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

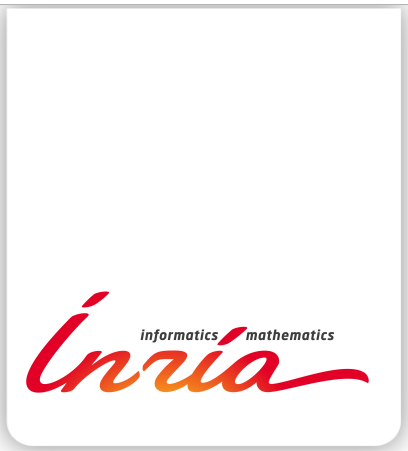
Anne Benoit, Aurélien Cavelan, Yves Robert, Hongyang Sun

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

Anne Benoit, Aurélien Cavelan, Yves Robert, and Hongyang Sun

RESEARCH REPORT
N° 8786
October 2015
Project-Team ROMA

ISSN 0249-6399 ISRN INRIA/RR--8786--FR+ENG



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

Anne Benoit*†, Aurélien Cavelan*†, Yves Robert*†‡, and Hongyang Sun*†

Project-Team ROMA

Research Report n° 8786 — October 2015 — 35 pages

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.

Key-words: resilience, fail-stop errors, silent errors, multi-level checkpoint, verification, optimal pattern.

* École Normale Supérieure de Lyon
† INRIA, France
‡ University of Tennessee Knoxville, USA

RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Schémas de résilience optimaux pour traiter les erreurs fatales et les erreurs silencieuses

Résumé : Ce travail s'intéresse aux techniques de résilience à très grande échelle. De nombreux articles considèrent les erreurs fatales, et d'autres considèrent les erreurs silencieuses. Mais peu considèrent que les deux types d'erreurs peuvent co-exister. Cependant, les applications de calcul haute performance devront nécessairement faire face aux deux sources d'erreurs. Ce travail présente un modèle et des solutions algorithmiques à ce double défi. Les erreurs silencieuses sont traitées grâce à des mécanismes de vérification (partiellement ou complètement précis) et des checkpoints en mémoire. Des checkpoints sur disques protègent des erreurs fatales. Tous les types de vérification et checkpoints sont combinés dans un schéma de calcul. Nous donnons un modèle et une caractérisation complète du schéma optimal. Nos résultats étendent de nombreuses solutions déjà publiées, et montrent comment utiliser différentes techniques pour faire face à la double menace des fautes fatales et silencieuses. Des simulations complètes, basées sur des données réelles, confirment la précision du modèle, et montrent que les schémas combinant tous les mécanismes de résilience sont nécessaires pour obtenir des surcoûts acceptables.

Mots-clés : résilience, erreurs fatales, erreurs silencieuses, checkpoint multi-niveaux, vérification, schéma optimal.

1 Introduction

Fault-tolerance techniques are mandatory at extreme scale [20, 13, 14]. 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 [16, 22], and many protocols are available (see [27] for a survey, as well as Section 7 on related work).

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 [31, 38, 29, 14]. 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 [17, 33, 7]. This verification mechanism can be general-purpose (e.g., based on replication [24] or even triplication [28]) or application-specific (e.g., based on Algorithm-based fault tolerance (ABFT) [26, 10, 34], on approximate re-execution for ODE and PDE solvers [8], or on orthogonality checks for Krylov-based sparse solvers [17, 33]).

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 [3, 14]. 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 [9]. 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

[1]The MTBF μ_p with p resources is $\mu_p = \mu_{\text{ind}}/p$, where μ_{ind} is the MTBF of each resource, see [27, Proposition 1.2].

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 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 sophisticated heuristics [19].

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 size) that are separated by partial verifications. See Figure 2 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 (see Section 2).
- 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 1 extends and unifies many results from the literature (see the discussion in Section 7).
- 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 2 introduces the model and notation. The following sections show how to determine the optimal pattern. We start with the simplest

pattern (a single one-chunk segment) in Section 3, extending Young and Daly's formula to two error sources. We discuss patterns with multiple one-chunk segments in Section 4.1, patterns with one multiple-chunk segment in Section 4.2, and finally the most general pattern in Section 4.3. Section 5 deals with errors during checkpoints and recoveries. Simulation results are presented in Section 6. Section 7 surveys related work. Finally, Section 8 provides concluding remarks and hints for future directions.

2 Model

2.1 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 λ_f and λ_s, respectively. Hence, 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 the probability 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 μ while accounting for both types of failures.

2.2 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 1. 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. In this case, we 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 check-

point and one disk checkpoint need to be maintained at any time during the execution of the application.

To simplify the analysis, we assume in Sections 3 and 4 that errors only strike the computations, while verifications, memory copies, and I/O transfers are protected from failures. In Section 5, we show how this assumption can be relaxed in the analysis.

2.3 Notation

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 [15, 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 an 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, 9]. This characteristic makes partial verification a highly attractive technique for detecting silent errors. Hence, we make use of partial verifications between memory checkpoints in the pattern.

For clarity, we refer to the computation between any two consecutive memory checkpoints as a *segment*, and refer to the computation between two consecutive verifications as a *chunk*. Formally, a pattern $\mathrm{P}(W, n, \boldsymbol{\alpha}, \boldsymbol{m}, \langle \boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_n \rangle)$ 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 computational segments within the pattern).
- $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, \ldots, \alpha_n]$: 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, we have $\sum_{i=1}^{n} \alpha_i = 1$.
- $\boldsymbol{m} = [m_1, m_2, \ldots, m_n]$: number of verifications inside each segment of the pattern (also number of chunks in that segment).
- $\boldsymbol{\beta}_i = [\beta_{i,1}, \beta_{i,2}, \ldots, \beta_{i,m_i}]$ $\forall i = 1, 2, \ldots, n$: 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 of the pattern. Hence, we have $\sum_{j=1}^{m_i} \beta_{i,j} = 1$ for all $i = 1, 2, \ldots, n$.

The simplest pattern is illustrated in Figure 1, and 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 P_D (only disk checkpoints, which are always preceded by a guaranteed verification and a memory checkpoint).

Figure 2 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.

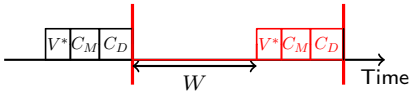


Figure 1: Pattern $\mathrm{P}_D = \mathrm{P}(W, 1, [1], [1], \langle [1] \rangle)$.

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. Additionally, silent errors may be detected earlier in the first and third segment thanks to the additional partial verifications.

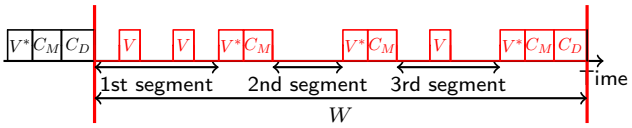


Figure 2: Pattern with three segments and six chunks.

2.4 Objective

The objective is to find a pattern that minimizes the expected execution time of the application. Let W_{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). Suppose the execution is divided into periodic patterns, defined by $\mathrm{P}(W, n, \boldsymbol{\alpha}, \boldsymbol{m}, \langle \boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_n \rangle)$. Let $\mathbb{E}(\mathrm{P})$ be the expected execution time of the pattern. For large jobs, the expected makespan W_{final} of the application when taking failures into account can then be approximated by:

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

Now, define $H(\mathrm{P}) = \frac{\mathbb{E}(\mathrm{P})}{W} - 1$ to be the expected *overhead* of the pattern. We obtain $W_{\text{final}} \approx W_{\text{base}} + H(\mathrm{P}) \times W_{\text{base}}$. Thus, minimizing the expected makespan is equivalent to minimizing the pattern overhead $H(\mathrm{P})$. Hence, we will focus on minimizing the pattern overhead in this paper.

3 Revisiting Young and Daly

In this section, we revisit Young [37] and Daly [18] on computing the best 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.

3.1 Optimal disk checkpointing interval

The classical formula by Young and Daly gives the optimal disk checkpointing interval without considering silent errors, thus does not include verification 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).

