Assessing the Impact of Partial Verifications Against Silent Data Corruptions

Aurélien Cavelan*, Saurabh K. Raina†, Yves Robert*‡, Hongyang Sun*
*Ecole Normale Superieure de Lyon & INRIA, France
†Jaypee Institute of Information Technology, India
‡University of Tennessee Knoxville, USA

***Abstract*—Silent errors, or silent data corruptions, constitute a major threat on very large scale platforms. When a silent error strikes, it is not detected immediately but only after some delay, which prevents the use of pure periodic checkpointing approaches devised for fail-stop errors. Instead, checkpointing must be coupled with some verification mechanism to guarantee that corrupted data will never be written into the checkpoint file. Such a guaranteed verification mechanism typically incurs a high cost. In this paper, we assess the impact of using partial verification mechanisms in addition to a guaranteed verification. The main objective is to investigate to which extent it is worthwhile to use some light cost but less accurate verifications in the middle of a periodic computing pattern, which ends with a guaranteed verification right before each checkpoint. Introducing partial verifications dramatically complicates the analysis, but we are able to analytically determine the optimal computing pattern (up to the first-order approximation), including the optimal length of the pattern, the optimal number of partial verifications, as well as their optimal positions inside the pattern. Performance evaluations based on a wide range of parameters confirm the benefit of using partial verifications under certain scenarios, when compared to the baseline algorithm that uses only guaranteed verifications.**

***Index Terms*—resilience; silent error; silent data corruption; partial/guaranteed verification; checkpoint; recall; optimal pattern.**

I. INTRODUCTION

As the number of components proliferate in High-Performance Computing (HPC) systems, resilience has become a major issue. Future exascale platforms are expected to be composed of one million computing nodes [12]. 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 50 minutes on average, which is shorter than the execution time of many HPC applications. Failures will become part of the norm when computing at scale, and effective resilient protocols will be the key to achieving sustained performance.

The de-facto general-purpose error recovery technique in HPC is checkpoint and rollback recovery [8], [15]. 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 and rollback recovery assumes instantaneous error detection, and therefore applies to fail-stop errors. Silent errors (a.k.a. silent data corruptions) constitute another source of error in HPC, whose threat can no longer be ignored [24], [28], [22]. The cause of silent errors could be soft faults in L1 cache or multiple bit flips due to cosmic radiation. In contrast to a fail-stop error whose detection is immediate, a silent error is identified only when the corrupted data is activated and/or 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 to restore the application.

One approach to dealing with silent errors is by maintaining several checkpoints in memory [20]. 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 the entire memory footprint of the application, which may well correspond to several terabytes. Second, the application cannot be recovered from fatal failures: suppose we keep k checkpoints in memory, and a silent error 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 from. However, due to the detection latency, we do not know when the silent error has occurred, hence we cannot identify the last valid checkpoint.

An arguably more effective approach is by employing some verification mechanism and combining it with checkpointing [9], [25], [1]. The simplest protocol with this approach would be to execute a verification procedure just 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 we can safely recover from that checkpoint to resume the execution of the application. This simple protocol eliminates the drawbacks of the multiple-checkpoint approach, provided that a guaranteed verification mechanism can be efficiently implemented. Of course, one can also design more sophisticated protocols by coupling multiple verifications with one checkpoint or even interleaving multiple checkpoints and verifications [1], [4]. The optimal pattern (i.e., number of verifications per checkpoint) in these protocols would be determined by the relative cost of executing a verification compared to checkpointing.

In practice, not all verification mechanisms are 100% accurate and at the same time admit fast implementations. In fact, to guarantee the accurate and efficient detection of silent errors for scientific applications is one of the hardest challenges in extreme-scale computing [2]. Indeed, thorough error detection is usually very costly and often involves expensive techniques, such as replication [16] or even triplication [21]. For many parallel applications, alternative techniques exist that are capable of detecting some but not all errors. We call these techniques *partial verifications*. One example is the data dynamic monitor (DADYMO) [2], which is a lightweight silent error detector designed to recognize anomalies in HPC datasets based on physical laws and spatial correlations. A similar fault filter has also been designed to detect silent errors in the temperature data of the Orbital Thermal Imaging Spectrometer (OTIS) [10]. Although not completely accurate, these partial verification techniques nevertheless cover a substantial amount of silent data corruptions, and more importantly, they incur very low overhead. These properties make them attractive candidates for designing more efficient resilient protocols.

The objective of this paper is to assess the potential benefits of using partial verifications against silent errors. The error detection accuracy of a partial verification is characterized by its *recall* r, which is the ratio between the number of detected errors and the total number of errors occurred during a computation. For example, the DADYMO tool has been shown to have a recall around 50% measured on synthetic scientific benchmarks, with negligible overhead [2]. 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 that of a guaranteed verification.

The problem under study can then be stated as follows: given the costs of checkpointing C and guaranteed verification V^* for an application, as well as the recall r and cost V of a partial verification, what is the *optimal pattern* that minimizes the expected execution time? As checkpointing is usually more expensive in terms of both time and space required, to avoid the risk of saving corrupted data, we only keep *verified checkpoints* by placing a guaranteed verification right before each checkpoint. Hence, a pattern refers to a work segment that repeats over time, and that is delimited by verified checkpoints, possibly with a sequence of partial verifications in between. Figure 1 shows a periodic pattern with two partial verifications followed by a verified checkpoint.

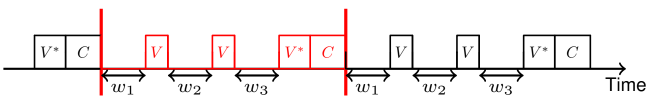


Figure 1: A periodic pattern (highlighted in red) with two partial verifications and a verified checkpoint.

Intuitively, including more partial verifications in a pattern increases the error detecting probability, thus reduces the waste due to re-executions, 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. Of course, the length of a pattern should also depend on the platform MTBF μ. For example, in the classical protocol for fail-stop errors where verification is not needed, the optimal checkpointing period is known to be $\sqrt{2\mu C}$ as given by Young [27] and Daly [11]. A similar result is also known for silent errors, and the optimal period in that case is $\sqrt{\mu(C + V^*)}$ if only verified checkpoints are used [4], [3]. These formulas provide first-order approximations to the optimal patterns in the respective scenarios, and are valid when the resilient parameters satisfy $C, V^* \ll \mu$.

