# Optimal resilience patterns to cope with fail-stop and silent errors

Anne Benoit*, Aurélien Cavelan*, Yves Robert*†, Hongyang Sun*
*Ecole Normale Supérieure de Lyon & Inria, France
†University of Tennessee Knoxville, USA

***Abstract*—This work focuses on resilience techniques at extreme scale. Many papers deal with fail-stop errors. Many others deal with silent errors (or silent data corruptions). But very few papers deal with fail-stop and silent errors simultaneously. However, HPC applications will obviously have to cope with both error sources. This paper presents a unified framework and optimal algorithmic solutions to this double challenge. Silent errors are handled via verification mechanisms (either partially or fully accurate) and in-memory checkpoints. Fail-stop errors are processed via disk checkpoints. All verification and checkpoint types are combined into computational patterns. We provide a unified model, and a full characterization of the optimal pattern. Our results nicely extend several published solutions and demonstrate how to make use of different techniques to solve the double threat of fail-stop and silent errors. Extensive simulations based on real data confirm the accuracy of the model, and show that patterns that combine all resilience mechanisms are required to provide acceptable overheads.**

***Index Terms*—resilience, fail-stop errors, silent errors, multi-level checkpoint, verification, optimal pattern.**

## I. Introduction

Fault-tolerance techniques are mandatory at extreme scale [12], [13]. The first source of problems is the frequent striking of fail-stop (unrecoverable) errors. This phenomenon is well understood: regardless of the robustness of each individual resource, aggregating too many resources will cause trouble at some point. Specifically, if the MTBF (Mean Time Between Failures) of each resource is ten years (a pretty optimistic figure for, say, a socket or a processor node), then the MTBF of a platform comprising one million of such resources is only five minutes[1]. The standard approach to cope with fail-stop errors is checkpoint and rollback recovery, and many protocols are available (see [19] for a survey).

The second source of problems is the frequent striking of silent errors (or SDCs, for Silent Data Corruptions). This phenomenon is not so well understood, but has been recently identified as one of the major challenges for Exascale [13]. 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 a 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 [15], [22], [8]. This verification mechanism can be general-purpose (e.g., based on replication [18] or even triplication [20]) or application-specific (e.g., based on Algorithm-based fault tolerance (ABFT) [11], on approximate re-execution for ODE and PDE solvers [9], or on orthogonality checks for Krylov-based sparse solvers [15], [22]).

Verification mechanisms are typically costly; in fact, replication is the only alternative in an application-agnostic framework. Guaranteeing accurate and efficient detection of silent errors for scientific applications is one of the hardest challenges in extreme-scale computing [13]. 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 silent 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 [10]. Although not completely accurate, these partial verification techniques nevertheless cover a substantial amount of silent errors, and more importantly, they incur low overhead. These properties make them attractive candidates for designing more efficient resilient protocols.

Altogether, the detection of silent errors seriously complicates the design of resilience protocols. What is the best type of verification, either guaranteed or partial? And what is the best combination with checkpoints? To further complicate the story, silent errors naturally call for in-memory checkpointing, because a local copy of the data can be used after corruption has been detected. On the contrary, fail-stop errors require to store the checkpoints on remote stable storage (disks) because the whole memory content can be lost when such a failure strikes. Granted, multi-level checkpointing protocols have been designed for several years, but we face two major difficulties when combining fail-stop and silent errors. First, and to the best of our knowledge, the interplay of verification mechanisms with two types of checkpoints, in-memory and disk-based, has never been investigated. Second, the inherent detection latency of silent errors renders the problem quite different from traditional multi-level checkpointing, where each failure, regardless of its level, is detected immediately

[1]The MTBF $\mu_p$ with $p$ resources is $\mu_p = \mu_{\text{ind}}/p$, where $\mu_{\text{ind}}$ is the MTBF of each resource, see [19, Proposition 1.2].

upon striking. In this work, after some quite technically involved derivations, we provide the optimal solution to the problem, either with guaranteed or with partial verifications. This was somewhat unexpected, because no optimal solution is known for two-level checkpointing with two levels of fail-stop errors; state-of-the-art protocols in that latter context rely on heuristics [17].

Our approach to solving the double problem of fail-stop and silent errors is to partition the execution of the application into *periodic patterns*, i.e., computational units that repeat over time. Each pattern ends with a guaranteed verification, an in-memory checkpoint and a disk checkpoint, so that errors do not propagate from a given pattern to the next one. Inside each pattern, there are several segments, each ending with a guaranteed verification and an in-memory checkpoint. In turn, each segment is partitioned into work chunks (possibly of different sizes) that are separated by partial verifications. See Figure 1(b) for an example with three segments and a total of six chunks. Several parameters should be given to fully characterize a pattern, namely the number of segments, and the number and size of each chunk inside each segment. The shape of a pattern is quite flexible, which enables us to provide the first model including two levels of checkpoints.

The main objective is to design an optimal pattern. Informally, consider a pattern P that includes $W$ units of work (the cumulated size of all the chunks within the pattern). Without loss of generality, assume unit speed computation, so that we can speak of time or work interchangeably. In the presence of fail-stop or silent errors, the expected execution time of the pattern will be $\mathbb{E}(\mathrm{P})$: we have to take expectations, as the computation time is no longer deterministic. Note that $\mathbb{E}(\mathrm{P}) > W$ for two reasons: the time spent in checkpoints and verifications, even if there is no error, and the time lost due to recovery and re-execution after an error. An optimal pattern is defined as the one minimizing the ratio $\frac{\mathbb{E}(\mathrm{P})}{W}$, or equivalently the ratio $\frac{\mathbb{E}(\mathrm{P})-W}{W} = \frac{\mathbb{E}(\mathrm{P})}{W} - 1$. This latter ratio is the relative overhead paid for executing the pattern. The smaller this overhead, the faster the progress of the execution.

The main contributions of this work are the following:

• The design of a detailed model based upon the computational patterns described above.

• The determination of the optimal pattern, first in some particular cases (one-chunk segments, one segment with multiple chunks), and then in the general case. The comprehensive list of results summarized in Table I extends and unifies many results from the literature.

• An extensive set of simulations that use data collected on real platforms, and extrapolate them to exascale platforms. The results confirm the accuracy of the model, as long as the MTBF is large enough in front of the resilience parameters. They also help assess the impact of each resilience mechanism, and show that patterns that combine all mechanisms (partial and guaranteed verifications and two checkpoint types) are required to provide acceptable overheads.

The rest of the paper is organized as follows. Section II introduces the model and notations. The following section shows how to determine the optimal pattern. We start with the simplest pattern (a single one-chunk segment) in Section III-A, extending Young and Daly's formula to two error sources. We discuss patterns with multiple one-chunk segments in Section III-B, and the most general pattern in Section III-C. Simulation results are presented in Section IV. Section V briefly discusses related work. Finally, Section VI provides concluding remarks and hints for future directions.

## II. Model

**Failure model.** We consider a realistic scenario in large-scale systems, where hardware faults and silent data corruptions coexist. They are commonly referred to as *fail-stop* errors and *silent* errors in the literature. Since these two types of errors are caused by different sources, we assume that they are independent and that both occurrences follow a *Poisson process* with arrival rates $\lambda_f$ and $\lambda_s$, respectively. The probability of having at least a fail-stop error during a computation of length $w$ is given by $p^f = 1 - e^{-\lambda_f w}$ and that of having at least a silent error during the same computation is $p^s = 1 - e^{-\lambda_s w}$. We also assume that both error rates are in the same order, i.e., $\lambda_f = \Theta(\lambda)$, and $\lambda_s = \Theta(\lambda)$, where $\lambda = \lambda_f + \lambda_s = 1/\mu$ denotes the reciprocal of the platform MTBF $\mu$ while accounting for both types of failures.