Obviously, the only parameter to determine is the work length W, which is also referred to as the checkpointing period by Young/Daly [37, 18]. 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*

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

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 can be expressed using the following recursive formula:

$$\begin{aligned}\mathbb{E}(\mathrm{P}) = p^f \Big(\mathbb{E}(T^{\mathrm{lost}}) + R_D + R_M + \mathbb{E}(\mathrm{P})\Big) \\ + (1 - p^f)\big(W + V^* + p^s(R_M + \mathbb{E}(\mathrm{P})) \\ + (1 - p^s)(C_M + C_D)\big) \ , \end{aligned} \tag{2}$$

where $\mathbb{E}(T^{\mathrm{lost}})$ 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 $\mathbb{E}(T^{\mathrm{lost}})$ 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 $\mathbb{E}(T^{\mathrm{lost}})$, which can be expressed as follows:

$$\begin{aligned}\mathbb{E}(T^{\mathrm{lost}}) &= \int_0^\infty x \mathbb{P}(X = x | X < W) dx \\ &= \frac{1}{\mathbb{P}(X < W)} \int_0^W x \mathbb{P}(X = x) dx \ , \end{aligned}$$

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

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

Now, substituting Equation (3) into Equation (2) and simplifying, we obtain

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

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*

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

The optimal expected overhead is

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

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

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

Assume that the platform MTBF $\mu = 1/\lambda$ is large in front of the resilience parameters. Then 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 of order $\Theta(\lambda^{1/2})$, 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 in the above expression, which gives Equation (4). Then, by substituting W^* back into $H(\mathrm{P})$, we get the optimal expected overhead as shown by Equation (5). □

Remarks. When only fail-stop errors exist, there is no need to perform verification and memory checkpointing: we retrieve the classical formula by Young [37] and Daly[18], which is given by $W^* = \sqrt{2C_D/\lambda_f}$. When there are only silent errors, we do not need to perform disk checkpointing, and the optimal work length is given by $W^* = \sqrt{(V^* + C_M)/\lambda_s}$.

3.2 Observations

First, we observe from Theorem 1 that the optimal work length W^* of a pattern is in the order of $\Theta\left(\lambda^{-1/2}\right)$ and the optimal overhead $H^*(\mathrm{P})$ is in the order of $\Theta(\lambda^{1/2})$. This allows us to express the expected execution overhead of a pattern in the following general form:

$$H(\mathrm{P}) = \frac{o_{\mathrm{ef}}}{W} + o_{\mathrm{rw}} W + O(\lambda) \ , \tag{7}$$

where o_{ef} and o_{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 $o_{\mathrm{ef}} = V^* + C_M + C_D$ and $o_{\mathrm{rw}} = \lambda_s + \frac{\lambda_f}{2}$, respectively.

Therefore, from Equation (7), the optimal pattern length and the optimal expected overhead can be expressed as

$$W^* = \sqrt{\frac{o_{\mathrm{ef}}}{o_{\mathrm{rw}}}}\ , \tag{8}$$

$$H^*(\mathrm{P}) = 2\sqrt{o_{\mathrm{ef}} \times o_{\mathrm{rw}}} + O(\lambda)\ . \tag{9}$$

We can see that minimizing the expected execution overhead $H(\mathrm{P})$ of a pattern becomes equivalent to minimizing the product $o_{\mathrm{ef}} \times o_{\mathrm{rw}}$ up to the dominating term, which is coherent with previous work [15, 2]. 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 that finds the optimal tradeoff between o_{ef} and o_{rw}. We will make use of this observation in the next section to derive the optimal patterns in more complicated protocols.

4 Optimal patterns

In this section, we derive the optimal pattern that involves two levels of checkpointing coupled with verifications. We start with simpler patterns that do not contain any intermediate verification nor memory checkpoint, and then move on to settle the complete full pattern.

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

We first consider a pattern that contains multiple segments, but each segment has only one chunk. In other words, the protocol performs multiple memory checkpoints between two disk checkpoints but without any intermediate verification. Figure 3 depicts the pattern P_{DM} in this protocol.

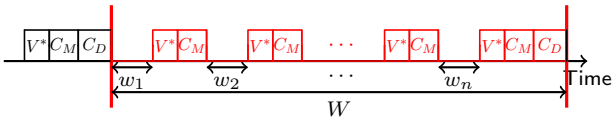


Figure 3: Pattern $\mathrm{P}_{DM} = \mathrm{P}(W, n, \boldsymbol{\alpha}, [1, \ldots, 1], \langle [1], \ldots, [1] \rangle)$.

The goal is to determine the pattern work length W, the number of memory checkpoints n, and the relative lengths of the segments $\boldsymbol{\alpha}$ inside 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*

$$\begin{aligned} \mathbb{E}(\mathrm{P}) &= W + n(V^* + C_M) + C_D \\ &\quad + \left(\lambda_s \sum_{i=1}^{n} \alpha_i^2 + \frac{\lambda_f}{2} \right) W^2 + O(\sqrt{\lambda})\ . \end{aligned} \tag{10}$$

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 on E_i:

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

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 3.1, 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 $\Theta\left(\lambda^{-1/2}\right)$, 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:

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

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 and p_i^s 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 compute the expected execution time of the pattern by summing up all the E_i's as follows:

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

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

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*

$$\alpha_i^* = \frac{1}{n^*} \text{ for } 1 \leq i \leq n^* \ , \tag{11}$$

$$W^* = \sqrt{\frac{n^*(V^* + C_M) + C_D}{\frac{\lambda_s}{n^*} + \frac{\lambda_f}{2}}} \ , \tag{12}$$

and n^* *is either* $\max(1, \lfloor \bar{n}^* \rfloor)$ *or* $\lceil \bar{n}^* \rceil$*, where*

$$\bar{n}^* = \sqrt{\frac{2\lambda_s}{\lambda_f} \cdot \frac{C_D}{V^* + C_M}} \ . \tag{13}$$

The optimal expected overhead is

$$H^*(\mathrm{P}) = 2\sqrt{\lambda_s (V^* + C_M)} + \sqrt{2\lambda_f C_D} + O(\lambda) \ . \tag{14}$$

Proof. 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 $1 \leq i \leq n$. Hence, we can derive the two types of overheads from Proposition 2 as follows:

$$\begin{aligned}
o_{\mathrm{ef}} &= n(V^* + C_M) + C_D \ , \\
o_{\mathrm{rw}} &= \frac{\lambda_s}{n} + \frac{\lambda_f}{2} \ .
\end{aligned}$$

For a given n, the optimal work length $W^* = \sqrt{\frac{o_{\mathrm{ef}}}{o_{\mathrm{rw}}}}$ is therefore given by Equation (12). Now, minimizing $F(n) = o_{\mathrm{ef}} \times o_{\mathrm{rw}} = (n(V^* + C_M) + C_D) \left(\frac{\lambda_s}{n} + \frac{\lambda_f}{2} \right)$, we get the optimal value of $\bar{n}^*$ as shown in Equation (13). 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 Equation (13) back into $H^*(\mathrm{P}) = 2\sqrt{o_{\mathrm{ef}} \times o_{\mathrm{rw}}}$, we obtain the optimal expected overhead as shown in Equation (14). □

Remarks. We can see why the analysis conducted here is different from multi-level checkpointing with two levels of fail-stop errors. In the latter case, one has to make a case study depending on which error type strikes first, while in our case, silent errors do not interrupt the execution and are detected at the end of the segments.

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

We now consider a pattern that contains only one segment, which 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. Figure 4 depicts the pattern P_{DV} in this protocol.

For simplicity, let m (instead of m_1) denote the number of chunks in the pattern, and let w_j (instead of $w_{1,j}$) denote the length of the j-th chunk for $1 \leq j \leq m$. We define $\beta_j = w_j/W$. The goal is to determine the pattern work length W, the number of chunks m as well as their relative lengths $\boldsymbol{\beta}$.

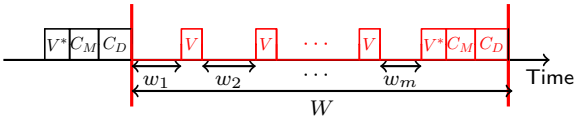


Figure 4: Pattern $\mathrm{P}_{DV} = \mathrm{P}(W, 1, [1], [m], \langle \boldsymbol{\beta} \rangle)$.

Proposition 3. *The expected execution time of a given pattern* $\mathrm{P}(W, 1, [1], [m], \langle \boldsymbol{\beta} \rangle)$ *is*

$$\mathbb{E}(\mathrm{P}) = W + (m-1)V + V^* + C_M + C_D + \left(\lambda_s \boldsymbol{\beta}^T A \boldsymbol{\beta} + \frac{\lambda_f}{2} \right) W^2 + O(\sqrt{\lambda}) \ , \tag{15}$$

where A is an $m \times m$ symmetric matrix defined by

$$A_{i,j} = \frac{1}{2} \left(1 + (1-r)^{|i-j|} \right) \ , \tag{16}$$

for all $1 \leq i, j \leq m$.

