Which Verification for Soft Error Detection?

Leonardo Bautista-Gomez, Anne Benoit, Aurélien Cavelan, Saurabh K. Raina, Yves Robert, Hongyang Sun

Which Verification for Soft Error Detection?

Leonardo Bautista-Gomez*, Anne Benoit†, Aurélien Cavelan†,
Saurabh K. Raina‡, Yves Robert†§ and Hongyang Sun†
*Argonne National Laboratory, USA
†Ecole Normale Superieure de Lyon & INRIA, France
‡Jaypee Institute of Information Technology, India
§University of Tennessee Knoxville, USA

***Abstract*—Many methods are available to detect silent errors in high-performance computing (HPC) applications. Each comes with a given cost and recall (fraction of all errors that are actually detected). The main contribution of this paper is to characterize the optimal computational 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 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.**

***Index Terms*—Fault Tolerance; High-Performance Computing; Silent Data Corruption; Partial Verification; Supercomputer; Exascale.**

I. 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 [17]. 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 checkpoint and rollback recovery [13], [19]. 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 [29], [33], [27]. There are several causes of silent errors, such as cosmic radiation, packaging pollution, among others. In contrast to a fail-stop error whose 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 [14], [30], [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 even interleaving multiple checkpoints and verifications [1], [7]. The optimal pattern (i.e., number of verifications per checkpoint) in these protocols would be determined by the 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 in extreme-scale computing [3]. Indeed, thorough error detection is usually very costly, and often involves expensive techniques, such as replication [20] or even triplication [26]. For many parallel applications, alternative techniques exist that are capable of detecting some but not all errors. We call these techniques *partial verifications*, while a *guaranteed verification* is capable of detecting all errors. 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 [9]. Although not completely accurate, these partial verification techniques nevertheless cover a substantial amount of silent errors, and more importantly, they incur very low overhead. 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 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. Figure 1 shows a periodic pattern with two partial verifications followed

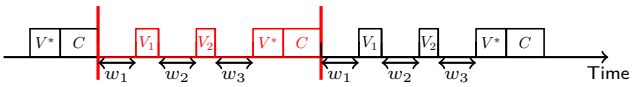

Figure 1. A periodic pattern (highlighted in red) with three segments, two partial verifications and a verified checkpoint.

by a verified checkpoint.

The error detection accuracy of a partial verification can be characterized by its *recall* r, which is the ratio between the number of detected errors and the total number of errors that occurred during a computation. For example, a basic spatial based SDC detector has been shown to have a recall around 0.5 measured on synthetic benchmarks [3], which means that it is capable of detecting half of the errors. Note that a guaranteed verification can be considered as a special type of partial verification with a recall $r^* = 1$. Each partial verification also has an associated *cost* V, which is typically much smaller than the cost V^* of a guaranteed verification.

An application can use several types of detectors with different overheads. For instance, to detect silent errors in HPC datasets, one has the option of using either the detector based on time series prediction [9], or the detector using spatial multivariate interpolation [3]. The first one needs more data to make a prediction, hence comes at a higher cost. However, its recall is also better. In the example of Figure 1, the second verification may use a detector whose cost is lower than that of the first one ($V_2 < V_1$), but is expected to have a lower recall as well ($r_2 < r_1$).

In this paper, we assume to have several detector types, whose costs and recalls 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), V^* (cost of guaranteed verification), the cost $V^{(j)}$ and recall $r^{(j)}$ of detector type $D^{(j)}$, the main question is which detector(s) to use? 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, and 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. 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 instance, 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 is the checkpoint time), as given by Young [32] and Daly [16]. A similar result is known for silent errors when using only verified checkpoints [7], [6]: in that case, the optimal period is $\sqrt{\mu(C+V^*)}$. These formulas provide first-order approximations to the length of the optimal pattern in the corresponding scenario, and are valid only if the resilience parameters satisfy $C, V^* \ll \mu$. To the best of our knowledge, the only analysis that includes partial verifications is the recent work [12], which deals with patterns that may include one or several detector(s), but all of the same type. While most applications accept several detector types, there has been no attempt to determine which and how many of these detectors should be used. This paper is the first to investigate the use of different types of partial verifications, i.e., different detectors.

We prove that this optimization problem is NP-complete. We show that a detector is most useful when it offers a high *accuracy-to-cost ratio*, defined as $\phi^{(j)} = \frac{r^{(j)}}{2-r^{(j)}} / \frac{V^{(j)}}{V^*+C}$. We then propose a greedy algorithm and a fully polynomial-time approximation scheme (FPTAS) to solve the problem. Simulation results, based on a wide range of parameters from realistic detectors, corroborate the theoretical study by showing that the detector with the best accuracy-to-cost ratio should be favored. In some particular cases with close accuracy-to-cost ratios, an optimal pattern may use two different detectors. Finally, the greedy algorithm has been shown to work quite well in practice.

The rest of this paper is organized as follows. Section II surveys related work. Section III introduces the model, notations and assumptions. Section IV presents key properties of optimal patterns. Section V provides a comprehensive complexity analysis. While the optimization problem is shown to be NP-complete, a simple greedy algorithm is presented, and a fully polynomial-time approximation scheme is described. Simulation results are presented in Section VI. Finally, Section VII provides concluding remarks and hints for future directions.

II. Related Work

Considerable efforts have been directed at detection techniques to reveal silent errors. Hardware mechanisms, such as ECC memory, can detect and even correct a fraction of errors. Unfortunately, future extreme scale systems are expected to observe an important increase in soft errors because of power constraints at increased system size. Most traditional resilient approaches maintain a single checkpoint. If the checkpoint file contains corrupted data, the application faces an irrecoverable failure and must restart from scratch. This is because error detection latency is ignored in traditional rollback and recovery schemes, which assume instantaneous error detection (therefore mainly targeting fail-stop errors) and are unable to accommodate SDC. This section describes some related work on detecting and handling silent errors.

A. Checkpoint versioning

One approach to dealing with silent errors is by maintaining several checkpoints in memory [25]. This multiple-checkpoint approach, however, has three major drawbacks. First, it is very demanding in terms of stable storage: each checkpoint typically represents a copy of a large portion of the memory footprint of the application, which may

well correspond to tens or even hundreds of terabytes. Second, the application cannot be recovered from fatal failures: suppose we keep k checkpoints in memory, and a silent error has struck before all of them. Then, all live checkpoints are corrupted, and one would have to re-execute the entire application from scratch. Third, even without memory constraints, we have to determine which checkpoint is the last valid one, which is needed to safely recover the application. However, due to the detection latency, we do not know when the silent error has occurred, hence we cannot identify the last valid checkpoint.

B. Process replication

There are few methods that can guarantee a perfect detection recall. Process replication is one of them. The simplest technique is triple modular redundancy and voting [26]. Elliot et al. [18] propose combining partial redundancy and checkpointing, and confirm the benefit of dual and triple redundancy. Fiala et al. [20] apply process replication (each process is equipped with a replica, and messages are quadruplicated) in the RedMPI library for high-performance scientific applications. Ni et al. [28] use checkpointing and replication to detect and enable fast recovery of applications from both silent errors and hard errors. However, full process replication is too expensive to be used in extreme scale HPC systems and is usually avoided for this reason.

C. Application-specific techniques

Application-specific information can be very useful to enable ad-hoc solutions, which dramatically decrease the cost of detection. Algorithm-based fault tolerance (ABFT) [23], [10], [31] is a well-known technique, which uses checksums to detect up to a certain number of errors in linear algebra kernels. Unfortunately, ABFT can only protect datasets in linear algebra kernels and it has to be implemented for each different kernel, which incurs a large amount of work for large HPC applications. Other techniques have also been advocated. Benson, Schmit and Schreiber [8] compare the result of a higher-order scheme with that of a lower-order one to detect errors in the numerical analysis of ODEs and PDEs. Sao and Vuduc [30] investigate self-stabilizing corrections after error detection in the conjugate gradient method. Heroux and Hoemmen [22] design a fault-tolerant GMRES capable of converging despite silent errors, and Bronevetsky and de Supinski [11] provide a comparative study of detection costs for iterative methods.

