Efficient checkpoint/verification patterns

Anne Benoit[a], Saurabh K. Raina[b], Yves Robert[a,c]

[a] *Laboratoire LIP, École Normale Supérieure de Lyon, France*
[b] *Department of CSE/IT, Jaypee Institute of Information Technology, Noida, India*
[c] *University of Tennessee Knoxville, USA*

Abstract

Errors have become a critical problem for high performance computing. Checkpointing protocols are often used for error recovery after fail-stop failures. However, silent errors cannot be ignored, and their peculiarity is that such errors are identified only when the corrupted data is activated. To cope with silent errors, we need a verification mechanism to check whether the application state is correct. Checkpoints should be supplemented with verifications to detect silent errors. When a verification is successful, only the last checkpoint needs to be kept in memory because it is known to be correct.

In this paper, we analytically determine the best balance of verifications and checkpoints so as to optimize platform throughput. We introduce a balanced algorithm using a pattern with p checkpoints and q verifications, which regularly interleaves both checkpoints and verifications across same-size computational chunks. We show how to compute the waste of an arbitrary pattern, and we prove that the balanced algorithm is optimal when the platform MTBF (Mean Time Between Failures) is large in front of the other parameters (checkpointing, verification and recovery costs). We conduct several simulations to show the gain achieved by this balanced algorithm for well-chosen values of p and q, compared to the base algorithm that always perform a verification just before taking a checkpoint ($p = q = 1$), and we exhibit gains of up to 19%.

Key words: High performance computing; fault tolerance; checkpointing; verification; silent error; silent data corruption.

Email addresses: Anne.Benoit@ens-lyon.fr (Anne Benoit), sk.raina@jiit.ac.in (Saurabh K. Raina), Yves.Robert@ens-lyon.fr (Yves Robert)

1. Introduction

With the advent of large-scale, massively parallel platforms, errors have become a critical problem for the HPC (High Performance Computing) community. Consider a large multicore node, say with one thousand cores, and assume, somewhat optimistically, that its MTBF (Mean Time Between Failures) is as large as 100 years. The path to Exascale computing [1] is to assemble a platform composed of one million such nodes. But with such a large number of nodes, the MTBF of the whole platform will only be 52 minutes, which means that most applications running on the platform for a few hours or more will experience a failure.

The de-facto general-purpose error recovery technique in HPC is checkpoint/restart [2, 3]. This technique employs checkpoints to periodically save the state of a parallel application, so that when an error strikes some process, the application can be restored into one of its former states. There are several families of checkpointing protocols, but they share a common feature: each checkpoint forms a consistent recovery line, i.e., when an error is detected, one can rollback to the last checkpoint and resume execution, after a downtime and a recovery time.

However, checkpoint and rollback recovery assumes instantaneous error detection, and therefore apply to fail-stop failures, such as the crash of a resource. In this work, we revisit checkpoint protocols in the context of *silent* errors, also called silent data corruptions. In HPC, it has been shown that such errors are not unusual, and must also be accounted for [4, 5, 6, 7, 8]. The cause may be for instance soft errors in L1 cache, or bit flips due to cosmic radiation. The problem is that the detection of a silent error is not immediate, because the error is identified only when the corrupted data is activated. If the error strikes a system before the last checkpoint, and is detected after that checkpoint, then the checkpoint is corrupted, and cannot be used to restore the application.

To alleviate this issue, one may envision to keep several checkpoints in memory, and to restore the application from the last *valid* checkpoint, thereby rolling back to the last *correct* state of the application [9]. This multiple-checkpoint approach has three major drawbacks. First, it is very demanding in terms of stable storage: each checkpoint typically represents a copy of the entire memory footprint of the application, which may well correspond to several terabytes. The second drawback is the possibility of fatal failures. Indeed, if we keep k checkpoints in memory, the approach assumes that the error that is currently detected did not strike before all the checkpoints still kept in memory, which would be fatal: in that latter case, all live checkpoints are corrupted, and one would have to re-execute the entire application from scratch. The probability of a fatal failure is evaluated in [10] for various error distribution laws and values of k. The third drawback of the approach is the most serious, and applies even without memory constraints, i.e., if we could store an infinite number of checkpoints in storage. The critical question is to determine which checkpoint is the last valid one. We need this information to safely recover from that point on. However, because of the detection latency, we do not know when the silent error has indeed occurred, hence we cannot identify the last valid checkpoint, unless some verification system is enforced.

The major objective of this paper is to introduce algorithms coupling verification and checkpointing, and to analytically determine the best balance of verifications between checkpoints so as to optimize platform throughput. In our (realistic) model, silent errors are detected only when some verification mechanism is executed. This approach is agnostic of the nature of this verification mechanism (checksum, error correcting code, coherence tests, etc.). This approach is also fully general-purpose, although application-specific information, if available, can always be used to decrease the cost of verification.