Proof. We first define some notations to be used in the proof. Let $p_j^f = 1 - e^{-\lambda_f w_j}$ and $p_j^s = 1 - e^{-\lambda_s w_j}$ denote the probabilities of having at least one fail-stop error and at least one silent error in chunk j, respectively. Let V_j denote the cost of the verification right after chunk j, so we have $V_j = V$ for $1 \leq j \leq m-1$ and $V_m = V^*$. Finally, let $\mathbb{E}(T_j^{\text{lost}})$ denote the expected time loss during the execution of chunk j if a fail-stop error strikes in this chunk. Based on Equation (3), we have $\mathbb{E}(T_j^{\text{lost}}) = \frac{1}{\lambda_f} - \frac{w_j}{e^{\lambda_f w_j} - 1}$.

To derive the expected execution time of the pattern, we need to know the probability that chunk j actually gets executed in the current attempt. Let q_j denote this probability; we compute it as follows. The first chunk is always executed, so we have $q_1 = 1$. Consider the second chunk, which is executed when there is no fail-stop error and no silent error in the first chunk. However, for silent errors that did occur in the first chunk, the partial verification V_1 may have missed them with probability $1 - r$. In this case, the second chunk also gets

executed. Hence, we have $q_2 = (1-p_1^f)\big((1-p_1^s)+p_1^s(1-r)\big)$. In general, the probability that the j-th chunk gets executed can be written as:

$$q_j = \left(\prod_{k=1}^{j-1}(1-p_k^f)\right)\left(\prod_{k=1}^{j-1}(1-p_k^s)+g_j\right) ,$$

where g_j denotes the probability that silent errors actually occurred before chunk j, but have been missed by all the partial verifications up to V_{j-1}, thus enabling chunk j to be executed. By enumerating all possible locations where silent errors could strike, we can express g_j as:

$$g_j = \sum_{\ell=1}^{j-1}\left(\prod_{k=1}^{\ell-1}(1-p_k^s)\right)p_\ell^s(1-r)^{j-\ell} .$$

Now, we are ready to compute the expected execution time of the pattern. The following gives the recursive expression:

$$\begin{aligned}\mathbb{E}(\mathrm{P}) = &\left(\prod_{k=1}^{m}(1-p_k^f)(1-p_k^s)\right)(C_M+C_D)\\ &+\left(1-\prod_{k=1}^{m}(1-p_k^f)(1-p_k^s)\right)(R_M+\mathbb{E}(\mathrm{P}))\\ &+\sum_{j=1}^{m}q_j\left(p_j^f\left(\mathbb{E}(T_j^{\mathrm{lost}})+R_D\right)+(1-p_j^f)(w_j+V_j)\right) . \qquad (17)\end{aligned}$$

Specifically, Line 1 of Equation (17) shows that the memory and disk checkpoints at the end of the pattern are performed only when neither fail-stop nor silent error has occurred in all chunks. Under all the other cases, we need to re-execute the pattern as shown in Line 2. Regardless of the type of error that triggered the re-execution, we always need to restore the memory checkpoint. Finally, Line 3 shows the condition for each chunk j to be executed. The execution of the chunk is either completed or interrupted by a fail-stop error, in which case we lose $\mathbb{E}(T_j^{\mathrm{lost}})$ time and need to additionally restore the disk checkpoint.

By simplifying Equation (17) and approximating the expression up to the second-order term, as in the proofs of Propositions 1 and 2, we obtain

$$\begin{aligned}\mathbb{E}(\mathrm{P}) = &\, W+(m-1)V+V^*+C_M+C_D\\ &+\lambda_s f W^2+\frac{\lambda_f}{2}W^2+O(\sqrt{\lambda}) ,\end{aligned}$$

where $f = \sum_{j=1}^{m}\beta_j\left(\sum_{k=1}^{j-1}\beta_k(1-r)^{j-k}+\sum_{k=j}^{m}\beta_k\right)$, and it can be concisely written as $f = \boldsymbol{\beta}^T M\boldsymbol{\beta}$, where M is the $m\times m$ matrix given by

$$M_{i,j} = \begin{cases}1 & \text{for } i\le j\\ (1-r)^{i-j} & \text{for } i>j\end{cases} .$$

By replacing M by $A = \frac{M+M^T}{2}$, which does not affect the value of f, we obtain the symmetric matrix A in Equation (16) and the expected execution time in Equation (15). □

Theorem 3. *A first-order approximation to the optimal parameters in pattern* $\mathrm{P}(W, 1, [1], [m], \langle \boldsymbol{\beta} \rangle)$ *is given by*

$$\beta_j^* = \begin{cases} \frac{1}{(m^*-2)r+2} & \text{for } j = 1, m^* \\ \frac{r}{(m^*-2)r+2} & \text{for } 2 \leq j \leq m^*-1 \end{cases} , \tag{18}$$

$$W^* = \sqrt{\frac{(m^*-1)V + V^* + C_M + C_D}{\frac{1}{2}\left(1 + \frac{2-r}{(m^*-2)r+2}\right)\lambda_s + \frac{\lambda_f}{2}}} , \tag{19}$$

and m^* *is either* $\max(1, \lfloor \bar{m}^* \rfloor)$ *or* $\lceil \bar{m}^* \rceil$*, where*

$$\bar{m}^* = 2 - \frac{2}{r} + \sqrt{\frac{\lambda_s}{\lambda_s+\lambda_f}\frac{2-r}{r}\left(\frac{V^*+C_M+C_D}{V} - \frac{2-r}{r}\right)} . \tag{20}$$

The optimal expected overhead is

$$H^*(\mathrm{P}) = \sqrt{2(\lambda_s+\lambda_f)\left(V^* - \frac{2-r}{r}V + C_M + C_D\right)} + \sqrt{2\lambda_s\frac{2-r}{r}V} + O(\lambda) . \tag{21}$$

Proof. Given the number of chunks m and subject to $\sum_{j=1}^{m} \beta_j = 1$, it has been shown in [15] that function $f = \boldsymbol{\beta}^T A \boldsymbol{\beta}$ is minimized when $\boldsymbol{\beta}$ follows Equation (18), and its minimum value is given by $f^* = \frac{1}{2}\left(1 + \frac{2-r}{(m-2)r+2}\right)$. From Proposition 2, we can derive the two types of overheads as follows:

$$o_{\mathrm{ef}} = (m-1)V + V^* + C_M + C_D ,$$
$$o_{\mathrm{rw}} = \frac{1}{2}\left(1 + \frac{2-r}{(m-2)r+2}\right)\lambda_s + \frac{\lambda_f}{2} .$$

The optimal work length $W^* = \sqrt{\frac{o_{\mathrm{ef}}}{o_{\mathrm{rw}}}}$ for any fixed m is thus given by Equation (19). The optimal number of chunks $\bar{m}^*$ shown in Equation (20) is obtained by minimizing $F(m) = o_{\mathrm{ef}} \times o_{\mathrm{rw}} = \frac{1}{2}\big((m-1)V + V^* + C_M + C_D\big)\left(\left(1 + \frac{2-r}{(m-2)r+2}\right)\lambda_s + \lambda_f\right)$. Again, the number of chunks in a pattern can only be a positive integer, so m^* is either $\max(1, \lfloor \bar{m}^* \rfloor)$ or $\lceil \bar{m}^* \rceil$, since $F(m)$ is a convex function of m. Finally, substituting Equation (20) back into $H^*(\mathrm{P}) = 2\sqrt{o_{\mathrm{ef}} \times o_{\mathrm{rw}}}$ gives rise to the optimal expected overhead as shown in Equation (21). □

Remarks. When only guaranteed verification is used, the optimal pattern contains equal-length chunks, which matches the result in [6]. In this case, the pattern is denoted P_{DV^*},

and we have:

$$\beta_j^* = \frac{1}{m^*} \text{ for } 1 \le j \le m^* ,$$
$$W^* = \sqrt{\frac{m^*V^* + C_M + C_D}{\frac{1}{2}\left(1 + \frac{1}{m^*}\right)\lambda_s + \frac{\lambda_f}{2}}} ,$$
$$\bar{m}^* = \sqrt{\frac{\lambda_s}{\lambda_s + \lambda_f} \cdot \frac{C_M + C_D}{V^*}} ,$$
$$H^*(\mathrm{P}) = \sqrt{2(\lambda_s + \lambda_f)(C_M + C_D)} + \sqrt{2\lambda_s V} + O(\lambda) .$$

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

Finally, we consider the complete pattern that contains multiple segments, each of which has multiple chunks. This represents the general two-level checkpointing protocol with intermediate verifications for silent error detection. Figure 4 depicts the pattern in this protocol.

