Partial Detectors Versus Replication To Cope With Silent Errors

Anne Benoit, Thomas Herault, Yves Robert, Alix Trémodeux

Partial Detectors Versus Replication To Cope With Silent Errors

Anne Benoit, Thomas Herault,
Yves Robert, Alix Trémodeux

RESEARCH REPORT
N° 9581
March 2025
Project-Team ROMA, Topal

ISSN 0249-6399 ISRN INRIA/RR--9581--FR+ENG

Partial Detectors Versus Replication To Cope With Silent Errors

Anne Benoit[*], Thomas Herault[†],

Yves Robert[‡], Alix Trémodeux[§]

Project-Team ROMA, Topal

Research Report n° 9581 — March 2025 — 26 pages

Abstract: This work considers an iterative algorithm that executes on an error-prone platform. Silent errors strike at each iteration with some probability. A detector is applied as a verification mechanism before taking a checkpoint. However, this mechanism is likely not perfect, and fails to detect some errors. More precisely, an error striking at iteration I will be detected only after iteration $(I-1)+X$, where X is a random variable obeying a probability distribution with bounded support $[1, D]$ such as a truncated geometric distribution. Intuitively, the error silently amplifies during some iterations before it can be detected at distance X or higher. As a consequence, when taking a verification before a checkpoint, there is the risk of missing an error that has struck recently but cannot be detected yet. The simplest solution is to keep two checkpoints in memory, and to use two segments of $D-1$ iterations, each followed by a verification and a checkpoint: in steady state, perform $D-1$ iterations after the last checkpoint and take a verification; (i) if successful, we can safely erase the oldest checkpoint, because the most recent checkpoint is necessarily verified. We take a new checkpoint to replace the old one, which is not verified yet; (ii) otherwise, we need to rollback to the oldest checkpoint, not to the last one which may not be verified. Can this simple scheme perform better than replication? What is the optimal number of segments (hence of checkpoints) to keep in memory, and what is the length of these segments? This work answers these questions, both theoretically and through Monte Carlo simulations.

Key-words: Silent errors, replication, partial detectors, bounded detection latency.

Authors' emails: {anne.benoit,thomas.herault,yves.robert,alix.tremodeux}@inria.fr.

[*] ENS Lyon, UCB Lyon, CNRS, Inria, LIP, F-69342, LYON Cedex 07, and Institut Universitaire de France, and Institute for Data Engineering and Science (IDEaS), Georgia Tech, USA
[†] Inria Bordeaux Sud-Ouest, Bordeaux, France
[‡] ENS Lyon, UCB Lyon, CNRS, Inria, LIP, F-69342, LYON Cedex 07, France
[§] ENS Lyon, UCB Lyon, CNRS, Inria, LIP, F-69342, LYON Cedex 07, France

Comparaison de la détection partielle et de la réplication pour les erreurs silencieuses

Résumé : Ce travail s'intéresse aux algorithmes itératifs qui s'exécutent sur des plates-formes soumises à des erreurs silencieuses et compare la réplication avec l'utilisation de détecteurs partiels (capables de détecter une partie des erreurs, mais pas toutes les erreurs).

Mots-clés : Erreurs silencieuses, réplication, détecteurs partiels, distance de détection bornée.

Contents

1 Introduction **4**

2 Related work **6**
2.1 Background 6
2.2 Partial detectors 7
2.3 Recall and precision 7

3 Framework **8**

4 Replication **9**

5 Partial Detectors **10**
5.1 Setting 10
5.2 Expectations 11
5.3 Recurrence formulas 14
5.4 Minimization 15

6 Evaluation **16**
6.1 Validation 16
6.2 Parameter exploration 18
6.3 Comparison with replication 22

7 Conclusion **23**

1 Introduction

Let us consider an iterative algorithm whose execution is struck by silent errors. The only application-independent approach to mitigate the impact of such errors is replication, which works as follows:

- The execution is partitioned into *segments* of M iterations, each followed by a checkpoint. Assume that the initial data can also be recovered if necessary;
- The execution of a new segment (after a checkpoint C) consists of at least two attempts, and possibly more:
  - Attempt 1: Execute the segment for the first time and checkpoint the result res_1;
  - While the results after $t \geq 1$ attempts are all different, execute new attempt $t+1$, i.e., recover from the checkpoint C, redo the M iterations and checkpoint the result res_{t+1};
  - Keep the outcome of the two identical checkpoints and proceed to the next segment.

Here, the classical hypothesis is that two errors will never lead to the same (incorrect) result; more precisely, one neglects this event whose probability is extremely low. Hence, the rationale is to make attempts until the same result is obtained twice, because that result will necessarily be correct. Interestingly, the optimal value of M (the segment length) can be obtained numerically as a function of the resilience parameters, namely the fault-rate and checkpoint cost. Optimality is defined as minimizing the expected time per iteration. We provide such a derivation in Section 4. Note that, to the best of our knowledge, this derivation is new, despite the many related studies on replication.

While replication is the only general-purpose approach to cope with silent errors, several application-specific methods have been introduced. We survey several examples of such detectors in Section 2, which is devoted to related work. In a nutshell, one uses a detector to verify that the current state of the computation is correct. The main problem with silent errors is their detection latency: when a silent error strikes, it does not manifest immediately to the application, but only after some non-deterministic and potentially high number of iterations. A detector must be applied to check whether the current state of the application is correct or not. In the absence of a detector (and without replication), the application must be re-executed from scratch when the error manifests: since one does not know when the error struck, it is impossible to restart from any given intermediate (checkpointed) state, if available, because it may well be corrupted. On the contrary, when applying a detector at the end of some iteration, one can verify that no error has struck so far, and safely take a checkpoint. Obviously, the key hypothesis here is that the detector must be *perfect*, i.e., it will identify all errors when applied. In other words, its recall must be equal to 1, meaning that any silent error can be detected at any stage, possibly at the end of the current iteration, or after any number of following iterations. With a perfect detector, a segment of computations would consist of M iterations, followed by the detector, whose verification is applied at cost V. If the verification is successful, no silent error has struck during the segment and one can safely take a checkpoint of cost C, which is guaranteed not to be corrupted. Otherwise, we recover from the last checkpoint, which is certified to be verified, and re-execute the segment. The optimal value of M is well-known [4], as well as a first-order approximation that is the counterpart for silent errors of the Young-Daly formula for fail-stop errors [25, 10].

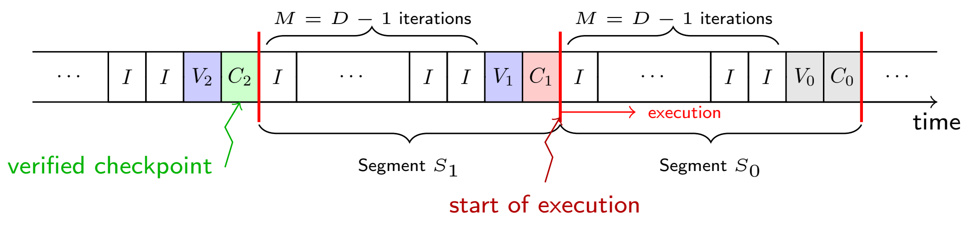
Unfortunately, it is very difficult, if at all possible, to design perfect detectors. More likely, one can design *partial* detectors that will detect many errors, but not all. One can envision several trade-offs where the recall[1] of the detector can be adjusted as a function of the cost of

[1]The recall of a detector is defined as the fraction of errors that it can detect. if the recall is 1, all errors are detected. If not, some errors are missed by the detector; these are called *false negatives.*

the verification mechanism. The more comprehensive the verification, the higher the recall of the detector, but also the higher its cost [1, 11].

Using partial detectors instead of replication seems a promising cost-effective alternative to replication. However, one must assume that the detection latency is bounded to guarantee correctness: indeed, if a silent error strikes at some iteration and fails to be detected for the whole remaining iterations because its detection latency is so high, the result will not be valid, and one will never know whether it is actually the case or not. Henceforth, we study partial detectors whose detection latency is bounded. More precisely, we envision the following scenario: when a silent error strikes during iteration I, one will be able to detect it by applying the partial detector at iteration $(I-1)+X$ or after, where X is a random variable obeying a probability distribution with bounded support $[1, D]$. For instance say a silent error strikes at iteration $I = 10$, and draw $X = 3$. If we apply the detector at the end of current iteration $I = 10$ or at the end of the following iteration $I = 11$, the error will not be detected. However, if we apply the detector at the end of iteration $I = 12$, i.e., $X = 3$ iterations later (including the original iteration $I = 10$), then it will be detected. The same holds true for any later iteration. Regardless of the value of X, the partial detector will systematically detect the error D iterations later, where D is the maximum detection latency, The rationale for this model if that we expect the impact of the silent error on the application data to grow and become more and more *detectable*. For computation errors, this corresponds to a numerical amplification of the error as the execution progresses.