In this paper, we focus on the design of resilient protocols for silent errors while embracing partial verifications. Given the values of C, V^*, V and r, and when the platform MTBF μ is large in front of these parameters, we derive an optimal pattern that characterizes: (1) the optimal length of the pattern; (2) the optimal number of partial verifications in the pattern; and (3) the optimal position of each partial verification inside the pattern. Furthermore, we determine the optimal configuration of a partial verification when its cost and recall can be traded off with each other. These results provide important extensions to the classical formulas in the field [27], [11], [4], [3], and to the best of our knowledge, are the first to include partial verifications. Unlike in the classical case, however, a silent error may not be detected by a partial verification and could get propagated to the subsequent work segments inside a pattern, thus significantly complicating the analysis. Our evaluation results based on a wide range of parameters also demonstrate that employing partial verifications indeed lead to performance gains compared to the baseline algorithm that relies only on guaranteed verifications.

The rest of this paper is organized as follows. Section II surveys the related work. Section III introduces the model, notations and assumptions. Section IV derives the optimal pattern using partial verifications and verified checkpoints. Performance evaluations are presented in Section V. Finally, Section VI provides concluding remarks and hints for future directions.

II. Related Work

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 silent errors. This section describes some related work on detecting and handling silent errors. A more comprehensive list of techniques and references is provided by Lu, Zheng and Chien [20].

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, but in practice they are complemented with software techniques. Guaranteed error detection is very costly, and usually involves expensive redundancy or replication techniques. The simplest technique is triple modular redundancy and voting [21]. Elliot et al. [14] propose combining partial redundancy and checkpointing, and confirm the benefit of dual and triple redundancy. Fiala et al. [16] 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. [23] use checkpointing and replication to detect and enable fast recovery of applications from both silent errors and hard errors.

Application-specific information can be very useful to enable ad-hoc solutions, which dramatically decrease the cost of detection. Besides the fault filtering technique applied in DADYMO [2] and OTIS [10] as mentioned previously, algorithm-based fault tolerance (ABFT) [18], [6], [26] is another well-known technique, which uses checksums to detect up to a certain number of errors in linear algebra kernels. Other techniques have also been advocated. Benson, Schmit and Schreiber [5] 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 [25] investigate self-stabilizing corrections after error detection in the conjugate gradient method. Heroux and Hoemmen [17] design a fault-tolerant GMRES capable of converging despite silent errors, and Bronevetsky and de Supinski [7] provide a comparative study of detection costs for iterative methods.

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 1 verification, and the other with k verifications and 1 checkpoint. The latter pattern, which needs to maintain only one checkpoint, is also analyzed in [3] to accommodate both fail-stop and silent errors. Benoit, Raina and Robert [4] 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. In this paper, we provide the first theoretical analysis that includes partial verifications.

III. MODEL

In this section, we state the problem and the assumptions, and present an analytical model for assessing the performance of a pattern.

A. Problem Statement

Consider the execution of a parallel application on a platform subject to silent errors. Let r denote the *recall* of a verification, which is defined as the fraction of detected errors over the total number of errors. We distinguish a *partial* verification with recall $r < 1$ and the *guaranteed* verification with recall $r = 1$. Then $1 - r$ is the probability that an error remains undetected by the partial verification, in which case the execution of the application goes on and reaches the next partial verification, and possibly further on, eventually leading to a rollback and recovery from the last valid checkpoint.

In this paper, we focus on the divisible-load application model, where checkpoints and verifications can be inserted at any point in execution of the application. We enforce resilience through the use of a periodic pattern as shown in Figure 1. A set of partial verifications can be placed at arbitrary locations within the pattern, but the pattern should always end with a *verified checkpoint*, that is, a guaranteed verification followed immediately by a checkpoint. This is to ensure that only one checkpoint needs to be maintained and that it is always valid, thereby ruling out the risk of a fatal failure.

Let C denote the cost of checkpointing, R the cost of recovery, V^* the cost of the guaranteed verification and V the cost of the partial verification with recall r. The objective is to find an optimal pattern that minimizes the expected execution time (or makespan) of the application. In particular, the optimal pattern should specify:

- The length of the pattern;
- The number of partial verifications in the pattern;
- The positions of partial verifications inside the pattern.

B. Assumptions

We assume that the platform MTBF μ is large in front of the resilience parameters C, R, V and V^*. This assumption is commonly made in the literature (see, e.g., [27], [11], [1], [3], [4]), which allows to make first-order approximations in the analysis to obtain close-form solutions.

As in [1], [4], we assume that the length W of a pattern also satisfies $W \ll \mu$, thus implicitly ruling out the possibility that more than one error could occur in the same period, including the re-executions. Moreover, the error distribution is assumed to be uniform inside a pattern, which is again a consequence of the first-order approximation to the Poisson process typically used to model the occurrence of failures.

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

C. Analytical Model

Suppose a pattern with work W contains m partial verifications. Thus, the total length of the pattern is $S = W + o_{\text{ff}}$, where $o_{\text{ff}} = mV + V^* + C$ is the overhead in a fault-free execution. The pattern is divided into $n = m + 1$ segments, each followed by a partial verification, except the last one, which is followed by a guaranteed verification.

Let T_{base} denote the base time of the application without any overhead due to resilience techniques (without loss of generality, assume unit-speed execution). First, imagine a fault-free execution: for every pattern of length S, only W units of work get executed, so the execution time T_{ff} in a fault-free execution is given by $T_{\text{ff}} = \frac{S}{W} T_{\text{base}}$.

Now, let T_{final} denote the expected execution time (or makespan) of the application when silent errors are taken into account. On average, errors occur every μ time units, where μ denotes the platform MTBF. For each error, suppose $\mathcal{F}$ time units are lost on average (where $\mathcal{F}$ will be computed later). Based on the first-order assumption, we expect $\frac{T_{\text{base}}}{\mu}$ errors during the entire execution of the application. Therefore, we derive that

$$\begin{aligned} T_{\text{final}} &= T_{\text{ff}} + \frac{T_{\text{base}}}{\mu}\mathcal{F} = \left(\frac{S}{W} + \frac{\mathcal{F}}{\mu}\right) T_{\text{base}} \\ &= \left(1 + \frac{o_{\text{ff}}}{W} + \frac{\mathcal{F}}{\mu}\right) T_{\text{base}} \ . \end{aligned} \tag{1}$$

It remains to determine $\mathcal{F}$, the expected time loss due to each failure. The value of $\mathcal{F}$ depends on the pattern used and it includes three components: re-executing a fraction of the total work W of the pattern, recovering from the last checkpoint, and re-executing some of the verifications in the pattern. Hence, the general form of $\mathcal{F}$ can be expressed as $\mathcal{F} = f_{\text{re}}W + R + \beta$, where f_{re} denotes the expected fraction of work that is re-executed, and β is a linear combination of V and V^*. Plugging the expression of $\mathcal{F}$ back into Equation (1), we get

$$T_{\text{final}} = \left(\frac{o_{\text{ff}}}{W} + \frac{f_{\text{re}}}{\mu}W + 1 + \frac{R+\beta}{\mu}\right) T_{\text{base}} \ . \tag{2}$$