D. Analytics-based corruption detection

Recently, several SDC detectors based on data analytics have been proposed, showing promising results. These detectors use several interpolation techniques such as time series prediction [9] and spatial multivariate interpolation [3], [4], [5]. Such techniques have the benefit of offering large detection coverage for a negligible overhead. However, these detectors do not guarantee full coverage; they can detect only a certain percentage of corruptions (i.e., partial verification). Nonetheless, the accuracy-to-cost ratios of these detectors are high, which makes them interesting alternatives at large scale. Most of the research work done in this domain focuses on how to increase the error detection accuracy while keeping low overhead, but there has been no theoretical attempt to find the optimal protocol the applications should use when multiple verification techniques are offered by the runtime.

E. Optimal strategies with guaranteed verifications

Theoretically, various protocols that couple verification and checkpointing have been studied. Aupy et al. [1] propose and analyze two simple patterns: one with k checkpoints and one verification, and the other with k verifications and one checkpoint. The latter pattern, which needs to maintain only one checkpoint, is also analyzed in [6] to accommodate both fail-stop and silent errors. Benoit et al. [7] extend the analysis of [1] by including p checkpoints and q verifications that are interleaved to form arbitrary patterns. All of these results assume the use of guaranteed verifications only.

As already mentioned, the only analysis that includes partial verifications in the pattern is the recent work of [12]. However, [12] restricts to a single type of partial verification. In this paper, we provide the first theoretical analysis that includes partial verifications of different types.

III. Model

We consider divisible-load applications, where checkpoints and verifications can be inserted anywhere in the execution of the application. The occurrence of silent errors follows a Poisson process with arrival rate $\lambda = 1/\mu$, where μ denotes the MTBF of the platform.

We enforce resilience through the use of a *pattern* that repeats periodically throughout the execution, as discussed in Section I. When an error is detected inside a pattern, either by a partial verification or by the guaranteed verification, we roll back to the beginning of the pattern and recover from the last checkpoint (taken at the end of the execution of the previous pattern, or initial data for the first pattern). Since the last verification of the pattern is guaranteed, we need to maintain only one checkpoint at any time, and it is always valid. The objective is to find a pattern that minimizes the expected execution time of the application.

Let C denote the cost of checkpointing, R the cost of recovery and V^* the cost of guaranteed verification. Furthermore, there are k types of detectors available, and the detector type $D^{(j)}$, where $1 \leq j \leq k$, is characterized by its cost $V^{(j)}$ and recall $r^{(j)}$. For convenience, we also define $g^{(j)} = 1 - r^{(j)}$ (proportion of undetected errors) and let D^* be the guaranteed detector with cost V^* and recall $r^* = 1$.

A pattern $\textsc{Pattern}(W, n, \boldsymbol{\alpha}, \mathbf{D})$ is defined by its total length W, the number n of segments in the pattern, a vector $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, \ldots, \alpha_n]^T$ containing the proportions of the segment sizes, and a vector $\mathbf{D} = [D_1, D_2, \ldots, D_{n-1}, D^*]^T$ containing the detectors used at the end of each segment. We also define the vector of segment sizes $\mathbf{w} = [w_1, w_2, \ldots, w_n]^T$. Formally, for each segment i, where $1 \leq i \leq n$, w_i is the size of the segment, $\alpha_i = \frac{w_i}{W}$ is

the proportion of the segment size in the whole pattern, and D_i is the detector used at the end of the segment. We have $\sum_{i=1}^{n} \alpha_i = 1$, and $\sum_{i=1}^{n} w_i = W$. If $i < n$, D_i has cost V_i and recall r_i (we have $D_i = D^{(j)}$ for some type j, $1 \le j \le k$), and $D_n = D^*$ of cost V^* and recall $r^* = 1$. Note that the same detector type $D^{(j)}$ may well be used at the end of several segments. For notational convenience, we let $g_i = 1 - r_i$ be the probability that the i-th detector of the pattern fails to detect an error (for $1 \le i < n$), and $g_{[i,j[} = \prod_{k=i}^{j-1} g_k$ be the probability that the error remains undetected by the detectors D_i to D_{j-1} (for $1 \le i < j < n$). In the example of Figure 1, we have $W = w_1 + w_2 + w_3$ and $n = 3$. The first partial verification has cost V_1 with recall r_1, and the second one has cost V_2 with recall r_2.

Let W_{base} denote the base time of an application without any overhead due to resilience techniques (without loss of generality, we assume unit-speed execution). Suppose the execution is divided into periodic patterns, defined by $\textsc{Pattern}(W, n, \boldsymbol{\alpha}, \mathbf{D})$. Let $\mathbb{E}(W)$ be the expected execution time of the pattern. Then, the expected makespan W_{final} of the application when taking silent errors into account can be bounded as follows:

$$\left\lfloor \frac{W_{\text{base}}}{W} \right\rfloor \times \mathbb{E}(W) \le W_{\text{final}} \le \left\lceil \frac{W_{\text{base}}}{W} \right\rceil \times \mathbb{E}(W).$$

This is because the execution involves $\lfloor \frac{W_{\text{base}}}{W} \rfloor$ full patterns, and terminates by a (possibly) incomplete one. For large jobs, we approximate the execution time as

$$W_{\text{final}} \approx \frac{\mathbb{E}(W)}{W} \times W_{\text{base}}.$$

Let $H(W) = \frac{\mathbb{E}(W)}{W} - 1$ denote the execution *overhead* of the pattern. We obtain $W_{\text{final}} \approx W_{\text{base}} + H(W) \times W_{\text{base}}$. Thus, minimizing the expected makespan is equivalent to minimizing the pattern overhead $H(W)$.

We assume that errors only strike the computations, while verifications and I/O transfers (checkpointing and recovery) are protected and are thus error-free.

IV. Properties of Optimal Patterns

In this section, we first show how to compute the expected execution time of any given pattern (Section IV-A), and then we derive a closed-form formula that is exact up to second-order terms. Based on this key result, we are able to characterize the optimal length of a pattern (Section IV-B), and to compute the optimal positions of the partial verifications (Section IV-C).

A. Expected execution time of a pattern

Consider any given pattern $\textsc{Pattern}(W, n, \boldsymbol{\alpha}, \mathbf{D})$. The following proposition computes the expected execution time of this pattern.

Proposition 1. *The expected execution time to execute a pattern* $\textsc{Pattern}(W, n, \boldsymbol{\alpha}, \mathbf{D})$ *is*

$$\mathbb{E}(W) = W + \sum_{i=1}^{n} V_i + C + \lambda W (R + W \boldsymbol{\alpha}^T A \boldsymbol{\alpha} + \mathbf{d}^T \boldsymbol{\alpha}) + o(\lambda), \quad (1)$$

where A is an $n \times n$ symmetric matrix defined by $A_{ij} = \frac{1}{2}\left(1 + g_{[i,j[}\right)$ for $i \le j$, and $\mathbf{d}$ is an $n \times 1$ vector defined by $\mathbf{d}_i = \sum_{j=i}^{n} g_{[i,j[} V_i$ for $1 \le i \le n$.