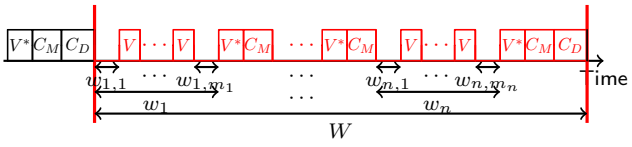


Figure 5: Pattern $\mathrm{P}_{DMV} = \mathrm{P}(W, n, \boldsymbol{\alpha}, \boldsymbol{m}, \langle \boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_n \rangle)$.

The goal is to determine all the parameters of the pattern. Again, we first derive the expected execution time of a pattern when all parameters are fixed.

Proposition 4. *The expected execution time of a given pattern* $\mathrm{P}(W, n, \boldsymbol{\alpha}, \boldsymbol{m}, \langle \boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_n \rangle)$ *is*

$$\mathbb{E}(\mathrm{P}) = W + \sum_{i=1}^{n}(m_i - 1)V + n(V^* + C_M) + C_D + \left(\lambda_s \sum_{i=1}^{n} \boldsymbol{\beta}_i^T A^{(m_i)} \boldsymbol{\beta}_i \cdot \alpha_i^2 + \frac{\lambda_f}{2}\right) W^2 + O(\sqrt{\lambda}) , \quad (22)$$

where $A^{(m)}$ *denotes an* $m \times m$ *symmetric matrix*[2] *defined by* $A^{(m)}_{i,j} = \frac{1}{2}\left(1 + (1-r)^{|i-j|}\right)$ *for all* $1 \le i, j \le m$.

Proof. Define E_i to be the expected execution time of the i-th segment up to the memory checkpoint at the end of the segment. We first show the following result:

$$E_i = w_i + (m_i - 1)V + V^* + C_M + \lambda_s \boldsymbol{\beta}_i^T A^{(m_i)} \boldsymbol{\beta} \cdot w_i^2 + \lambda_f \left(\frac{w_i^2}{2} + \sum_{k=1}^{i-1} w_k w_i\right) + O(\sqrt{\lambda}) .$$

[2]Matrices $A^{(m)}$ only differ in their dimensions; they have the same components.

The proof combines the techniques from those of Propositions 2 and 3. Specifically, as in the proof of Proposition 2, we go by induction on i. The base case is equivalent to the pattern $\mathrm{P}(W, 1, [1], [m], \langle \boldsymbol{\beta} \rangle)$ analyzed in Section 4.2, except that there is no disk checkpoint at the end of the segment. Hence, from Proposition 3, we get

$$\begin{aligned} E_1 &= w_1 + (m_1 - 1)V + V^* + C_M \\ &\quad + \lambda_s \boldsymbol{\beta}_1^T A^{(m_1)} \boldsymbol{\beta}_1 \cdot w_1^2 + \frac{\lambda_f}{2} w_1^2 + O(\sqrt{\lambda}) \ . \end{aligned}$$

Suppose the claim holds up to E_{i-1}. Then, by following the proof of Proposition 3, in particular, Equation (17), we can express E_i recursively as follows:

$$\begin{aligned} E_i &= \left(\prod_{j=1}^{m_i} (1 - p_{i,j}^f)(1 - p_{i,j}^s) \right) C_M \\ &\quad + \left(1 - \prod_{j=1}^{m_i} (1 - p_{i,j}^f)(1 - p_{i,j}^s) \right) (R_M + E_i) \\ &\quad + \sum_{j=1}^{m_i} q_{i,j} \left(p_{i,j}^f \left(\mathbb{E}(T_{i,j}^{\mathrm{lost}}) + R_D + \sum_{k=1}^{i-1} E_k \right) + (1 - p_{i,j}^f)(w_{i,j} + V_{i,j}) \right), \end{aligned} \tag{23}$$

where $q_{i,j}$ denotes the probability that the j-th chunk of the i-th segment gets executed, and it is given by

$$q_{i,j} = \left(\prod_{k=1}^{j-1} (1 - p_{i,k}^f) \right) \left(\prod_{k=1}^{j-1} (1 - p_{i,k}^s) + g_{i,j} \right) \ ,$$

with

$$g_{i,j} = \sum_{\ell=1}^{j-1} \left(\prod_{k=1}^{\ell-1} (1 - p_{i,k}^s) \right) p_{i,\ell}^s (1 - r)^{j-\ell} \ .$$

We point out two differences between Equations (17) and (23). First, we do not need to perform a disk checkpoint when there is no error in segment i (Line 1). Second, if a fail-stop error occurred in the j-th chunk, we need to additionally re-execute all the segments before i (Line 3). Simplifying and approximating Equation (23) as in the proof of Proposition 3, we get:

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

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

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

This completes the proof of the proposition. □

Theorem 4. *A first-order approximation to the optimal parameters in pattern* $\mathrm{P}(W, n, \boldsymbol{\alpha}, \boldsymbol{m}, \langle \boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_n \rangle)$ *is given by*

$$
\alpha_i^* = \frac{1}{n^*} \text{ for } i = 1 \ldots n^* \ , \tag{24}
$$

$$
\beta_{i,j}^* = \begin{cases} \frac{1}{(m_i^* - 2)r + 2} & \text{for } 1 \le i \le n^*, j = 1, m_i^* \\ \frac{r}{(m_i^* - 2)r + 2} & \text{for } 1 \le i \le n^*, 2 \le j \le m_i^* - 1 \end{cases} \ , \tag{25}
$$

$$
W^* = \sqrt{\frac{n^*(m^* - 1)V + n^*(V^* + C_M) + C_D}{\frac{1}{2}\left(1 + \frac{2 - r}{(m^* - 2)r + 2}\right)\frac{\lambda_s}{n^*} + \frac{\lambda_f}{2}}} \ , \tag{26}
$$

and n^* *is either* $\max(1, \lfloor \bar{n}^* \rfloor)$ *or* $\lceil \bar{n}^* \rceil$*, and* m_i^* *is either* $\max(1, \lfloor \bar{m}^* \rfloor)$ *or* $\lceil \bar{m}^* \rceil$ *for all* $1 \le i \le$

n^*, *where*

$$\bar{n}^* = \sqrt{\frac{\lambda_s}{\lambda_f} \cdot \frac{C_D}{V^* - \frac{2-r}{r}V + C_M}} \ , \tag{27}$$

$$\bar{m}^* = 2 - \frac{2}{r} + \sqrt{\frac{2-r}{r}\left(\frac{V^* + C_M}{V} - \frac{2-r}{r}\right)} \ . \tag{28}$$

The optimal expected overhead is

$$H^*(\mathrm{P}) = \sqrt{2\lambda_f C_D} + \sqrt{2\lambda_s\left(V^* - \frac{2-r}{r}V + C_M\right)} + \sqrt{2\lambda_s \frac{2-r}{r}V} + O(\lambda) \ . \tag{29}$$

Proof. For any given n and $\boldsymbol{m}$, we perform a series of optimizations on the expected execution time shown in Equation (22). First, minimizing function $f_i = \boldsymbol{\beta}_i^T A^{(m_i)} \boldsymbol{\beta}_i$ subject to $\sum_{j=1}^{m_i} \beta_{i,j} = 1$ (as in the proof of Theorem 3), we get $f_i^* = \frac{1}{2}\left(1 + \frac{2-r}{(m_i-2)r+2}\right)$, obtained when $\boldsymbol{\beta}_i^*$ satisfies Equation (25). Next, minimizing $h = \sum_{i=1}^n f_i^* \alpha_i^2$ subject to $\sum_{i=1}^n \alpha_i = 1$, we get $h^* = \frac{1}{\sum_{i=1}^n 1/f_i^*}$, which is obtained at $\alpha_i^* = \frac{1/f_i^*}{\sum_{k=1}^n 1/f_k^*}$. Finally, subject to $\sum_{i=1}^n m_i = nm$, where m is the average number of chunks per segment, $\sum_{i=1}^n 1/f_i^*$ is maximized when $m_i = m$ for all $1 \le i \le n$. This means that $\boldsymbol{\alpha}^*$ satisfies Equation (24) and the minimum value of h^* is given by $h^* = \frac{1}{2n}\left(1 + \frac{2-r}{(m-2)r+2}\right)$.

Hence, we can write the two types of overheads as follows:

$$o_{\mathrm{ef}} = n(m-1)V + n(V^* + C_M) + C_D \ ,$$
$$o_{\mathrm{rw}} = \frac{1}{2}\left(1 + \frac{2-r}{(m-2)r+2}\right)\frac{\lambda_s}{n} + \frac{\lambda_f}{2} \ .$$

The optimal work length $W^* = \sqrt{\frac{o_{\mathrm{ef}}}{o_{\mathrm{rw}}}}$ for any fixed n and $\boldsymbol{m}$ is thus given by Equation (26).