For a given pattern, the optimal work length that minimizes T_{final} can then be computed from Equation (2) as

$$W^* = \sqrt{\mu \cdot \frac{o_{\text{ff}}}{f_{\text{re}}}} \ , \tag{3}$$

and the optimal period is $S = W^* + o_{\text{ff}}$. The expectation of the optimal *execution overhead* H can be expressed as

$$H = \frac{T^*_{\text{final}} - T_{\text{base}}}{T_{\text{base}}} = 2\sqrt{\frac{o_{\text{ff}} f_{\text{re}}}{\mu}} + \frac{R+\beta}{\mu} \ .$$

When the platform MTBF μ is large in front of all resilience parameters, we can identify the dominant term in the above expression. Indeed, in that case, the value $R + \beta$ becomes negligible in front of μ, and we have

$$H = 2\sqrt{o_{\text{ff}} f_{\text{re}}}\sqrt{\frac{1}{\mu}} + o(\sqrt{\frac{1}{\mu}}) \ . \tag{4}$$

Equation (4) shows that the optimal pattern when μ is large is obtained when the product $o_{\text{ff}} f_{\text{re}}$ is minimized. This calls for a tradeoff, as a smaller value of o_{ff} with fewer verifications leads to a larger re-execution time, thus a larger value of f_{re}. In the next section, we show how to compute f_{re} for a given pattern, and use this characterization in terms of the product $o_{\text{ff}} f_{\text{re}}$ to derive the optimal pattern.

IV. Optimal Pattern With Partial Verifications

We present the optimal pattern in this section. First, we compute the value of f_{re} for a give pattern (Section IV-A). Then, we derive the optimal positions of partial verifications (Section IV-B), followed by the optimal number of them in a pattern (Section IV-C). After that, we determine the optimal tradeoff between the cost and recall of a partial verification (Section IV-D). We end this section with an example illustrating the benefit of using partial verifications (Section IV-E).

A. Computing f_{re}

Consider a pattern with W work, m partial verifications and a verified checkpoint. The pattern is divided into $n = m + 1$ segments, whose work sizes are denoted by $w_1, w_2, \ldots, w_n$. For each segment i, we define $\alpha_i = w_i/W$ to be the fraction of its work in the pattern. Hence, we have $\sum_{i=1}^n \alpha_i = 1$. The following proposition expresses the expected re-execution fraction when an error occurs in the pattern.

Proposition 1. *Suppose a pattern contains m partial verifications, thus $n = m + 1$ segments, followed by a verified checkpoint. Then, the expected re-execution fraction in case of an error is given by*

$$f_{re} = \boldsymbol{\alpha}^T A \boldsymbol{\alpha} \ , \tag{5}$$

where $\boldsymbol{\alpha} = \begin{bmatrix}\alpha_1 & \alpha_2 & \ldots & \alpha_n\end{bmatrix}^T$ is a vector containing the fraction of work of each segment and A is the symmetric matrix defined by $A_{ij} = \frac{1}{2}\left(1 + (1-r)^{|i-j|}\right)$.

Proof. First, note that if no error occurs during the execution of a pattern, then the execution time is exactly the size of the pattern itself and there is no re-execution. When an error does occur, we assumed in Section III-B that no other error would occur again in the same pattern, including during the re-execution, because of the large platform MTBF μ. Moreover, the occurrence of the error is uniformly distributed.

With partial verifications, an error occurred in a segment may not be detected immediately, and may propagate to the following segments, thereby increasing the execution time until it gets eventually detected (in the worst case by the guaranteed verification at the end). Once the error is detected, all the previous work is lost, and we have to recover from the last checkpoint and re-execute the whole pattern again. Therefore, in order to express f_{re}, we need to compute the expected amount of time between the moment when the error strikes and the moment when it is actually detected.

Consider the pattern given in Figure 1 as an example. Suppose an error strikes in the pattern. We have the following scenarios:

- with probability α_1, the error strikes in the first segment and we lose $\alpha_1 W$ work. With probability $1-r$, the error is not detected by the first verification, so we further lose $\alpha_2 W$ work executing the second segment. With probability $(1-r)^2$, the error remains undetected by the second verification; in this case, it propagates to the third and last segment, where it is eventually detected by the guaranteed verification, so we lose an additional amount of work $\alpha_3 W$.
- with probability α_2, the error strikes in the second segment and we lose $(\alpha_1+\alpha_2)W$ work. With probability $1-r$, the error is not detected by the second verification, so we further lose $\alpha_3 W$ work executing the last segment.
- with probability α_3, the error strikes in the third and last segment. In this case, the error will be detected by the guaranteed verification, and the whole work pattern $(\alpha_1 + \alpha_2 + \alpha_3)W$ is lost.

Altogether, the total expected re-execution fraction for this pattern with three segments can be expressed as

$$\begin{aligned} f_{\text{re}} &= \alpha_1\left(\alpha_1+(1-r)\alpha_2+(1-r)^2\alpha_3\right)\\ &\quad+\alpha_2\left(\alpha_1+\alpha_2+(1-r)\alpha_3\right)\\ &\quad+\alpha_3(\alpha_1+\alpha_2+\alpha_3)\ . \end{aligned}$$

More generally, for a pattern with m partial verifications and $n = m + 1$ segments, we can derive

$$f_{\text{re}} = \sum_{i=1}^{n}\alpha_i\left(\sum_{j=1}^{i}\alpha_j + \sum_{j=i+1}^{n}(1-r)^{j-i}\alpha_j\right)\ ,$$

which can be rewritten in the following matrix form:

$$f_{\text{re}} = \boldsymbol{\alpha}^T B\boldsymbol{\alpha}, \tag{6}$$

where B is the following $n \times n$ Toeplitz matrix

$$B = \begin{bmatrix} 1 & 1-r & (1-r)^2 & \dots & (1-r)^{n-1}\\ 1 & 1 & 1-r & \dots & (1-r)^{n-2}\\ \vdots & \vdots & \vdots & \ddots & \vdots\\ 1 & 1 & 1 & \dots & 1 \end{bmatrix}\ .$$

Replacing B by $A = \frac{B+B^T}{2}$ in Equation (6), we have the same result, and we obtain

$$A = \frac{1}{2}\begin{bmatrix} 2 & 1+(1-r) & \dots & 1+(1-r)^{n-1}\\ 1+(1-r) & 2 & \dots & 1+(1-r)^{n-2}\\ \vdots & \vdots & \ddots & \vdots\\ 1+(1-r)^{n-1} & 1+(1-r)^{n-2} & \dots & 2 \end{bmatrix}$$

which concludes the proof. □

B. Optimal positions of partial verifications

For a given number m of partial verifications to be used in a pattern, the following theorem shows their optimal positions and the corresponding expected re-execution fraction.