The main focus of this work is to provide a comprehensive assessment of the above scenario. Given a partial detector of bounded latency, what is the best (less costly) execution scheme to ensure correctness? How does it compare to replication? The simplest scheme is to have segments of $M = D - 1$ iterations and to keep two checkpoints in memory, as illustrated below:



Here, we are executing segment S_0, checkpoints C_1 and C_2 are stored in memory. By induction, the oldest checkpoint C_2 is certified to be verified, but this is not yet the case for checkpoint C_1. When we complete the execution of the current segment S_0, we apply the detector through verification V_0. There are two cases:

- No error is detected. Then checkpoint C_1 is verified, because all errors that might have struck during segment S_1 did so at least D iterations before and would have been detected by V_0. For instance, if an error struck during the last iteration of S_1, then its distance to the last iteration of segment S_0 is the maximum detection latency D (remember that we include the original struck iteration in the distance count). We can safely take checkpoint C_0 and overwrite the older checkpoint C_2.
- An error is detected. We must roll back. But there is no way of knowing whether the error struck during segment S_0 or earlier on during S_1; in the latter case, the error went through undetected when verification V_1 was applied. Hence, we need to roll back to checkpoint C_2, which is known to be verified, and to re-execute S_1 followed by S_0. But now, if an error is detected by V_1, we can roll back to C_2 again, because C_2 is verified.

Despite simple to understand, this scheme is not easy to evaluate and compare with replication. Our main contribution is to assess the efficiency of this scheme and more complicated ones, involving many segments. In fact, we solve the main optimization problem: how many segments, or equivalently how many checkpoints to keep in memory? And how many iterations to execute within each segment? These results lay the theoretical foundations for the problem of comparing partial detectors and replication.

The rest of the paper is organized as follows. Section 2 surveys related work. In Section 3, we detail the framework. Section 4 is devoted to analyzing replication and computing the optimal number of iterations per segment. Section 5 is the heart of the paper, where we assess partial detectors: we compute the expected time per iteration for a general scheme with k segments / checkpoints and M iterations per segment, and we provide the optimal values of k and M. This theoretical analysis is complemented by Monte-carlo simulations in Section 6. Finally, we give concluding remarks and hints for future work in Section 7.

2 Related work

Section 2.1 provides some background on silent errors, while Section 2.2 reviews partial detectors. Finally, we discuss recall and precision in Section 2.3.

2.1 Background

While fail-stop errors lead to fatal interruptions (such as a crash) and cause the loss of the entire memory of the processor, silent errors, a.k.a. silent data corruptions (SDCs), only impact a given process and lead to incorrect results. But a silent error strikes undetected and the processor can continue its execution; sometimes the silent error can be detected and corrected, and some other times it degenerates into a fatal fail-stop error.

Silent errors may be caused, for instance, by arithmetic errors in the Arithmetic and Logic Unit (ALU), soft errors in the L1 cache which is usually not well protected, or in the L2 cache which might be protected by one parity bit, or bit flips in the dynamic random-access memory (DRAM) due to cosmic radiation, overheat and other sources [18, 19, 27, 26].

There are several hardware mechanisms to detect and correct silent errors, such as parity bits, error correcting codes (ECCs), and Chip-kill technology. They have been implemented to protect the DRAM and different cache layers to some extent. However, the closer the data is to the processing unit, the more frequent the access to that data and therefore the higher the overhead of these methods. Thus, processor caches are not protected by ECC in general, but by weaker mechanisms, like simple parity, exposing a higher risk of undetectable error in case of multiple simultaneous bit flips. Buses also often are a weak link in the protecting chain, making all data transfers at higher risk. In addition, the constant need to reduce component size and voltage increases the likelihood of silent errors.

Although many silent errors caused by one or multiple bits that spontaneously flip to the opposite state are caught by the above-mentioned hardware mechanisms, in reality, some bit flips still manage to pass undetected [23, 3]. In a nutshell, silent errors have become a major threat due to the increase in problem size [22]: the larger the problem, the more memory to be used to store the data, the more frequent the errors, and the higher the probability of overriding ECC protection, generating multiple errors.

A major problem with silent errors is *detection latency*: contrarily to a fail-stop error whose detection is immediate, a silent error is identified only when the corrupted data is activated and/or leads to an unusual application behavior. However, checkpoint and rollback recovery assumes instantaneous error detection, and this raises a new difficulty: if the error stroke before the last checkpoint, and is detected after that checkpoint, then the checkpoint is corrupted and cannot be used to restore the application. To solve this problem, one may envision keeping several checkpoints in memory and restoring the application from the last *valid* checkpoint, thereby rolling back to the last *correct* state of the application [16]. But even if it was at all possible to store many checkpoints (which is very demanding in memory), one would not know how to identify the last valid one. Some verification mechanism, a.k.a. detector, must be enforced.

2.2 Partial detectors

Considerable efforts have been directed at designing detectors to reveal silent errors, because error detection is usually very costly. As already discussed, the only general-purpose method is to replicate the execution of the target computational kernel on two sets of processors (i.e., duplication) and to compare the results of both executions. If they do not coincide, an error has been detected, and the application must be executed a third time. To avoid a-posteriori re-execution, triplication (i.e., using three parallel executions of the same work) can be enforced, which allows for error correction in addition to error detection, using a simple majority vote. However, triplication (originally known as triple modular redundancy and voting [17]) is even more costly than duplication, which already requires half the resources to execute redundant operations.

Application-specific information can be very useful to enable ad-hoc solutions, which dramatically decreases the cost of detection. Many techniques have been advocated. They include memory scrubbing [14] and Algorithm-Based Fault Tolerance (ABFT) techniques [13, 7, 21], such as coding for the sparse-matrix vector multiplication kernel [21], blockwise checksum calculation for error-bounded lossy compressor [15], and coupling a higher-order with a lower-order scheme for PDEs [5]. Self-stabilizing corrections after error detection in the conjugate gradient method are investigated in [20]. Fault-tolerant iterative solvers for sparse linear algebra are introduced in [9, 12, 8], using extra checks such as re-computing inner products of vectors that should be orthogonal, or even re-computing the residual. Another example is the lightweight SDC detector based on data dynamic monitoring [2], 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 [6]. Although not completely accurate, these partial verification techniques nevertheless cover a substantial number of silent errors, and more importantly, they incur very low overheads. These properties make them attractive candidates for designing more efficient resilient protocols.

To summarize, perfect detectors are very appealing because they can be applied before taking a checkpoint, which by definition will be verified since all potential errors will have been detected. With a perfect detector, the approach is to use of a single segment of execution and a single checkpoint is kept in memory. Partial detectors can be added in the middle of that segment to speed-up detection. However, perfect detectors are more a perspective than an actual proposal. Without a perfect detector, replication was the only known approach. Introducing a partial detector with bounded detection latency, and assessing its performance, is the key contribution of this work.

2.3 Recall and precision

Partial detectors are characterized by their recall and their precision. The recall, denoted by r, is the ratio between the number of detected errors and the total number of errors that occurred during a computation. The precision, denoted by p, is the ratio between the number of true errors and the total number of errors detected by the verification. For example, the basic spatial based SDC detector [2],has been shown to have a recall value around 0.5 and a precision value very close to 1, which means that it is capable of detecting half of the errors with almost no false alarm. A perfect detector can be considered as a special type of partial detector with recall

$r^* = 1$ and precision $p^* = 1$. Each partial detector also has an associated cost V , which is typically much smaller than the cost V^* of a perfect detector.

A detailed study [1] considers several partial detectors and shows how to decide which ones to include and which ones to discard in a computing pattern that involves several segments. Each segment ends with a partial detector, while the last segment ends with a perfect detector followed by a (verified) checkpoint. Surprisingly, it is proven in [1] that partial detectors whose precision is not 1 should always be discarded. A major difference with this work, beyond the existence of a perfect detector, is that [1] assumes statistical independence of the detectors: if the same detector is applied twice, at two different time-steps, then its recall and precision are the same for both instances. On the contrary, we assume here that the error amplifies and is detected with higher probability as the execution progresses.

3 Framework

This section details the framework and the objective function. We focus on a generic iterative application, because it is easier to express every quantity in terms of numbers of iterations. In particular, we account for the cost of every resilience mechanism (detector, checkpoint, recovery) as a number of iterations. This makes the approach agnostic of the granularity of the application, which can range from sequential to massively parallel, and of the nature of errors, either silent or transient.

The application is subject to the occurrence of errors, which can be mitigated either by replication or by applying a partial detector. We consider two main probability distribution laws to model error rates and latency detections:

- Occurrence: A silent error may strike each iteration independently and with a fixed probability f (hence obeying a simple Bernoulli law of parameter f).
- Detection: When an error strikes during iteration I, it can be detected (by applying the partial detector) only at iteration $(I-1)+X$ or after, where X is the detection distance, or latency. X obeys a truncated Geometric distribution law: we have $X = \min(Y, D)$, where Y is a random Geometric variable of parameter θ, and D is the maximum detection latency. This leads to

$$\begin{cases} P(X=d) = P(Y=d) = (1-\theta)^{d-1}\theta & \text{if } 1 \le d < D \\ P(X=D) = P(Y \ge D) = (1-\theta)^{D-1} \end{cases} \tag{3.0.1}$$

The support of X is the interval $[1, D]$, and $X = 0$ elsewhere.

