J. Parallel Distrib. Comput. 98 (2016) 8–24

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Coping with recall and precision of soft error detectors☆

Leonardo Bautista-Gomez [a], Anne Benoit [b], Aurélien Cavelan [b], Saurabh K. Raina [c], Yves Robert [b,d], Hongyang Sun [b,*]

[a] Argonne National Laboratory, USA
[b] Ecole Normale Superieure de Lyon & INRIA, France
[c] Jaypee Institute of Information Technology, India
[d] University of Tennessee Knoxville, USA

HIGHLIGHTS

- Resilience algorithms to cope with silent errors for HPC applications.
- Characterization of optimal patterns using partial error detectors.
- Imprecise detectors offer limited usefulness.
- Optimization problem is NP-complete with multiple detector types.
- Construction of an FPTAS and a greedy approximation algorithm.

ARTICLE INFO

Article history:
Received 18 December 2015
Received in revised form
6 July 2016
Accepted 22 July 2016
Available online 29 July 2016

Keywords:
Fault tolerance
High-performance computing
Silent data corruption
Partial verification
Recall and precision
Exascale

ABSTRACT

Many methods are available to detect silent errors in high-performance computing (HPC) applications. Each method comes with a cost, a recall (fraction of all errors that are actually detected, i.e., false negatives), and a precision (fraction of true errors amongst all detected errors, i.e., false positives). The main contribution of this paper is to characterize the optimal computing pattern for an application: which detector(s) to use, how many detectors of each type to use, together with the length of the work segment that precedes each of them. We first prove that detectors with imperfect precisions offer limited usefulness. Then we focus on detectors with perfect precision, and we conduct a comprehensive complexity analysis of this optimization problem, showing NP-completeness and designing an FPTAS (Fully Polynomial-Time Approximation Scheme). On the practical side, we provide a greedy algorithm, whose performance is shown to be close to the optimal for a realistic set of evaluation scenarios. Extensive simulations illustrate the usefulness of detectors with false negatives, which are available at a lower cost than the guaranteed detectors.

1. Introduction

Failures in high-performance computing (HPC) systems have become a major issue as the number of components proliferates. Indeed, future exascale platforms are expected to be composed of hundreds of thousands of computing nodes [25]. Even if each individual node provides an optimistic mean time between failures (MTBF) of, say 100 years, the whole platform will experience a failure around every few hours on average, which is shorter than the execution time of most HPC applications. Thus, effective resilient protocols will be essential to achieve efficiency.

The de-facto general-purpose error recovery technique in HPC is checkpointing and rollback recovery [18,29]. Such protocols employ checkpoints to periodically save the state of a parallel application so that when an error strikes some process, the application can be restored to one of its former states. However, checkpoint/restart assumes instantaneous error detection, and therefore applies to fail-stop errors. Silent errors, a.k.a. silent data corruptions (SDC), constitute another source of failures in HPC, whose threat can no longer be ignored [38,42,36]. There are several causes of silent errors, such as cosmic radiation, packaging pollution, among others. In contrast to a fail-stop error whose

☆ A preliminary version of this paper has appeared in the Proceedings of the IEEE International Conference on High Performance Computing, December 2015.

* Corresponding author.
E-mail address: hongyang.sun@ens-lyon.fr (H. Sun).

http://dx.doi.org/10.1016/j.jpdc.2016.07.007
0743-7315/

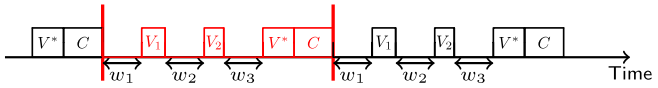

Fig. 1. A periodic pattern (highlighted in red) with three segments, two partial verifications and a verified checkpoint. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

detection is immediate, a silent error is identified only when the corrupted data leads to an unusual application behavior. Such detection latency raises a new challenge: if the error struck before the last checkpoint, and is detected after that checkpoint, then the checkpoint is corrupted and cannot be used for rollback.

In order to avoid corrupted checkpoints, an effective approach consists in employing some verification mechanism and combining it with checkpointing [19,39,1]. The simplest protocol with this approach would be to execute a verification procedure before taking each checkpoint. If the verification succeeds, then one can safely store the checkpoint. Otherwise, it means that an error has struck since the last checkpoint, which was duly verified, and one can safely recover from that checkpoint to resume the execution of the application. Of course, more sophisticated protocols can be designed, by coupling multiple verifications with one checkpoint, or interleaving multiple checkpoints and verifications [1,9]. The optimal parameter (e.g., number of verifications per checkpoint) in these protocols would be determined by the relative cost of executing a verification.