The simplest protocol (see Figure 1) would be to perform a verification just before taking each checkpoint. If the verification succeeds, then one can safely store the checkpoint and mark it as *valid*. If the verification fails, then an error has struck since the last checkpoint, which was duly verified, and one can safely recover from that checkpoint to resume the execution of the application. This protocol with verifications eliminates fatal errors that would corrupt all live checkpoints and cause to restart execution from scratch. However, we still need to assume that both checkpoints and verifications are executed in a reliable mode.

There is room for optimization. Consider the second pattern illustrated in Figure 2 with three verifications per checkpoint. There are three chunks of size w, each followed by a verification. Every third verification is followed by a checkpoint. We assume that $w = W/3$ to ensure that both patterns correspond to the same amount of work, W. Just as for the first pattern, a single checkpoint needs to be kept in memory, owing to

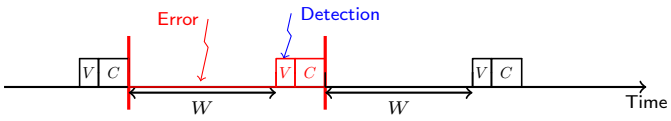


Figure 1: The first pattern with one verification before each checkpoint.

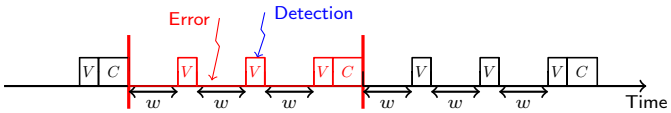


Figure 2: The second pattern with three verifications per checkpoint.

the verifications. Also, as before, each error leads to re-executing the work since the last checkpoint. But detection occurs much more rapidly in the second pattern, owing to the intermediate verifications. If the error strikes in the first of the three chunks, it is detected by the first verification, and only the first chunk is re-executed. Similarly, if the error strikes in the second chunk (as illustrated in the figure), it is detected by the second verification, and the first two chunks are re-executed. The entire pattern of work needs to be re-executed only if the error strikes during the third chunk. On average, the amount of work to re-execute is $(1 + 2 + 3)w/3 = 2w = 2W/3$. On the contrary, in the first pattern of Figure 1, the amount of work to re-execute always is W, because the error is never detected before the end of the pattern. Hence the second pattern leads to a 33% gain in re-execution time. However, this comes at the price of three times as many verifications. This overhead is paid in every failure-free execution, and may be an overkill if the verification mechanism is too costly.

This little example shows that the optimization problem looks difficult. It can be stated as follows: given the cost of checkpointing C, recovery R, and verification V, what is the optimal strategy to minimize the (expectation of the) execution time? A strategy is a periodic pattern of checkpoints and verifications, interleaved with work segments, that repeats over time. The length of the work segments also depends upon the platform MTBF μ. For example, with a single checkpoint and no verification (which corresponds to the classical approach for fail-stop failures), the optimal length of the work segment is known to be $\sqrt{2\mu C}$ (as given by Young [11]) or $\sqrt{2C(\mu + R)}$ (as given by Daly [12]). These well-known formulas are first-order approximations and are valid only if $C, R \ll \mu$ (in which case they collapse). Given a periodic pattern with checkpoints and verifications, can we extend these formulas and compute similar approximations?

This paper is an important extension of [10], where the idea of coupling checkpoints and verifications was originally introduced. In [10], only two simple patterns were proposed and analyzed: one pattern with k verifications, 1 checkpoint, and k same-size chunks (as in the example of Figure 2), and another pattern with k checkpoints, 1 verification, and k same-size chunks. These two imbalanced patterns were chosen for simplicity. The first pattern has much greater applicability, since a single checkpoint needs to be kept in memory. The second pattern requires k checkpoints to be kept in memory simultaneously, which is reasonable only for very small values of k. One major contribution of this paper is to analyze arbitrary patterns where p checkpoints and q verifications are interleaved, and where different-size chunks are allowed.

Given values of C and V, the cost of re-executing the work will dominate the waste due to failures as soon as the platform MTBF μ is large in front of these parameters. Here, the waste is defined as the fraction of time during which the processors do not perform useful work. As already mentioned, the waste due to failures must be traded-off with the waste incurred in a failure-free execution. In this context, we succeed in characterizing the optimal pattern with p checkpoints and q verifications, and we show that this pattern regularly interleaves both checkpoints and verifications across same-size chunks (algorithm BalancedAlgorithm, see Figure 3 for an example with $p = 2$ and $q = 5$). This important result fully characterizes the optimal strategy when μ is large in front of the resilience parameters C, R, and V. This result also shows (a posteriori) that using

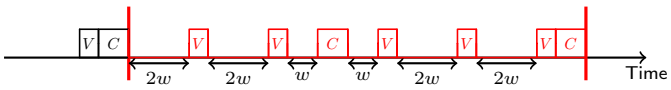


Figure 3: The BalancedAlgorithm with five verifications for two checkpoints.

same-size chunks in the case $p = 1$ and $q = k$, as proposed in [10], is indeed asymptotically optimal.

Finally, another important contribution of this paper is an exact computation of the waste for an arbitrary pattern with p checkpoints and q verifications, including the overhead of re-executing some verifications and recovering from invalid checkpoints when $p, q \geq 2$. These exact values of the waste are also compared through extensive simulations that involve realistic scenarios for petascale and exascale platforms.