Again, once an error manifests, it never auto-corrects and keeps manifesting during the following iterations until it can be detected. This is like tossing a biased coin (with probability θ for heads and $1-\theta$ for tails) every iteration during which or after the error struck, until the first head is drawn, but we bound the maximum number of tossings to ensure correctness in the worst case.

Table 1 helps get an insight on typical values for the maximum detection distance D. For an efficient partial detector ($\theta = 0.9$), distance detection never exceeds 10 in practice. For a poor partial detector ($\theta = 0.2$, capturing only 20% of errors), distance detection never exceeds 100 in practice.

All the random variables for occurrence and detection are assumed to be independent. We point out that the study can be easily conducted for different probability distribution laws, as long as independence is preserved.

We further assume that the partial detector has precision 1 (no false alarm). This assumption is only a simplification rather than an intrinsic limitation of partial detectors. Indeed, we can

θ	$\min\{d \mid P(X \geq d) \leq 10^{-6}\}$	$\min\{d \mid P(X \geq d) \leq 10^{-9}\}$
0.2	62	93
0.4	28	41
0.9	6	9

Table 1: Minimum distance to be detected with high probability as a function of θ.

extend the complicated derivation provided in Section 5 to account for the extra rollbacks, recoveries and re-executions that would be caused by false alarms. Such an extended derivation is set for future work.

The execution scheme partitions the application into segments of M iterations followed by a checkpoint. We always apply the partial detector, whose cost is V iterations, before taking a checkpoint, whose cost is C iterations. Note that: (i) we take that checkpoint only if no error is detected; and (ii) the checkpoint may still not be verified because some error has struck and remained undetected. However, some of the checkpoints are guaranteed to include no error; we call such checkpoint a *verified checkpoint*. If an error is detected by the verification, the application rolls back to the last verified checkpoint and pays a recovery, whose cost is R iterations, before resuming the execution from that point on. The existence of a verified checkpoint is an invariant of the execution scheme and is proven by induction (see details in Section 5.2). There are no errors during verifications, checkpoints and recoveries.

Finally, the objective function is to minimize the expected slowdown $\mathcal{S}$ per iteration, which characterizes the diminution of the application progress due to error mitigation. If the cost to execute a segment of M iterations is $cost(M)$, then the slowdown is $\mathcal{S} = \frac{cost(M)}{M}$.

4 Replication

As discussed in Section 1, replication uses segments of M iterations followed by a checkpoint. Each segment is executed several times, until the results of two execution attempts are identical. The cost of the first attempt is $M + C$, while the cost of the following attempts is $R + M + C$ because each of them starts with a recovery. While we need to perform further attempts, we have keep the checkpoints of all previous attempts in memory so that we can check whether the results of two attempts are identical.

What is the expected number of attempts? This is the expected number of attempts until two of them are successful, and it also corresponds to the number of checkpoints that have to be stored. The number of attempts until one is successful obeys a Geometric distribution law of parameter p_S, the probability of executing a segment of M iterations without any error. We have

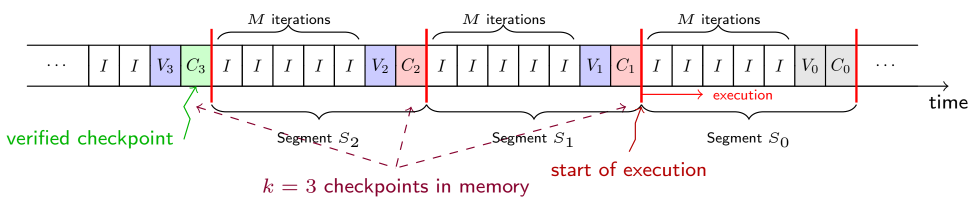
$$p_S = (1 - f)^M,$$

and the expected number of attempts until one is successful is $\frac{1}{p_S}$. The expected number of attempts until two are successful is $\frac{2}{p_S}$. Since the first attempt has cost $M + C$ and the following ones have cost $R + M + C$, the expected cost to execute a segment is

$$cost(M) = (M + C) + \left(\frac{2}{p_S} - 1\right)(R + M + C) = \frac{2(R + M + C)}{p_S} - R,$$

and the expected slowdown is

$$\mathcal{S} = \frac{cost(M)}{M} = \frac{2(R + C)}{M p_S} + \frac{2}{ps} - \frac{R}{M}.$$



M iterations
M iterations
M iterations
$\cdots$ I I V_3 C_3 I I I I I V_2 C_2 I I I I I V_1 C_1 I I I I I V_0 C_0 $\cdots$
execution
time
verified checkpoint
Segment S_2
Segment S_1
Segment S_0
start of execution
$k = 3$ checkpoints in memory

Figure 1: Example with $k = 3$ and $M = 5$.

With $p_S = (1-f)^M$, we let

$$g(M) = \frac{2(R+C)}{M(1-f)^M} + \frac{2}{(1-f)^M} - \frac{R}{M}$$

and aim at minimizing $g(M)$. Differentiating, we get

$$g'(M) = \frac{(1-f)^{-M}}{M^2}\left(R((1-f)^M - 2) - 2M\ln(1-f)(C+R+M) - 2C\right).$$

The optimal value M_{opt}^{rep} of M is such that $g'(M_{opt}^{rep}) = 0$. Unfortunately, this equation does not have a closed-form solution when f, C and R are unknown parameters, even when $R = C$. We need to resort to numerical methods to find the optimal value M_{opt}^{rep} when the values of the parameters are given.

Note that the replication strategy has significant advantages: it is application-independent, and its performance depends only on the statistical properties of the risk of errors. However, as we will see in Section 6, it comes with a high cost, which may be prohibitive in practice.

5 Partial Detectors

5.1 Setting

In this section, we assess the use of the partial detector. We consider an execution scheme with k segments and k checkpoints stored in memory. Each segment consists of M iterations, followed by a verification and a checkpoint. Segment S_i, $0 \le i \le k-1$, is followed by verification V_i and checkpoint C_i. Figure 1 provides an illustration with $k = 3$ and $M = 5$.

First, there is some relation to enforce between M, k and the maximum detection latency D for the execution scheme to be valid. We are currently executing segment S_0 and aiming to ensure that checkpoint C_{k-1} (C_2 in the example) is verified, so that we can overwrite C_3 by C_0 if verification V_0 is successful. This requires that any error that may have struck within segment S_{k-1} (S_2 in the example) is detected by V_0 eventually, if it has not been detected by previous verifications $V_{k-1}, V_{k-2}, \ldots, V_1$ earlier on. This requirement is fulfilled if we enforce that

$$(k-1)M \ge D-1,$$

because an error that struck at the last iteration of segment S_{k-1} is at distance $(k-1)M+1 \ge D$ from the last iteration of segment S_0, and it will be detected by V_0 if it has not been detected before. Given a value for M, we will use

$$k = \left\lceil \frac{D-1}{M} \right\rceil + 1 \tag{5.1.1}$$

as the minimum number of checkpoints that must be kept in memory; this is also the minimum number of segments for the execution scheme. Note that $k \geq 2$ unless $D = 1$, which corresponds to a perfect detector.

In the example, $M = 5$ and $k = 3$, hence we must have $D \leq 11$. Conversely, for $D = 11$, the values of M that lead to choosing $k = 3$ in the execution scheme are $5 \leq M \leq 9$.

When executing the current segment S_0, checkpoints C_1 to C_k are in memory, and checkpoint C_k is known to be verified, since all errors previous to C_k would have been detected by V_1 at the latest. However, errors having struck segments S_{k-1} to S_1 may have remained undetected up to verification V_1 included.

Our objective is to determine the expected time needed to complete the M iterations of segment S_0 and to replace the oldest checkpoint C_k by a new one at the end of S_0. Once V_0 is successful, we know for sure that C_{k-1} is a verified checkpoint; but if V_0 detects an error, we have to roll back to C_k, the last verified checkpoint, and to re-execute all the k segments.

We start by computing the probability that an error is detected during a given verification. Let $P_{i,j}$ be the probability that an error that struck at the i-th iteration of the ℓ-th segment (for $1 \leq i \leq M$ and $0 \leq \ell \leq k-1$) is detected at $V_{\ell-j}$ (for $0 \leq j \leq \ell$). Hence, for $j = 0$, this is the probability that the error is detected immediately at the end of the segment where it struck; otherwise, it is the probability of detection after j other segments have been executed. We have

$$P_{i,j} = P\big(X \leq jM + (M-i+1)\big) \quad - \quad P\big(X \leq (j-1)M + (M-i+1)\big). \tag{5.1.2}$$

To help understand Equation (5.1.2): (i) For $i = 1$ and $j = 0$, the error occurred at the very beginning of the segment, and hence we detect it at the end of the segment with a probability $P(X \leq M)$. (ii) In general, the distance between the iteration where the error occurred and the next verification is $M-i+1$, and we then enforce the segment number by bounding X according to jM.