**Two-level checkpointing.** To deal with both fail-stop and silent errors, resilience is provided through the use of a two-level checkpointing scheme coupled with an error detection (or verification) mechanism. The protocol is enforced by a periodic computing *pattern* as discussed in Section I. When a fail-stop error strikes inside a pattern, the computation is interrupted immediately due to a hardware fault, so all the memory content is destroyed: we then roll back to the beginning of the pattern and recover from the last disk checkpoint (taken at the end of the previous pattern, or the initial data for the first pattern). On the contrary, when a silent error is detected inside a pattern, either by a partial verification or by a guaranteed one, we roll back to the nearest memory checkpoint in the pattern, and recover from the memory copy there, which is much cheaper than recovering from the last disk checkpoint. We enforce the following two properties for a pattern:

• *A memory checkpoint is always taken immediately before each disk checkpoint*. Since performing an I/O operation requires first flushing the data to a memory buffer, this process incurs little extra overhead and hence has a natural justification. Indeed, such a property has been enforced in some practical multi-level checkpointing systems [5]. Similarly, when we recover from a disk checkpoint, we also restore the corresponding memory copy, which was destroyed due to the last fail-stop error.

• *A guaranteed verification is always executed immediately before each memory checkpoint*. Since storing a checkpoint can be expensive even for the memory, this property guarantees that all (memory and disk) checkpoints are valid, and hence avoids the need of maintaining multiple checkpoints, which is known to be difficult to recover from (one has to decide which checkpoint is valid, etc.). With this property, only one memory checkpoint and one disk checkpoint need to be maintained at any time during the execution of the application.

To simplify the presentation, we assume that errors only strike the computations, while verifications, memory copies, and I/O transfers are protected from failures. However, all optimality results hold without this assumption [7].

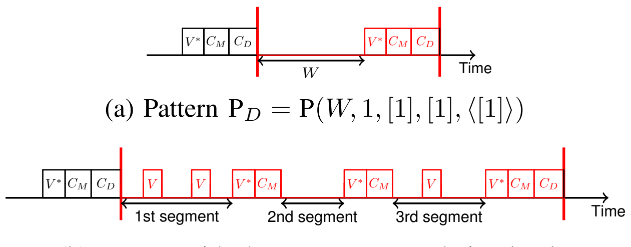
**Notations.** Let $C_D$ denote the cost of disk checkpointing, $C_M$ the cost of memory checkpointing, $R_D$ the cost of disk recovery, and $R_M$ the cost of memory recovery. Recall that when a disk recovery is done, we also need to restore the memory state, hence a cost $R_D + R_M$ is paid. Also, let $V^*$ denote the cost of guaranteed verification and $V$ the cost of a partial verification. The partial verification is also characterized by its *recall*, which is denoted by $r$ and represents the proportion of detected errors over all silent errors that have occurred during the execution. If multiple partial verifications are available, our previous work [14], [2] has suggested to use the one with the largest *accuracy-to-cost* ratio, which is defined as $\frac{r}{2-r}/\frac{V}{V^*+C_M}$. Note that the guaranteed verification can be considered as one with recall $r^* = 1$ and hence with accuracy-to-cost ratio $\frac{C_M}{V^*}+1$. Since a partial verification usually incurs a much smaller cost yet has a reasonable recall, its accuracy-to-cost ratio can be orders of magnitude (e.g., 100x) better than that of the guaranteed verification [3], [10]. Hence partial verifications are highly attractive for detecting silent errors, and we make use of them between memory checkpoints in the pattern.

For clarity, we refer to the computation between two consecutive memory checkpoints as a *segment*, and refer to the computation between two consecutive verifications as a *chunk*. Formally, a pattern $\mathrm{P}(W, n, \boldsymbol{\alpha}, \boldsymbol{m}, \langle \boldsymbol{\beta}_1, \dots, \boldsymbol{\beta}_n \rangle)$ is defined by the following parameters:

- $W$: total amount of computation (or work) of the pattern.
- $n$: number of memory checkpoints inside the pattern (also number of segments within the pattern).
- $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, \dots, \alpha_n]$: proportion of the segment sizes, i.e., $\alpha_i = w_i/W$, where $w_i$ denotes the amount of work in the $i$-th segment of the pattern. Hence $\sum_{i=1}^{n} \alpha_i = 1$.
- $\boldsymbol{m} = [m_1, m_2, \dots, m_n]$: number of verifications inside each segment (also number of chunks in that segment).
- $\boldsymbol{\beta}_i = [\beta_{i,1}, \beta_{i,2}, \dots, \beta_{i,m_i}]$ $\forall i = 1, 2, \dots, n$: proportion of the chunk sizes in the segments, i.e., $\beta_{i,j} = w_{i,j}/w_i$, where $w_{i,j}$ denotes the amount of work in the $j$-th chunk of the $i$-th segment. Hence $\sum_{j=1}^{m_i} \beta_{i,j} = 1$ for all $i = 1, 2, \dots, n$.

The simplest pattern is illustrated in Figure 1(a), and it consists of a single segment ($n = 1$, $W = w_1$), which comprises a single chunk ($\boldsymbol{m} = [1]$). By construction, this chunk is followed by a guaranteed verification, followed immediately by a memory checkpoint and a disk checkpoint. With our notations, this pattern is denoted as $\mathrm{P}(W, 1, [1], [1], \langle [1] \rangle)$, or $\mathrm{P}_D$ (only disk checkpoints, which are always preceded by a guaranteed verification and a memory checkpoint). Figure 1(b) shows a more complicated pattern, with three segments. The first segment has three chunks, the second segment has one chunk, and the third segment has two chunks. Therefore, if a silent error is detected by the guaranteed verification at the end of the second segment, it is possible to recover from the memory checkpoint preceding it, rather than starting the whole pattern again. Moreover, silent errors, if occurred in the first and third segments, may be detected earlier thanks to the additional partial verifications.

**Objective.** The objective is to find a pattern that minimizes the expected execution time of the application. Let $W_{\text{base}}$ denote the base execution time of an application without any overhead due to resilience techniques (without loss of generality, we assume unit-speed execution). Since the execution is divided into periodic patterns, defined by $\mathrm{P}(W, n, \boldsymbol{\alpha}, \boldsymbol{m}, \langle \boldsymbol{\beta}_1, \dots, \boldsymbol{\beta}_n \rangle)$, let $\mathbb{E}(\mathrm{P})$ be the expected execution time of the pattern. For large jobs, the expected makespan $W_{\text{final}}$ of the application when taking failures into account can then be approximated by $W_{\text{final}} \approx \frac{\mathbb{E}(\mathrm{P})}{W} \cdot W_{\text{base}}$. Now, define $H(\mathrm{P}) = \frac{\mathbb{E}(\mathrm{P})}{W} - 1$ to be the expected *overhead* of the pattern. We obtain $W_{\text{final}} \approx W_{\text{base}} + H(\mathrm{P}) \cdot W_{\text{base}}$. Thus, minimizing the expected makespan is equivalent to minimizing the pattern overhead $H(\mathrm{P})$. Hence, we will focus on minimizing the overhead in this paper.



(a) Pattern $\mathrm{P}_D = \mathrm{P}(W, 1, [1], [1], \langle [1] \rangle)$

(b) Pattern with three segments and six chunks

Figure 1. Two examples of patterns.

## III. Optimal Patterns

### A. *Pattern $\mathrm{P}_D$, or revisiting Young and Daly*

In this section, we revisit Young [23] and Daly [16] on computing the optimal periodic checkpointing interval, and extend their formula to include both fail-stop and silent errors. The result on the order of the optimal interval and the observations established in this case will pave the way for the subsequent analysis on more advanced patterns.

The classical formula by Young and Daly gives the optimal disk checkpointing interval without considering silent errors, thus does not include verifications and memory checkpoints in the pattern. To cope with both fail-stop and silent errors, we analyze the pattern $\mathrm{P}_D = \mathrm{P}(W, 1, [1], [1], \langle [1] \rangle)$, which contains one single segment with a unique chunk followed by a guaranteed verification, a memory checkpoint and a disk checkpoint (see Figure 1(a)). Obviously, the only parameter to determine is the work length $W$, which is commonly referred to as the checkpointing period in the literature. The following proposition shows the expected execution time of a pattern with a fixed work length.