We conclude this introduction by providing a practical example of the checkpoint and verification mechanisms that we study in this paper. A nice instantiation of this approach is given by Chen [13], who deals with sparse iterative solvers. Chen considers a simple method such as the PCG, the Preconditioned Conjugate Gradient method, and aims at protecting the execution from arithmetic errors in the ALU. Chen's approach performs a periodic verification every d iterations, and a periodic checkpoint every $d \times c$ iterations, which is a particular case of the pattern with $p = 1$ and $q = c$. For PCG, the verification amounts to checking the orthogonality of two vectors and to recomputing and checking the residual, while the cost of checkpointing is that of storing three vectors. The cost of a checkpoint is smaller than the cost of the verification, which itself is smaller than the cost of an iteration, especially when the preconditioner requires much more flops than a sparse matrix-vector product. In this context, Chen [13] shows how to numerically estimate the best values of the parameters d and c. Our results show using equidistant verifications, as suggested in [13], is asymptotically optimal when using a pattern with a single checkpoint (p =1), and enable to determine the best pattern with p checkpoints and q verifications as a function of C, R, and V, and the MTBF μ.

The rest of the paper is organized as follows. We survey related work in Section 2. We describe the performance model in Section 3. We show how to compute the waste of an arbitrary pattern in Section 4. In Section 5, we analyze the performance of a generic balanced algorithm, BalancedAlgorithm, which equipartitions p checkpoints and q verifications inside a pattern, for arbitrary values of p and q such that $p \leq q$. In Section 6, we show that this algorithm is optimal when the platform MTBF μ is large in front of the resilience parameters C, R and V, and we explain how to choose the optimal pattern given a set of parameters. In Section 7, we conduct several simulations that show the gain achieved by the balanced algorithm over the base algorithm with $p = q = 1$. We provide final remarks and hints for future work in Section 8.

2. Related work

Most traditional approaches maintain a single checkpoint. If the checkpoint file includes errors, the application faces an irrecoverable failure and must restart from scratch. This is because error detection latency is ignored in traditional rollback and recovery schemes. These schemes assume instantaneous error detection (therefore mainly targeting fail-stop failures) and are unable to accommodate silent errors. We focus in this section on related work about silent errors. A comprehensive list of techniques and references is provided by Lu, Zheng and Chien in [9].

Considerable efforts have been directed at error-checking to reveal silent errors. Error detection is usually very costly. Hardware mechanisms, such as ECC memory, can detect and even correct a fraction of errors, but in practice they are complemented with software techniques. The simplest technique is triple modular redundancy and voting [14]. For high-performance scientific applications, process replication (each process is equipped with a replica, and messages are quadruplicated) is proposed in the RedMPI library [15]. Another approach based on checkpointing and replication is proposed in [16], in order to detect and enable fast recovery of applications from both silent errors and hard errors.

Application-specific information can be very useful to enable ad-hoc solutions, that dramatically decrease the cost of detection. Many techniques have been advocated. They include memory scrubbing [17], but also

ABFT techniques [18, 19, 20], such as coding for the sparse-matrix vector multiplication kernel [20], and coupling a higher-order with a lower-order scheme for Ordinary Differential Equations [21]. These methods can only detect an error but do not correct it. Self-stabilizing corrections after error detection in the conjugate gradient method are investigated by Sao and Vuduc [22]. Also, Heroux and Hoemmen [23] design a fault-tolerant GMRES capable of converging despite silent errors, and Bronevetsky and de Supinski [24] provide a comparative study of detection costs for iterative methods. Elliot et al. [25] combine partial redundancy and checkpointing, and confirm the benefit of dual and triple redundancy. The drawback is that twice the number of processing resources is required (for dual redundancy).

As already mentioned, our work is agnostic of the underlying error-detection technique and takes the cost of verification as an input parameter to the model. To the best of our knowledge, the closest work is the preliminary study in [10], which we considerably extend by considering arbitrary patterns and different size chunks, as explained in Section 1.

3. Performance model

In this section, we introduce a performance model to assess the efficiency of any checkpoint/verification pattern. We enforce resilience through the use of a periodic pattern with p checkpoints and q verifications, and whose total length is $S = pC + qV + W$. Here, W is the work that is executed during the whole pattern, and it is divided into several chunks that are each followed by a verification, or a checkpoint, or both. Checkpoints and verifications are at arbitrary location within the pattern. The only constraint is that the pattern always ends by a verification immediately followed by a checkpoint: this is to enforce that the last checkpoint is always valid, thereby ruling out the risk of a fatal failure. In the example of Figure 2, we have three chunks of same size w, hence $W = 3w$ and $S = C + 3V + 3w$. The example of Figure 3 uses six chunks of size either w or $2w$, for a total work $W = 10w$, and $S = 2C + 5V + 10w$. The rationale for using such chunk sizes in Figure 3 is given in Section 5.