We also introduce the notation for the probability to detect an error striking at iteration i in segment ℓ to be detected after $V_{j-\ell-j}$:

$$P_{i,>j} = P\big(X > jM + (M-i+1)\big).$$

5.2 Expectations

Let E_0 be the expected time required to process segment S_0 entirely, and then to take a new checkpoint C_0 after a successful verification, hence deleting C_k from memory and moving on to the next segment. By induction, C_k is a verified checkpoint when we start executing S_0, but the more recent $k-1$ checkpoints are not yet verified. The objective is to compute E_0.

Because different rollbacks may be needed to complete the process of segment S_0, we introduce E_j for $1 \leq j \leq k-1$ as the expected time required to go all the way up to taking C_0, but starting at the first iteration of segment S_j (instead of the first iteration of segment S_0 for E_0), and knowing that no error has been detected by the previous verifications $V_{j'}$, where $j' \geq j+1$: hence, no error has been detected in the first segments before C_{j+1} and the start of segment S_j.

The execution goes as follows: we execute S_0 and V_0. If V_0 is successful, we take checkpoint C_0 and we are done. Otherwise, we roll back to C_k and re-execute all the last k segments, recovering from C_k which is the only verified checkpoint. Now, during this re-execution, some errors may strike, among which some may be detected. The execution progresses until an error is detected, in which case we roll back to C_k.

Here is an execution scenario for the example with $k = 3$: to account time for E_0, we execute S_0 for the first time. If V_0 detects no error, we take C_0 and we are done. Otherwise, we roll back to C_3, and start accounting time for E_2: recover, execute segment S_2 and take verification

V_2. If V_2 detects an error, roll back to C_2 and call E_2 recursively. Otherwise, start accounting time for E_1, starting segment S_1. When taking verification V_1, there are again two possibilities: an error is detected, in which case we roll back to C_3 and re-start from segment S_2, or not, in which case we start accounting time for E_0, starting segment S_0. The process goes on until C_0 can be taken eventually.

First segment S_{k-1}. We start by considering the first segment S_{k-1}, just after the verified checkpoint C_k. Let Q_0 be the probability that either there is no error during the M iterations of S_{k-1}, or that none of the errors that struck during these iterations are detected by V_{k-1}. In other words, Q_0 is the probability to go through V_{k-1} (so to speak), checkpoint and proceed to segment S_{k-2}:

$$\begin{aligned} E_{k-1} = {} & R + M + V && \text{recovery, computation and verification done in all cases} \\ & + Q_0(C + E_{k-2}) && \text{no error in } S_{k-1} \text{ that is detected by } V_{k-1} \\ & + (1 - Q_0)E_{k-1} && \text{rollback and start over: detected error} \end{aligned}$$

Since C_k is correct by induction hypothesis, an error detected by V_{k-1} can only have occurred in S_{k-1}, not before. Hence:

$$Q_0 = \prod_{i=1}^{M}(1 - fP_{i,0}).$$

Indeed, an error occurs at iteration i of S_{k-1} with probability f, it is detected at V_{k-1} with probability $P_{i,0}$ (current segment), hence there is a probability $1 - fP_{i,0}$ that there were no error nor any detected error from iteration i. Note that Q_0 can actually be used for any segment, since it only considers errors occurring during the segment and detected at the end of that segment.

Next segments S_j, $1 \le j \le k-2$. We extend the formula to any E_j as follows, for $j \ge 1$ (the case E_0 is slightly different):

$$\begin{aligned} E_j = {} & M + V && \text{computation and verification done in all cases} \\ & + \left(\prod_{\ell=0}^{k-1-j} Q_\ell\right)(C + E_{j-1}) && \text{no error that is detected by } V_j\text{, move to } S_{j-1} \\ & + \left(1 - \prod_{\ell=0}^{k-1-j} Q_\ell\right) E_{k-1} && \text{rollback and start over from first segment: detected error} \end{aligned}$$

Here is how we compute the different probabilities. In order to be able to move to the next segment S_{j-1}, we now need to consider several cases:

- No error in S_j has been detected by V_j at the end of the segment: this happens with probability Q_0 as defined above.
- We must however also look for errors that may come from previous segments, and hence generalize the formula. Q_ℓ is the probability that there is no error coming from the ℓ-th previous segment that is detected by V_j, for $0 \le \ell \le k-2$. Hence, for $\ell = 1$, we are looking at errors coming from the segment just before S_j (hence in S_{j+1}). For each previous segment, there are three possibilities for any of the M possible error locations:
  - There is no error, with a probability $(1 - f)$.

  - There is an error that is not detected by V_j, but that will be detected later. By conditional definition of E_j, the error was not detected before either (with $V_{j'}$, where $j' > j$), and hence this is the probability $fP_{i,>\ell}$ (there is an error, it is detected later).
  - Finally, there might have been an error that is detected by the current verification V_j, and this comes with a probability $fP_{i,\ell}$.

All probabilities must therefore be normalized by the sum of the probabilities of the three cases, which is $(1-f)+f(P_{i,>\ell}+P_{i,\ell})$. For $\ell = 0$, this sums to 1 since the error is either detected at the end of the current segment ($P_{i,0}$) or later ($P_{i,>0}$), and therefore we obtain the previous formula of Q_0 again.

We finally obtain the general expression for Q_ℓ:

$$Q_\ell = \prod_{i=1}^{M} \left(1 - \frac{fP_{i,\ell}}{(1-f)+f(P_{i,>\ell}+P_{i,\ell})}\right).$$

- By combining all possible locations of errors that remain undetected at V_j, coming from the current segment S_j or previous ones (up to $k-1-j$ previous segments), we obtain the probability that no error is detected at V_j and that we can move forward to S_{j-1}, expressed as $\prod_{\ell=0}^{k-1-j} Q_\ell$.

Last segment E_0. Finally, for E_0, when we consider faults from segment S_{k-1}, they are necessarily detected by V_0, which slightly changes the computation of probabilities to complete. Indeed, the expression of Q_{k-1} further considers that any error at S_{k-1} will necessarily be detected at most at V_0, because the distance of detection D is exceeded. Again, because of the conditional expression of E_0, we know that no error in S_{k-1} has been detected before, and therefore, either there was no error (probability $(1-f)$), or the error is detected at V_0 (probability $P_{i,k-1}$). Hence, we write:

$$Q_{k-1} = \prod_{i=1}^{M} \left(1 - \frac{fP_{i,k-1}}{(1-f)+f(P_{i,k-1})}\right).$$

We point out that this is the same formula as above because $P_{i,>k-1} = 0$ (remember that the distance detection is upper bounded by D).

We can finally write E_0, which is the quantity that we aim at minimizing, i.e., the time to progress to the next segment, add a new checkpoint and delete checkpoint C_k:

$$\begin{aligned} E_0 = {} & M + V && \text{computation and verification done in all cases} \\ & + \left(\prod_{\ell=0}^{k-1} Q_\ell\right) C && \text{no error that is detected at } V_0 \text{, we checkpoint} \\ & + \left(1 - \prod_{\ell=0}^{k-1} Q_\ell\right) E_{k-1} && \text{rollback and start over from the start: detected error} \end{aligned}$$

We check that this is indeed the same expression as for E_j with $1 \le j \le k-2$ if we let $E_{-1} = 0$, which is comforting.

5.3 Recurrence formulas

We show by induction that we have the following formulas:

Theorem 1. *Letting* $E_{-1} = 0$*, and* $\Phi_j = \prod_{\ell=0}^{j} Q_\ell$ *for* $0 \leq j \leq k-1$*, we have:*

$$\begin{array}{lll} \text{Equation } (A_j)\text{:} & E_{k-1} = E_{k-j} + u_j C + v_j(M+V) + w_j R & \text{for } 1 \leq j \leq k \\ \text{Equation } (B_j)\text{:} & E_{k-j} = E_{k-j-1} + a_j C + b_j(M+V) + c_j R & \text{for } 1 \leq j \leq k \end{array}$$

where

$$\begin{array}{ll} u_1 = v_1 = w_1 = 0 & \\ a_1 = 1, b_1 = c_1 = \frac{1}{\Phi_0} & \\ u_j = u_{j-1} + a_{j-1} & \text{for } 2 \leq j \leq k \\ v_j = v_{j-1} + b_{j-1} & \text{for } 2 \leq j \leq k \\ w_j = w_{j-1} + c_{j-1} & \text{for } 2 \leq j \leq k \\ a_j = 1 + \left(\frac{1}{\Phi_{j-1}} - 1\right) u_j & \text{for } 2 \leq j \leq k \\ b_j = \frac{1}{\Phi_{j-1}} + \left(\frac{1}{\Phi_{j-1}} - 1\right) v_j & \text{for } 2 \leq j \leq k \\ c_j = \left(\frac{1}{\Phi_{j-1}} - 1\right) w_j & \text{for } 2 \leq j \leq k \end{array}$$

Proof. We have $u_1 = v_1 = w_1 = 0$ directly from Equation (A_1). We have shown before that

$$E_{k-1} = R + M + V + Q_0(C + E_{k-2}) + (1 - Q_0)E_{k-1},$$

which we rewrite into

$$E_{k-1} = E_{k-2} + C + \frac{1}{Q_0}(R + M + V).$$

Since $\Phi_0 = Q_0$ by definition, we derive that $a_1 = 1$ and $b_1 = c_1 = \frac{1}{\Phi_0}$, which shows Equations (A_1) and (B_1).