**Proposition 1.** *The expected execution time of a given pattern* $\mathrm{P}(W, 1, [1], [1], \langle[1]\rangle)$ *is*

$$\mathbb{E}(\mathrm{P}) = W + V^* + C_M + C_D + \left(\lambda_s + \frac{\lambda_f}{2}\right) W^2 \\ + \lambda_s W(V^* + R_M) + \lambda_f W(R_M + R_D) + O(\lambda^2 W^3) \ . \quad (1)$$

*Proof.* Let $p^f = 1 - e^{-\lambda_f W}$ and $p^s = 1 - e^{-\lambda_s W}$ denote the probabilities of having at least one fail-stop error and at least one silent error, respectively, in the pattern. The expected execution time obeys the following recursive formula:

$$\mathbb{E}(\mathrm{P}) = p^f \left(\mathbb{E}(T^{\text{lost}}) + R_D + R_M + \mathbb{E}(\mathrm{P})\right) \\ + (1 - p^f)\big(W + V^* + p^s \left(R_M + \mathbb{E}(\mathrm{P})\right) \\ + (1 - p^s)(C_M + C_D)\big) \ , \quad (2)$$

where $\mathbb{E}(T^{\text{lost}})$ denotes the expected time loss during the execution of the pattern if a fail-stop error strikes. Equation (2) can be interpreted as follows: if a fail-stop error occurs, we lose $\mathbb{E}(T^{\text{lost}})$ time, perform a recovery from both disk and memory, and then re-execute the pattern (Line 1). If no fail-stop error strikes during the execution, we run the guaranteed verification to detect silent errors, which if indeed occurred involves a memory recovery only followed by a re-execution (Line 2). Otherwise, if no silent error strikes either, we can proceed with the memory and disk checkpointing (Line 3).

To derive the expected execution time, we need to compute $\mathbb{E}(T^{\text{lost}})$, which can be expressed as follows: $\mathbb{E}(T^{\text{lost}}) = \int_0^\infty x\mathbb{P}(X = x | X < W)dx = \frac{1}{\mathbb{P}(X<W)} \int_0^W x\mathbb{P}(X = x)dx$, where $\mathbb{P}(X = x)$ denotes the probability that a fail-stop error strikes at time $x$. By definition, we have $\mathbb{P}(X = x) = \lambda_f e^{-\lambda_f x}$ and $\mathbb{P}(X < W) = 1 - e^{-\lambda_f W}$. Integrating by parts, we get

$$\mathbb{E}(T^{\text{lost}}) = \frac{1}{\lambda_f} - \frac{W}{e^{\lambda_f W} - 1} \ . \quad (3)$$

Substituting Equation (3) into Equation (2) and solving the recursion, we obtain

$$\mathbb{E}(\mathrm{P}) = \frac{e^{(\lambda_f + \lambda_s)W} - e^{\lambda_s W}}{\lambda_f} + e^{\lambda_s W} V^* + C_D + C_M \\ + \left(e^{(\lambda_f+\lambda_s)W} - e^{\lambda_s W}\right) R_D + \left(e^{(\lambda_f+\lambda_s)W} - 1\right) R_M \ .$$

By approximating $e^{\lambda x} = 1 + \lambda x + \frac{\lambda^2 x^2}{2}$ up to the second-order term, we can further simplify the expected execution time, which turns out to be given by Equation (1). □

**Theorem 1.** *A first-order approximation to the optimal work length in pattern* $\mathrm{P}(W, 1, [1], [1], \langle[1]\rangle)$ *is given by*

$$W^* = \sqrt{\frac{V^* + C_M + C_D}{\lambda_s + \frac{\lambda_f}{2}}} \ . \quad (4)$$

*The optimal expected overhead in this case is*

$$H^*(\mathrm{P}) = 2\sqrt{\left(\lambda_s + \frac{\lambda_f}{2}\right)(V^* + C_M + C_D)} + O(\lambda) \ . \quad (5)$$

*Proof.* From the result of Proposition 1, the expected overhead of the pattern can be computed as

$$H(\mathrm{P}) = \frac{V^* + C_M + C_D}{W} + \left(\lambda_s + \frac{\lambda_f}{2}\right) W \\ + \lambda_s(V^* + R_M) + \lambda_f(R_M + R_D) + O(\lambda^2 W^2) \ . \quad (6)$$

Assuming that the platform MTBF $\mu = 1/\lambda$ is large in front of the resilience parameters, consider the first two terms of $H(\mathrm{P})$ (Line 1 of Equation (6)): the overhead is minimal when the pattern has length $W = \Theta(\lambda^{-1/2})$, and in that case both terms are in the order of $\lambda^{1/2}$, so that we have

$$H(\mathrm{P}) = \Theta(\lambda^{1/2}) + O(\lambda).$$

Indeed, the last term $O(\lambda^2 W^2)$ becomes also negligible compared to $\Theta(\lambda^{1/2})$. Hence, the optimal pattern length $W^*$ can be obtained by balancing the first two terms of Equation (6), which gives rise to Equation (4). Then, by substituting $W^*$ back into $H(\mathrm{P})$, we get the optimal expected overhead as shown by Equation (5). □

When only fail-stop errors exist, we retrieve the classical formula by Young [23] and Daly [16], which is given by $W^* = \sqrt{2C_D/\lambda_f}$. When there are only silent errors, the optimal work length is given by $W^* = \sqrt{(V^* + C_M)/\lambda_s}$.

We observe from Theorem 1 that the optimal work length $W^*$ of a pattern is in the order of $\lambda^{-1/2}$ and the optimal overhead $H^*(\mathrm{P})$ is in the order of $\lambda^{1/2}$. Theorem 1 also shows that we can express the expected execution overhead of a pattern as $H(\mathrm{P}) = \frac{o_{\text{ef}}}{W} + o_{\text{rw}} W + O(\lambda)$, where $o_{\text{ef}}$ and $o_{\text{rw}}$ are two key parameters that characterize two different types of overheads in the execution, and they are defined below.

**Definition 1.** *For a given pattern,* $o_{ef}$ *denotes the* error-free *overhead due to the resilience operations (e.g., verification, checkpointing), and* $o_{rw}$ *denotes the* re-executed work *overhead, in terms of the fraction of re-executed work due to errors.*

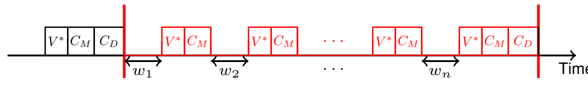
In the simple pattern $\mathrm{P}(W, 1, [1], [1], \langle[1]\rangle)$ analyzed above, these two overheads are given by $o_{\text{ef}} = V^* + C_M + C_D$ and $o_{\text{rw}} = \lambda_s + \frac{\lambda_f}{2}$, respectively. The optimal pattern length and the optimal expected overhead can thus be expressed as

$$W^* = \sqrt{\frac{o_{\text{ef}}}{o_{\text{rw}}}} \text{ and } H^*(\mathrm{P}) = 2\sqrt{o_{\text{ef}} \cdot o_{\text{rw}}} + O(\lambda) \ . \quad (7)$$

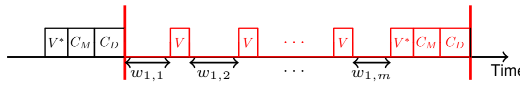
We see that minimizing the expected execution overhead $H(\mathrm{P})$ of a pattern becomes equivalent to minimizing the product $o_{\text{ef}} \cdot o_{\text{rw}}$ up to the dominating term. Intuitively, including more resilience operators reduces the re-executed work overhead but adversely increases the error-free overhead, and vice versa. This requires a resilience protocol to find the optimal trade-off between $o_{\text{ef}}$ and $o_{\text{rw}}$. We will make use of this observation in the subsequent sections to derive the optimal patterns in more complicated protocols.