Consider a parallel application, and let T_{base} be the base time of its execution without any overhead due to resilience techniques (without loss of generality, assume unit-speed execution). We enforce resilience through the use of the periodic pattern described above, with p checkpoints, q verifications, work W, and total length $S = pC + qV + W$. We assume a a *selective reliability* model where checkpoint, recovery and verification are error-free operations. The problem is to compute the execution time of the application when silent errors can strike during execution. The input parameters are the following:

- the cost V of the verification mechanism;
- the cost C of a checkpoint;
- the cost R of a recovery;
- the platform MTBF μ.

First, assume a fault-free execution of the application: every pattern of length S, only W units of work are executed, hence the time T_{ff} for a fault-free execution is $T_{\text{ff}} = \frac{S}{W} T_{\text{base}}$. Now, let T_{final} denote the expectation of the execution time with silent errors taken into account. On average, errors occur every μ time-units. For each error, we lose $\mathcal{F}$ time-units on average (where $\mathcal{F}$ will be computed later), and there are $\frac{T_{\text{final}}}{\mu}$ errors during the execution. The value of $\mathcal{F}$ depends upon the pattern, see the examples below and the full derivation in Section 5. Altogether, we derive that

$$T_{\text{final}} = T_{\text{ff}} + \frac{T_{\text{final}}}{\mu} \mathcal{F}, \tag{1}$$

which we rewrite as

$$\begin{aligned} \big(1 - \textsc{Waste}\big) T_{\text{final}} &= T_{\text{base}}, \\ \text{with } \textsc{Waste} &= 1 - \big(1 - \frac{\mathcal{F}}{\mu}\big)\big(1 - \frac{pC + qV}{S}\big). \end{aligned} \tag{2}$$

The waste is the fraction of time where nodes do not perform useful computations. Minimizing execution time is equivalent to minimizing the waste. In Equation (2), we identify two sources of overhead: (i) the term $\textsc{Waste}_{\text{ff}} = \frac{pC+qV}{S}$, which is the waste due to checkpointing in a fault-free execution, by construction of the algorithm; and (ii) the term $\textsc{Waste}_{\text{fail}} = \frac{\mathcal{F}}{\mu}$, which is the waste due to errors striking during execution. With these notations, we have

$$\textsc{Waste} = \textsc{Waste}_{\text{fail}} + \textsc{Waste}_{\text{ff}} - \textsc{Waste}_{\text{fail}}\textsc{Waste}_{\text{ff}}. \tag{3}$$

To fully characterize the efficiency of a given pattern, there remains to determine $\mathcal{F}$, the (expected) time lost due to each failure. The value of $\mathcal{F}$ depends upon which pattern is used, and we compute it for arbitrary values of p and q in Section 5. We now give two examples.

The first example is for the simple protocol of Figure 1. We have $p = q = 1$, a single chunk of size $w = W$, and a pattern of size $S = C + V + W$. Computing $\mathcal{F}$ for this pattern goes as follows: whenever an error strikes, it is detected at the end of the work, during the verification. We first recover from the last checkpoint, then re-execute the entire work, and finally redo the verification. This leads to $\mathcal{F} = R+W+V = R+S-C$. From Equation (2), we obtain that

$$\textsc{Waste} = 1 - \big(1 - \frac{R+S-C}{\mu}\big)\big(1 - \frac{C+V}{S}\big) = aS + \frac{b}{S} + c, \tag{4}$$

where $a = \frac{1}{\mu}$, $b = (C+V)(1+\frac{C-R}{\mu})$ and $c = \frac{R-V-2C}{\mu}$. The value that minimizes the waste is $S = S_{\text{opt}}$, and the optimal waste is $\textsc{Waste}_{\text{opt}}$, where

$$S_{\text{opt}} = \sqrt{\frac{b}{a}} = \sqrt{(C+V)(\mu+C-R)} \quad \text{and} \quad \textsc{Waste}_{\text{opt}} = 2\sqrt{ab} + c. \tag{5}$$

We point out that this approach leads to a first-order approximation of the optimal pattern, not to an optimal value. This is because we have neglected the possibility of having more than one error within a pattern. In fact, this approach is valid when μ is large in front of S (and of all parameters R, C and V). When this is the case, we derive that $S_{\text{opt}} \approx \sqrt{(C+V)\mu}$ and $\textsc{Waste}_{\text{opt}} \approx 2\sqrt{\frac{C+V}{\mu}}$. It is very interesting to make a comparison with Young's or Daly's formula for the optimal checkpointing period T_{opt} when dealing with fatal failures: their formula writes $T_{\text{opt}} \approx \sqrt{2C\mu}$. In essence, the factor 2 comes from the fact that we re-execute only half the period on average with a fatal failure, because the detection is instantaneous. In our case, we always have to re-execute the entire pattern. And of course, we have to replace C by $C+V$, to account for the cost of the verification mechanism.