Now, we need to find values for n and m that minimize the function $F(n,m) = o_{\mathrm{ef}} \times o_{\mathrm{rw}} = \frac{1}{2}\big(n(m-1)V + n(V^* + C_M) + C_D\big)\left(\left(1 + \frac{2-r}{(m-2)r+2}\right)\frac{\lambda_s}{n} + \lambda_f\right)$. For the solution $(\bar{n}^*, \bar{m}^*)$ given in Equations (27) and (28), we can verify that

$$\frac{\partial F(\bar{n}^*, \bar{m}^*)}{\partial n} = 0 \ ,$$
$$\frac{\partial F(\bar{n}^*, \bar{m}^*)}{\partial m} = 0 \ ,$$

and moreover

$$\frac{\partial^2 F(\bar{n}^*, \bar{m}^*)}{\partial n^2} > 0 \ ,$$
$$\frac{\partial^2 F(\bar{n}^*, \bar{m}^*)}{\partial n^2} \cdot \frac{\partial^2 F(\bar{n}^*, \bar{m}^*)}{\partial n^2} - \left(\frac{\partial^2 F(\bar{n}^*, \bar{m}^*)}{\partial n \partial m}\right)^2 > 0 \ ,$$

which shows that $(\bar{n}^*, \bar{m}^*)$ is indeed a global minimum of $F(n,m)$. Since the number of segments and number of chunks per segment can only be positive integers, and $F(n,m)$ is a

convex function, the optimal integer solution is one of the four integer combinations around the optimal rational solution.

Finally, substituting Equations (27) and (28) into $H^*(\mathrm{P}) = 2\sqrt{o_{\mathrm{ef}} \times o_{\mathrm{rw}}}$ and simplifying, we get the optimal expected overhead as shown in Equation (29). □

Remarks Theorem 4 shows that the optimal pattern has identical segments (same size and identical number and sizes of chunks). However, inside each segment, chunks do not necessarily have the same size. With partial verifications, the first and last chunk in each segment are larger than the other ones.

When only guaranteed verifications are used, all chunks in a segment (and hence in the whole pattern) have the same length. In this case, the pattern is denoted P_{DMV^*} and we have:

$$
\begin{aligned}
\alpha_i^* &= \frac{1}{n^*} \text{ for } 1 \le i \le n^* \ , \\
\beta_{i,j}^* &= \frac{1}{m^*} \ \text{ for } 1 \le i \le n^*, 1 \le j \le m^* \ , \\
W^* &= \sqrt{\frac{n^* m^* V^* + n^* C_M + C_D}{\frac{1}{2}\left(1 + \frac{1}{m^*}\right)\frac{\lambda_s}{n^*} + \frac{\lambda_f}{2}}} \ , \\
\bar{n}^* &= \sqrt{\frac{\lambda_s}{\lambda_f}\frac{C_D}{C_M}} \ , \\
\bar{m}^* &= \sqrt{\frac{C_M}{V^*}} \ , \\
H^*(\mathrm{P}) &= \sqrt{2\lambda_f C_D} + \sqrt{2\lambda_s C_M} + \sqrt{2\lambda_s V^*} + O(\lambda) \ .
\end{aligned}
$$

4.4 Summary of results

Table 1 summarizes the results. P_D, P_{DV^*} and P_{DV} are patterns with only one level of checkpointing, i.e., we always perform the memory checkpoint just before the disk checkpoint. P_{DV^*} adds extra guaranteed verifications between two disk checkpoints, while P_{DV} adds partial verifications. Similarly, P_{DM}, P_{DMV^*} and P_{DMV} correspond to two-levels checkpointing patterns, with extra guaranteed verifications for P_{DMV^*} and partial verifications for P_{DMV}.

We report in each case the optimal pattern length W^*, the optimal overhead $H(\mathrm{P})$, 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.

5 Errors in verifications, checkpoints and recoveries

So far, we have assumed error-free execution during verifications, checkpoints and recoveries. In this section, we show how to handle fail-stop errors during these operations[3], and that

[3]Checkpoints and recoveries do not suffer from silent errors, since silent errors typically do not strike I/O and protected memory space, where memory checkpoints are assumed to be stored. Verifications can be protected from silent errors by using redundancy techniques to ensure correct results.

Table 1: Summary of results

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

the first-order approximations derived in the preceding section remain valid as long as the platform MTBF $\mu = 1/\lambda$ is large in front of the resilience parameters.

First, we handle errors during checkpoints and recoveries. The probability of experiencing at least one error during a (checkpoint or recovery) process of length L is given by $p_L^f = 1 - e^{-\lambda_f L}$ and, according to Equation (3), the expected time loss in executing this process is given by $\mathbb{E}(T_L^{\mathrm{lost}}) = \frac{1}{\lambda_f} - \frac{L}{e^{\lambda_f L}-1}$. Let $\mathbb{E}(C_D)$, $\mathbb{E}(C_M)$, $\mathbb{E}(R_D)$ and $\mathbb{E}(R_M)$ denote the expected time to perform disk checkpointing, memory checkpointing, disk recovery and memory recovery, respectively. Since each disk checkpoint is always preceded by a memory checkpoint, and each disk recovery is immediately followed by a memory recovery, we can express these expected execution times recursively as follows:

$$\mathbb{E}(R_D) = p_{R_D}^f\Big(\mathbb{E}(T_{R_D}^{\mathrm{lost}}) + \mathbb{E}(R_D)\Big) + \Big(1 - p_{R_D}^f\Big)\, R_D \ , \tag{30}$$

$$\begin{aligned}\mathbb{E}(R_M) = {} & p_{R_M}^f\Big(\mathbb{E}(T_{R_M}^{\mathrm{lost}}) + \mathbb{E}(R_D) + \mathbb{E}(R_M) + \mathbb{E}(T^{\mathrm{rec}})\Big) \\ & + \Big(1 - p_{R_M}^f\Big)\, R_M \ ,\end{aligned} \tag{31}$$

$$\begin{aligned}\mathbb{E}(C_D) = {} & p_{C_D}^f\Big(\mathbb{E}(T_{C_D}^{\mathrm{lost}}) + \mathbb{E}(R_D) + \mathbb{E}(R_M) + \mathbb{E}(T^{\mathrm{rec}}) + \mathbb{E}(C_M) + \mathbb{E}(C_D)\Big) \\ & + \Big(1 - p_{C_D}^f\Big)\, C_D \ ,\end{aligned} \tag{32}$$

$$\begin{aligned}\mathbb{E}(C_M) = {} & p_{C_M}^f\Big(\mathbb{E}(T_{C_M}^{\mathrm{lost}}) + \mathbb{E}(R_D) + \mathbb{E}(R_M) + \mathbb{E}(T^{\mathrm{rec}}) + \mathbb{E}(C_M)\Big) \\ & + \Big(1 - p_{C_M}^f\Big)\, C_M \ ,\end{aligned} \tag{33}$$

where $\mathbb{E}(T^{\mathrm{rec}})$ denotes the expected time to re-execute the whole pattern, or part of it, depending on when the fault strikes. If the fault strikes during disk checkpointing (Equation (32)), the entire pattern needs to be re-executed. But if the fault strikes during memory checkpointing (Equation (33)), then only part of the pattern up to the given memory checkpoint needs to be re-executed. In all cases, $\mathbb{E}(T^{\mathrm{rec}})$ is upper bounded by the expected execution time of the whole pattern. Recall from our previous analysis that the optimal pattern length satisfies $W^* = \Theta\left(\lambda^{-1/2}\right)$ and the optimal overhead satisfies $H^*(\mathrm{P}) = \Theta\left(\lambda^{1/2}\right)$. Hence, in an

optimized pattern, we will have $\mathbb{E}(T^{\text{rec}}) \leq \mathbb{E}(\text{P}) = W^* + H^*(\text{P}) \times W^* = \Theta\left(\lambda^{-1/2}\right)$. Solving Equations (30) to (33), we can then derive the following results:

$$\begin{aligned}
\mathbb{E}(R_D) &= R_D + O(\lambda) \ , \\
\mathbb{E}(R_M) &= R_M + O(\sqrt{\lambda}) \ , \\
\mathbb{E}(C_D) &= C_D + O(\sqrt{\lambda}) \ , \\
\mathbb{E}(C_M) &= C_M + O(\sqrt{\lambda}) \ ,
\end{aligned}$$

which suggest that the expected costs to perform checkpoints and recoveries are dominated by their original costs under the assumption of a large platform MTBF. Intuitively, this is due to the small probability of encountering an error during these operations. Thus, in Propositions 1 to 4, replacing C_D, C_M, R_D and R_M by their expected values does not affect the expected execution times of the patterns in the first-order approximation.