Theorem 1. *Consider a pattern with m partial verifications thus $n = m + 1$ work segments. Suppose the work fraction of the i-th segment is α_i, so $\sum_{i=1}^{n}\alpha_i = 1$. Then, the expected fraction of work f_{re} that is re-executed in case of error is minimized when $\boldsymbol{\alpha} = \boldsymbol{\alpha}^*$, where*

$$\alpha_i^* = \begin{cases} \frac{1}{(n-2)r+2} & \text{for } i = 1 \text{ and } i = n\\ \frac{r}{(n-2)r+2} & \text{for } 2 \le i \le n-1 \end{cases}\ , \tag{7}$$

and the minimal re-execution fraction is

$$f_{re}^* = \frac{1}{2}\left(1 + \frac{2-r}{(n-2)r+2}\right)\ . \tag{8}$$

Somewhat unexpectedly, the n segments do not share the same length in the optimal solution: the first and last segments are longer than the others. When $r = 1$, we retrieve equal-length segments, which is in accordance with the results of [4].

Proof. The goal is to minimize $f_{\text{re}} = \boldsymbol{\alpha}^T A\boldsymbol{\alpha}$ (from Equation (5)) subject to the constraint $\sum_{i=1}^{n}\alpha_i = 1$. We rewrite the constraint as $\mathbf{c}^T\boldsymbol{\alpha} = 1$, where $\mathbf{c} = \begin{bmatrix}1 & 1 & \dots & 1\end{bmatrix}^T$.

Hence, we have a quadratic minimization problem under a linear constraint. If the matrix A is symmetric positive definite, it can be shown that this minimization problem admits a unique solution

$$f_{\text{re}}^{\text{opt}} = \frac{1}{\mathbf{c}^T A^{-1}\mathbf{c}}\ , \tag{9}$$

which is obtained at

$$\boldsymbol{\alpha}^{\text{opt}} = \frac{A^{-1}\mathbf{c}}{\mathbf{c}^T A^{-1}\mathbf{c}}\ . \tag{10}$$

In the following, we first show that A is indeed symmetric positive definite. Then we prove Equations (9) and (10). Finally, we prove that $\boldsymbol{\alpha}^{\text{opt}} = \boldsymbol{\alpha}^*$ and $f_{\text{re}}^{\text{opt}} = f_{\text{re}}^*$, where $\boldsymbol{\alpha}^*$ and f_{re}^* are given by Equations (7) and (8). This will conclude the proof of Theorem 1.

1) A is symmetric positive definite (SPD): In this section, we write A_n instead of simply A for a problem of size n (with n segments). We know that A_n is symmetric by construction. To show that A_n is positive definite, we show that all its principal minors are strictly positive. Recall that the principal minor of order k of A_n is the determinant of the submatrix of size k that consists of the first k rows and columns of A_n. But this submatrix is exactly A_k, the matrix for the problem of size k, so the result will follow if we show that $\det(A_n) > 0$ for all $n \ge 1$. We prove by induction on n that

$$\det(A_n) = \frac{r^{n-1}(2-r)^{n-2}((n-3)r+4)}{2^n}\ . \tag{11}$$

For $n = 1$, Equation (11) gives $\det(A_1) = 1$, which is correct. Assume that the result holds up to $n-1$. Computing the first coefficient $(A_n^{-1})_{11}$ of the inverse of A_n using the co-factor method, we get that

$$\left(A_n^{-1}\right)_{11} = \frac{\det(A_{n-1})}{\det(A_n)}\ .$$

And now comes the magic! It turns out that A_n is an extended KMS matrix (such that $A_{ij} = u + v\sigma^{|i-j|}$ with $u = v = \frac{1}{2}$ and $\sigma = 1-r$ in our case). Dow [13, Section 1.5] provides the inverse of such matrices and shows that they can be expressed using only seven coefficients. In particular, we have

$$\left(A_n^{-1}\right)_{11} = \frac{2((n-4)r+4)}{r(2-r)((n-3)r+4)}\ .$$

Thus, we can derive

$$\begin{aligned} \det(A_n) &= \frac{\det(A_{n-1})}{\left(A_n^{-1}\right)_{11}}\\ &= \frac{\det(A_{n-1})r(2-r)((n-3)r+4)}{2((n-4)r+4))}\\ &= \frac{r^{n-2}(2-r)^{n-3}((n-4)r+4)r(2-r)((n-3)r+4)}{2^{n-1}2((n-4)r+4)}\\ &= \frac{r^{n-1}(2-r)^{n-2}((n-3)r+4)}{2^n}\ , \end{aligned}$$

where the third line uses the inductive hypothesis for $\det(A_{n-1})$. This shows that Equation (11) holds for $\det(A_n)$ and completes the proof that A_n is SPD.

2) Optimal solution: We aim at minimizing $f_{\text{re}} = \boldsymbol{\alpha}^T A\boldsymbol{\alpha}$ subject to $\mathbf{c}^T\boldsymbol{\alpha} = 1$. Let $\boldsymbol{\alpha}^{\text{opt}} = f_{\text{re}}^{\text{opt}} A^{-1}\mathbf{c}$, where $f_{\text{re}}^{\text{opt}} = \frac{1}{\mathbf{c}^T A^{-1}\mathbf{c}}$, as in Equations (9) and (10). We check that $\mathbf{c}^T\boldsymbol{\alpha}^{\text{opt}} =$

$f_{\text{re}}^{\text{opt}}(\mathbf{c}^T A^{-1}\mathbf{c}) = 1$, so $\boldsymbol{\alpha}^{\text{opt}}$ is indeed a valid solution.

Because A is SPD, we have $X = (\boldsymbol{\alpha}-\boldsymbol{\alpha}^{\text{opt}})^T A(\boldsymbol{\alpha}-\boldsymbol{\alpha}^{\text{opt}}) \geq 0$ for any valid vector $\boldsymbol{\alpha}$, and $X = 0$ if and only if $\boldsymbol{\alpha} = \boldsymbol{\alpha}^{\text{opt}}$. Developing X, we get

$$X = \boldsymbol{\alpha}^T A\boldsymbol{\alpha} - 2\boldsymbol{\alpha}^T A\boldsymbol{\alpha}^{\text{opt}} + (\boldsymbol{\alpha}^{\text{opt}})^T A\boldsymbol{\alpha}^{\text{opt}} .$$

We have $\boldsymbol{\alpha}^T A\boldsymbol{\alpha}^{\text{opt}} = f_{\text{re}}^{\text{opt}}\boldsymbol{\alpha}^T\mathbf{c} = f_{\text{re}}^{\text{opt}}$ because $\mathbf{c}^T\boldsymbol{\alpha} = 1$. Similarly, we get $(\boldsymbol{\alpha}^{\text{opt}})^T A\boldsymbol{\alpha}^{\text{opt}} = f_{\text{re}}^{\text{opt}}$. Hence, we derive that $X = \boldsymbol{\alpha}^T A\boldsymbol{\alpha} - f_{\text{re}}^{\text{opt}} \geq 0$, with equality if and only if $\boldsymbol{\alpha} = \boldsymbol{\alpha}^{\text{opt}}$. This shows that the optimal value of f_{re} is achieved at $\boldsymbol{\alpha}^{\text{opt}}$, and is equal to $f_{\text{re}}^{\text{opt}}$.