The second example is for the BalancedAlgorithm illustrated in Figure 3. We have $p = 2$, $q = 5$, six chunks of size w or $2w$, $W = 10w$, and a pattern of size $S = 2C + 5V + W$. Note that it may now be the case that we store an invalid checkpoint, if the error strikes during the third chunk (of size w, just before the non-verified checkpoint), and therefore we must keep two checkpoints in memory to avoid the risk of fatal failures. When the verification is done at the end of the fourth chunk, if it is correct, then we can mark the preceding checkpoint as valid and keep only this checkpoint in memory. Because $q > p$, there are never two consecutive checkpoints without a verification between them, and at most two checkpoints need to be kept in memory.

The time lost due to an error depends upon where it strikes:

- With probability $2w/W$, the error strikes in the first chunk. It is detected by the first verification, and the time lost is $R + 2w + V$, since we recover, and re-execute the work and the verification.
- With probability $2w/W$, the error strikes in the second chunk. It is detected by the second verification, and the time lost is $R + 4w + 2V$, since we recover, re-execute the work and both verifications.
- With probability w/W, the error strikes in the third chunk. It is detected by the third verification, and we roll back to the last checkpoint, recover and verify it. We find it invalid, because the error struck

before taking it. We roll back to the beginning of the pattern and recover from that checkpoint. The time lost is $2R+6w+C+4V$, since we recover twice, re-execute the work up to the third verification, re-do the checkpoint and the three verifications, and add the verification of the invalid checkpoint.

- With probability w/W, the error strikes in the fourth chunk. It is detected by the third verification. We roll back to the previous checkpoint, recover and verify it. In this case, it is valid, since the error struck after the checkpoint. The time lost is $R+w+2V$.
- With probability $2w/W$, the error strikes in the fifth chunk. Because there was a valid verification after the checkpoint, we do not need to verify it again, and the time lost is $R+3w+2V$.
- With probability $2w/W$, the error strikes in the sixth and last chunk. A similar reasoning shows that the time lost is $R+5w+3V$.

Averaging over all cases, we derive that $\mathcal{F} = \frac{11R}{10} + \frac{35w}{10} + \frac{C}{10} + \frac{22V}{10}$. We then proceed as with the first example to derive the optimal size S of the pattern. We optain $S_{\text{opt}} = \sqrt{\frac{b}{a}}$ and $\text{WASTE}_{\text{opt}} = 2\sqrt{ab} + c$ (see Equation (5)), where $a = \frac{7\mu}{20}$, $b = (2C+5V)(1-\frac{1}{20\mu}(22R-12C+9V))$ and $c = \frac{1}{20\mu}(22R-26C-17V)$.

When μ is large, we have $S_{\text{opt}} \approx \sqrt{\frac{20}{7}(2C+5V)\mu}$ and $\text{WASTE}_{\text{opt}} \approx 2\sqrt{\frac{7(2C+5V)}{20\mu}}$.

These examples are intended to prepare the reader for the computation of the waste in the general case. This computation is conducted in Section 4. Also, the examples are helpful to introduce the analysis of the waste when the platform MTBF μ is large in front of all resilience parameters R, C and V (Section 4.2).

4. Computing the waste

In this section, we generalize from the examples and provide a generic expression for the waste (Section 4.1). Then we derive the dominant term when the platform MTBF μ is large in front of all resilience parameters R, C and V (Section 4.2).

4.1. Exact expression

Consider a general pattern of size $S = pC+qV+W$, with $p \leq q$. Recall from Equation (3) that the total waste is $\text{WASTE} = \text{WASTE}_{\text{ff}} + \text{WASTE}_{\text{fail}} - \text{WASTE}_{\text{ff}}\text{WASTE}_{\text{fail}}$, where WASTE_{ff} is the waste without failures, that is the fraction of the time spent to do useless work each period, and $\text{WASTE}_{\text{fail}}$ is the waste due to errors striking during execution. We have $\text{WASTE}_{\text{ff}} = \frac{o_{\text{ff}}}{S}$, where $o_{\text{ff}} = pC+qV$ is the fault-free overhead due to inserting p checkpoints and q verifications within the pattern. We also have $\text{WASTE}_{\text{fail}} = \frac{\mathcal{F}}{\mu}$, where $\mathcal{F}$ is the time lost each time an error strikes.

The time lost $\mathcal{F}$ includes two components: re-executing a fraction of the total work W of the pattern, and computing additional verifications, checkpoints and recoveries (see both examples in Section 3). The general form of $\mathcal{F}$ is thus $\mathcal{F} = f_{\text{re}}W + \alpha$ where f_{re} stands for *fraction* of work that is *re-executed* due to failures; α is a constant that is a linear combination of C, V and R. For the first example (Figure 1), we have $f_{\text{re}} = 1$. For the second example (Figure 3), we have $f_{\text{re}} = \frac{7}{20}$ (recall that $w = W/10$).

