, Jeffrey S Vetter, and Weikuan Yu. Classifying soft error vulnerabilities in extreme-scale scientific applications using a binary instrumentation tool. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 57. IEEE Computer Society Press, 2012.

[4] Tezzaron Semiconductor. Soft errors in electronic memory-a white paper, 2004.

This work was supported by the U. S. Department of Energy, Office of Science, under Contract No. DE-AC02-06CH11357.