3) $\boldsymbol{\alpha}^{opt} = \boldsymbol{\alpha}^*$ and $f_{re}^{opt} = f_{re}^*$: We now show that $A\boldsymbol{\alpha}^* = f_{\text{re}}^*\mathbf{c}$. From that we can directly derive $\boldsymbol{\alpha}^* = f_{\text{re}}^* A^{-1}\mathbf{c}$ and $1 = \mathbf{c}^T\boldsymbol{\alpha}^* = f_{\text{re}}^*(\mathbf{c}^T A^{-1}\mathbf{c})$ hence $f_{\text{re}}^{\text{opt}} = f_{\text{re}}^*$, and finally $\boldsymbol{\alpha}^{\text{opt}} = \boldsymbol{\alpha}^*$.

To show that $A\boldsymbol{\alpha}^* = f_{\text{re}}^*\mathbf{c}$, we proceed as follows. Since

$$\boldsymbol{\alpha}^* = \frac{1}{(n-2)r+2}\begin{bmatrix}1 & r & \ldots & r & 1\end{bmatrix}^T = \frac{r\mathbf{c}+(1-r)\mathbf{d}}{D_{\text{n}}} ,$$

where $D_{\text{n}} = (n-2)r+2$ and $\mathbf{d} = \begin{bmatrix}1 & 0 & \ldots & 0 & 1\end{bmatrix}^T$, we can compute

$$A\boldsymbol{\alpha}^* = \frac{rA\mathbf{c}+(1-r)A\mathbf{d}}{D_{\text{n}}} .$$

For $1 \leq i \leq n$, we get

$$\begin{aligned}(A\mathbf{c})_i &= \frac{1}{2}\left(\sum_{j=1}^{i-1}\left(1+(1-r)^{i-j}\right) + \sum_{j=0}^{n-i}\left(1+(1-r)^j\right)\right) \\ &= \frac{1}{2}\left(n + \frac{(1-r)-(1-r)^i}{r} + \frac{1-(1-r)^{n-i+1}}{r}\right)\end{aligned}$$

and

$$r(A\mathbf{c})_i = \frac{nr+(1-r)-(1-r)^i+1-(1-r)^{n-i+1}}{2} .$$

Then, we get

$$(A\mathbf{d})_i = \frac{1+(1-r)^{i-1}+1+(1-r)^{n-i}}{2}$$

and

$$(1-r)(A\mathbf{d})_i = \frac{2(1-r)+(1-r)^i+(1-r)^{n-i+1}}{2} .$$

Finally, we can compute

$$\begin{aligned}(A\boldsymbol{\alpha}^*)_i &= \frac{nr+(1-r)+1+2(1-r)}{2D_{\text{n}}} = \frac{nr+3(1-r)+1}{2D_{\text{n}}} \\ &= \frac{1}{2}\cdot\frac{(n-3)r+4}{(n-2)r+2} = \frac{1}{2}\left(1+\frac{2-r}{(n-2)r+2}\right) = f_{\text{re}}^* .\end{aligned}$$

This concludes the proof of Theorem 1. □

C. Optimal number of partial verifications

The following theorem shows the number of partial verifications used in an optimal pattern.

Theorem 2. *If $\frac{r}{2-r} > \frac{2V}{C+V^*}$, then the optimal number of partial verifications in a pattern is either $\lfloor m^* \rfloor$ or $\lceil m^* \rceil$, where*

$$m^* = -\frac{2-r}{r} + \sqrt{\left(\frac{2-r}{r}\right)\left(\frac{C+V^*}{V} - \frac{2-r}{r}\right)} . \quad (12)$$

If $\frac{r}{2-r} \leq \frac{2V}{C+V^*}$, then the optimal pattern contains no partial verification.

Proof. Consider a pattern containing m partial verifications, thus $n = m+1$ work segments. The fault-free overhead is $o_{\text{ff}}(m) = mV + V^* + C$. From Theorem 1, the minimum re-execution fraction is

$$f_{\text{re}}^*(m) = \frac{1}{2}\left(1+\frac{2-r}{(m-1)r+2}\right) .$$

Define $F(m) = o_{\text{ff}}(m)f_{\text{re}}^*(m)$. From the analysis outlined in Section III-C, the optimal m should minimize $F(m)$.

Differentiating $F(m)$ with respect to m and setting $\partial F(m)/\partial m = 0$, we get

$$m^2+2\left(\frac{2-r}{r}\right)m+2\left(\frac{2-r}{r}\right)^2-\left(\frac{2-r}{r}\right)\left(\frac{C+V^*}{V}\right) = 0 .$$

Solving the above equation gives us a critical point m^* as shown in Equation (12), which is positive (hence a potential solution) if $\frac{r}{2-r} > \frac{2V}{C+V^*}$.

Now, taking the second-order derivative of $F(m)$, we get

$$\frac{\partial^2 F(m)}{\partial m^2} = \frac{r(2-r)\left((C+V^*)r - V(2-r)\right)}{\left(2+(m-1)r\right)^3} ,$$

which is positive (hence ensures the solution is minimum) for all $m \in [0,\infty)$ if $\frac{r}{2-r} > \frac{V}{C+V^*}$.

In practice, the number of partial verifications can only be an integer. Thus, the optimal number, if $m^* > 0$, is either $\lceil m^* \rceil$ or $\lfloor m^* \rfloor$, whichever leads to a smaller $F(m)$. □

Altogether, we have completely characterized the optimal pattern of length $S = W + o_{\text{ff}}$:

- The number m of partial verifications in the pattern is given by Theorem 2, so that $o_{\text{ff}} = mV + V^* + C$;
- The positions of these partial verifications within the pattern, together with the optimal value of f_{re}, are given by Theorem 1;
- The work length W of the pattern is given by Equation (3).

D. Optimal cost-recall tradeoff

Now, we determine the optimal tradeoff between the cost and recall of a partial verification. Consider a partial verification, whose cost V and recall r could be traded off against each other by adjusting the parameters of the verification mechanism. For instance, the error detection capability of ABFT can be improved by adding more checksums to the matrices at the cost of additional computations [19]. The question is to determine the optimal configuration in order to minimize the execution overhead.