In practice, not all verification mechanisms are 100% accurate and at the same time admit fast implementations. In fact, guaranteeing accurate and efficient detection of silent errors for scientific applications is one of the hardest challenges towards extreme-scale computing [15,16]. Indeed, thorough and general-purpose error detection is usually very costly, and often involves expensive techniques, such as replication [30] or even triplication [35]. Many applications have developed specific verification mechanisms that leverage detailed knowledge of the physics behind the simulation to determine whether the output of a simulation is corruption-free or not. While such application-specific mechanisms do not detect the totality of SDC affecting the hardware, they can guarantee to detect all the corruptions relevant for the end user, thus they can be called arguably *perfect detectors* or *guaranteed verifications*, at least from the user's perspective. For many parallel applications, alternative techniques exist that are capable of detecting silent errors but with lower accuracy. We call these techniques *partial verifications*. One example is the lightweight SDC detector based on data dynamic monitoring [3], designed to recognize anomalies in HPC datasets based on physical laws and spatial interpolation. Similar fault filters have also been designed to detect silent errors based on time series predictions [11]. Although not completely accurate, these partial verification techniques nevertheless cover a substantial number of silent errors, and more importantly, they incur very low overheads. These properties make them attractive candidates for designing more efficient resilient protocols.

Since checkpointing is often expensive in terms of both time and space required, to avoid saving corrupted data, we only keep *verified checkpoints* by placing a guaranteed verification right before each checkpoint. Such a combination ensures that the checkpoint contains valid data and can be safely written onto stable storage. The execution of the application is partitioned into *periodic patterns*, i.e., computational chunks that repeat over time, and that are delimited by verified checkpoints, possibly with a sequence of partial verifications in between. Fig. 1 shows a periodic pattern with two partial verifications followed by a verified checkpoint.

The error detection accuracy of a partial verification can be characterized by two parameters: recall and precision. The *recall*, denoted by r, is the ratio between the number of detected errors and the total number of errors that occurred during a computation. The *precision*, denoted by p, is the ratio between the number of true errors and the total number of errors detected by the verification. For example, a basic spatial based SDC detector [3] has been shown to have a recall value around 0.5 and a precision value very close to 1, which means that it is capable of detecting half of the errors with almost no false alarm. A guaranteed verification can be considered as a special type of partial verification with recall $r^* = 1$ and precision $p^* = 1$. Each partial verification also has an associated *cost* V, which is typically much smaller than the cost V^* of a guaranteed verification. Note that precision and recall are conflicting objectives as they both are directly related to the allowed prediction error of the detector. If the prediction error is too small, then small changes in data behavior will produce false positives. On the other hand, if the allowed prediction error is too large, important corruption could be absorbed in the error corrupting the execution. Thus, one usually sets a target for one of them (e.g., precision $= 0.999$) and then measures the recall obtained with such a level of precision. Therefore, although it is hard to know in advance the precision and recall of a given detector for a particular application, it is possible to set a target for either one, and then quickly measure the complementary parameter.

An application can use several types of detectors with different overheads and accuracies. For instance, to detect silent errors in HPC datasets, one has the option of using either a detector based on time series prediction [11], or a detector using spatial multivariate interpolation [3]. The first one needs more data to make a prediction, hence comes at a higher cost. However, its accuracy is also better. In the example of Fig. 1, the second verification may use a detector whose cost is lower than that of the first one, i.e., $V_2 < V_1$, but is expected to have a lower accuracy as well, i.e., $r_2 < r_1$ and/or $p_2 < p_1$. This is due to the fact that less accurate detectors perform a much simpler approximation, leading to more prediction errors.

In this paper, we assume that we have several detector types, whose costs and accuracies may differ. At the end of each segment inside the pattern, any detector can be used. The only constraint is to enforce a guaranteed verification after the last segment. Given the values of C (cost to checkpoint) and V^* (cost of guaranteed verification), as well as the cost $V^{(j)}$, recall $r^{(j)}$ and precision $p^{(j)}$ of each detector type $D^{(j)}$, the main question is which detector(s) to use? Note that we do not assume that all detectors perform equally on all applications, nor that their efficiency can be easily predicted for each type of application. The only requirement is that the accuracy and cost of those detectors can be measured in a relatively easy way. The objective is to find the optimal pattern that minimizes the expected execution time of the application. Intuitively, including more partial verifications in a pattern allows us to detect more errors earlier in the execution, thereby reducing the waste due to re-execution; but that comes at the price of additional overhead in an error-free execution, and in case of bad precision, of unnecessary rollbacks and recoveries. Therefore, an optimal strategy must seek a good tradeoff between error-induced waste and error-free overhead. The problem is intrinsically combinatorial, because there are many parameters to choose: the length of the pattern, the number of partial verifications, and the type and location of each partial verification within the pattern. Of course, the length of an optimal pattern will also depend on the platform MTBF μ.

Only very specific instances of the problem have received a solution yet. For example, when there is a single segment in the pattern without intermediate verification, the only thing to determine is the size of the segment. In the classical protocol for fail-stop errors (where verification is not needed), the optimal checkpointing period is known to be $\sqrt{2\mu C}$ (where C