### *B. Pattern* $\mathrm{P}_{DM} = \mathrm{P}(W, n, \boldsymbol{\alpha}, [1, \ldots, 1], \langle[1], \ldots, [1]\rangle)$

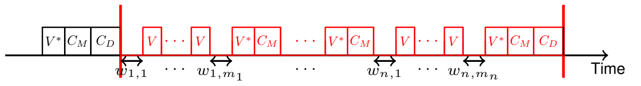
We first consider a pattern with multiple segments, but each segment has only one chunk. In other words,



(a) Pattern $\mathrm{P}_{DM} = \mathrm{P}(W, n, \boldsymbol{\alpha}, [1, \ldots, 1], \langle [1], \ldots, [1] \rangle)$



(b) Pattern $\mathrm{P}_{DV} = \mathrm{P}(W, 1, [1], [m], \langle \boldsymbol{\beta} \rangle)$



$V^*$ $C_M$ $C_D$ $V$
$w_{1,1}$ $\cdots$ $w_{1,m_1}$ $\cdots$ $w_{n,1}$ $\cdots$ $w_{n,m_n}$
Time

(c) Pattern $\mathrm{P}_{DMV} = \mathrm{P}(W, n, \boldsymbol{\alpha}, \boldsymbol{m}, \langle \boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_n \rangle)$

Figure 2. Patterns $\mathrm{P}_{DM}$, $\mathrm{P}_{DV}$ and $\mathrm{P}_{DMV}$.

the protocol performs multiple memory checkpoints between two disk checkpoints but without any intermediate verification. Figure 2(a) depicts the pattern $\mathrm{P}_{DM} = \mathrm{P}(W, n, \boldsymbol{\alpha}, [1, \ldots, 1], \langle [1], \ldots, [1] \rangle)$ in this protocol. The goal is to determine the work length $W$, the number of memory checkpoints $n$, and the relative lengths of the segments $\boldsymbol{\alpha}$ in the pattern. The following proposition shows the expected execution time of a pattern when these parameters are fixed.

**Proposition 2.** *The expected execution time of a given pattern* $\mathrm{P}(W, n, \boldsymbol{\alpha}, [1, \ldots, 1], \langle [1], \ldots, [1] \rangle)$ *is*

$$\mathbb{E}(\mathrm{P}) = W + n(V^* + C_M) + C_D + \left( \lambda_s \sum_{i=1}^{n} \alpha_i^2 + \frac{\lambda_f}{2} \right) W^2 + O(\sqrt{\lambda}) \ . \quad (8)$$

*Proof.* Define $E_i$ as the expected time to execute the $i$-th segment of the pattern up to the memory checkpoint at the end of the segment. We first show the following result:

$$E_i = w_i + V^* + C_M + \lambda_s w_i^2 + \lambda_f \left( \frac{w_i^2}{2} + \sum_{k=1}^{i-1} w_k w_i \right) + O(\sqrt{\lambda}) \ ,$$

where $w_i = \alpha_i W$ denotes the work length of the $i$-th segment.

We prove the above claim by induction on $i$. For the base case, the problem is reduced to the simple pattern shown in Section III-A, except that there is no disk checkpoint. Since we know from Theorem 1 that the work length of a pattern is in the order of $\lambda^{-1/2}$, we get the following result from Proposition 1: $E_1 = w_1 + V^* + C_M + \lambda_s w_1^2 + \frac{\lambda_f}{2} w_1^2 + O(\sqrt{\lambda})$. Suppose the claim holds up to $E_{i-1}$. Then, $E_i$ can be expressed recursively as follows:

$$E_i = p_i^f \left( \mathbb{E}(T_i^{\text{lost}}) + R_D + R_M + \sum_{k=1}^{i-1} E_k + E_i \right) + (1 - p_i^f)\big( w_i + V^* + p_i^s (R_M + E_i) + (1 - p_i^s) C_M \big) \ ,$$

where $\mathbb{E}(T_i^{\text{lost}})$ denotes the expected time loss during the execution of segment $i$ when a fail-stop error strikes, which according to Equation (3) is given by $\mathbb{E}(T_i^{\text{lost}}) = \frac{1}{\lambda_f} - \frac{w_i}{e^{\lambda_f w_i} - 1}$, and $p_i^f = 1 - e^{-\lambda_f w_i}$ and $p_i^s = 1 - e^{-\lambda_s w_i}$ denote the probabilities of having at least one fail-stop error and at least one silent error in segment $i$, respectively. By following the reasoning of the proof of Proposition 1, we obtain:

$$\begin{aligned}
E_i &= w_i + V^* + C_M + \lambda_s w_i^2 + \frac{\lambda_f}{2} w_i^2 + \lambda_f w_i \sum_{k=1}^{i-1} E_k + O(\sqrt{\lambda}) \\
&= w_i + V^* + C_M + \lambda_s w_i^2 + \frac{\lambda_f}{2} w_i^2 + \lambda_f w_i \sum_{k=1}^{i-1} (w_k + O(1)) + O(\sqrt{\lambda}) \\
&= w_i + V^* + C_M + \lambda_s w_i^2 + \lambda_f \left( \frac{w_i^2}{2} + \sum_{k=1}^{i-1} w_k w_i \right) + O(\sqrt{\lambda}) \ .
\end{aligned}$$

Now, we can compute the expected execution time of the pattern by summing up all the $E_i$'s as follows:

$$\begin{aligned}
\mathbb{E}(\mathrm{P}) &= \sum_{i=1}^{n} E_i + C_D = \sum_{i=1}^{n} w_i + n(V^* + C_M) + C_D \\
&\quad + \lambda_s \sum_{i=1}^{n} w_i^2 + \lambda_f \sum_{i=1}^{n} \left( \frac{w_i^2}{2} + \sum_{k=1}^{i-1} w_k w_i \right) + O(\sqrt{\lambda}) \\
&= W + n(V^* + C_M) + C_D + \left( \lambda_s \sum_{i=1}^{n} \alpha_i^2 + \frac{\lambda_f}{2} \right) W^2 + O(\sqrt{\lambda}) \ ,
\end{aligned}$$

since $\sum_{i=1}^{n} \left( w_i^2 + 2 \sum_{k=1}^{i-1} w_k w_i \right) = W^2$. □

**Theorem 2.** *A first-order approximation to the optimal parameters in pattern* $\mathrm{P}(W, n, \boldsymbol{\alpha}, [1, \ldots, 1], \langle [1], \ldots, [1] \rangle)$ *is given by* $\alpha_i^* = \frac{1}{n^*}$ *for* $1 \le i \le n^*$, $W^* = \sqrt{\frac{n^*(V^* + C_M) + C_D}{\frac{\lambda_s}{n^*} + \frac{\lambda_f}{2}}}$, *and* $n^*$ *is either* $\max(1, \lfloor \bar{n}^* \rfloor)$ *or* $\lceil \bar{n}^* \rceil$, *where* $\bar{n}^* = \sqrt{\frac{2\lambda_s}{\lambda_f} \cdot \frac{C_D}{V^* + C_M}}$. *The optimal expected overhead in this case is* $H^*(\mathrm{P}) = 2\sqrt{\lambda_s (V^* + C_M)} + \sqrt{2 \lambda_f C_D} + O(\lambda)$.

*Proof.* (Sketch) Given the number of segments $n$ and subject to $\sum_{i=1}^{n} \alpha_i = 1$, we know that $\sum_{i=1}^{n} \alpha_i^2$ is minimized when $\alpha_i = \frac{1}{n}$ for all $1 \le i \le n$. Hence, we can derive the two types of overheads from Proposition 2 as follows:

$$o_{\text{ef}} = n(V^* + C_M) + C_D \text{ and } o_{\text{rw}} = \frac{\lambda_s}{n} + \frac{\lambda_f}{2} \ .$$

For a given $n$, we can then retrieve the value of the optimal work length $W^* = \sqrt{\frac{o_{\text{ef}}}{o_{\text{rw}}}}$. Now, minimizing $F(n) = o_{\text{ef}} \cdot o_{\text{rw}} = \big( n(V^* + C_M) + C_D \big) \left( \frac{\lambda_s}{n} + \frac{\lambda_f}{2} \right)$, we get the optimal value of $\bar{n}^*$ as shown in the theorem. Since the number of segments can only be a positive integer, and $F(n)$ is a convex function of $n$, the optimal integer solution is either $\max(1, \lfloor \bar{n}^* \rfloor)$ or $\lceil \bar{n}^* \rceil$, whichever one leads to a smaller value of $F(n)$. Substituting all these values back into $H^*(\mathrm{P}) = 2\sqrt{o_{\text{ef}} \cdot o_{\text{rw}}} + O(\lambda)$, we obtain the optimal expected overhead. □