To find the optimal tradeoff, we define $a = \frac{r}{2-r}$ to be the *accuracy* of the partial verification, and define $b = \frac{V}{C+V^*}$ to be its *normalized cost*. The following theorem states that the optimal configuration is achieved when the *accuracy-to-cost ratio (ACR)* is maximized.

Theorem 3. *Suppose the cost V and recall r of a partial verification can be traded off against each other. Then, the minimum execution overhead is achieved when the accuracy-to-cost ratio a/b is maximized.*

Proof. Consider a partial verification configuration with fixed cost V and recall r.

Suppose $\frac{r}{2-r} \geq \frac{2V}{C+V^*}$, then Theorem 2 gives the optimal number m^* of partial verifications that minimizes $F(m) = o_{\text{ff}}(m) f^*_{\text{re}}(m)$. From Equation (12), we have

$$m^* = -\frac{1}{a} + \sqrt{\frac{1}{a}\left(\frac{1}{b} - \frac{1}{a}\right)} .$$

Plugging the above Equation into $F(m) = o_{\text{ff}}(m) f^*_{\text{re}}(m)$ and simplifying, we can get the optimal value of F as follows:

$$\begin{aligned} F^* &= o_{\text{ff}}(m^*) f^*_{\text{re}}(m^*) \\ &= \frac{C+V^*}{2}(bm^*+1)\left(1 + \frac{1}{am^*+1}\right) \\ &= \frac{C+V^*}{2}\left(\sqrt{1-\frac{b}{a}} + \sqrt{\frac{b}{a}}\right)^2 . \end{aligned}$$

When the platform MTBF μ is large, the optimal execution overhead (ignoring the lower-order term in Equation (4)) can then be expressed as

$$H^* = \sqrt{\frac{2(C+V^*)}{\mu}}\left(\sqrt{1-\frac{b}{a}} + \sqrt{\frac{b}{a}}\right) . \qquad (13)$$

Since $\frac{r}{2-r} \geq \frac{2V}{C+V^*}$, we have $0 \leq b/a \leq 1/2$. As the function $f = \sqrt{1-x} + \sqrt{x}$ is increasing in $[0, 1/2]$, the minimum execution overhead in Equation (13) is achieved when b/a is minimized, or equivalently when a/b is maximized. $\square$

We point out that the derivation in Theorem 3 is based on the fractional number m^* of partial verifications instead of the optimal integer value. As a result, the optimal configuration and overhead are subject to rounding error. The magnitude of such error has been evaluated numerically in Section V-B and is shown to be small for practical parameter settings.

E. An example

The analysis in the preceding subsections has completely characterized the optimal pattern and configuration using partial verifications. To demonstrate the results, let us consider an example.

Suppose a platform consists of 10^5 nodes, each with a MTBF of 100 years. Then, the overall MTBF of the system is $\mu = 100 \times 365 \times 24 \times 3600/10^5 = 31536$ seconds. Suppose the costs of checkpointing and guaranteed verification are $C = 600$ seconds and $V^* = 300$ seconds, respectively. A partial verification has three configurations with a cost-recall (V, r) tradeoff given by $(20, 0.5)$, $(30, 0.8)$ and $(50, 0.9)$. The following computes the optimal pattern.

First, we calculate the accuracy-to-cost ratio (ACR)

$$\frac{a}{b} = \frac{r(C+V^*)}{(2-r)V}$$

for the three partial verifications, which are 15, 20, and 14.73, respectively. Theorem 3 suggests to configure the partial verification with the highest ACR, i.e., $V = 30$ and $r = 0.8$. Then, Theorem 2 states that the minimum overhead in this configuration is achieved when $m^* \approx 5.0383$, and the optimal integer number of partial verifications in a pattern is $\lfloor m^* \rfloor = 5$. The optimal period, according to Equation (3), is computed to be $W^* = \sqrt{\mu \cdot o_{\text{ff}}(5)/f^*_{\text{re}}(5)} \approx 7335$ seconds, and Theorem 1 indicates that the six segments in the optimal pattern are approximately 1411, 1128, 1128, 1128, 1128, 1411 seconds, respectively. Finally, Equation (13) shows that the optimal expected overhead (ignoring the lower-order term) is about 28.6%.

When the application uses only guaranteed verifications, which is a special case of our analysis with $V = V^*$ and $r = 1$, the optimal pattern contains roughly $\sqrt{C/V} - 1$ equidistanced verifications followed by a verified checkpoint. In this example, the optimal pattern uses either 0 or 1 guaranteed verification as intermediate verification. Both cases turn out to lead to the same execution overhead around 33.8% with a checkpointing period of 5328 seconds.

This example illustrates that the use of partial verifications provides more than 5% improvement in the expected execution time of the application. This means saving 1 hour for every 20 hours of execution on the platform, which is significant in terms of cost and resource usage. In the next section, we will evaluate the impact of partial verifications with a wider range of parameters.

V. Performance Evaluation

In order to assess the impact of partial verifications and to determine the performance improvement they can provide, we evaluate the performance of the optimal algorithm described in the preceding section that employs partial verifications. Experiments are conducted using Maple for two different scenarios exhibiting a wide and realistic gamut of parameters. The usefulness of partial verifications is evaluated by comparing with the baseline algorithm that uses only guaranteed verifications. Section V-A describes the experimental setup, including the scenarios and range of parameter values. Section V-B presents the results through various plots and highlights the improvements over the baseline algorithm.

A. Evaluation framework

We present two scenarios used for instantiating the performance model for the algorithm with partial verifications. The target platform consists of 10^5 components whose individual MTBF is 100 years, which depicts a typical large-scale platform. This amounts to a platform MTBF of $\mu = 31536$ seconds. The other parameters depend on the scenario. For scenario 1, the checkpointing time is fixed at 600 seconds (10 minutes) and the guaranteed verification mechanism (with recall $r = 1$) takes 300 seconds to detect all the errors. For this scenario, performance is estimated by varying the cost V of partial verifications from 20 to 300 and varying the recall r from 0.1 to 0.9. Scenario 2, being more optimistic, fixes the parameter values at $C = 100$ seconds, $V^* = 30$ seconds and varies V from 3 to 30. For both scenarios, we also conduct evaluations by varying the number m of partial verifications (from 0 up to 15) to be able to monitor the behavior of the overhead in its entirety.

Regarding the accuracy of the performance model, we point out that, for the considered platform MTBF, the total length of the optimal interval S is always bounded by 0.3μ for scenario 1 and by 0.1μ for scenario 2, thereby resulting in a high accuracy of approximation.

B. Results and analysis

Based on the above framework, we report the evaluation results through various plots highlighting the behavior of the overhead, and the improvements achieved by employing partial verifications.