Assume now by induction that both Equations (A_m) and (B_m) hold for $1 \leq m \leq j-1$. First, we plug Equation (B_{j-1}), namely

$$E_{k-j+1} = E_{k-j} + a_{j-1}C + b_{j-1}(M+V) + c_{j-1}R$$

into Equation (A_{j-1}), namely

$$E_{k-1} = E_{k-j+1} + u_{j-1}C + v_{j-1}(M+V) + w_{j-1}R,$$

which leads to

$$E_{k-1} = E_{k-j} + (u_{j-1} + a_{j-1})C + (v_{j-1} + b_{j-1})(M+V) + (w_{j-1} + c_{j-1})R$$

hence Equation (A_j) holds with $u_j = u_{j-1} + a_{j-1}$, $v_j = v_{j-1} + b_{j-1}$ and $w_j = w_{j-1} + c_{j-1}$.

Now, we have shown before that

$$E_{k-j} = M + V + \Phi_{j-1}(C + E_{k-j-1}) + (1 - \Phi_{j-1})E_{k-1},$$

which we rewrite using Equation (A_j) into

$$E_{k-j} = E_{k-j-1} + \left(1 + (\frac{1}{\Phi_{j-1}} - 1)u_j\right) C + \left(\frac{1}{\Phi_{j-1}} + (\frac{1}{\Phi_{j-1}} - 1)v_j\right)(M+V) + + \left((\frac{1}{\Phi_{j-1}} - 1)w_j\right) R.$$

We obtain Equation (B_j) with $a_j = 1 + \left(\frac{1}{\Phi_{j-1}} - 1\right) u_j$ $b_j = \frac{1}{\Phi_{j-1}} + \left(\frac{1}{\Phi j-1} - 1\right) v_j$ and $c_j = \left(\frac{1}{\Phi j-1} - 1\right) w_j$. □

We derive a closed-form expression for E_0:

Corollary 1. *$E_0 = a_k C + b_k(M+V) + c_k R$, where*

$$\begin{aligned}
a_k &= 1 + \left(\tfrac{1}{\Phi_{k-1}} - 1\right) u_k \\
b_k &= \tfrac{1}{\Phi_{k-1}} + \left(\tfrac{1}{\Phi_{k-1}} - 1\right) v_k \\
c_k &= \left(\tfrac{1}{\Phi_{k-1}} - 1\right) w_k \\
u_k &= 1 + \textstyle\sum_{m=0}^{k-3} \prod_{\ell=0}^{m} \tfrac{1}{\Phi_{k-2-l}} \\
v_k &= \textstyle\sum_{m=0}^{k-2} \prod_{\ell=0}^{m} \tfrac{1}{\Phi_{k-2-l}} \\
w_k &= \textstyle\prod_{\ell=0}^{k-2} \tfrac{1}{\Phi_\ell}
\end{aligned}$$

Proof. Using Theorem 1, we have $u_1 = 0$, $u_2 = 1$, and for $3 \le j \le k$:

$$u_j = u_{j-1} + a_{j-1} = u_{j-1} + \left(1 + (\frac{1}{\Phi_{j-2}} - 1)u_{j-1}\right) = 1 + \frac{1}{\Phi_{j-2}} u_{j-1}.$$

We get $u_3 = 1 + \frac{1}{\Phi_1}$, $u_4 = 1 + \frac{1}{\Phi_2} + \frac{1}{\Phi_1\Phi_2}$, hence, by direct induction, the formula for $j \ge 3$:

$$u_j = 1 + \sum_{m=0}^{j-3} \prod_{\ell=0}^{m} \frac{1}{\Phi_{j-2-l}}.$$

Similarly, we have $v_1 = 0$, $v_2 = \frac{1}{\Phi_0}$ and for $3 \le j \le k$:

$$v_j = v_{j-1} + b_{j-1} = v_{j-1} + \left(\frac{1}{\Phi_{j-2}} + \left(\frac{1}{\Phi_{j-2}} - 1\right) v_{j-1}\right) = \frac{1}{\Phi_{j-2}}(1 + v_{j-1}).$$

We get $v_3 = \frac{1}{\Phi_1} + \frac{1}{\Phi_0\Phi_1}$, hence, by direct induction, the formula for $j \ge 3$:

$$v_j = 1 + \sum_{m=0}^{j-2} \prod_{\ell=0}^{m} \frac{1}{\Phi_{j-2-l}}.$$

Finally, we have $w_1 = 0$, $w_2 = \frac{1}{\Phi_0}$ and for $3 \le j \le k$:

$$w_j = w_{j-1} + c_{j-1} = w_{j-1} + \left(\frac{1}{\Phi_{j-2}} - 1\right) w_{j-1} = \frac{1}{\Phi_{j-2}} w_{j-1}.$$

We get $w_3 = \frac{1}{\Phi_0\Phi_1}$, hence, by direct induction, the formula for $j \ge 3$:

$$w_j = \prod_{\ell=0}^{j-2} \frac{1}{\Phi_\ell}.$$

This concludes the proof. □

5.4 Minimization

We aim at minimizing the expected slowdown $\mathcal{S} = \frac{E_0}{M}$, for all possible values of M. For each value of M, the value of k is given by Equation (5.1.1) and is large enough to guarantee correctness. We do this with a simple exhaustive loop:

```
$M^{par}_{opt} = 0; k_{opt} = 0; \mathcal{S} = \infty$
for $M = 1$ to $UB$
  $k = \lceil \frac{D-1}{M} \rceil + 1$
  $E^{iter} = \frac{E_0}{M} = a_k \frac{C}{M} + b_k(1 + \frac{V}{M}) + c_k \frac{R}{M}$
  if $E^{iter} < \mathcal{S}$ then
     $M^{par}_{opt} = M; k_{opt} = k; \mathcal{S} = E^{iter}$
```

We choose the upper bound *UB* experimentally, using a very large value. Indeed, if there are (almost) no errors, we should use two arbitrarily long segments. Using the computation of E_0 above, the loop returns the best number of checkpoints, and the best number of iterations per segment, that minimizes the expected time per iteration.

6 Evaluation

In order to evaluate the different models, we have implemented a set of discrete events simulations that follow the protocols described above. The simulations are written in C, and are publicly available for reproducibility purposes at https://gitlab.inria.fr/therault/partial-detector-code. Each execution that we simulate is 100,000 iterations long, and as per our models above, C, V, R are all measured in terms of iterations. For each set of parameters, we simulate 10,000 executions and measure for each the total time of execution (*walltime*, in number of iterations), the number of errors, rollbacks, and the number of checkpoints taken. We then compute the mean and the variance of these values. Errors are injected following the model described in Section 3.

We have set the following parameters for the simulations: C and R are both equal and set to 3 iterations to account for the overheads in communications and I/O; V is set to 1 iteration, as verification is usually a fast operation (see [24] and the references therein). The other parameters of the simulation are varied to evaluate their respective impacts on the performance. f, the probability of occurrence of an error during an iteration, ranges between 10^{-4} (producing only a handful of errors during the 100,000 iterations of an execution) and $8.6 \cdot 10^{-3}$ (leading up to 4,500 errors during the execution); we consider all values for θ (probability of detecting an error during a verification phase) between 10% and 90%; as suggested by Table 1, we consider values of D ranging from 10 to 100.

In order to validate the model, we evaluate the protocols for all values of M, and set k using Equation (5.1.1). We compare the optimal value for (k, M^{par}_{opt}) predicted by the model with the experimental value.

6.1 Validation

Figure 2 shows, for a specific configuration ($D = 80, \theta = 0.4$), the optimal interval between checkpoints for the Partial Detector strategy (M^{par}_{opt}, on the left axis), and the walltime of a run of 100,000 iterations (on the right axis), as calculated by the theoretical model and as measured by the simulations. The goal of this figure is to validate that both the mathematical model and the simulation reach the same measurement on the key metrics. The walltime predicted by the model is slightly smaller than the walltime measured during simulations for very high f (when the frequency of errors is high), but the difference remains under 5%. Conjointly, M^{par}_{opt} according to the theoretical model is different by 1 iteration from the value extracted from the simulations, on a small range of low f values. These small differences are due to rounding errors in the simulation framework.