Now, we briefly discuss the impact of having errors during verifications. Basically, as far as fail-stop errors are concerned, we can consider any (partial or guaranteed) verification together with the work segment (or chunk) immediately preceding it. Two expressions need to be modified in the analysis. First, the probability of experiencing at least one error during the execution of any segment (or chunk) of length w becomes $p^f = 1 - e^{-\lambda_f(w+V)}$, where V denotes the cost of the verification at the end of the work length w. Second, the expected time loss in executing this segment (or chunk) becomes $\mathbb{E}(T^{\text{lost}}) = \frac{1}{\lambda_f} - \frac{w+V}{e^{\lambda_f(w+V)}-1}$. It turns out that both changes do not affect the first-order approximation, because the extra terms (involving V) contribute $O(\sqrt{\lambda})$ to the expected execution time of a pattern, which again is negligible compared to the dominant terms under a large platform MTBF.

6 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 computing patterns at extreme scale, and we instantiate the model with three scenarios. In the first scenario, we evaluate the performance of each pattern 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 of the method. The simulator code is publicly available at `http://graal.ens-lyon.fr/~yrobert/two-level`, so interested readers can experiment with it and build relevant scenarios of their choice.

6.1 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 [9, 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

Table 2: Platform parameters

platform	#nodes	λ_f	λ_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	$2500s$	$180.0s$

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 λ_f for fail-stop errors and λ_s for silent errors. The simulator allows fail-stop errors to occur during computations, verifications, checkpoints and recoveries, while silent errors are only allowed during actual computations, which is in accordance with our model.

An experiment goes as follows. We feed the simulator with the description of the platform, consisting of the parameters λ_f, λ_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 1. 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.

6.2 Assessing resilience mechanisms on real platforms

In the first scenario, we assess the performance of the optimal patterns on four different platforms with real parameter settings. We compare the results for the six optimal patterns of Table 1.

6.2.1 Platform settings

Table 2 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. [29], who provide accurate measurements for λ_f, λ_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 technology for memory checkpointing, which provides more data space, at the cost of higher checkpointing costs.

6.2.2 Pattern overhead

Figure 6a presents, for each pattern, the predicted overhead $H^*(\mathrm{P})$ (in blue) versus the simulated one (in yellow) on each platform. Remember that the formula used to compute the

expected overhead is the result of a first-order approximation, meaning that we are ignoring some low-order terms in the computation. The consequence is that the predicted overhead, being a little optimistic, is always slightly inferior compared to the simulated one. However, the difference between both overheads 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 (P_D, P_{DV^*}, P_{DV}), which use single-level checkpointing, and the last three patterns (P_{DM}, P_{DMV^*}, 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.

6.2.3 Pattern periods

Figure 6b shows the work lengths (periods) of the patterns on each platform. We observe that 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.

Nevertheless, silent errors are more prominent on these platforms and when one 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.

6.2.4 Pattern checkpoints

Figure 6c presents the average number of disk checkpoints, memory checkpoints and verifications taken each day by each pattern. We take all checkpoints and verifications from the simulations 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 (P_{DV} and P_{DMV}) tend to take full advantage of this mechanism. On Hera, P_{DV} generates an average of 13 verifications per hour (including the guaranteed ones), which is slightly more than its two-level counterpart (P_{DMV}), which generates 12 verifications per hour. On Coastal, there are more than 20 verifications per hour for P_{DV} and 19 for P_{DMV}. Note that the verifications before each memory checkpoint are included.

In order to have a closer look at the number of checkpoints, which is dwarfed by the number of verifications, Figure 6d presents the checkpointing frequencies alone. Naturally, for the two-level patterns, whose periods are longer, the disk checkpointing frequencies are smaller. 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. Note that the memory checkpoints before each disk checkpoint are also taken into account. 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 higher on this platform ($180s$) as opposed to the costs on other platforms ($15.4s$ on Hera and $4.5s$ on Coastal).

6.2.5 Pattern recoveries

Finally, Figure 6e shows the number of recoveries per day for each pattern on each platform. We can see that the number of disk recoveries follows closely the fail-stop error rate of a given platform, and it is not affected by the patterns. Indeed, when a fail-stop error strikes, a disk recovery is performed regardless of the pattern used. 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 for Atlas and 29 days for Coastal).

The number of memory recoveries may be influenced by several factors. This is 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 whether partial verifications are used in a pattern and the length of the pattern. This also 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.

6.3 Weak scaling experiment

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

6.3.1 Platform settings

We first calculate the MTBF of one computing node, namely 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 used in the simulation. For example, when 2^{17} nodes are used, the MTBF decreases to about $2064s$ 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 needed to perform a memory checkpoint C_M remains constant. In addition, we make the optimistic assumption that the cost of a disk checkpoint C_D also remains constant by scaling the I/O bandwidth of the file system[4]. We explore two scenarios. In the first scenario, we set the cost of a disk checkpoint to be the same as on Hera, i.e., $300s$. In the second scenario, we reduce the cost of a disk checkpoint to $90s$ to account for improved disk technology.

[4]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.

6.3.2 Results

Figure 7a presents the impact of the number of nodes on the overheads for the simplest pattern P_D and the most advanced pattern P_{DMV}. From the simulation results, we can see that the performance remains acceptable up to $2^{15} = 32768$ nodes, with an overhead of 100% for P_D and 64% for P_{DMV}. Beyond that number, the overhead increases 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 down to less than 10 minutes, which is in the same order as the period of a pattern (Figure 7b). At this point, the application experiences a few errors per pattern (Figure 7c) and nearly a dozen errors per hour (Figure 7f). In order to minimize the impact of the errors, the pattern P_{DMV} places more than 200 verifications and more than 10 memory checkpoints per hour (Figures 7e and 7d). As a result, a lot of time is wasted on resilience operations, and the model starts to show its limits. However, when the error rate is this high, there is not much flexibility left in the optimization, and no pattern is able to offer satisfactory performance.

Finally, Figure 8 presents the results when we repeat the weak scaling experiment with a disk checkpointing cost of $90s$ instead of $300s$. Since writing a disk checkpoint is cheaper now, the period is reduced and checkpointing frequency is increased. Overall, the overheads become much better, around 200% at 2^{18} nodes, as opposed to 500% observed in Figure 7a. The behavior of other parameters is similar to the ones presented in Figure 7.

6.4 Impact of error rates

We assess the impact of the error rates on the performance of the computing patterns. Again, we focus on the Hera platform but scale its number of nodes to 10^5. We vary the error rates λ_f and λ_s with respect to their nominal values while keeping the other parameters fixed.

Figures 9a and 9b present the impact of λ_f and λ_s on the simulated overheads of the two patterns P_D and P_{DMV}. For the P_{DMV} pattern, we observe that 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, simply because silent errors have a much higher rate. Figure 9c shows the difference between the overheads of both patterns. We observe a similar performance when most errors are fail-stop, due to their 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 execution overhead.

We now study the impact of error rates on the checkpointing period and frequency. Figure 9d presents the impact of fail-stop errors on the period of both patterns when the silent error rate is fixed at its nominal value. We can see that the period for P_D remains constant,

while the period for P_{DMV} decreases with increased λ_f. This is because the high silent error rate has already driven the P_D pattern period very low (< 10 minutes), so increasing the fail-stop error rate has a limited impact. On the other hand, the period for the P_{DMV} pattern is primarily driven by the fail-stop error rate, so it decreases quickly, allowing more disk checkpoints to be taken. In addition, Figures 9e and 9f show that the number of checkpoints successfully taken in each hour remain 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. Figure 9g shows the corresponding number of recoveries, which is again in accordance with the MTBF of the platform. Note that the number of memory recoveries decreases slightly, as some silent errors are masked by fail-stop errors.

Figure 9h shows the impact of silent errors on the performance of both patterns when the fail-stop error rate is fixed at the nominal value. Now, the role is reversed. Since the P_{DMV} pattern is equipped with more memory checkpoints and verifications, silent errors have no impact on the period. On the contrary, the period of the P_D pattern decreases in order to detect silent errors earlier, which is the only way to protect the application from increased silent error rate. As shown by Figures 9i and 9j, the number of verifications and 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. Finally, Figure 9k shows the corresponding number of recoveries. When silent errors are more prominent, two-level checkpointing helps to detect them before the end of the pattern, as shown by the higher number of memory recoveries. This results in faster recoveries and overall better performance.

6.5 Summary

Through the simulation results of this section, 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.