For convenience, we use an equivalent form

$$\mathcal{F} = f_{\text{re}}S + \beta, \tag{6}$$

where $\beta = \alpha - f_{\text{re}}(pC+qV)$ is another constant. Plugging this expression back into the waste, we can generalize Equation (4) and derive that $\text{WASTE} = aS + \frac{b}{S} + c$, where

$$a = \frac{f_{\text{re}}}{\mu}, \quad b = o_{\text{ff}}\left(1 - \frac{\beta}{\mu}\right), \quad \text{and } c = \frac{1}{\mu}(\beta - o_{\text{ff}}f_{\text{re}}).$$

We then compute $S_{\text{opt}} = \sqrt{\frac{b}{a}}$ and $\text{WASTE}_{\text{opt}} = 2\sqrt{ab} + c$ as before. There remains to compute f_{re} and β, which is done in Section 5 for the general case of BALANCEDALGORITHM.

A word of caution. This approach is valid only when the length of the pattern $S = pC + qV + pqw$ is small in front of the MTBF μ: we need to enforce $S \ll \mu$. Indeed, we made a first-order approximation when implicitly assuming that we do not have more than one failure during the same period. This hypothesis is required to allow the expression of the model in a closed form. In fact, the number of failures during a pattern of length S can be modeled as a Poisson process of parameter $\frac{S}{\mu}$; the probability of having $k \geq 0$ failures is $\frac{1}{k!}(\frac{S}{\mu})^k e^{-\frac{S}{\mu}}$. Hence the probability of having two or more failures is $\pi = 1 - (1 + \frac{S}{\mu})e^{-\frac{S}{\mu}}$. For instance, enforcing the constraint $S \leq 0.1\mu$ leads to $\pi \leq 0.005$, hence a valid approximation when capping S to that value. Indeed, we have overlapping faults every 200 periods on average, so that our model is accurate for 99.5% of the checkpointing segments, hence it is quite reliable. In addition to the previous constraint, we must enforce the condition $S \geq pC + qV$, because the number of intervals and interval lengths are positive. The optimal value of S must therefore be chosen in the interval $[pC + qV,\ 0.1\mu]$.

4.2. When μ is large

When the platform MTBF μ is large in front of all resilience parameters R, C and V, we can identify the dominant term in the optimal waste $\text{WASTE}_{\text{opt}}$. Indeed, in that case, the constant β becomes negligible in front of μ, and we derive that

$$S_{\text{opt}} = \sqrt{\frac{o_{\text{ff}}}{f_{\text{re}}}} \times \sqrt{\mu} + o(\sqrt{\mu}), \tag{7}$$

and that the optimal waste is

$$\text{WASTE}_{\text{opt}} = 2\sqrt{o_{\text{ff}} f_{\text{re}}}\sqrt{\frac{1}{\mu}} + o(\sqrt{\frac{1}{\mu}}). \tag{8}$$

This equation shows that the optimal pattern when μ is large is obtained when the product $o_{\text{ff}} f_{\text{re}}$ is minimal. This calls for a trade-off, as a smaller value of o_{ff} with few checkpoints and verifications leads to a larger re-execution time, hence to a larger value of f_{re}.

We will use this characterization in terms of the product $o_{\text{ff}} f_{\text{re}}$ (and extend it to different-size chunks) to derive the optimal pattern in Section 6. For instance, coming back to the examples of Figures 1 and 3, we readily see that the second pattern is better than the first one for large values of μ whenever $V > 2C/5$, which corresponds to the condition $\frac{7}{20} \times (5V + 2C) > 1 \times (V + C)$.

5. The balanced algorithm

In this section, we analyze the generic balanced algorithm BALANCEDALGORITHM to place p checkpoints and q verifications for given values of W, p and q. As discussed before, we assume that $p \leq q$, so that no more than two checkpoints need to be kept in memory at each time. Then, we explain how to compute the optimal total workload W, given the parameters μ, V, C and R.

When $p = 1$, the algorithm follows the pattern illustrated in Figure 2 for $q = 3$: the total workload W is divided into q intervals of equal size $w = W/q$. The first $q - 1$ intervals are followed only by a verification, while the last interval is followed both by a checkpoint and a verification. With this pattern, note that only one checkpoint is kept in memory at all time, because the checkpoint is always verified by a verification immediately preceding it.

Other cases are such that $p > 1$ and p does not divide q; otherwise, we are back to the previous case with one checkpoint and q/p verifications. The idea is then to equally space each checkpoint and each verification, as was done in the pattern of Figure 3. Hence, we divide the total workload into $p \times q$ same-size intervals of size $w = W/pq$. Some intervals may be ended neither by a checkpoint nor by a verification, hence we end up for instance with chunks of size $2w$ in the example. Then, we place checkpoints at the end of intervals $i \times q$, for $1 \leq i \leq p$, and verifications at the end of intervals $j \times p$, for $1 \leq j \leq q$. If both a checkpoint and a verification happen at the end of the same interval (this is always the case for the last interval $p \times q$), then the checkpoint follows the verification. With this pattern, however, some intervals are ended only with a checkpoint, and hence this checkpoint is not verified. Therefore, we need to keep two checkpoints in memory while there has not been a valid verification after the non-verified checkpoint.

Next, we compute the waste in both cases ($p = 1$ and $p > 1$), as a function of p, q, the interval lengths w. As explained in Section 4, we have $o_{\text{ff}} = pC + qV$ and we only need to compute $\mathcal{F}$, the expectation of the amount of time spent to recover from a failure.