Proof. Let q_i denote the probability that an error occurs in the execution of segment i. We can express the expected execution time of the pattern recursively as follows:

$$\mathbb{E}(W) = \left(\prod_{k=1}^{n}(1-q_k)\right) C + \left(1 - \prod_{k=1}^{n}(1-q_k)\right)(R + \mathbb{E}(W)) + \sum_{i=1}^{n}\left(\sum_{j=1}^{i-1}\left(\prod_{k=1}^{j-1}(1-q_k)\right) q_j g_{[j,i[} + \prod_{k=1}^{i-1}(1-q_k)\right)(w_i + V_i). \quad (2)$$

The first line shows that checkpointing will only be taken if no error has occurred in all the segments, which happens with probability $\prod_{k=1}^{n}(1-q_k)$, and in all the other cases, the application needs to recover from the last checkpoint and then re-computes the entire pattern. The second line shows the expected cost involved in the execution of each segment of the pattern and the associated verification. To better understand it, consider the third segment of size w_3 and the verification D_3 right after it, which will be executed only when the following events happen (with the probability of each event in brackets):

- There is a fault in the first segment (q_1), which is missed by the first verification ($1 - r_1 = g_1$) and again missed by the second verification ($1 - r_2 = g_2$).
- There is no fault in the first segment ($1 - q_1$), and there is a fault in the second segment (q_2), which is missed by the second verification ($1 - r_2 = g_2$).
- There is no fault in the first segment ($1 - q_1$) and no fault in the second segment ($1 - q_2$).

Thus, the expected cost involved in the execution of this segment is given by

$$\Big(q_1 g_1 g_2 + (1-q_1) q_2 g_2 + (1-q_1)(1-q_2)\Big)(w_3 + V_3) = \Big(q_1 g_{[1,3[} + (1-q_1) q_2 g_{[2,3[} + (1-q_1)(1-q_2)\Big)(w_3 + V_3).$$

We can generalize this reasoning to express the expected cost to execute the i-th segment of the pattern, which leads to Equation (2).

Since errors arrive according to the Poisson process, by definition, we have $q_i = 1 - e^{-\lambda w_i}$. Substituting it into the recursive formula and solving for $\mathbb{E}(W)$, we obtain the expected execution time as

$$\mathbb{E}(W) = C + (e^{\lambda W} - 1)R + \sum_{i=1}^{n}\left(\sum_{j=1}^{i-1}(e^{\lambda W_{j,n}} - e^{\lambda W_{j+1,n}}) g_{[j,i[} + e^{\lambda W_{i,n}}\right)(w_i + V_i),$$

where $W_{i,j} = \sum_{k=i}^{j} w_k$. Approximating $e^{\lambda x} = 1 + \lambda x + o(\lambda)$ to the first-order term, we can further simplify the expected

execution time as

$$\begin{aligned}\mathbb{E}(W) &= C + \lambda WR + o(\lambda) \\ &\quad + \sum_{i=1}^{n} \left(\sum_{j=1}^{i-1} \lambda w_j g_{[j,i[} + 1 + \lambda \sum_{j=i}^{n} w_j \right) (w_i + V_i) \\ &= W + \lambda WR + \sum_{i=1}^{n} V_i + C + o(\lambda) \\ &\quad + \lambda \sum_{i=1}^{n} \left(\sum_{j=1}^{i-1} w_j g_{[j,i[} + \sum_{j=i}^{n} w_j \right) (w_i + V_i).\end{aligned}$$

Letting $F = \sum_{i=1}^{n} \left(\sum_{j=1}^{i-1} w_j g_{[j,i[} + \sum_{j=i}^{n} w_j \right) (w_i + V_i)$, we have the following matrix form:

$$F = \mathbf{w}^T M \mathbf{w} + \mathbf{d}^T \mathbf{w},$$

where M is the following $n \times n$ matrix

$$M = \begin{bmatrix} 1 & g_{[1,2[} & g_{[1,3[} & \cdots & g_{[1,n[} \\ 1 & 1 & g_{[2,3[} & \cdots & g_{[2,n[} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & 1 & \dots & 1 \end{bmatrix}.$$

For instance, when $n = 4$ we have

$$M = \begin{bmatrix} 1 & g_1 & g_1 g_2 & g_1 g_2 g_3 \\ 1 & 1 & g_2 & g_2 g_3 \\ 1 & 1 & 1 & g_3 \\ 1 & 1 & 1 & 1 \end{bmatrix}.$$

Replacing M by $A = \frac{M + M^T}{2}$ gives the same value for F, and we obtain the following symmetric matrix

$$A = \frac{1}{2} \begin{bmatrix} 2 & 1 + g_{[1,2[} & \dots & 1 + g_{[1,n[} \\ 1 + g_{[1,2[} & 2 & \dots & 1 + g_{[2,n[} \\ \vdots & \vdots & \ddots & \vdots \\ 1 + g_{[1,n[} & 1 + g_{[2,n[} & \dots & 2 \end{bmatrix}.$$

Now, by using $\mathbf{w} = W\boldsymbol{\alpha}$, we obtain Equation (1). This completes the proof of the proposition. □

B. Optimal length of a pattern to minimize the overhead

In this section, we compute the overhead $H(W)$ of a pattern $\text{PATTERN}(W, n, \boldsymbol{\alpha}, \mathbf{D})$ and show how to compute its length W to minimize this overhead. We introduce two key parameters.

Definition 1. *Consider a pattern* $\text{PATTERN}(W, n, \boldsymbol{\alpha}, \mathbf{D})$. *The* fault-free overhead o_{ff} *of the pattern is*

$$o_{ff} = \sum_{i=1}^{n} V_i + C, \tag{3}$$

and the fraction of re-executed work in case of faults is

$$f_{re} = \boldsymbol{\alpha}^T A \boldsymbol{\alpha}. \tag{4}$$

Proposition 2. *The execution overhead of a pattern* $\text{PATTERN}(W, n, \boldsymbol{\alpha}, \mathbf{D})$ *is minimized when its length is*

$$W^* = \sqrt{\frac{o_{ff}}{\lambda f_{re}}}. \tag{5}$$

In that case, we have

$$H(W^*) = 2\sqrt{\lambda o_{ff} f_{re}} + o(\sqrt{\lambda}). \tag{6}$$

Proof. From Equation (1), we directly derive

$$H(W) = \frac{\mathbb{E}(W)}{W} - 1 = \lambda R + \frac{o_{\text{ff}}}{W} + \lambda f_{\text{re}} W + \lambda \mathbf{d}^T \boldsymbol{\alpha} + o(\lambda).$$

The optimal pattern length that minimizes the execution overhead can now be computed from the above equation, and it is given by Equation (5).

Substituting W^* back into $H(W)$, the execution overhead is given by

$$\begin{aligned} H(W^*) &= 2\sqrt{\lambda o_{\text{ff}} f_{\text{re}}} + \lambda(R + \mathbf{d}^T \boldsymbol{\alpha}) + o(\lambda) \\ &= 2\sqrt{\lambda o_{\text{ff}} f_{\text{re}}} + o(\sqrt{\lambda}). \end{aligned}$$

This completes the proof of the proposition. □

When the platform MTBF $\mu = 1/\lambda$ is large in front of the resilience parameters, Equation (6) shows that the expected execution overhead of the optimal pattern is dominated by $2\sqrt{\lambda o_{\text{ff}} f_{\text{re}}}$, and the last term becomes negligible. The problem is then reduced to the minimization of the product $o_{\text{ff}} f_{\text{re}}$. Intuitively, this calls for a tradeoff between fault-free overhead and fault-induced re-execution, as a smaller fault-free overhead o_{ff} tends to induce a larger re-execution fraction f_{re}, and vice versa.

C. Optimal positions of verifications to minimize f_{re}

To fully characterize an optimal pattern, we have to determine its number of segments, and the type and position of each partial verification. In this section, we consider a pattern whose number of segments is given, as well as the type of all partial verifications, that is, the value of o_{ff} (see Equation (3)) is given. We show how to determine the optimal length of each segment (or equivalently, the optimal position of each verification), so as to minimize the value of f_{re} (see Equation (4)). The result is the most technically involved contribution of this paper, and its lengthy proof is available in the companion research report [2].