## C. General patterns

Due to lack of space, we refer to [7] for a detailed (and technical) proof for the characterization of the optimal pattern in the general case. We report all results in Table I, which contains six optimal patterns. In addition to $\mathrm{P}_D$ (one single-chunk segment, Section III-A) and $\mathrm{P}_{DM}$ (several one-chunk segments, Section III-B), we have:

| Pattern | $W^*$ | $n^*$ | $\boldsymbol{m}^*$ | $H^*(\mathrm{P})$ |
|---|---|---|---|---|
| $\mathrm{P}_D$ | $\sqrt{\frac{V^*+C_M+C_D}{\lambda_s+\frac{\lambda_f}{2}}}$ | – | – | $2\sqrt{\left(\lambda_s+\frac{\lambda_f}{2}\right)(V^*+C_M+C_D)}$ |
| $\mathrm{P}_{DV^*}$ | $\sqrt{\frac{m^*V^*+C_M+C_D}{\frac{1}{2}\left(1+\frac{1}{m^*}\right)\lambda_s+\frac{\lambda_f}{2}}}$ | – | $\sqrt{\frac{\lambda_s}{\lambda_s+\lambda_f}\cdot\frac{C_M+C_D}{V^*}}$ | $\sqrt{2(\lambda_s+\lambda_f)C_M+C_D}+\sqrt{2\lambda_s V^*}$ |
| $\mathrm{P}_{DV}$ | $\sqrt{\frac{(m^*-1)V+V^*+C_M+C_D}{\frac{1}{2}\left(1+\frac{2-r}{(m^*-2)r+2}\right)\lambda_s+\frac{\lambda_f}{2}}}$ | – | $2-\frac{2}{r}+\sqrt{\frac{\lambda_s}{\lambda_s+\lambda_f}\cdot\frac{2-r}{r}\left(\frac{V^*+C_M+C_D}{V}-\frac{2-r}{r}\right)}$ | $\sqrt{2(\lambda_s+\lambda_f)\left(V^*-\frac{2-r}{r}V+C_M+C_D\right)}+\sqrt{2\lambda_s\frac{2-r}{r}V}$ |
| $\mathrm{P}_{DM}$ | $\sqrt{\frac{n^*(V^*+C_M)+C_D}{\frac{\lambda_s}{n^*}+\frac{\lambda_f}{2}}}$ | $\sqrt{\frac{2\lambda_s}{\lambda_f}\cdot\frac{C_D}{V^*+C_M}}$ | – | $2\sqrt{\lambda_s(V^*+C_M)}+\sqrt{2\lambda_f C_D}$ |
| $\mathrm{P}_{DMV^*}$ | $\sqrt{\frac{n^*m^*V^*+n^*C_M+C_D}{\frac{1}{2}\left(1+\frac{1}{m^*}\right)\frac{\lambda_s}{n^*}+\frac{\lambda_f}{2}}}$ | $\sqrt{\frac{\lambda_s}{\lambda_f}\cdot\frac{C_D}{C_M}}$ | $\sqrt{\frac{C_M}{V^*}}$ | $\sqrt{2\lambda_f C_D}+\sqrt{2\lambda_s C_M}+\sqrt{2\lambda_s V^*}$ |
| $\mathrm{P}_{DMV}$ | $\sqrt{\frac{n^*(m^*-1)V+n^*(V^*+C_M)+C_D}{\frac{1}{2}\left(1+\frac{2-r}{(m^*-2)r+2}\right)\frac{\lambda_s}{n^*}+\frac{\lambda_f}{2}}}$ | $\sqrt{\frac{\lambda_s}{\lambda_f}\cdot\frac{C_D}{V^*-\frac{2-r}{r}V+C_M}}$ | $2-\frac{2}{r}+\sqrt{\frac{2-r}{r}\left(\frac{V^*+C_M}{V}-\frac{2-r}{r}\right)}$ | $\sqrt{2\lambda_f C_D}+\sqrt{2\lambda_s\left(V^*-\frac{2-r}{r}V+C_M\right)}+\sqrt{2\lambda_s\frac{2-r}{r}V}$ |

Table I

THE SIX OPTIMAL PATTERNS. $\mathrm{P}_D$, $\mathrm{P}_{DV^*}$ AND $\mathrm{P}_{DV}$ HAVE ONLY ONE LEVEL OF CHECKPOINTING, WHILE $\mathrm{P}_{DM}$, $\mathrm{P}_{DMV^*}$ AND $\mathrm{P}_{DMV}$ HAVE TWO LEVELS. THE TABLE REPORTS THE OPTIMAL PATTERN LENGTH $W^*$, THE OPTIMAL OVERHEAD $H^*(\mathrm{P})$ (IGNORING LOWER-ORDER TERMS), THE OPTIMAL NUMBER OF MEMORY CHECKPOINTS $n^*$ FOR THE TWO-LEVEL CHECKPOINTING PATTERNS, AND THE OPTIMAL NUMBER OF VERIFICATIONS $\boldsymbol{m}^*$ WITHIN A SEGMENT WHEN ADDITIONAL VERIFICATIONS ARE ADDED.

• $\mathrm{P}_{DV}$, the optimal pattern with a single segment that has multiple chunks in it. Each chunk ends with a partial verification, except the last one, which ends with a guaranteed verification followed by a memory checkpoint and a disk checkpoint (see Figure 2(b)). In the optimal solution, all chunks have same length, except the first and last one, which are larger.
• $\mathrm{P}_{DV^*}$, the variant of $\mathrm{P}_{DV}$ with only guaranteed verifications. For this variant, all chunks have same length.
• $\mathrm{P}_{DMV}$, the optimal pattern in the general case, with several multiple-chunk segments (see Figure 2(c)). In the optimal solution, all segments are identical. In each segment, all chunks have the same length, except the first and last one, which are larger.
• $\mathrm{P}_{DMV^*}$, the variant of $\mathrm{P}_{DMV}$ with only guaranteed verifications. For this variant, all chunks have the same length.

## IV. Performance evaluation

In this section, we conduct a set of simulations whose goal is twofold: (i) corroborate the theoretical study, and (ii) assess the relative efficiency of each checkpoint and verification type under realistic scenarios. We rely on simulations to evaluate the performance of the patterns at extreme scale, and we instantiate the model with three scenarios. In the first scenario, we compare all patterns using real parameters from the literature. The second scenario is a weak scaling experiment, whose purpose is to assess the scalability of the approach on increasingly large platforms. In the last scenario, we study the impact of varying error rates on the overhead. The simulator code is publicly available at http://graal.ens-lyon.fr/~yrobert/two-level.zip, so interested readers can experiment with it and build relevant scenarios of their choice.

### A. Simulation setup

We make several assumptions on the input parameters. First, we assume that the recovery cost is equivalent to the corresponding checkpointing cost, i.e., $R_D = C_D$ and $R_M = C_M$. This is reasonable because writing a checkpoint and reading one typically takes the same amount of time. Then, we assume that a guaranteed verification must check all the data in memory, making its cost in the same order as that of a memory checkpoint, i.e., $V^* = C_M$. Furthermore, we assume partial verifications similar to those proposed in [10], [3], [4], with very low cost while offering good recalls. In the following, we set $V = \frac{V^*}{100}$ and $r = 0.8$. All these choices are somewhat arbitrary and can easily be modified in the simulator; we believe they represent reasonable values for current and next-generation HPC applications.