5.1. Computing $\mathcal{F}$ with $p = 1$

We first consider that $p = 1$. Recall that the total workload is divided into q intervals of size w. Consider that the failure strikes in interval i, with $1 \leq i \leq q$. Then $\mathcal{F}(i) = R + i(w + V)$: we need to recover from the failure, and we go back to the checkpoint (that was already verified) at the end of the previous period, hence we need to redo i intervals with their verifications. Therefore, because the probability that the error strikes during an interval is $1/q$, we obtain

$$\mathcal{F}(w, 1, q) = \frac{1}{q} \sum_{i=1}^{q} (R + i(w + V)) = R + \frac{q+1}{2}(w + V) = \frac{q+1}{2} w + c_1(q),$$

with $c_1(q) = R + \frac{q+1}{2} V$.

Finally, note that if $p = q = 1$, we have $w = W$ and we obtain the same result as in Section 3:

$$\mathcal{F}(w, 1, 1) = w + R + V \ .$$

5.2. Computing $\mathcal{F}$ with $p > 1$

We compute the time lost $\mathcal{F}(i)$ if failure strikes during interval i, for $1 \leq i \leq pq$. We express i as

$$i = ap + b, \quad i = a'q + b',$$

by making an euclidian division by p and q. For instance, $a = \lfloor i/p \rfloor$ and $b = i \mod p$.

Recall that checkpoints occur at the end of intervals $\ell \times q$, for $1 \leq \ell \leq p$, and verifications occur at the end of intervals $\ell' \times p$, for $1 \leq \ell' \leq q$. Therefore, the first verification following i occurs at the end of interval $NV(i)$ (next verification), where

$$NV(i) = \begin{cases} ap & \text{if } b = 0 \\ (a+1)p & \text{otherwise} \end{cases}$$

Also, the first checkpoint preceding i occurs at the end of interval $PC(i)$ (preceding checkpoint), where

$$PC(i) = \begin{cases} (a'-1)q & \text{if } b' = 0 \\ a'q & \text{otherwise} \end{cases}$$

Intervals $PC(i) + 1$ to $NV(i)$ will therefore need to be re-executed, because the failure is detected at the end of interval $NV(i)$, and the first valid checkpoint from which we will recover is at the end of interval $PC(i)$. We need to compute the number of verifications and checkpoints that occur during these intervals, that will have to be redone. Let $NbV(i)$ be the number of verifications between $PC(i)+1$ and $NV(i)$, and $NbC(i)$ be the number of checkpoints between $PC(i)+1$ and $NV(i)-1$ (a checkpoint at the end of interval $NV(i)$ will not be re-executed or loaded, because the verification occurs just before). For the number of verifications, let $PC(i) + 1 = cp + d$ (euclidian division) and $NV(i) = a''p$ (where $a'' = a$ or $a'' = a + 1$). If $d = 0$, then $NbV(i) = a'' - c' + 1$, otherwise $NbV(i) = a'' - c'$. For the number of checkpoints, let $NV(i) - 1 = c'q + d'$ (euclidian division). Then, $NbC(i) = c' - a' + 1$ if $b' = 0$, and $NbC(i) = c' - a'$ otherwise. Note that we count the number of checkpoints between i and $NV(i)$, because there are no additional checkpoints between $PC(i) + 1$ and i, by definition of $PC(i)$.

Let $Int(x)$ be a function returning 0 if x is an integer, and 1 otherwise. Indeed, for the first checkpoint from which we recover (at the end of $PC(i)$), we need to verify it only if it was not already verified, i.e., if $PC(i)/p$ is not an integer. Finally we can express $\mathcal{F}(i)$:

$\mathcal{F}(i) = (NV(i) - PC(i))w + NbV(i) \times V + NbC(i) \times (R + V + C) + R + Int(PC(i)/p)V.$

$(NV(i) - PC(i))w$ corresponds to the re-executed work; $NbV(i) \times V$ corresponds to the re-executed verifications (including the last one at the end of interval $NV(i)$); $NbC(i) \times (R+V+C)$ corresponds to the checkpoints from which we recover, between i and $NV(i)$ (none of them are already verified by definition of $NV(i)$, and therefore we pay $R+V+C$ for each of them); and finally R corresponds to the actual recovery from $PC(i)$, plus an eventual verification cost if this checkpoint was not already verified (i.e., if $PC(i)/p$ is not an integer).

By summing over all intervals, since the probabilities that the failure strikes in an interval are evenly balanced and equal to $1/pq$, we obtain the time lost with BalancedAlgorithm:

$$\mathcal{F}(w,p,q) = \frac{1}{pq}\sum_{i=1}^{pq}(NV(i) - PC(i))w + NbV(i) \times V + NbC(i) \times (R+V+C) + R + Int(PC(i)/p)V.$$

Note that with $p = 1$, we have $NV(i) = NbV(i) = i$, $PC(i) = NvC(i) = 0$, and $Int(PC(i)/p) = 0$, hence we obtain the same formula as in Section 5.1.