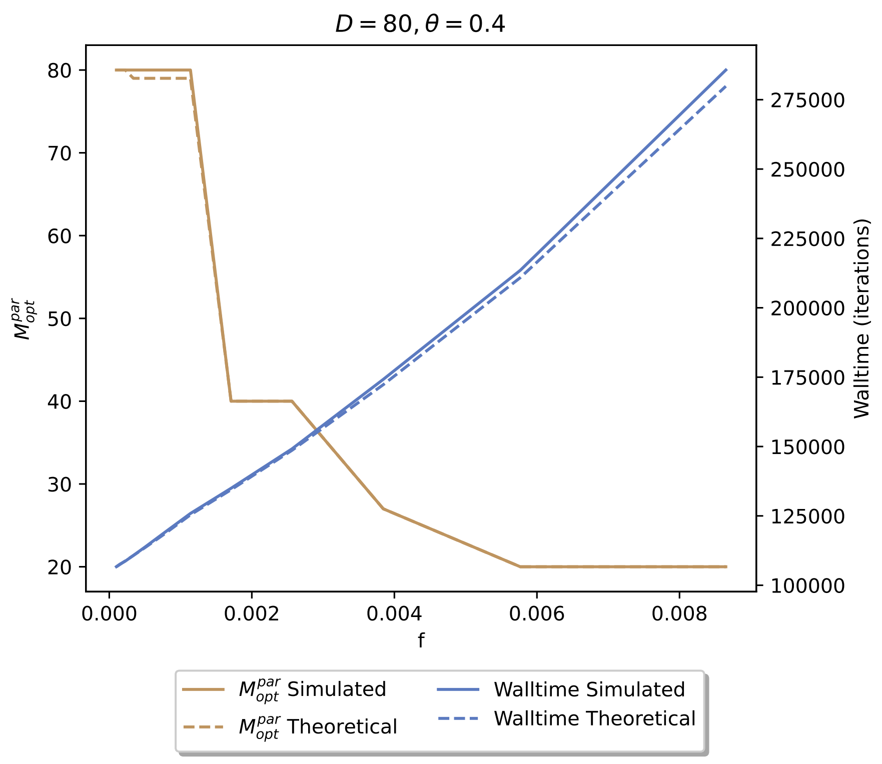


Figure 2: Comparison between the optimal values of (k, M_{opt}^{par}), and the walltime, predicted by the model and the simulations.

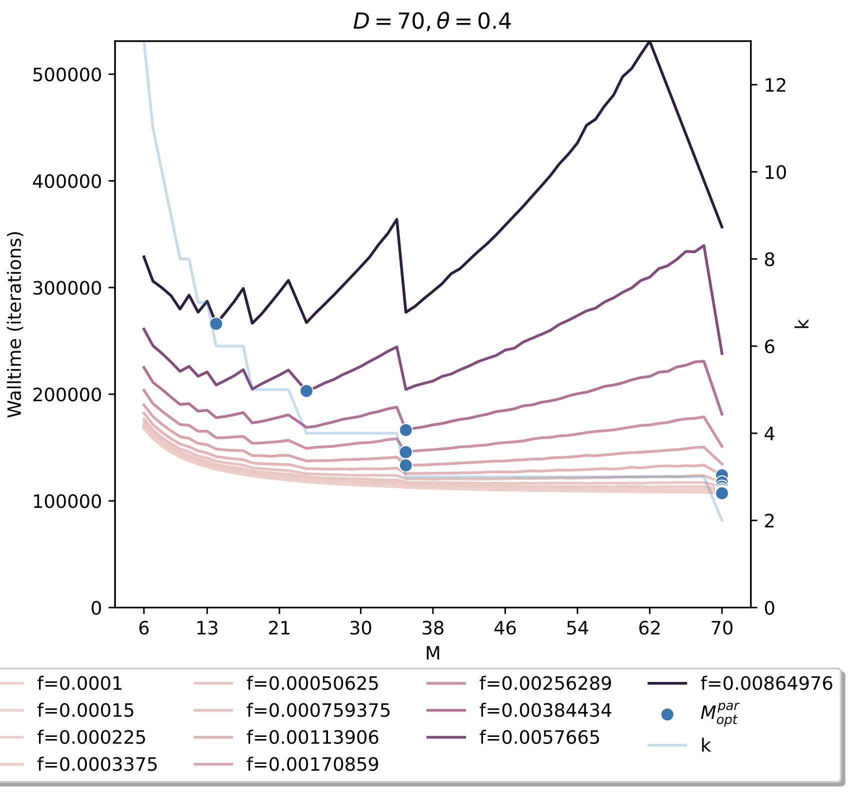


Figure 3: Simulated mean walltime for $\theta = 0.4$ and $D = 70$, varying the risk of error (f) and the number of iterations between two checkpoints (M).

6.2 Parameter exploration

Figure 3 shows the simulated mean walltime for $\theta = 0.4$ and $D = 70$, varying the risk of error (f) and the number of iterations between two checkpoints (M). For each value of f, we consider all possible values of M between C (number of iterations to take a single checkpoint) and D (maximum latency to detect an error). For each M, we set k, the number of checkpoints to keep, according to Equation (5.1.1). First, we observe that simulations with a low f ($f \leq 0.00113906$), taking more checkpoints than the minimal two or using an interval between checkpoints lower than D is detrimental to performance. Errors are so rare in this situation (low f), and the chance that they are detected right away is so high ($\theta = 0.4, D = 70$), that the most efficient strategy consists in waiting as long as possible before introducing a verification and a checkpoint.

When the risk of error increases, however, we see that the optimal strategy changes. The walltime figure then looks like a step-wise function, with steps happening every time M divides D into equal intervals. The low points of this step-wise function define a parabolic curve that accepts a minimal value of the walltime for various couples (k, M_{opt}^{par}) that are identified in the figure. For example, when $f = 0.00864976$, $k = 6, M = 14$ defines a low point, where the walltime is at 266,027 (so 2.66 times slower than the execution without faults nor fault tolerance enabled).

Figure 4 shows the number of errors observed during the simulations under the same conditions. We can see that in the worst case considered, the system could sustain close to 2,000 errors

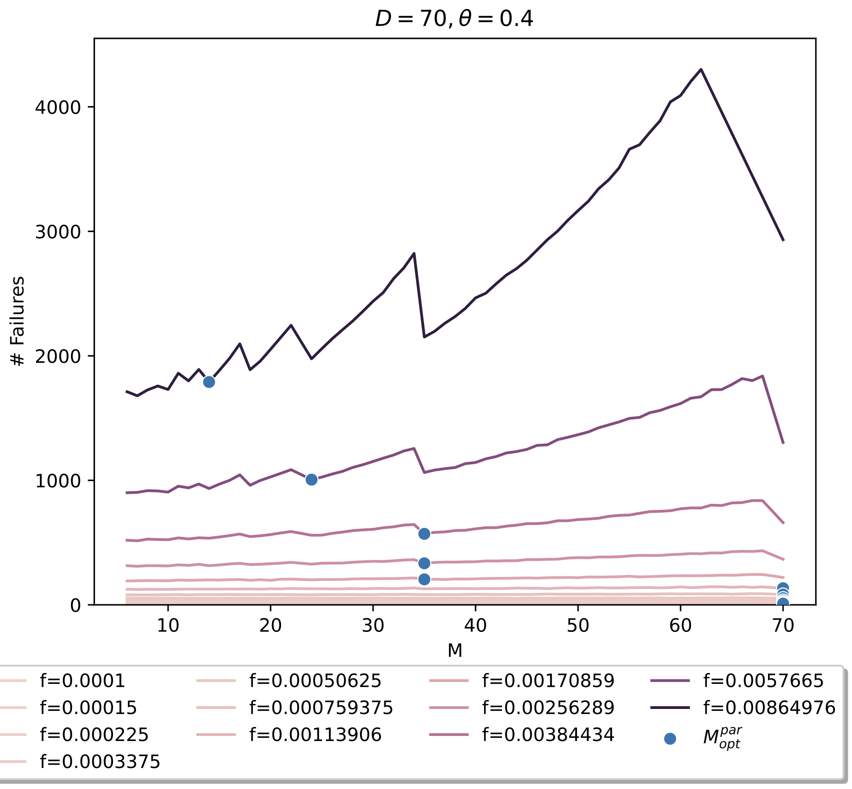


Figure 4: Number of errors observed during the simulations for $\theta = 0.4$ and $D = 70$, varying the risk of error (f) and the number of iterations between two checkpoints (M).