Theorem 1. *Consider a pattern* $\text{PATTERN}(W, n, \boldsymbol{\alpha}, \mathbf{D})$ *where* W, n, *and* $\mathbf{D}$ *are given. The fraction of re-executed work* f_{re} *is minimized when* $\boldsymbol{\alpha} = \boldsymbol{\alpha}^*$, *where*

$$\alpha_k^* = \frac{1}{U_n} \times \frac{1 - g_{k-1} g_k}{(1 + g_{k-1})(1 + g_k)} \quad \text{for } 1 \le k \le n, \tag{7}$$

with $g_0 = g_n = 0$ *and*

$$U_n = 1 + \sum_{i=1}^{n-1} \frac{1 - g_i}{1 + g_i}. \tag{8}$$

In that case, the value of f_{re} *is*

$$f_{re}^* = \frac{1}{2} \left(1 + \frac{1}{U_n} \right). \tag{9}$$

Proof sketch. The goal is to minimize $f_{\text{re}} = \boldsymbol{\alpha}^T A \boldsymbol{\alpha}$ subject to the constraint $\mathbf{c}^T \boldsymbol{\alpha} = 1$, where $\mathbf{c} = [1, 1, \cdots, 1]^T$. We can show the following properties (proofs are omitted; see [2] for the details):

- A is symmetric positive definite (SPD);
- $\boldsymbol{\alpha}^*$ is a valid solution, i.e., $\mathbf{c}^T\boldsymbol{\alpha}^* = 1$;
- $A\boldsymbol{\alpha}^* = f^*_{\text{re}}\mathbf{c}$.

From the fact that A is SPD, we can derive a unique optimal solution $f^{\text{opt}}_{\text{re}} = \frac{1}{\mathbf{c}^T A^{-1}\mathbf{c}}$ to the optimization problem obtained at $\boldsymbol{\alpha}^{\text{opt}} = \frac{A^{-1}\mathbf{c}}{\mathbf{c}^T A^{-1}\mathbf{c}}$. Since $A\boldsymbol{\alpha}^* = f^*_{\text{re}}\mathbf{c}$, we have $\boldsymbol{\alpha}^* = f^*_{\text{re}}A^{-1}\mathbf{c}$ and hence $1 = \mathbf{c}^T\boldsymbol{\alpha}^* = f^*_{\text{re}}(\mathbf{c}^T A^{-1}\mathbf{c})$. This leads to $f^{\text{opt}}_{\text{re}} = f^*_{\text{re}}$ and $\boldsymbol{\alpha}^{\text{opt}} = \boldsymbol{\alpha}^*$. □

When all the partial verifications in the pattern have the same type, i.e., $g_k = g$ for all $1 \le k < n$, we retrieve the result of [12], obtaining $f^*_{\text{re}} = \frac{1}{2}\left(1 + \frac{1+g}{n(1-g)+2g}\right)$ with

$$\alpha^*_k = \begin{cases} \frac{1}{n(1-g)+2g} & \text{for } k = 1, n \\ \frac{1-g}{n(1-g)+2g} & \text{for } k = 2, \ldots, n-1 \end{cases}.$$

Theorem 1 also shows that, for a given set of partial verifications in a pattern, the minimum value of f_{re} does not depend upon their ordering within the pattern.

Corollary 1. *For a given set of partial verifications within a pattern, the minimum fraction of re-executed work f^*_{re} is independent of their ordering.*

V. Complexity

This section builds upon the previous results to provide a comprehensive complexity analysis. We introduce the *accuracy-to-cost ratio* of a detector and show that it is the key parameter to compute the optimal rational solution (Section V-A). Then we establish the NP-completeness to determine the optimal integer solution (Section V-B). On the positive side, we design a simple greedy algorithm whose performance is guaranteed, and sketch the construction of an FPTAS for the problem (Section V-C).

A. Accuracy-to-cost ratio and rational solution

Consider a pattern Pattern$(W, n, \boldsymbol{\alpha}, \mathbf{D})$. Let m_j denote the number of partial verifications using detector type $D^{(j)}$ in the pattern (the number of indices $i < n$ such that D_i is of type $D^{(j)}$), and define $\mathbf{m} = [m_1, m_2, \ldots, m_k]$. Section IV-B shows that minimizing the execution overhead of the pattern is equivalent to minimizing the product $o_{\text{ff}} f_{\text{re}}$. From Equations (3) and (9), we have $o_{\text{ff}} f_{\text{re}} = \frac{V^*+C}{2} f(\mathbf{m})$, where

$$f(\mathbf{m}) = \left(1 + \frac{1}{1 + \sum_{j=1}^k m_j a^{(j)}}\right)\left(1 + \sum_{j=1}^k m_j b^{(j)}\right). \tag{10}$$

In Equation (10), we define $a^{(j)} = \frac{1-g^{(j)}}{1+g^{(j)}}$ to be the *accuracy* of detector $D^{(j)}$ and define $b^{(j)} = \frac{V^{(j)}}{V^*+C}$ to be the *relative cost* of $D^{(j)}$. Furthermore, we define $\phi^{(j)} = \frac{a^{(j)}}{b^{(j)}}$ to be the *accuracy-to-cost ratio* of $D^{(j)}$. We will show that this ratio plays a key role in selecting the best detector(s).

Altogether, minimizing the pattern overhead amounts to finding the solution $\mathbf{m} = [m_1, m_2, \ldots, m_k]$ that minimizes $f(\mathbf{m})$, with $m_j \in \mathbb{N}_0$ for all $1 \le j \le k$. Indeed, once $\mathbf{m}$ is given, Proposition 2 and Theorem 1 completely characterize the optimal pattern, giving its length W, the number of segments $n = \sum_{j=1}^k m_j + 1$, and the locations $\boldsymbol{\alpha}$ of all partial detectors (whose ordering does not matter).

We first derive the optimal solution if we relax the integer constraint on $\mathbf{m}$. A rational solution in this case is denoted by $\bar{\mathbf{m}} = [\bar{m}_1, \bar{m}_2, \ldots, \bar{m}_k]$ with $\bar{m}_j \ge 0$ for all $1 \le j \le k$. The optimal value of $f(\bar{\mathbf{m}})$ is a lower bound on the optimal integer solution.

Lemma 1. *Suppose there are k types of detectors sorted in non-increasing order of accuracy-to-cost ratio, i.e., $\phi^{(1)} \ge \phi^{(2)} \ge \cdots \ge \phi^{(k)}$. Then,*

$$f^*(\bar{\mathbf{m}}) = \begin{cases} \left(\sqrt{\frac{1}{\phi^{(1)}}} + \sqrt{1 - \frac{1}{\phi^{(1)}}}\right)^2 & \text{if } \phi^{(1)} > 2 \\ 2 & \text{otherwise} \end{cases}.$$

Proof. First, we prove that the optimal rational solution is achieved when only the detector with the largest accuracy-to-cost ratio $\phi^{(1)}$ is used. Specifically, given any rational solution $\bar{\mathbf{m}} = [\bar{m}_1, \bar{m}_2, \ldots, \bar{m}_k]$, we show that there exists a solution $\bar{\mathbf{m}}' = [\bar{m}'_1, 0, \ldots, 0]$, which satisfies $f(\bar{\mathbf{m}}') \le f(\bar{\mathbf{m}})$. We have

$$\begin{aligned} f(\bar{\mathbf{m}}) &= \left(1 + \frac{1}{1 + \sum_{j=1}^k \bar{m}_j a^{(j)}}\right)\left(1 + \sum_{j=1}^k \bar{m}_j b^{(j)}\right) \\ &= \left(1 + \frac{1}{1 + a^{(1)} \sum_{j=1}^k \frac{\bar{m}_j a^{(j)}}{a^{(1)}}}\right)\left(1 + b^{(1)} \sum_{j=1}^k \frac{\bar{m}_j b^{(j)}}{b^{(1)}}\right). \end{aligned} \tag{11}$$