The simulator generates errors following an exponential distribution of parameter $\lambda_f$ for fail-stop errors and $\lambda_s$ for silent errors. An experiment goes as follows. We feed the simulator with the description of the platform, consisting of the parameters $\lambda_f$, $\lambda_s$, $C_D$ and $C_M$ (since the other parameters can be deduced from the above assumptions). For each pattern, we compute the optimal length $W^*$, the optimal overhead $H^*(\mathrm{P})$, as well as the optimal number of memory checkpoints $n^*$ and the optimal number of verifications $\boldsymbol{m}^*$ (when applicable), using the formulas from Table I. The total amount of work for the application is set to that of $1000$ optimal patterns, and the simulator runs each experiment $1000$ times. For each pattern, it outputs the simulated overhead, the simulated number of disk checkpoints, memory checkpoints, verifications, disk recoveries and memory recoveries, by averaging the values from the $1000$ runs.

### B. Assessing resilience mechanisms on real platforms

In the first scenario, we assess the performance of the six optimal patterns shown in Table I on four different platforms with real parameter settings.

*1) Platform settings:* Table II presents the four platforms used in this experiment and their main parameters. These platforms have been used to evaluate the Scalable Checkpoint/Restart (SCR) library by Moody et al. [21], who provide accurate measurements for $\lambda_f$, $\lambda_s$, $C_D$ and $C_M$ using real applications. Note that the Hera platform has the worst error rates, with a platform MTBF of 12.2 days for fail-stop errors and 3.4 days for silent errors. In comparison, and despite its higher number of nodes, the Coastal platform features a platform MTBF of 28.8 days for fail-stop errors and 5.8 days for silent errors. In addition, the last platform uses SSD

| platform | #nodes | $\lambda_f$ | $\lambda_s$ | $C_D$ | $C_M$ |
|---|---|---|---|---|---|
| Hera | 256 | 9.46e-7 | 3.38e-6 | $300s$ | $15.4s$ |
| Atlas | 512 | 5.19e-7 | 7.78e-6 | $439s$ | $9.1s$ |
| Coastal | 1024 | 4.02e-7 | 2.01e-6 | $1051s$ | $4.5s$ |
| Coastal SSD | 1024 | 4.02e-7 | 2.01e-6 | $2500s$ | $180.0s$ |

Table II
PLATFORM PARAMETERS.

technology for memory checkpointing, which provides more data space, at the cost of higher checkpointing costs.

*2) Pattern overhead:* The first row of Figure 3 presents, for each pattern, the predicted overhead $H^*(\text{P})$ (in blue) versus the simulated one (in yellow) on each platform. Remember that the derivation of the expected overhead uses first-order approximation, ignoring some low-order terms in the computation. As a result, the predicted overhead, being a little optimistic, is always slightly less than the simulated one. However, the difference between the two is very small (less than $1\%$), which validates the model quite satisfactorily. Overall, the overhead oscillates between $4\%$ and $7\%$ on Hera, where checkpoints are relatively cheaper, to just over $15\%$ on Coastal SSD, where checkpoints are more expensive. Regardless of the platform, the more advanced patterns always result in smaller overheads. In particular, we observe a significant difference between the first three patterns ($\text{P}_D$,$\text{P}_{DV^*}$,$\text{P}_{DV}$), which use single-level checkpointing, and the last three patterns ($\text{P}_{DM}$,$\text{P}_{DMV^*}$,$\text{P}_{DMV}$), which use two-level checkpointing. The gap is more visible for Atlas ($5\%$) and Coastal ($4\%$), where the difference between the costs of a disk checkpoint and a memory checkpoint is larger, thus making memory checkpoints more valuable.

*3) Pattern periods:* The second row of Figure 3 shows the work lengths (periods) of the patterns on each platform. Single-level patterns are associated with shorter periods (around 3 hours on Hera and 10 hours on Coastal), as opposed to the longer periods shown by the two-level patterns (around 8 hours on Hera and 20 hours on Coastal). Indeed, when a fail-stop error strikes, the only choice is to recover from the last disk checkpoint, losing all the work done so far. In that case, a short period helps to mitigate the amount of time lost. However, silent errors are more prominent on these platforms and when a silent error occurs, two-level patterns can recover from an intermediate memory checkpoint instead. Not only does that provide a faster recovery, but also it does not require the application to restart from the very beginning of the pattern. As a result, disk checkpoints are only used for fail-stop errors, and a longer period is favored in order to accommodate more but cheaper intermediate memory checkpoints.

*4) Pattern checkpoints and verifications:* The third row of Figure 3 presents the average number of disk checkpoints, memory checkpoints and verifications taken by each pattern. We take all checkpoints and verifications into account, including the ones performed in recoveries and re-executions. Since a partial verification is much cheaper than a guaranteed one, the two patterns that are allowed to use them ($\text{P}_{DV}$ and $\text{P}_{DMV}$) tend to take full advantage of this mechanism. On Hera, $\text{P}_{DV}$ generates an average of 13 verifications per hour, which is slightly more than its two-level counterpart ($\text{P}_{DMV}$), which generates 12 verifications per hour. On Coastal, there are more than 20 verifications per hour for $\text{P}_{DV}$ and 19 for $\text{P}_{DMV}$. For the two-level patterns, whose periods are longer, the disk checkpointing frequencies become lower. However, their memory checkpointing frequencies are higher, because the cheaper memory checkpoints are favored in these two-level schemes in order to better protect the application from silent errors. Lastly, we observe that the Coastal SSD platform requires very few verifications and memory checkpoints. This is because the cost of a memory checkpoint is much larger on this platform ($180s$) as opposed to the costs on other platforms ($15.4s$ on Hera and $4.5s$ on Coastal).

*5) Pattern recoveries:* The last row of Figure 3 shows the number of recoveries performed per day by each pattern on each platform. The number of disk recoveries follows closely the fail-stop error rate of a given platform, and it is not affected by the pattern used. Indeed, when a fail-stop error strikes, a disk recovery is performed regardless of the pattern. On Hera, we observe $0.083$ disk recovery per day on average, translating to approximately one recovery every 12 days, which is in accordance with the platform MTBF of 12.2 days for fail-stop errors. The same applies to Atlas and Coastal, which show respectively $0.044$ and $0.034$ disk recoveries per day on average (equivalent to a platform MTBF of 22 days and 29 days). The number of memory recoveries is more complicated to analyze, because a memory recovery is not performed immediately after the occurrence of a silent error. Instead, it is performed only when an alarm is raised by a verification, or when a fail-stop error strikes. In both cases, more than one silent error could have occurred before the memory recovery. In the latter case, a memory recovery is triggered right after a disk recovery, possibly without any silent error. In general, the memory recovery frequency could well depend on the pattern used, which explains the slight difference under different patterns. Nevertheless, the simulation results show that the silent error rate is a good indicator of the memory recovery frequency. For instance, on Hera, we observe $0.285$ memory recovery per day on average, which is approximately one memory recovery every 3.5 days. This is very close to the MTBF of 3.4 days for silent errors.

## C. Weak scaling experiment

We now present the results of a weak scaling experiment to assess the scalability of the model. This experiment is based on the Hera platform, whose disk checkpoint cost is the closest to state-of-the-art platforms (5 minutes).

*1) Platform settings:* We first calculate the MTBF of one computing node, which is 8.57 years for fail-stop errors and 2.4 years for silent errors. The platform MTBF is obtained by dividing the per-node MTBF by the number of nodes. For example, when $2^{17}$ nodes are used, the MTBF decreases to $2064s$ for fail-stop errors and $577s$ for silent errors. Under weak scaling, the problem size grows linearly with the number of nodes, so the time to perform a memory checkpoint $C_M$ remains constant. In addition, we make the optimistic