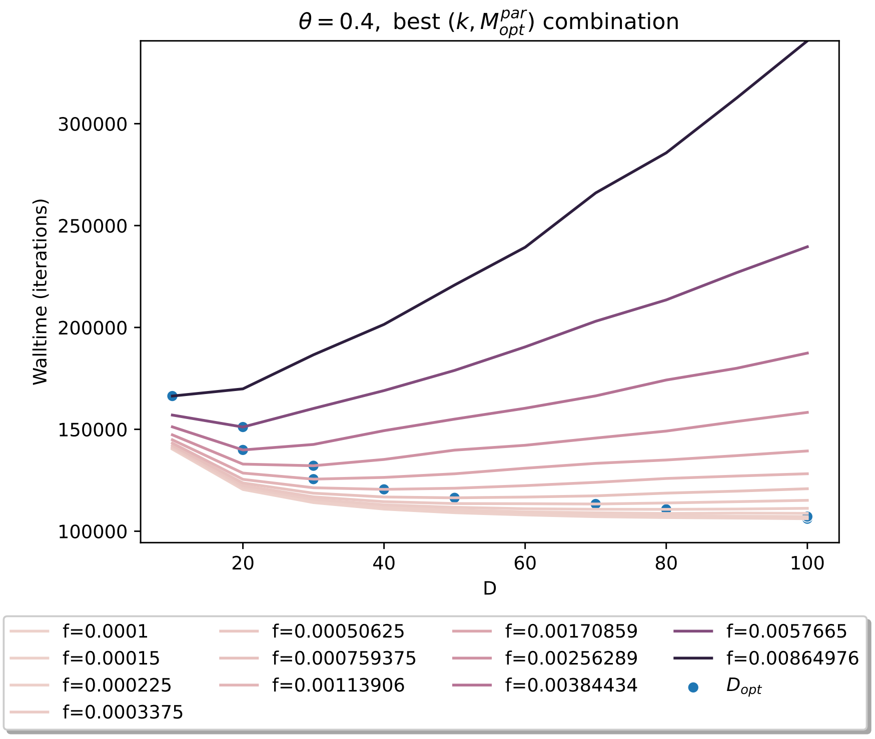


Figure 5: Impact of the detection bound (D) on the walltime for $\theta = 0.4$, varying the risk of error (f).

(distributed over a 100,000 compute iterations) while exhibiting the slowdown of 2.66 described above. With the wrong strategy, for example keeping the interval between two checkpoints at D, under the same conditions the slowdown would have been of 4 for a total of about 3,000 errors occurring during the execution.

Figure 5 shows the impact of the detection bound (D) on the walltime for $\theta = 0.4$, varying the risk of error (f). All curves show a parabolic behavior, with an optimal that depends on D. For experiments where the risk of error is low, a high D value turns out more efficient than a lower D value, which is to be expected as errors being rare, longer working periods allow to reduce the overheads due to forced (and useless) checkpoints. Three of the curves shown in this figure are limited to values of D that are suboptimal ($D = 100$), and even higher values of D could be explored. These curves represent the runs in which errors are the most unlikely to occur. For more volatile experiments, when f has a high value and many errors impact the system, a lower D value is optimal. However, D is not a parameter that the user can choose for performance reasons, but is strongly constrained by the properties of the partial detector. It is possible, depending on the detector properties, that D becomes higher than the optimal value. If this is the case, this figure also shows that performance can be significantly impacted, for highly volatile systems.

Figure 6 shows the impact of the detection probability (θ) on the walltime for $D = 10$, varying the risk of error (f). The figure shows that θ has no measurable impact on the performance, even for very low values of D (10 iterations). The same behavior is observed for higher values of D. This is because even for very unreliable error detectors ($\theta = 0.1$ for example), the vast

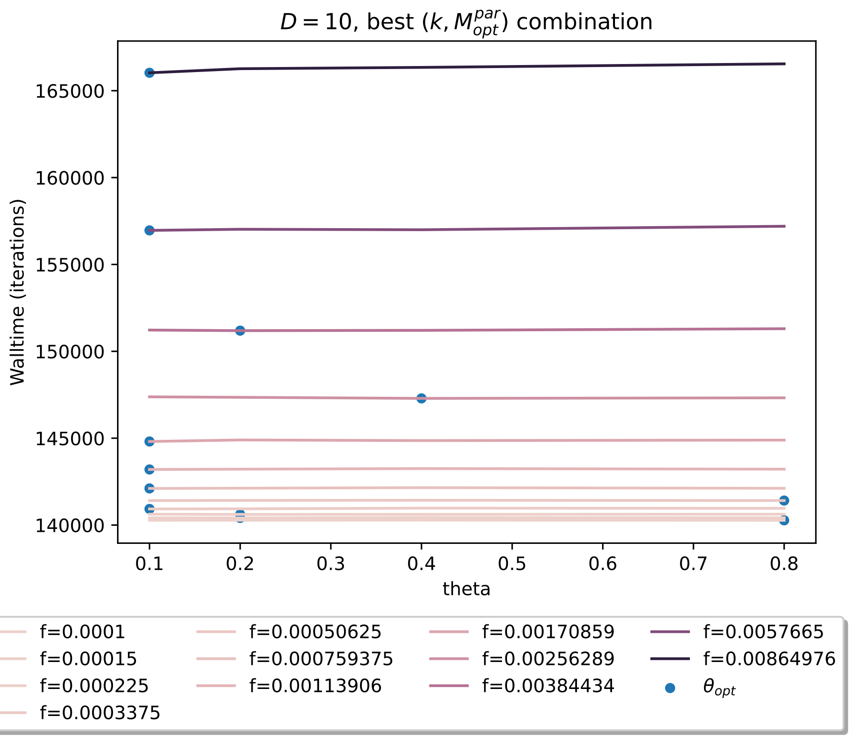


Figure 6: Impact of the detection probability (θ) on the walltime for $D = 10$, varying the risk of error (f).

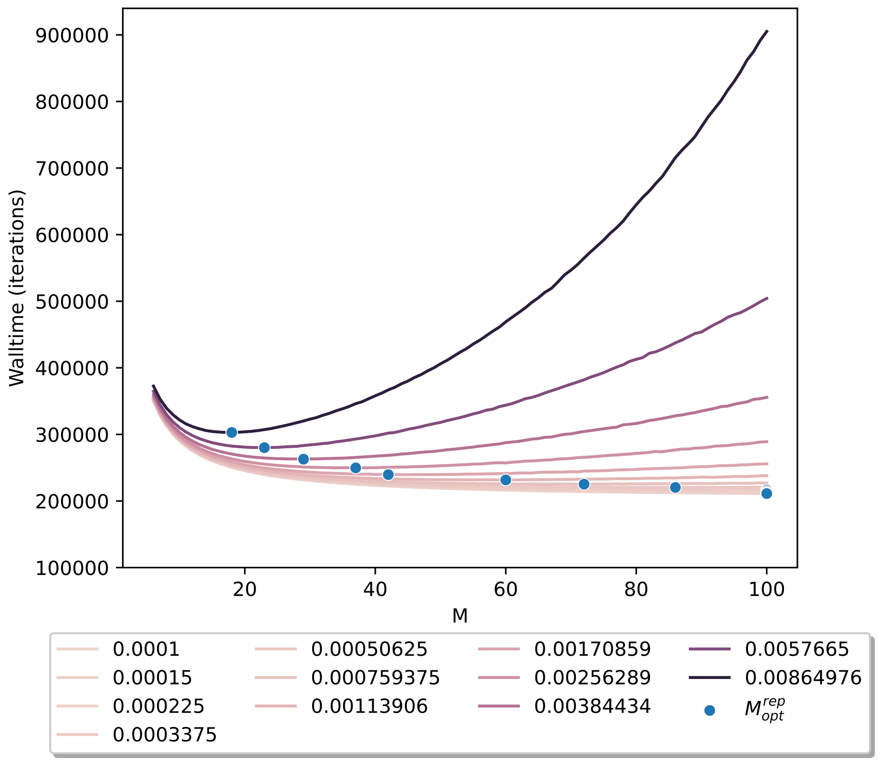


Figure 7: Walltime for replication, as a function of the interval between two checkpoints (M) and the risk of error (f).

majority of errors are detected before the bound D is reached. Thus, finding a tight D bound is critical for performance, much more than estimating closely the value of θ. Of course, both parameters are linked in real systems, and a low θ value will imply a high D bound, while a high θ value allows to consider lower values of D with a high probability.

6.3 Comparison with replication

Figure 7 shows the walltime for the replication strategy, as a function of the interval between two checkpoints (M) and the risk of error (f). As for the partial detector strategy, the walltime exhibits a parabolic behavior, with an optimal value M^{rep}_{opt} that depends on f. The figure shows the location of this optimal value for each different value of f. As f decreases, M^{rep}_{opt} increases, and more importantly, the walltime decreases to asymptotically reach a factor two when f is low enough and almost no errors hit the system. By construction, the replication strategy cannot be more efficient than this limit, as each segment must be executed at least twice to validate the result. This is the cost to pay to have a generic and perfect error mitigation mechanism.