Let $\bar{m}'_1 = \sum_{j=1}^k \frac{\bar{m}_j a^{(j)}}{a^{(1)}}$ and $\bar{n}'_1 = \sum_{j=1}^k \frac{\bar{m}_j b^{(j)}}{b^{(1)}}$. Since $\frac{b^{(j)}}{b^{(1)}} \ge \frac{a^{(j)}}{a^{(1)}}$ for all $1 \le j \le k$, we get $\bar{n}'_1 = \sum_{j=1}^k \frac{\bar{m}_j b^{(j)}}{b^{(1)}} \ge \sum_{j=1}^k \frac{\bar{m}_j a^{(j)}}{a^{(1)}} = \bar{m}'_1$. Hence, Equation (11) can be written as

$$\begin{aligned} f(\bar{\mathbf{m}}) &= \left(1 + \frac{1}{1 + a^{(1)}\bar{m}'_1}\right)\left(1 + b^{(1)}\bar{n}'_1\right) \\ &= \left(1 + \frac{1}{1 + a^{(1)}\bar{m}'_1}\right)\left(1 + \frac{b^{(1)}\bar{n}'_1}{\bar{m}'_1} \cdot \bar{m}'_1\right) \\ &\ge \left(1 + \frac{1}{1 + a^{(1)}\bar{m}'_1}\right)\left(1 + b^{(1)}\bar{m}'_1\right) \\ &= f(\bar{\mathbf{m}}'). \end{aligned}$$

Now, define $f(\bar{m}) = \left(1 + \frac{1}{1+a^{(1)}\bar{m}}\right)\left(1 + b^{(1)}\bar{m}\right)$. The following derives the minimum value of $f(\bar{m})$. Differentiating $f(\bar{m})$ with respect to $\bar{m}$ and solving $\frac{\partial f(\bar{m})}{\partial \bar{m}} = 0$, we get

$$\bar{m}^* = -\frac{1}{a^{(1)}} + \sqrt{\frac{1}{a^{(1)}}\left(\frac{1}{b^{(1)}} - \frac{1}{a^{(1)}}\right)}, \tag{12}$$

which is positive (hence a potential solution) if $\phi^{(1)} = \frac{a^{(1)}}{b^{(1)}} > 2$. Taking the second-order derivative of $f(\bar{m})$, we get

$$\frac{\partial^2 f(\bar{m})}{\partial \bar{m}^2} = \frac{2a^{(1)}(a^{(1)} - b^{(1)})}{(a^{(1)}\bar{m} + 1)^3},$$

which is positive (hence ensures that the solution is the unique minimum) for all $\bar{m} \in [0, \infty)$ if $\phi^{(1)} = \frac{a^{(1)}}{b^{(1)}} > 1$.

Thus, when $\phi^{(1)} > 2$, the optimal solution is obtained by substituting $\bar{m}^*$ into $f(\bar{m})$, and we get

$$\begin{aligned} f(\bar{m}^*) &= \left(1 + \frac{1}{1 + a^{(1)}\bar{m}^*}\right)\left(1 + b^{(1)}\bar{m}^*\right) \\ &= \left(1 + \frac{1}{\sqrt{\phi^{(1)} - 1}}\right)\left(1 - \frac{1}{\phi^{(1)}} + \sqrt{\frac{1}{\phi^{(1)}}\left(1 - \frac{1}{\phi^{(1)}}\right)}\right) \\ &= \left(\sqrt{\frac{1}{\phi^{(1)}}} + \sqrt{1 - \frac{1}{\phi^{(1)}}}\right)^2. \end{aligned}$$

When $\phi^{(1)} \leq 2$, the minimum value of $f(\bar{m})$ is achieved at $\bar{m} = 0$, which gives $f(0) = 2$. □

Lemma 1 shows that the optimal rational solution is achieved with only one detector, namely, the one with the highest accuracy-to-cost ratio. The optimal integer solution, however, may use more than one detector. The following shows that finding the optimal integer solution is NP-complete.

B. NP-completeness

We show that finding the optimal integer solution $\mathbf{m}$ is NP-complete, even when all detectors share the same accuracy-to-cost ratio. In particular, we consider the following decision problem.

Definition 2 (Multiple Partial Verifications (MPV))**.** *Given k detectors with the same accuracy-to-cost ratio ϕ, i.e., $\frac{a^{(j)}}{b^{(j)}} = \phi$ for all $1 \leq j \leq k$, and a bound K, is there a solution $\mathbf{m}$ that satisfies*

$$\left(1 + \frac{1}{1 + \sum_{j=1}^{k} m_j a^{(j)}}\right)\left(1 + \sum_{j=1}^{k} m_j b^{(j)}\right) \leq K? \quad (13)$$

Theorem 2. *The MPV problem is NP-complete.*

Proof. The MPV problem is obviously in NP. We prove the completeness by a reduction from the *Unbounded Subset Sum (USS)* problem, which is known to be NP-complete [21]. Given a multiset $S = \{s_1, s_2, \ldots, s_k\}$ of k positive integers and a positive integer I, the USS problem asks if there exists a subset $S' \subseteq S$ whose sum is exactly I, i.e., $\sum_{j=1}^{k} m_j s_j = I$, where $m_j \in \mathbb{N}_0$. We can further assume that I/s_j is not an integer for $1 \leq j \leq k$, since otherwise we would have a trivial solution.

Given an instance of the USS problem, we construct an instance of the MPV problem with k detectors. First, choose any $\phi \in \left(2, (I/s_{\max} + 1)^2 + 1\right)$, where $s_{\max} = \max_{j=1..k} s_j$. Then, let $\frac{a}{b} = \phi$ and $-\frac{1}{a} + \sqrt{\frac{1}{a}\left(\frac{1}{b} - \frac{1}{a}\right)} = I$, so we can get $a = \frac{\sqrt{\phi-1}-1}{I}$ and $b = \frac{\sqrt{\phi-1}-1}{\phi I}$. For each $1 \leq j \leq k$, define $a^{(j)} = s_j a$ and $b^{(j)} = s_j b$. According to the range of ϕ, we have $a^{(j)} < 1$ and $b^{(j)} < 1$ for all $1 \leq j \leq k$. Finally, let $K = \left(\sqrt{\frac{1}{\phi}} + \sqrt{1 - \frac{1}{\phi}}\right)^2$.

If we use only one detector, say $D^{(j)}$, then Lemma 1 shows that Equation (13) is satisfied with the following unique solution:

$$\begin{aligned} m_j^* &= -\frac{1}{a^{(j)}} + \sqrt{\frac{1}{a^{(j)}}\left(\frac{1}{b^{(j)}} - \frac{1}{a^{(j)}}\right)} \\ &= \frac{1}{s_j}\left(-\frac{1}{a} + \sqrt{\frac{1}{a}\left(\frac{1}{b} - \frac{1}{a}\right)}\right) = \frac{I}{s_j}, \end{aligned}$$

which is not an integer by hypothesis, but achieves the lower bound $\left(\sqrt{\frac{1}{\phi}} + \sqrt{1 - \frac{1}{\phi}}\right)^2 = K$. Now, we show that, by using multiple detectors, an integer solution to the MPV instance exists if and only if there is an integer solution to the USS instance.