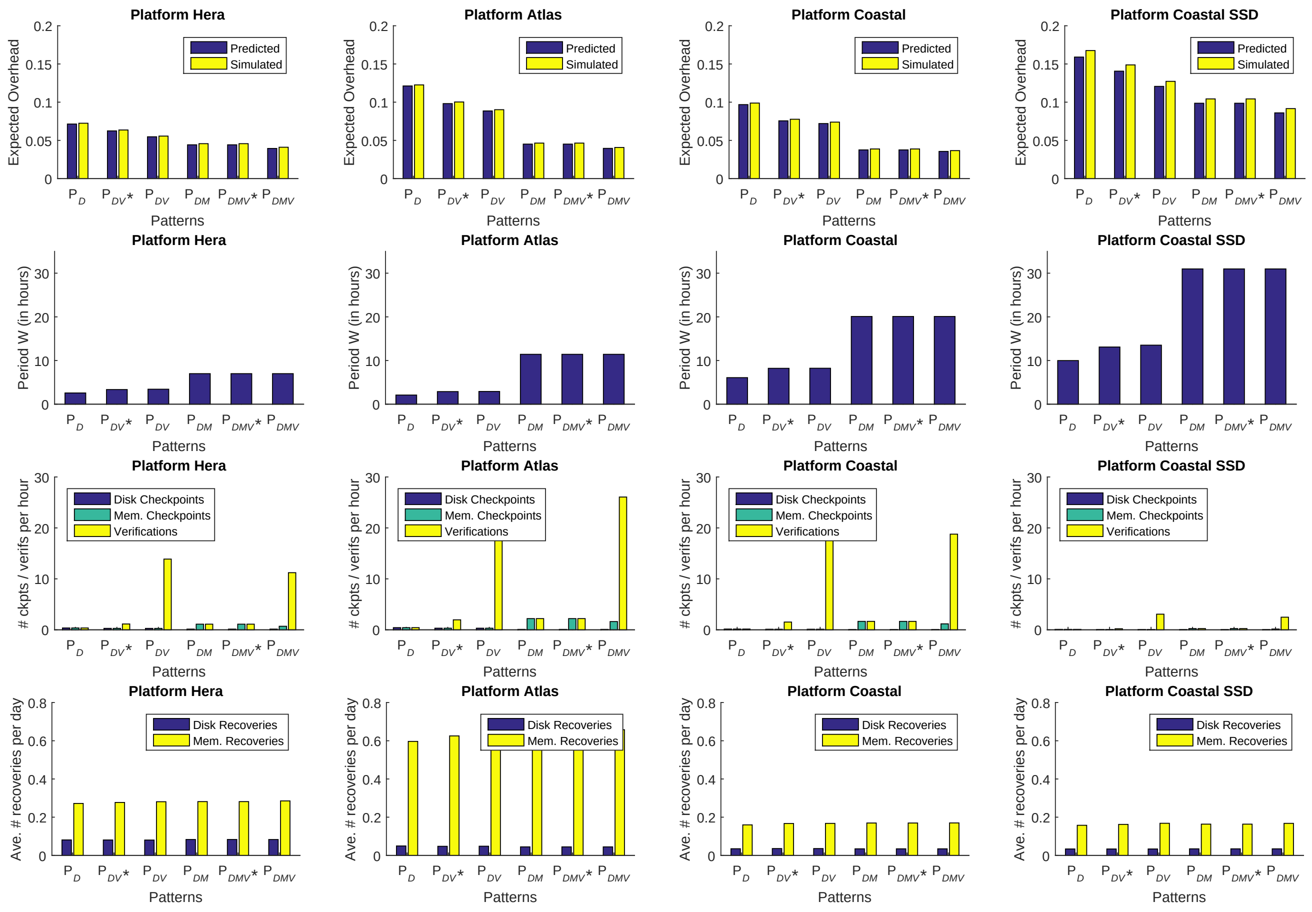


Figure 3. Performance of the six optimal patterns on the four platforms. Each column represents one platform.

assumption that the disk checkpointing time $C_D$ also remains constant by scaling the I/O bandwidth of the file system[2].

*2) Results:* Figure 4 presents the simulation results for the simplest pattern $P_D$ and the most advanced pattern $P_{DMV}$. We can see that the overheads remain acceptable up to $2^{15} = 32768$ nodes, which are $100\%$ for $P_D$ and $64\%$ for $P_{DMV}$. Beyond that, the overheads increase drastically for both patterns, eventually exceeding $500\%$ for $2^{18} = 262144$ nodes. However, compared to the simple pattern $P_D$, the two-level pattern $P_{DMV}$ improves the overhead by a few percent on 256 nodes up to over $150\%$ on $2^{18}$ nodes.

We also observe the difference between the simulated overhead and the predicted one, which starts negligible for a small number of nodes but reaches more than a factor of three for $2^{18}$ nodes. The reason is the use of first-order approximation to compute the predicted overhead, which is only accurate when the platform MTBF is large in front of the other parameters. Obviously, this is no longer the case for a large number of nodes. For instance, when the number of nodes reaches $10^5$ (almost $2^{17}$ nodes), the MTBF of the whole platform reduces to less than 10 minutes, which is in the same order as the pattern period. At this point, the application experiences more than half a dozen errors per hour. In order to minimize the impact of errors, the pattern $P_{DMV}$ places a large number of verifications and more than 10 memory checkpoints per hour. As a result, a lot of time is wasted on resilience operations. When the error rate is this high, however, there is not much flexibility in the optimization, and no pattern is able to offer satisfying performance.

Similar results are also obtained when we repeat the weak scaling experiment with a disk checkpointing cost of $90s$ instead of $300s$ to account for improved disk technology [7].

## D. Impact of error rates

Finally, we study the impact of the error rates on the performance of the patterns. Again, we focus on the Hera platform but scale its number of nodes to $10^5$. We vary the error rates $\lambda_f$ and $\lambda_s$ with respect to their nominal values while keeping the other parameters fixed.

The first row of Figure 5 presents the impact of $\lambda_f$ and $\lambda_s$ on the simulated overheads of two patterns $P_D$ and $P_{DMV}$. For the $P_{DMV}$ pattern, the overhead is affected more by the fail-stop errors than by the silent errors. This is because the intermediate memory checkpoints better protect the application from silent errors. On the other hand, the overhead of the single-level pattern $P_D$ is affected more by the silent errors,

[2]In actual systems, the I/O bandwidth could become a bottleneck, resulting in increased disk checkpointing cost. This would further widen the performance gap between single-level and two-level patterns.

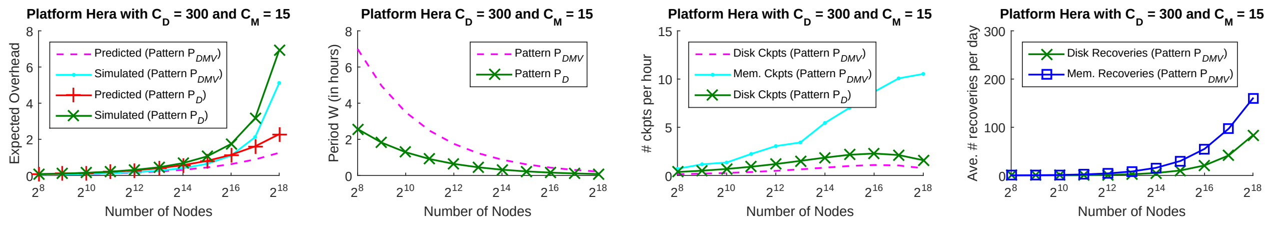


Figure 4. Weak scaling experiment on the Hera platform.

because silent errors have a much higher rate. The figure also shows the difference between the overheads of the two patterns, which is small when most errors are fail-stop, due to the relatively small rate. However, when the silent error rate increases, the two-level pattern achieves a much better performance than the single-level pattern, by saving up to $200\%$ on the overhead.

The second row of Figure 5 presents the impacts of $\lambda_f$ and $\lambda_s$ separately on the periods and checkpointing frequencies of the two patterns. First, when the silent error rate is fixed at its nominal value, the period of $P_D$ remains constant with varying $\lambda_f$, while the period of $P_{DMV}$ decreases as $\lambda_f$ increases. This is because the high silent error rate has already driven the period of $P_D$ very low ($< 10$ minutes), so increasing the fail-stop error rate has a limited impact. On the other hand, the period of $P_{DMV}$ is primarily driven by the fail-stop error rate, so it decreases quickly, allowing more disk checkpoints to be taken. In addition, the number of checkpoints successfully taken in an hour remains stable for both patterns. Since the period of $P_{DMV}$ decreases while the period of $P_D$ remains constant, it implies degraded performance for the two-level pattern and stable performance for the single-level one, corroborating the previous analysis. The role is reversed when the fail-stop error rate is fixed at the nominal value and the silent error rate is changed. Since the $P_{DMV}$ pattern is equipped with more memory checkpoints and verifications, silent errors have little impact on its period. On the contrary, the period of $P_D$ decreases in order to detect silent errors earlier, which is the only way to protect the application from increased silent error rate. Lastly, the number of memory checkpoints performed by $P_{DMV}$ increases with the silent error rate in order to compensate for the fixed number of disk checkpoints. For the $P_D$ pattern, the checkpointing frequency remains the same, implying degraded performance with decreased period.