Figure 8 shows the best interval between two checkpoints in the case of the replication strategy (M^{rep}_{opt}), compared between the simulation results and the theoretical model. Additionally, the second axis shows how many checkpoints need to be kept in the worst cases. The figure shows that the theoretical model is very close to the simulation results, for values of f bigger than 0.00050625. For $f < 0.00050625$, the simulation artificially bounded the maximum value of

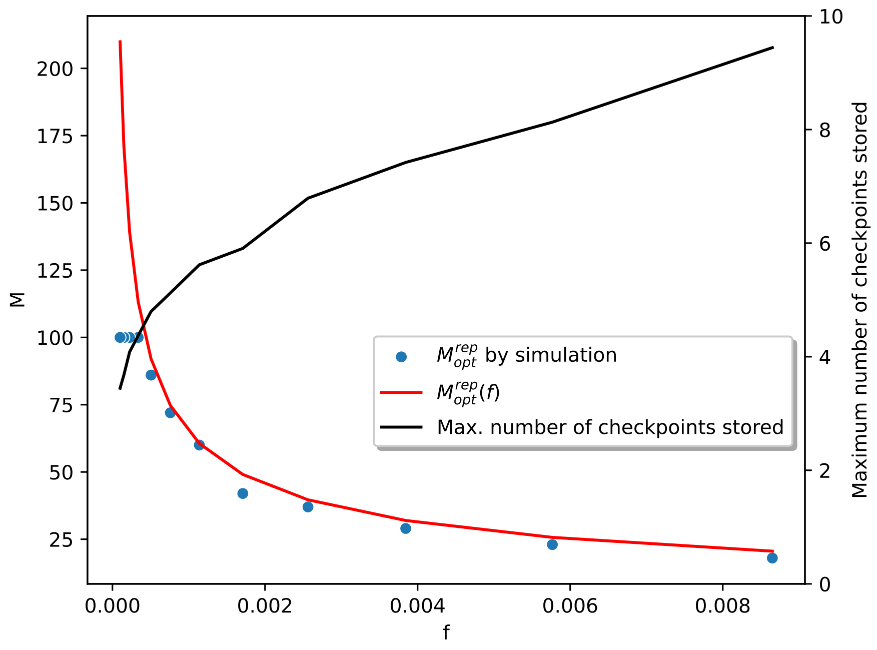


Figure 8: Best interval between two checkpoints for replication, compared between the simulation results and the theoretical model. Second axis shows how many checkpoints are required in the worst simulation.

M_{opt}^{rep} to D (as it was done in the case of the partial detector strategy), and the minimum value is observed in simulations for that bound, while the model predicts much higher values of M_{opt}^{rep}. The figure also shows that the number of checkpoints required in the worst simulation is relatively small, with a maximum of 10 checkpoints required in the worst case thatwe have considered.

Figure 9 compares the performance of the replication strategy with the partial detector strategy for a few values of D and θ. As we have observed that θ does not have a significant impact on the performance, we have chosen to show only the results for $\theta = 0.4$. The figure shows that the replication strategy is less efficient than the partial detector strategy in a large range of D and f values. Only when errors are very frequent (for $f = 0.008$, executions typically experience more than 4,000 errors to execute the 100,000 iterations), and the error detector is so unreliable that the bound for the maximum number of iterations to detect an error is 100, the replication strategy becomes more efficient.

7 Conclusion

This work is the first study devoted to comparing replication with the use of partial detectors with bounded latency. We have formalized the problem, and provided a detailed analysis of the optimal schemes for both approaches. Given the application and platform parameters, we derived the optimal number of iterations that should be executed between two checkpoints, as well as the number of checkpoints to keep in memory in the case of partial detectors.

Next, we have assessed and compared the performance of both strategies through a comprehensive set of Monte-Carlo simulations. Simulated results perfectly match the theoretical

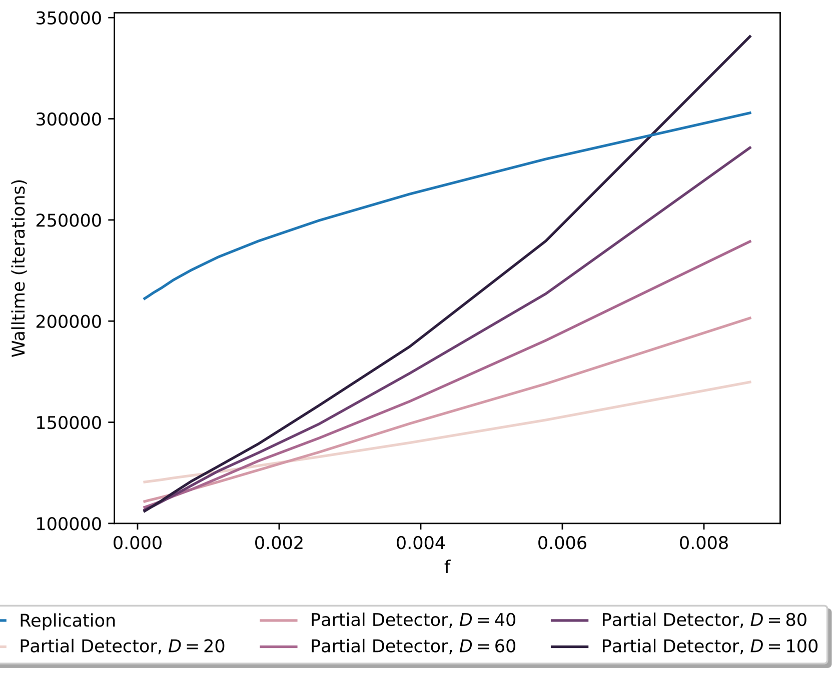


Figure 9: Comparison of the walltime of replication with partial detection for various values of D.

predictions, and illustrate the gain that results from using non-trivial patterns in terms of number of segments and number of iterations. More importantly, the simulations demonstrate that the use of partial detectors can massively outperform replication, allowing the application to complete within a significantly smaller time (typically, twice faster in the presence of few errors). Furthermore, the number of checkpoints that need to be stored to guarantee that we can always recover from a verified checkpoint is fixed with partial detectors, while it is unclear how many checkpoints will have to remain in memory when using replication.

While we have focused in this work on partial detectors with a perfect precision, our future work will extend the analysis to partial detectors whose precision is strictly lower than 1, i.e., with some false positives. We would need to account for the extra rollbacks, recoveries and re-executions that would be caused by the false alarms, even though there was no actual error, since we cannot take the risk of loosing the only verified checkpoint. An experimental validation with some iterative numerical application (the Preconditioned Conjugate Gradient method would be an ideal candidate) would also further demonstrate the usefulness of partial detectors.

References

[1] L. Bautista-Gomez, A. Benoit, A. Cavelan, S. Raina, Y. Robert, and H. Sun. Coping with recall and precision of soft error detectors. *JPDC*, 98:8–24, 2016.

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

[3] L. Bautista-Gomez, F. Zyulkyarov, O. Unsal, and S. McIntosh-Smith. Unprotected computing: A large-scale study of dram raw error rate on a supercomputer. In *SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 645–655, 2016.

[4] A. Benoit, A. Cavelan, Y. Robert, and H. Sun. Assessing general-purpose algorithms to cope with fail-stop and silent errors. *ACM TPC*, 3(2), 2016.

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

[6] 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 *HPDC*. ACM, 2015.

[7] 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.

[8] G. Bronevetsky and B. de Supinski. Soft error vulnerability of iterative linear algebra methods. In *ICS*. ACM, 2008.

[9] 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.

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

[11] T. Herault and Y. Robert, editors. *Fault-Tolerance Techniques for High-Performance Computing.* Springer Verlag, 2015.

[12] M. Heroux and M. Hoemmen. Fault-tolerant iterative methods via selective reliability. Research report SAND2011-3915 C, Sandia Nat. Lab., 2011.

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

[14] A. A. Hwang, I. A. Stefanovici, and B. Schroeder. Cosmic rays don't strike twice: understanding the nature of DRAM errors and the implications for system design. *SIGARCH Comput. Archit. News*, 40(1):111–122, 2012.

[15] S. Li, S. Di, K. Zhao, X. Liang, Z. Chen, and F. Cappello. Resilient error-bounded lossy compressor for data transfer. In *SC'94*, SC '21. ACM, 2021.

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

[17] 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.

[18] A. Moody, G. Bronevetsky, K. Mohror, and B. R. De Supinski. Design, modeling, and evaluation of a scalable multi-level checkpointing system. In *SC'10: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–11. IEEE, 2010.

[19] 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.

[20] P. Sao and R. Vuduc. Self-stabilizing iterative solvers. In *ScalA '13*, 2013.

[21] M. Shantharam, S. Srinivasmurthy, and P. Raghavan. Fault tolerant preconditioned conjugate gradient for sparse linear system solution. In *ICS*. ACM, 2012.

[22] M. Snir and et al. Addressing failures in exascale computing. *Int. J. High Perform. Comput. Appl.*, 28(2):129–173, 2014.

[23] V. Sridharan, N. DeBardeleben, S. Blanchard, K. B. Ferreira, J. Stearley, J. Shalf, and S. Gurumurthi. Memory errors in modern systems: The good, the bad, and the ugly. In *20th Int. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 297–310. ACM, 2015.

[24] A. Tremodeux, E. Agullo, A. Benoit, L. Giraud, T. Herault, and Y. Robert. Fault-tolerant numerical iterative algorithms at scale. Technical Report RR-9567, Inria Lyon, 2025.

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

[26] J. Ziegler, M. Nelson, J. Shell, R. Peterson, C. Gelderloos, H. Muhlfeld, and C. Montrose. Cosmic ray soft error rates of 16-Mb DRAM memory chips. *IEEE Journal of Solid-State Circuits*, 33(2):246–252, 1998.

[27] 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.

RESEARCH CENTRE
Centre Inria de Lyon

Bâtiment CEI-2, Campus La Doua
56, Boulevard Niels Bohr - CS 52132
69603 Villeurbanne

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

ISSN 0249-6399