($\Rightarrow$) Suppose there is an integer solution $\mathbf{m} = [m_1, m_2, \ldots, m_k]$ such that $\sum_{j=1}^{k} m_j s_j = I$. Then, by employing m_j partial verifications of detector type $D^{(j)}$ for $1 \leq j \leq k$, we get

$$\begin{aligned} &\left(1 + \frac{1}{1 + \sum_{j=1}^{k} m_j a^{(j)}}\right)\left(1 + \sum_{j=1}^{k} m_j b^{(j)}\right) \\ &= \left(1 + \frac{1}{1 + a\sum_{j=1}^{k} m_j s_j}\right)\left(1 + b\sum_{j=1}^{k} m_j s_j\right) \\ &= \left(1 + \frac{1}{1 + aI}\right)(1 + bI) = \left(\sqrt{\frac{1}{\phi}} + \sqrt{1 - \frac{1}{\phi}}\right)^2 = K. \end{aligned}$$

($\Leftarrow$) Suppose there is an integer solution $\mathbf{m} = [m_1, m_2, \ldots, m_k]$ to the MPV instance such that

$$\left(1 + \frac{1}{1 + \sum_{j=1}^{k} m_j a^{(j)}}\right)\left(1 + \sum_{j=1}^{k} m_j b^{(j)}\right) = K.$$

This implies

$$\begin{aligned} &\left(1 + \frac{1}{1 + a\sum_{j=1}^{k} m_j s_j}\right)\left(1 + b\sum_{j=1}^{k} m_j s_j\right) \\ &= 1 + 2\sqrt{\frac{1}{\phi}\left(1 - \frac{1}{\phi}\right)}. \end{aligned}$$

Let $T = \sum_{j=1}^{k} m_j s_j$. Solving T from the equation above, we get the following unique solution:

$$T = -\frac{1}{a} + \sqrt{\frac{1}{a}\left(\frac{1}{b} - \frac{1}{a}\right)} = I.$$

This completes the proof of the theorem. □

C. Greedy algorithm and FPTAS

To cope with the NP-completeness of minimizing $o_{\text{ff}} f_{\text{re}}$, there is a simple and intuitive greedy algorithm. This greedy algorithm uses only the detector with the highest accuracy-to-cost ratio $\phi^{(1)}$. We compute the optimal rational number of partial verifications $\bar{m}^*$ and then round it up if it is not an integer. In Section VI, we show that this algorithm performs quite well in practice.

Interestingly, we can guarantee the performance of this simple algorithm. From Lemma 1, we can assume

$\phi^{(1)} = \frac{a^{(1)}}{b^{(1)}} > 2$. Since $a^{(1)} < 1$, we can get $b^{(1)} < 1/2$. If the optimal fractional solution $\bar{m}^*$ given in Equation (12) happens to be an integer, then we get the optimal solution. Otherwise, rounding it to $\lceil \bar{m}^* \rceil$ increases the objective function $f(\mathbf{m})$ shown in Equation (10) by at most a factor of $\delta = 1 + b^{(1)} < 3/2$. According to Equation (6), this gives a $\sqrt{3/2}$-approximation algorithm for minimizing the expected execution overhead (and hence the makespan).

In the following, we show that it is possible to have a fully polynomial-time approximation scheme (FPTAS), which ensures, for any $\epsilon > 0$, that the solution is within $1+\epsilon$ times the optimal, and that the running time of the algorithm is polynomial in the input size and $1/\epsilon$. To develop the FPTAS, we perform the following transformations to the problem.

First, we convert all parameters in Equation (10) to integers. Since $a^{(j)} = \frac{1-g^{(j)}}{1+g^{(j)}} = \frac{r^{(j)}}{2-r^{(j)}} \leq 1$ and $r^{(j)}$ is rational, we can write $a^{(j)} = \frac{p_j}{q_j}$, where p_j and q_j are positive integers with $p_j \leq q_j$. We assume that C, V^* and all the $V^{(j)}$'s are also integers. Thus, minimizing $f(\mathbf{m})$ is equivalent to minimizing the following function:

$$F(\mathbf{m}) = \left(1 + \frac{L}{L + \sum_{j=1}^{k} m_j L^{(j)}}\right)\left(C + V^* + \sum_{j=1}^{k} m_j V^{(j)}\right),$$

where L denotes the least common multiple (LCM) of $q_1, q_2, \ldots, q_k$, and $L^{(j)} = \frac{p_j}{q_j} L \leq L$. Clearly, L and all the $L^{(j)}$'s can be represented by a polynomial function of the original input size.

Next, we compute an upper bound on the number of partial verifications. Observe that $F(\mathbf{0}) = 2(C + V^*)$ and $F(\mathbf{m}) \geq C + V^* + \sum_{j=1}^{k} m_j V^{(j)}$. This implies that the optimal solution must satisfy $m_j \leq \frac{C+V^*}{V^{(j)}}$ for all $1 \leq j \leq k$. Therefore, it follows that $\sum_{j=1}^{k} m_j V^{(j)} \leq k(C + V^*)$. The bound on m_j allows us to transform the unbounded problem to the 0-1 problem by providing $\lfloor \log m_j \rfloor$ additional copies of each item type j with doubling $V^{(j)}$ and $L^{(j)}$ values. This is a standard technique also used in transforming the bounded and unbounded knapsack problems to the 0-1 knapsack problem [24]. The total number of items becomes $K = \sum_{j=1}^{k} (1 + \lfloor \log m_j \rfloor) = O(k \log(C + V^*))$, which stays polynomial in the input size.

Define $\mathbf{x} = [x_1, x_2, \ldots, x_K]$, and let L_j and V_j be the value and cost of item j, respectively. We can now formulate the optimization problem as follows:

$$\text{minimize } F(\mathbf{x}) = \left(1 + \frac{L}{L + \sum_{j=1}^{K} x_j L_j}\right)\left(C + V^* + \sum_{j=1}^{K} x_j V_j\right)$$

$$\text{subject to} \sum_{j=1}^{K} x_j V_j \leq k(C + V^*)$$

$$x_j \in \{0, 1\} \quad \forall j = 1, 2, \ldots, K$$

and the size of all parameters is a polynomial function of the input size of the original problem. To find an FPTAS for the problem above, we adopt the technique used in [15] for designing an FPTAS for the *Maximum Density Knapsack (MDK)* problem described below.

Maximum Density Knapsack (MDK): Given a set $S = \{s_1, s_2, \ldots, s_K\}$ of K items, where each item $s_j \in S$ has a positive integer profit p_j and a positive integer weight w_j, a total capacity W, and an initial weight w_0, the MDK problem is formulated as:

$$\text{maximize } \frac{\sum_{j=1}^{K} x_j p_j}{w_0 + \sum_{j=1}^{K} x_j w_j}$$

$$\text{subject to} \sum_{j=1}^{K} x_j w_j \leq W$$

$$x_j \in \{0, 1\} \quad \forall j = 1, 2, \ldots, K$$

Cohen and Katzir [15] give an FPTAS for the MDK problem by using the existing FPTAS for the knapsack problem [24]. In particular, their algorithm relies on the property that, for every profit P, a minimum weight solution $\mathbf{x}$ is found such that $P(\mathbf{x}) = \sum_{j=1}^{K} x_j p_j \geq \lfloor \frac{P}{1+\epsilon'} \rfloor$, for any $\epsilon' > 0$. This immediately gives rise to an FPTAS for MDK.

We can apply the same technique to construct an FPTAS for minimizing $F(\mathbf{x})$. Let $\mathbf{x}_{opt}$ denote the optimal solution. By considering V_j as weight and L_j as profit, we can run the FPTAS for knapsack and return in polynomial time a solution $\mathbf{x}$ that satisfies $P(\mathbf{x}) \geq \lfloor \frac{P(\mathbf{x}_{opt})}{1+\epsilon'} \rfloor$ and $W(\mathbf{x}) \leq W(\mathbf{x}_{opt})$. By setting carefully the value of ϵ' as a function of ϵ, the solution yields $F(\mathbf{x}) \leq (1 + \epsilon) F(\mathbf{x}_{opt})$. The detail is similar to the one presented in [15] and is omitted here.