1) Impact of m: We first evaluate the impact of the number m of partial verifications on the expected execution overhead. Figure 2 shows the overhead behavior when fixing the cost of the partial verification to be $V = 20$ for scenario 1 and $V = 3$ for scenario 2. For both scenarios, the overhead is greatly diminished by employing partial verifications, except when $r = 0.1$, in which case the overhead almost overlaps with that of the baseline algorithm before rising. However, forcing too many verifications eventually makes the error-free overhead rise, while forcing too few increases the error-induced overhead. The range of m shows that the overhead indeed falls down to the optimal value and then starts rising. A similar behavior can be observed among the curves with different recall values. The plots indicate an improvement in overhead of approximately 6% for scenario 1 and 3% for scenario 2 over the baseline algorithm for the verification with the highest recall $r = 0.9$.

2) Impact of r and V: Figures 3 and 4 illustrate, for scenarios 1 and 2 respectively, as a function of r and V, the optimal expected overhead (on the left) and the corresponding optimal number of partial verifications (on the right). We can see in both 3D plots that the optimal overhead is the same for many r, V combinations, which is obtained when no partial verification is used, either due to an expensive cost or due to a low recall value. In such cases, the baseline algorithm has better performance. On the other hand, there exist many cases where partial verifications should be used. This is evident from the improvements in the expected overhead, which in scenario 1 is up to 6.3% and in scenario 2 is up to 2.3%.

It can also be inferred from the contour plots that employing partial verifications in a pattern is beneficial for scenario 1 in 40% of the cases and for scenario 2 in 60% of the cases. For all these cases, the optimal pattern uses at least one partial verification. Although an increase in verification cost adversely influences the optimal overhead and we quickly move to the yellow region (representing all r, V combinations for which $m = 0$) in both scenarios, a sufficiently high recall value would still offset that impact. For example, Figure 4 shows that, for $r \geq 0.7$, it is always beneficial to use partial verification(s) irrespective of its cost.

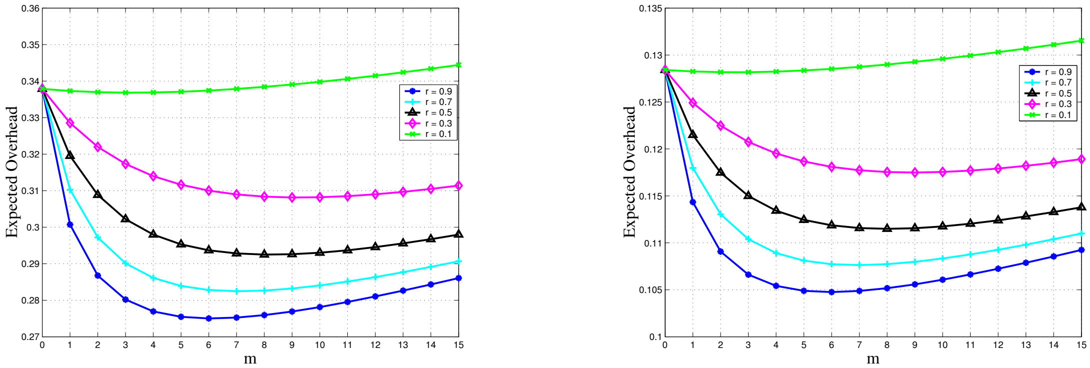
3) Impact of ACR: Figure 5 shows, for scenario 1, the expected execution overhead as a function of the accuracy-to-cost ratio (ACR) discussed in Section IV-D. In Figure 5(a), both the optimal and worst-case overheads are plotted as ACR is varied from 0 to 30. The optimal overhead is computed based on Equation (13), which gives the ideal value by using the optimal fractional m^* as shown in Equation (12). However, since practical number of partial verifications can only be an integer, the worst overhead reflects the maximum possible overhead for each ACR value. This is obtained by varying r from 0.01 to 0.99 and V from 20 to 300, and by computing among them the worst optimal integer solution. We can see that the two curves exhibit almost negligible difference, especially for higher ACR values. This provides a strong support to the result of Theorem 3 on the selection of optimal partial verification configuration.

Figure 5(b) further shows, between the optimal and worst overhead curves, multiple overheads that can be obtained as scattered points. For the sake of clarity, the number of data points has been reduced in the plot to show the configuration with $m = 0$ up to $m = 5$ only within the ACR range of 2 to 5. This plot shows, for each ACR value, the existence of multiple configurations that lead to different expected overheads and number of partial verifications. For example, for ACR = 2.85, one configuration results in an overhead of 33.8% without using any partial verification while another configuration has an overhead of 33.4% with three partial verifications. In this case, a difference of 0.4% is observed in terms of the overhead, and the gap becomes smaller as ACR increases.

VI. Conclusion and Future Work

In this paper, we have evaluated the impact of partial verifications in the detection of silent data corruptions. Since silent errors are only identified when the corrupted data is activated, enforcing some verification mechanism is a promising approach to tackling them. For many parallel applications, partial verification offers a low-overhead alternative to the guaranteed counterpart, at the expense of reduced error detection accuracy. By incorporating partial verifications, we have derived the optimal pattern (up to the first-order approximation) in a resilient protocol, including the optimal configuration, the optimal checkpointing period, the optimal number of verifications, as well as their optimal positions inside the pattern. These results provide dramatic extensions to the existing formulas in the field. Evaluations based on a wide range of parameters confirm the benefit of using partial verifications in certain scenarios, when compared to the baseline algorithm that uses only guaranteed verifications.

In future work, we will investigate the use of multiple configurations of a partial verification with different costs and recalls. Note that Theorem 3 does not consider the case where different configurations can be mixed together in a single pattern. The question is whether better performance can be achieved by utilizing more than one configuration simultaneously. Another direction is to consider “false positives”, which



(a) Scenario 1: $C = 600$, $V^* = 300$ and $V = 20$

(b) Scenario 2: $C = 100$, $V^* = 30$ and $V = 3$

Figure 2: Expected overhead for two scenarios with different recall r and number m of partial verifications.

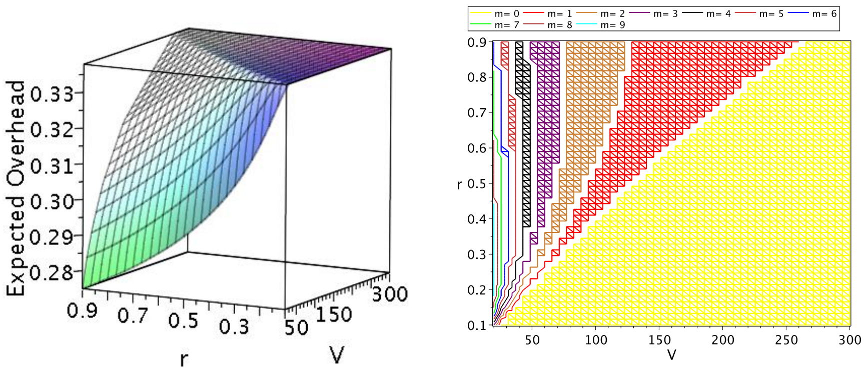


Figure 3: Optimal expected overhead on the left and a contour plot on the right showing the optimal number of partial verification(s) for varying r and V values when $C = 600$ and $V^* = 300$. The contour plot also shows the r and V combinations for each value of the optimal m.

are present in many fault filters, such as the ones described in [2], [10]. False positives are measured by the *precision* value, defined as the number of actual errors over the total number of detected errors. Indeed, there exists a tradeoff between the recall and precision by adjusting the range parameters, based on which the computed data is examined by such filters. Analyzing the performance of verification mechanisms in the presence of both false positives and false negatives will be a challenge.