### *E. Summary*

From the simulation results, we conclude that the first-order approximation for the resilience patterns provides an accurate performance model for systems with up to tens of thousands of nodes. Overall, the complex pattern that combines all resilience mechanisms offers significantly better performance, improving the base pattern by up to 150% in the execution overhead. The findings are consistent on different platforms and with varying error rates. The results nicely corroborate the analytical study, and demonstrate the benefit of using two-level patterns for dealing with both fail-stop and silent errors.

## V. Related work

Due to lack of space, we refer to [7] for a detailed list of related work on multi-level checkpointing and silent error detection. Here we only outline differences with our previous work. In a nutshell, our previous work investigates the design of special patterns. Aupy et al. [1] analyzed two simple patterns: one with $k$ checkpoints and one guaranteed verification, and the other with $k$ verifications and one checkpoint. Benoit et al. [6] studied the latter pattern and gave explicit formulas to accommodate both fail-stop and silent errors. The idea of interleaving $p$ checkpoints and $q$ verifications has also been explored in [8] to achieve more optimized computing patterns. The first analysis of a pattern using partial verification for silent error detection was given by Cavelan et al. [14]. This analysis has been recently extended to the case with multiple partial verifications [2]. All these results apply to a single level of checkpointing only, which considerably simplifies the design of optimal solutions. To the best of our knowledge, this work is the first to investigate the combination of memory checkpoints, disk checkpoints, partial verifications and guaranteed verifications.

Finally, we stress that this work is along the same direction as multi-level checkpointing, but the two levels we propose target different error sources, namely, fail-stop errors and silent errors. This dramatically changes the computation of the expected re-execution time, because we do not have to distinguish which error type strikes first. Moreover, as in Young [23] and Daly [16], we provide explicit formulas on the optimal checkpointing intervals for both levels (up to a first-order approximation), while previous work relies on numerical methods to find the optimal solution [17].

## VI. Conclusion

When computing at extreme scale, both fail-stop errors and silent errors are major threats to executing HPC applications with acceptable overhead. While several techniques have been developed to cope with either threat, few approaches are devoted to addressing both of them simultaneously. Although surprising—because dealing with both error sources is unavoidable on large-scale platforms—, this lack of solutions may be explained by the new challenges raised by silent errors, whose detection is not immediate and requires the use of verification mechanisms, either partial or guaranteed. Also, the interplay of two levels of checkpoints and of two types of verifications raises difficult optimization challenges. The major contribution of this paper is the characterization of the

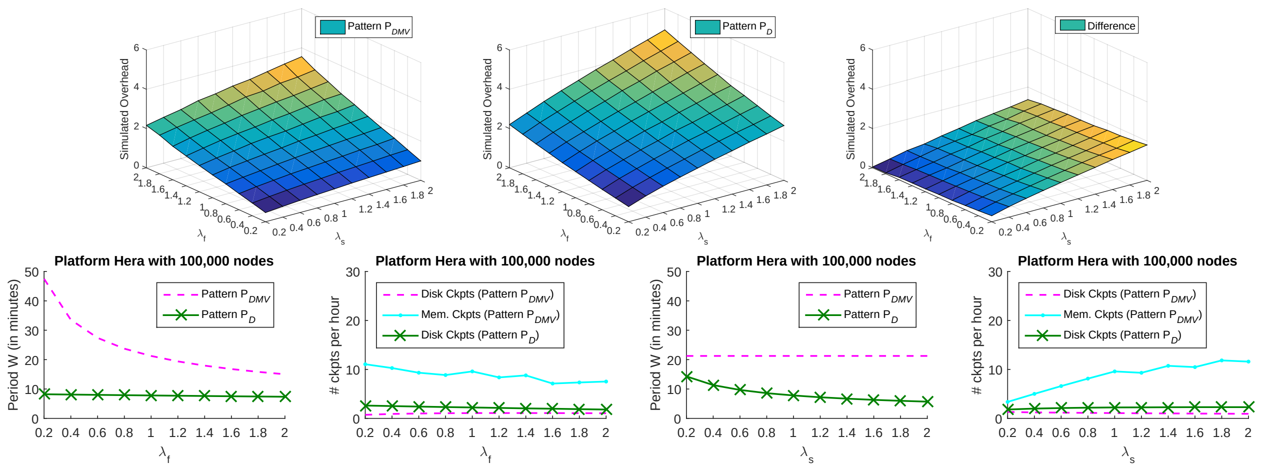


Figure 5. Impact of error rates $\lambda_f$ and $\lambda_s$ on the performance of the patterns on the Hera platform with $10^5$ nodes.

optimal computational pattern. The derivation is technically involved, but the results are easy to use in real-life scenarios: one has just to look at Table I and pick the optimal pattern that fits their resilience needs. The accuracy of our model as well as the analysis have been nicely corroborated by extensive simulations. The results show acceptable difference in the predicted overhead and the simulated one on systems with up to tens of thousands of nodes. Also, the complex pattern that combines all resilience mechanisms provides up to 150% improvement in the execution overhead compared to the base pattern dictated by the classical Young/Daly's formula.

Finally, our approach is application-agnostic. Future work will be devoted to the study of application-specific verification and checkpoint mechanisms, in particular for sparse iterative solvers. It will be interesting to assess the impact of ad-hoc techniques (ABFT, orthogonality checks, incremental checkpointing, etc) on the overhead of computational patterns in the latter framework.

*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 *Proc. 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? In *Proc. HiPC*, 2015. Available as INRIA RR-8741.

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

[4] L. Bautista Gomez and F. Cappello. Detecting and correcting data corruption in stencil applications through multivariate interpolation. In *Proc.1st Int. Workshop on Fault Tolerant Systems (FTS)*, 2015.

[5] L. Bautista-Gomez, S. Tsuboi, D. Komatitsch, F. Cappello, N. Maruyama, and S. Matsuoka. FTI: High performance fault tolerance interface for hybrid systems. In *Proc. SC'11*, 2011.

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

[7] A. Benoit, A. Cavelan, Y. Robert, and H. Sun. Optimal resilience patterns to cope with fail-stop and silent errors. Research report RR-8786, INRIA, 2015. Available at graal.ens-lyon.fr/~yrobert/rr8786.pdf. Short version appears in IPDPS'15.

[8] A. Benoit, Y. Robert, and S. K. Raina. Efficient checkpoint/verification patterns. *Int. J. High Performance Computing Applications*, 2015.

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

[10] 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. HPDC*, 2015.

[11] 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.

[12] F. Cappello, A. Geist, B. Gropp, L. Kale, B. Kramer, and M. Snir. Toward Exascale Resilience. *Int. Journal of High Performance Computing Applications*, 23(4):374–388, 2009.

[13] F. Cappello, A. Geist, W. Gropp, S. Kale, B. Kramer, and M. Snir. Toward exascale resilience: 2014 update. *Supercomputing frontiers and innovations*, 1(1), 2014.

[14] A. Cavelan, S. K. Raina, Y. Robert, and H. Sun. Assessing the impact of partial verifications against silent data corruptions. In *Proc. ICPP*, 2015.

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

[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] S. Di, M. S. Bouguerra, L. Bautista-Gomez, and F. Cappello. Optimization of multi-level checkpoint model for large scale HPC applications. In *Proc. IPDPS'14*, 2014.

[18] 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.

[19] T. Hérault and Y. Robert, editors. *Fault-Tolerance Techniques for High-Performance Computing*, Computer Communications and Networks. Springer Verlag, 2015.

[20] 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.

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

[22] P. Sao and R. Vuduc. Self-stabilizing iterative solvers. In *Proc. ScalA*, 2013.

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