7 Related work

7.1 Checkpointing

The most commonly deployed strategy to cope with fail-stop errors is checkpointing, in which processes periodically save their state, so that computation can be resumed from that point when some failure disrupts the execution. Checkpointing strategies are numerous, ranging from fully coordinated checkpointing [16] to uncoordinated checkpoint and recovery with message logging [22]. Despite a very broad applicability, all these fault-tolerance methods suffer from the intrinsic limitation that both protection and recovery generate an I/O workload that grows with failure probability, and becomes unsustainable at large scale [23, 11] (even when considering optimizations such as diskless or incremental checkpointing [32]).

To reduce the checkpointing overhead, many authors have proposed multi-level checkpointing protocols, which combine global disk checkpointing with local or in-memory checkpointing (also known as *diskless checkpointing*). Most of these protocols handle fail-stop errors only, and recover from different levels of checkpoints according to the severity of the failures. Vaidya [36] proposed a two-level recovery scheme that tolerates a single node failure using a local checkpoint stored on a parter node. If more than one failure occurs during any local checkpointing interval, the scheme then resorts to the global checkpoint. Silva and Silva [35] advocated a similar scheme by using memory to store the local checkpoints, which is protected by XOR encoding. Moody et al. [29] generalized this idea to account for an arbitrary number of levels with increasing failure handling capability, and implemented it in a Scalable Checkpoint/Restart (SCR) library. Bautista-Gomez et al. [5] also designed a multi-level checkpointing library, called Fault Tolerance Interface (FTI), but they employed a more efficient Reed-Solomon encoding scheme to handle multiple failures without the need to access the parallel file system.

Our 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. As mentioned before, 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 [37] and Daly [18], 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 [19].

7.2 Silent error detection

Considerable efforts have been directed at verification techniques to reveal silent errors. A guaranteed verification is often only achievable with expensive techniques, such as process replication [24, 30] or redundancy [28, 21]. Application-specific information can be very useful in decreasing the verification cost. Algorithm-based fault tolerance (ABFT) [26, 10, 34] is a well-known technique to detect errors in linear algebra kernels using checksums. Various techniques have been proposed in other application domains. Benson et al. [8] compared a higher-order scheme with a lower-order one to detect errors in the numerical analysis of ODEs. Sao and Vuduc [33] investigated self-stabilizing corrections after error detection in the conjugate gradient method. Heroux and Hoemmen [25] designed a fault-tolerant GMRES capable of converging despite silent errors. Bronevetsky and de Supinski [12] provided a comparative study of detection costs for iterative methods. Recently, detectors based on data analytics have been proposed to serve as partial verifications [9, 3, 4]. These detectors use interpolation techniques, such as time series prediction and spatial multivariate interpolation, on scientific dataset to offer large error coverage for a negligible overhead. Although not perfect, their accuracy-to-cost ratios tend to be very high, which makes them interesting alternatives at large scale.

7.3 Optimization of computing patterns

Given the checkpointing cost and the platform MTBF, classical formulas due to Young [37] and Daly [18] are well known to determine the optimal checkpointing intervals in the presence of fail-stop errors. These formulas have been extended to account for silent errors in various ways. By coupling checkpointing with guaranteed verification, Aupy et al. [1] analyzed two

simple patterns: one with k checkpoints and one 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 [7] to achieve more optimized computing patterns. The first analysis of a pattern that utilizes partial verification for silent error detection was given by Cavelan et al. [15]. 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. To the best of our knowledge, this work is the first to investigate the combination of in-memory checkpoints, disk checkpoints, partial verifications and guaranteed verifications.

8 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 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 1 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. 5th Int. Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*, 2014.

[7] A. Benoit, Y. Robert, and S. K. Raina. Efficient checkpoint/verification patterns. *ICL Research report RR-1403. To appear in IJHPCA*, 2015.

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

[9] E. Berrocal, L. Bautista-Gomez, S. Di, Z. Lan, and F. Cappello. Lightweight silent data corruption detection based on runtime data analysis for HPC applications. In *Proc. HPDC*, 2015.

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

[11] G. Bosilca et al. Unified model for assessing checkpointing protocols at extreme-scale. *Concurrency and Computation: Practice and Experience*, 2013.

[12] 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.

[13] 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.

[14] 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.

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

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

[17] 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.

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

[19] 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.

[20] J. Dongarra et al. The International Exascale Software Project: a Call To Cooperative Action By the Global High-Performance Community. *Int. J. High Performance Computing Applications*, 23(4):309–322, 2009.

[21] 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.

[22] 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.

[23] K. Ferreira, J. Stearley, J. H. I. Laros, R. Oldfield, K. Pedretti, R. Brightwell, R. Riesen, P. G. Bridges, and D. Arnold. Evaluating the Viability of Process Replication Reliability for Exascale Systems. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 44:1–44:12, 2011.

[24] 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.

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

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

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

[28] 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.

[29] 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*. ACM/IEEE, 2010.

[30] 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.

[31] 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.

[32] J. Plank, K. Li, and M. Puening. Diskless checkpointing. *IEEE Trans. Parallel Dist. Systems*, 9(10):972–986, 1998.

[33] 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.

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

[35] L. Silva and J. Silva. Using two-level stable storage for efficient checkpointing. *IEE Proceedings - Software*, 145(6):198–202, 1998.

[36] N. H. Vaidya. A case for two-level distributed recovery schemes. *SIGMETRICS Perform. Eval. Rev.*, 23(1):64–73, 1995.

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

[38] 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.

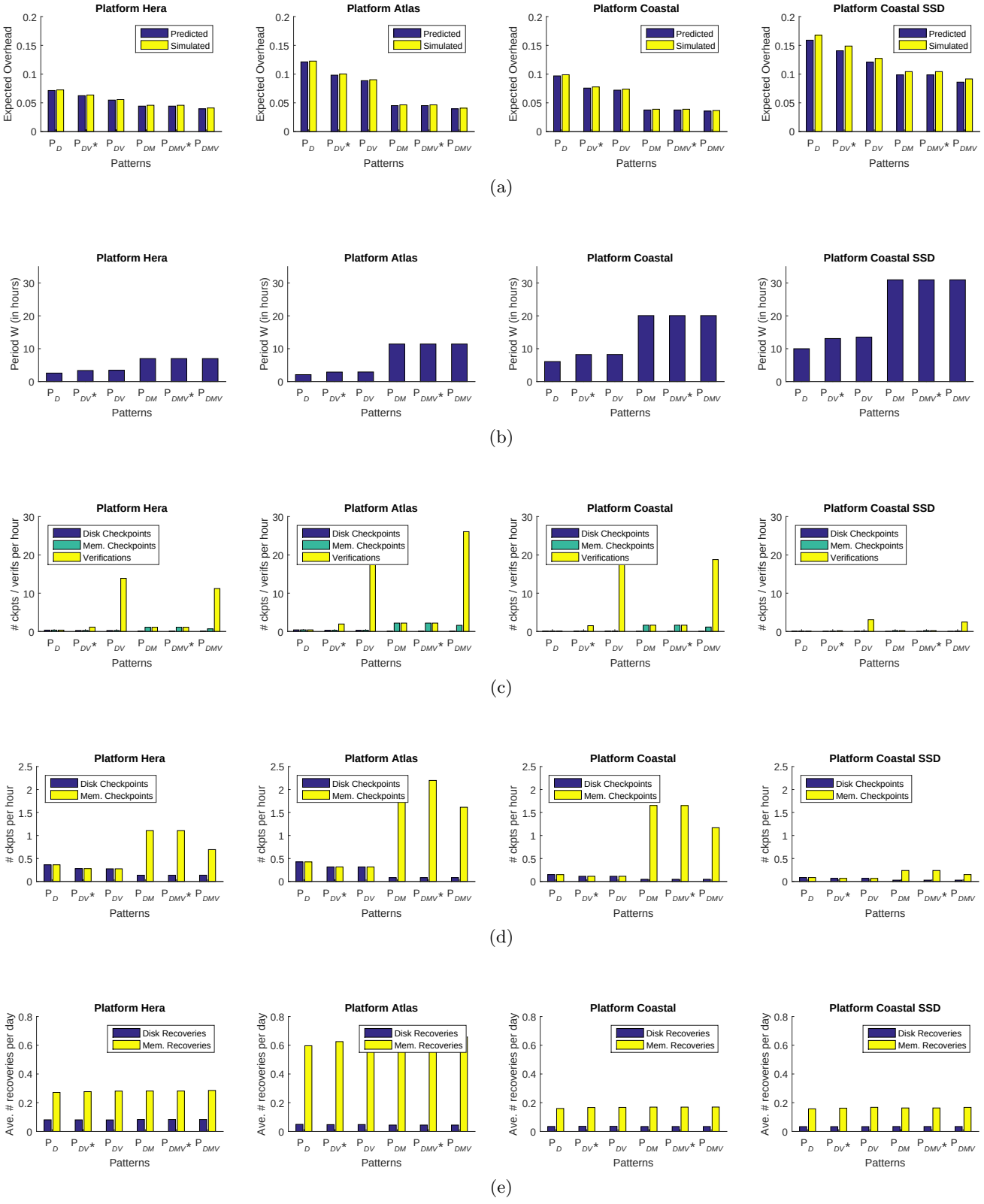


Figure 6: Performance of different patterns on the four platforms. Each column represents one platform.