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, and by the ANR Rescue project. 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 the 19th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC)*, pages 11–20, 2013.

[2] L. Bautista Gomez and F. Cappello. Detecting silent data corruption through data dynamic monitoring for scientific applications. *SIGPLAN Notices*, 49(8):381–382, 2014.

[3] A. Benoit, A. Cavelan, Y. Robert, and H. Sun. Assessing general-purpose algorithms to cope with fail-stop and silent errors. In *Proceedings of the 5th International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*, pages 215–236, 2014.

[4] A. Benoit, Y. Robert, and S. K. Raina. Efficient checkpoint/verification patterns for silent error detection. *ICL Research report RR-1403*, 2014.

[5] 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.

[6] 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.

[7] G. Bronevetsky and B. de Supinski. Soft error vulnerability of iterative linear algebra methods. In *Proceedings of the International Conference on Supercomputing (ICS)*, pages 155–164, 2008.

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

[9] Z. Chen. Online-ABFT: An online algorithm based fault tolerance scheme for soft error detection in iterative methods. In *Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*, pages 167–176, 2013.

[10] E. Ciocca, I. Koren, Z. Koren, C. M. Krishna, and D. S. Katz. Application-level fault tolerance in the orbital thermal imaging spectrometer. In *Proceedings of the 10th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC'04)*, pages 43–48, 2004.

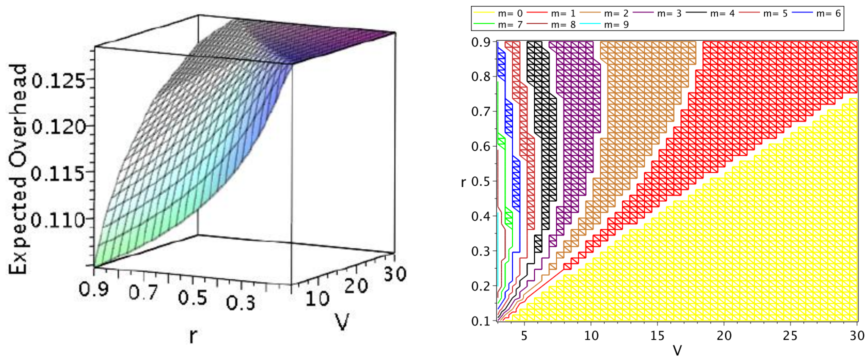
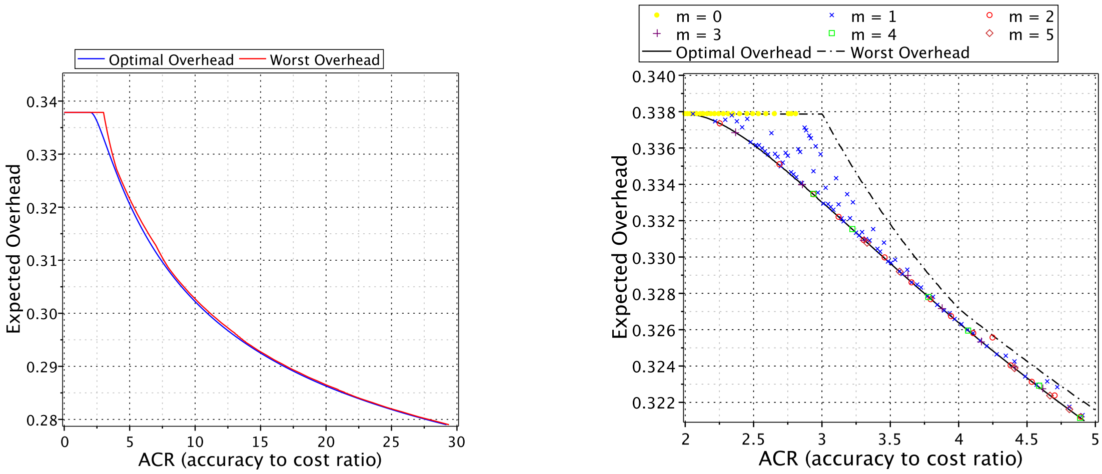


Figure 4: Optimal expected overhead on the left and a contour plot on the right showing the optimal number of partial verification(s) for varying r and V values when $C = 100$ and $V^* = 30$. The contour plot also shows the r and V combinations for each value of the optimal m.



(a) (b)

Figure 5: Optimal and worst overhead curves against accuracy-to-cost ratio (ACR) when $C = 600$ and $V^* = 300$.

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

[12] 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. *HJPCA*, 23(4):309–322, 2009.

[13] M. Dow. Explicit inverses of toeplitz and associated matrices. *ANZIAM J.*, 44(E):185–215, 2003.

[14] J. Elliott, K. Kharbas, D. Fiala, F. Mueller, K. Ferreira, and C. Engelmann. Combining partial redundancy and checkpointing for HPC. In *Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 615–626, 2012.

[15] 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.

[16] 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 *Proc. SC'12*, page 78, 2012.

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

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

[19] J. Jou and J. Abraham. Fault tolerant matrix operations on multiple systems using weighted checksums. In *Proc. SPIE 0495, Real-Time Signal Processing VII*, pages 94–101, 1984.

[20] G. Lu, Z. Zheng, and A. A. Chien. When is multi-version checkpointing needed? In *Proc. 3rd Workshop on Fault-tolerance for HPC at extreme scale (FTXS)*, pages 49–56, 2013.

[21] 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.

[22] A. Moody, G. Bronevetsky, K. Mohror, and B. R. d. Supinski. Design, Modeling, and Evaluation of a Scalable Multi-level Checkpointing System. In *Proc. of the ACM/IEEE SC Conf.*, pages 1–11, 2010.

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

[24] 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.

[25] 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.

[26] M. Shantharam, S. Srinivasmurthy, and P. Raghavan. Fault tolerant preconditioned conjugate gradient for sparse linear system solution. In *Proceedings of the ACM International Conference on Supercomputing (ICS)*, pages 69–78, 2012.

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

[28] 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.