VI. Simulations

Simulations are conducted to evaluate the performance improvement that partial verifications can provide. Unlike with a single detector type, the number of different patterns to consider, while searching for the optimal one, grows exponentially with multiple detector types (e.g., there are 64 different patterns with three partial verifications of four types). Since Corollary 1 shows that the ordering of partial verifications does not matter, only unique patterns need to be considered to find the optimal one (20 different patterns remain in the above example). We implement the optimal algorithms using a single detector type as well as multiple detector types in Maple, and conduct simulations with realistic detector values. The expected overhead of the algorithm employing partial verifications is compared with that of the baseline algorithm, which uses only a single guaranteed verification. The reduction in overhead illustrates the usefulness of partial verifications. Section VI-A describes the experimental setup, including the scenarios and the range of parameter values. Results are presented in Section VI-B by exploring two problem instances.

A. Simulation framework

This section provides information about the parameters used for instantiating the performance model. We have chosen realistic parameters that depict a typical expected exascale platform. The target platform consists of 10^5 nodes whose individual MTBF is 100 years, which amounts to a platform MTBF of $\mu = 31536$ seconds (i.e., about

8.7 hours). The global size of the memory for an exascale machine is expected to be between 32 PB and 64 PB; divided by the number of nodes (10^5), the memory size per node goes from 320 GB to 640 GB. Most HPC applications try to populate 90% of the node memory but only $10\% - 50\%$ of the memory is checkpointed. That makes the checkpoint size between 30 GB and 300 GB. At exascale, most checkpoints will be done in local non-volatile memory (NVM), which is known to be slower than DRAM. We assume checkpoint throughput between 0.5 GB/s and 1 GB/s.

Concerning the SDC detectors, we assume a first detector $D^{(1)}$ with a throughput of about 200 MB/s/process and a recall of 0.5 [3], [5]. The second one $D^{(2)}$ has a throughput of about 20 MB/s/process and a recall of 0.95 [9]. If we assume 512 processes per node at exascale, then the node throughput of the detectors becomes 100 GB/s for $D^{(1)}$ and 10 GB/s for $D^{(2)}$. Finally, we assume a third detector $D^{(3)}$, which is an optimized version that combines the features of the first two detectors, achieving a recall of 0.8 and a throughput of 50 GB/s. Concerning the perfect detector D^*, we assume a throughput of 0.5 GB/s based on the fact that application-specific detectors are usually based on physical properties such as mass or energy conservation, which requires global communications and is therefore more expensive than purely local checks.

In the first instance, we evaluate our model with three detectors $D^{(1)}$, $D^{(2)}$ and $D^{(3)}$ (with respective costs and recall values $V^{(1)} = 3$ seconds, $V^{(2)} = 30$ seconds, $V^{(3)} = 6$ seconds and $r^{(1)} = 0.5$, $r^{(2)} = 0.95$, $r^{(3)} = 0.8$). The checkpointing cost and the perfect detector cost (with recall $r^* = 1$) are fixed at $C = V^* = 600$ seconds. We compute the expected overhead for each detector as a function of number of partial verifications (m from 0 to up to 35). We then assess the performance of a detector and its effect on the execution overhead.

The second instance targets applications with various datasets that expose a change in detection recall [5], [4]. Therefore, it assumes a range of the recall for each detector rather than a single value: $r^{(1)} = [0.5, 0.9]$, $r^{(2)} = [0.75, 0.95]$, and $r^{(3)} = [0.8, 0.99]$. Given a dataset, we obtain a value of recall for each detector, within the interval. This is because different datasets might expose different levels of entropy and therefore the detectors might expose different prediciton accuracy, hence different recall. We note that although the recall might be different for different datasets, the work done, hence the detection cost, is the same. We compare the greedy algorithm to the optimal solution for different datasets, keeping the same values for C, V^* and μ.

B. Results and analysis

Based on the above framework, we report the simulation results highlighting the observed overheads, and the improvements achieved by employing partial verifications.

1) Performance of detectors and impact of m: Figure 2 shows the performance of the three detectors $D^{(1)}$, $D^{(2)}$ and $D^{(3)}$, and assesses the impact of the number m of partial verifications on the execution overhead. The

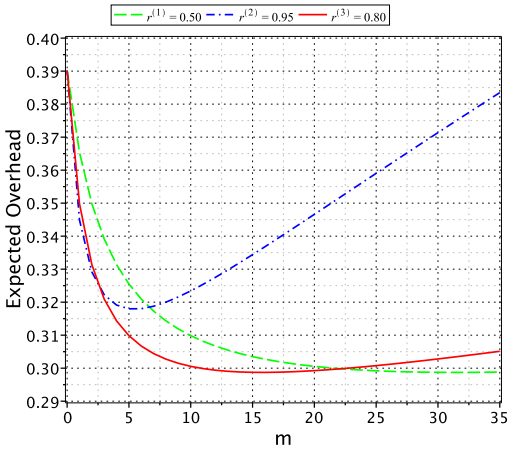

Figure 2. Expected overhead against the number of partial verifications when a single type of detector is employed for three different detectors (with their respective cost and recall values). In the simulation, $C = V^* = 600$, and $\mu = 31536$.

plot also shows the overhead of the baseline algorithm, which is approximately 39%, represented by $m = 0$. The expected overhead is reduced for all three detectors by employing partial verifications. For each detector, the optimal overhead is attained for a particular value of m, corroborating the theoretical study. After this point, it starts rising again due to the fact that forcing too many verifications will eventually lead the error-free overhead to rise. The improvement in overhead over the baseline algorithm is 9% for detectors $D^{(1)}$ and $D^{(3)}$ (optimal overhead for both is $\approx 30\%$), and 7% for detector $D^{(3)}$ (optimal overhead is $\approx 32\%$). Concerning the performance of detectors, we can see that $D^{(1)}$ and $D^{(3)}$ are slightly better than $D^{(2)}$, due to their higher accuracy-to-cost ratios. However, for $m \leq 2$, $D^{(2)}$ is better due to its higher recall, while its performance degrades as more $D^{(2)}$ detectors are employed due to its high cost.

2) Performance of the greedy algorithm: In this second instance, the recall of each detector is within a recall interval, and its value depends on the dataset. Even with the intervals, because of the high cost, $D^{(2)}$ always has a lower accuracy-to-cost ratio when compared to $D^{(1)}$ and $D^{(3)}$, which share similar ratios. We show that the greedy algorithm presented in Section V-C performs extremely well in practice, even though the optimal pattern

Table I
PERFORMANCE COMPARISON OF GREEDY ALGORITHM AND OPTIMAL SOLUTION. IN ALL SCENARIOS, $C = V^* = 600$, $V^{(1)} = 3$, $V^{(3)} = 6$.

	m	overhead H	diff. from opt.
Scenario 1: $r^{(1)} = 0.51$, $r^{(3)} = 0.82$, $\phi^{(1)} \approx 137$, $\phi^{(3)} \approx 139$			
Optimal solution	(1, 15)	29.828%	0%
Greedy with $D^{(3)}$	(0, 16)	29.829%	0.001%
Scenario 2: $r^{(1)} = 0.58$, $r^{(3)} = 0.9$, $\phi^{(1)} \approx 163$, $\phi^{(3)} \approx 164$			
Optimal solution	(1, 14)	29.659%	0%
Greedy with $D^{(3)}$	(0, 15)	29.661%	0.002%
Scenario 3: $r^{(1)} = 0.64$, $r^{(3)} = 0.97$, $\phi^{(1)} \approx 188$, $\phi^{(3)} \approx 188$			
Optimal solution	(1, 13)	29.523%	0%
Greedy with $D^{(1)}$	(27, 0)	29.524%	0.001%
Greedy with $D^{(3)}$	(0, 14)	29.525%	0.002%

may employ both $D^{(1)}$ and $D^{(3)}$ in the solution.