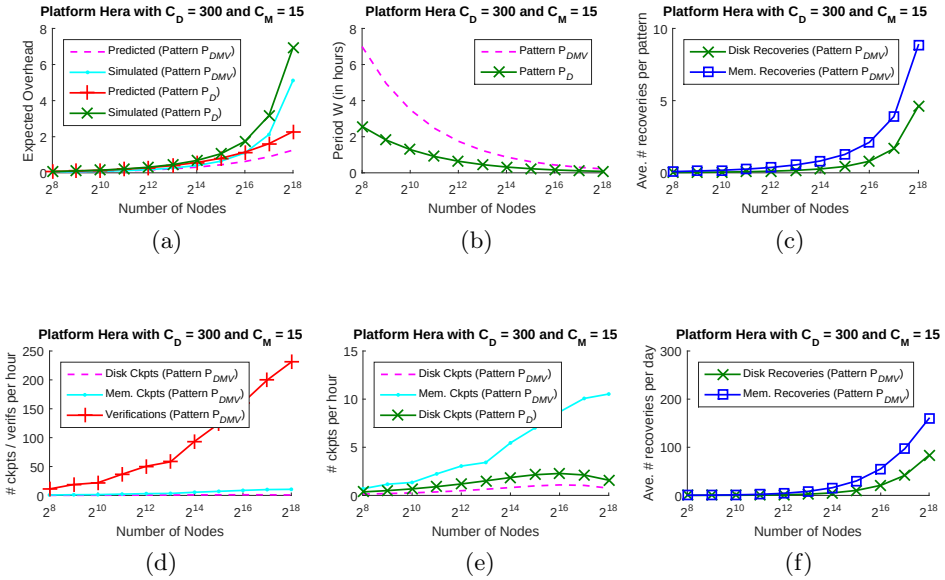


Figure 7: Weak scaling experiment on the Hera platform.

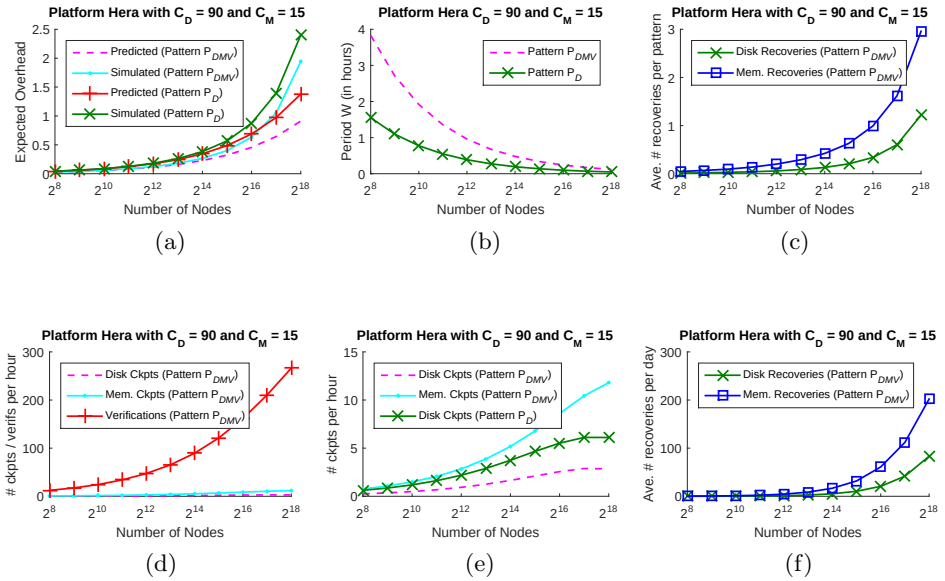


Figure 8: Weak scaling experiment on the Hera platform with reduced disk checkpointing cost.

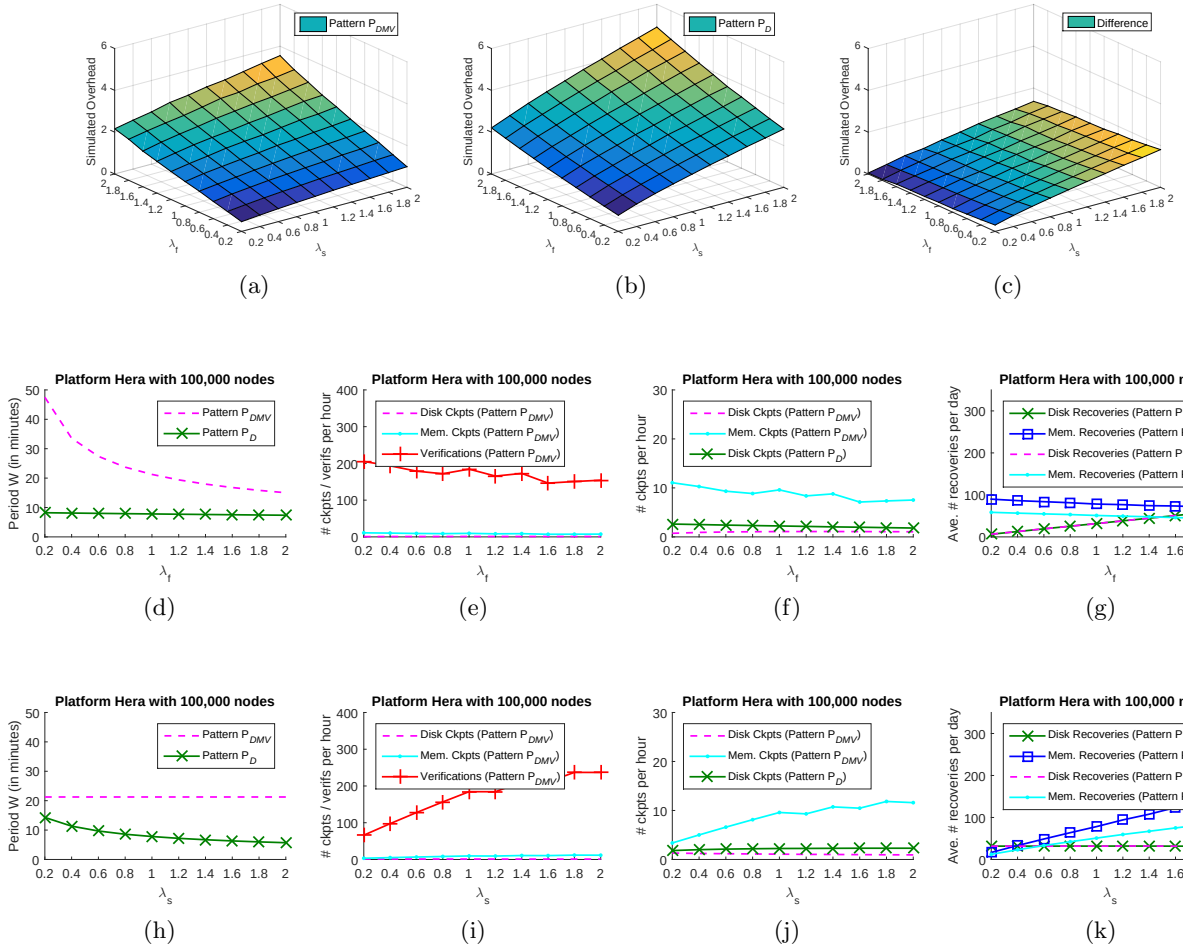
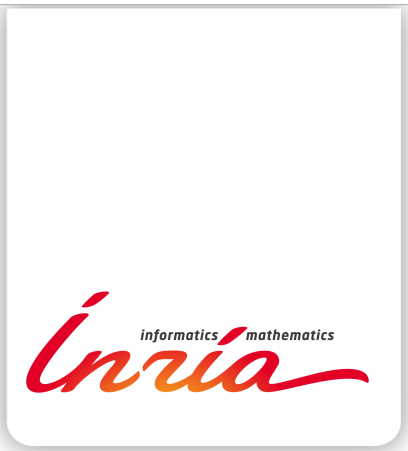


Figure 9: Impact of error rates λ_f and λ_s on the performance on the Hera platform with 10^5 nodes.



RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée

655 avenue de l'Europe Montbonnot

38334 Saint Ismier Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399