Indeed, we identify three scenarios where a combination of $D^{(1)}$ and $D^{(3)}$ constitutes the optimal pattern, as presented in Table I. In all of them, the greedy algorithm, which uses only the detector with the highest accuracy-to-cost ratio, performs within 0.002% of the optimal solution.

VII. Conclusion and Future Work

In this paper, we provided a comprehensive analysis of a computing pattern that employs different types of partial verifications for detecting silent errors in HPC applications. We showed that the optimization problem is NP-complete in general, and proposed a greedy algorithm as well as an FPTAS for choosing the number of detectors to be used, as well as their types and locations in the pattern. Simulations based on realistic detector settings show that the greedy algorithm works well in practice, and confirm their usefulness in tackling SDCs on exascale systems.

In future work, we will investigate the impact of false positives on the performance of partial verification patterns. False positives are characterized by the *precision* value, which is the number of true errors over the total number of errors raised by a detector. In many partial detectors, a tradeoff exists between the recall and precision by adjusting the detection parameters. Analyzing a pattern in the presence of both false positives and false negatives will refine the assessment of the usefulness of partial detectors.

Acknowledgment

This research was funded in part by the European project SCoRPiO, by the LABEX MILYON (ANR-10-LABX-0070) of Université de Lyon, within the program "Investissements d'Avenir" (ANR-11-IDEX-0007) operated by the French National Research Agency (ANR), by the PIA ELCI project, by the ANR Rescue project, by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research Program, under Contract DE-AC02-06CH11357; by the INRIA-Illinois-ANL-BSC Joint Laboratory on Extreme Scale Computing, and Center for Exascale Simulation of Advanced Reactors at Argonne. Yves Robert is with Institut Universitaire de France.

References

[1] G. Aupy, A. Benoit, T. Hérault, Y. Robert, F. Vivien, and D. Zaidouni. On the combination of silent error detection and checkpointing. In *Proceedings of PRDC*, pages 11–20, 2013.

[2] L. Bautista-Gomez, A. Benoit, A. Cavelan, S. K. Raina, Y. Robert, and H. Sun. Which verification for soft error detection? Research report RR-8741, INRIA, June 2015.

[3] L. Bautista Gomez and F. Cappello. Detecting silent data corruption through data dynamic monitoring for scientific applications. In *Proceedings of PPoPP'14*, pages 381–382, New York, NY, USA, 2014. ACM.

[4] L. Bautista Gomez and F. Cappello. Detecting and correcting data corruption in stencil applications through multivariate interpolation. In *Proceedings of FTS'15*, 2015.

[5] L. Bautista Gomez and F. Cappello. Exploiting Spatial Smoothness in HPC Applications to Detect Silent Data Corruption. In *Proceedings of HPCC'15*, 2015.

[6] A. Benoit, A. Cavelan, Y. Robert, and H. Sun. Assessing general-purpose algorithms to cope with fail-stop and silent errors. In *Proceedings of PMBS, held as part of SC'14*, 2014.

[7] A. Benoit, Y. Robert, and S. K. Raina. Efficient checkpoint/verification patterns. *Int. J. High Performance Computing Applications*, DOI: 10.1177/1094342015594531, Published online, July 2015. Available as ICL Research report RR-1403.

[8] A. R. Benson, S. Schmit, and R. Schreiber. Silent error detection in numerical time-stepping schemes. *Int. J. High Performance Computing Applications*, DOI: 10.1177/1094342014532297, 2014.

[9] 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 *Proc. of HPDC'15*, 2015.

[10] G. Bosilca, R. Delmas, J. Dongarra, and J. Langou. Algorithm-based fault tolerance applied to high performance computing. *J. Parallel Distrib. Comput.*, 69(4):410–416, 2009.

[11] G. Bronevetsky and B. de Supinski. Soft error vulnerability of iterative linear algebra methods. In *Proceedings of ICS*, pages 155–164, 2008.

[12] A. Cavelan, S. K. Raina, Y. Robert, and H. Sun. Assessing the impact of partial verifications against silent data corruptions. In *Proceedings of the 44th Annual International Conference on Parallel Processing (ICPP)*, 2015.

[13] K. M. Chandy and L. Lamport. Distributed snapshots: Determining global states of distributed systems. *ACM Transactions on Computer Systems*, 3(1):63–75, 1985.

[14] Z. Chen. Online-ABFT: An online algorithm based fault tolerance scheme for soft error detection in iterative methods. In *Proceedings of PPoPP'13*, pages 167–176, 2013.

[15] R. Cohen and L. Katzir. The generalized maximum coverage problem. *Inf. Process. Lett.*, 108(1):15–22, 2008.

[16] J. T. Daly. A higher order estimate of the optimum checkpoint interval for restart dumps. *Future Generation Comp. Syst.*, 22(3):303–312, 2006.

[17] J. Dongarra, P. Beckman, P. Aerts, F. Cappello, T. Lippert, S. Matsuoka, P. Messina, T. Moore, R. Stevens, A. Trefethen, and M. Valero. The international exascale software project: a call to cooperative action by the global high-performance community. *IJHPCA*, 23(4):309–322, 2009.

[18] J. Elliott, K. Kharbas, D. Fiala, F. Mueller, K. Ferreira, and C. Engelmann. Combining partial redundancy and checkpointing for HPC. In *Proceedings of ICDCS'12*, pages 615–626, 2012.

[19] E. N. M. Elnozahy, L. Alvisi, Y.-M. Wang, and D. B. Johnson. A survey of rollback-recovery protocols in message-passing systems. *ACM Computing Survey*, 34:375–408, 2002.

[20] 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 *Proceedings of SC'12*, page 78, 2012.

[21] M. R. Garey and D. S. Johnson. *Computers and Intractability, a Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.

[22] M. Heroux and M. Hoemmen. Fault-tolerant iterative methods via selective reliability. Research report SAND2011-3915 C, Sandia National Laboratories, 2011.

[23] K.-H. Huang and J. A. Abraham. Algorithm-based fault tolerance for matrix operations. *IEEE Trans. Comput.*, 33(6):518–528, 1984.

[24] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer, 2004.

[25] G. Lu, Z. Zheng, and A. A. Chien. When is multi-version checkpointing needed? In *Proceedings of FTXS'13*, pages 49–56, 2013.

[26] R. E. Lyons and W. Vanderkulk. The use of triple-modular redundancy to improve computer reliability. *IBM J. Res. Dev.*, 6(2):200–209, 1962.

[27] A. Moody, G. Bronevetsky, K. Mohror, and B. R. d. Supinski. Design, Modeling, and Evaluation of a Scalable Multi-level Checkpointing System. In *Proceedings of SC'10*, 2010.

[28] X. Ni, E. Meneses, N. Jain, and L. V. Kalé. ACR: Automatic Checkpoint/Restart for Soft and Hard Error Protection. In *Proceedings of SC'13*. ACM, 2013.

[29] T. O'Gorman. The effect of cosmic rays on the soft error rate of a DRAM at ground level. *IEEE Trans. Electron Devices*, 41(4):553–557, 1994.

[30] P. Sao and R. Vuduc. Self-stabilizing iterative solvers. In *Proceedings of the Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA)*, 2013.

[31] M. Shantharam, S. Srinivasmurthy, and P. Raghavan. Fault tolerant preconditioned conjugate gradient for sparse linear system solution. In *Proceedings of ICS'12*, pages 69–78, 2012.

[32] J. W. Young. A first order approximation to the optimum checkpoint interval. *Comm. of the ACM*, 17(9):530–531, 1974.

[33] J. F. Ziegler, H. W. Curtis, H. P. Muhlfeld, C. J. Montrose, and B. Chin. IBM experiments in soft fails in computer electronics. *IBM J. Res. Dev.*, 40(1):3–18, 1996.