6. Asymptotic analysis

In this section, we focus on the case where the MTBF μ is large in front of the resilience parameters C, R and V. Recall from Equation (8) that the optimal waste is $\text{WASTE}_{\text{opt}} = 2\sqrt{o_{\text{ff}} f_{\text{re}}}\sqrt{\frac{1}{\mu}} + o(\sqrt{\frac{1}{\mu}})$. Consider a given pattern with p checkpoints and q verifications, where $p \leq q$. We have $o_{\text{ff}}(p,q) = pC + qV$ and we aim at minimizing $f_{\text{re}}(p,q)$, the expected fraction of the work that is re-executed. The major result of this section is that $f_{\text{re}}(p,q)$ is minimized when the pattern has pq same-size intervals and when the checkpoints and verifications are equally spaced among these intervals as in the BalancedAlgorithm, in which case $f_{\text{re}}(p,q) = \frac{p+q}{2pq}$. We first prove this important result for $p = 1$ in Section 6.1 before moving to the general case in Section 6.2. Finally, we explain how to choose the optimal pattern given values of C and V in Section 6.3.

6.1. Computing f_{re} when $p = 1$

Theorem 1. *The minimal value of $f_{re}(1,q)$ is obtained for same-size chunks and it is $f_{re}(1,q) = \frac{q+1}{2q}$.*

Proof. For $q = 1$, we already know from Section 3 that $f_{\text{re}}(1,1) = 1$. Consider a pattern with $q \geq 2$ verifications, executing a total work W. Let $\alpha_i W$ be the size of the i-th chunk, where $\sum_{i=1}^{q} \alpha_i = 1$ (see Figure 4). We compute the expected fraction of work that is re-executed when a failure strikes the pattern as follows. With probability α_i, the failure strikes in the i-th chunk. The error is detected by the i-th verification, we roll back to the beginning of the pattern, so we re-execute the first i chunks. Altogether, the amount of work that is re-executed is $\sum_{i=1}^{q}\left(\alpha_i \sum_{j=1}^{i} \alpha_j W\right)$, hence

$$f_{\text{re}}(1,q) = \sum_{i=1}^{q}\left(\alpha_i \sum_{j=1}^{i} \alpha_j\right). \tag{9}$$

What is the minimal value of $f_{\text{re}}(1,q)$ in Equation (9) under the constraint $\sum_{i=1}^{q} \alpha_i = 1$? We rewrite

$$f_{\text{re}}(1,q) = \frac{1}{2}\left(\sum_{i=1}^{q} \alpha_i\right)^2 + \frac{1}{2}\sum_{i=1}^{q} \alpha_i^2 = \frac{1}{2}\left(1 + \sum_{i=1}^{q} \alpha_i^2\right),$$

and by convexity, we see that f_{re} is minimal when all the α_i's have the same value $1/q$. In that case, we derive that $f_{\text{re}}(1,q) = \frac{1}{2}(1 + \sum_{i=1}^{q} \frac{1}{q^2}) = \frac{q+1}{2q}$, which concludes the proof. □

When $p = 1$, BalancedAlgorithm uses q same-size chunks. Theorem 1 shows that this is optimal.

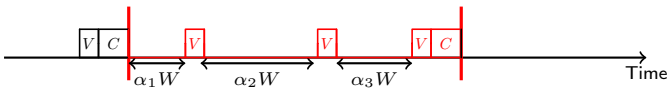


Figure 4: A pattern with different-size chunks, for $p = 1$ and $q = 3$.

6.2. Computing f_{re} when $p \geq 1$

Theorem 2. *For a pattern with $p \geq 1$, the minimal value of $f_{re}(p, q)$ is $f_{re}(p, q) = \frac{p+q}{2pq}$, and it is obtained with the* BalancedAlgorithm.

Proof. Consider an arbitrary pattern with p checkpoints, $q \geq p$ verifications and total work W. The repartition of the checkpoints and verifications is unknown, and different-size chunks can be used. The only assumption is that the pattern ends by a verification followed by a checkpoint.

The main idea of the proof is to compare the gain in re-execution time due to the $p-1$ intermediate checkpoints. Let $f_{\text{re}}^{(p)}$ be the fraction of work that is re-executed for the pattern, and let $f_{\text{re}}^{(1)}$ be the fraction of work that is re-executed for the same pattern, but where the $p-1$ first checkpoints have been suppressed. Clearly, $f_{\text{re}}^{(p)}$ is smaller than $f_{\text{re}}^{(1)}$, because the additional checkpoints save some roll-backs, and we aim at maximizing their difference.

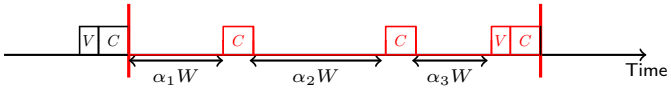


Figure 5: A pattern with different-size chunks, with 3 checkpoints (we do not show where intermediate verifications are located).

In the original pattern, let $\alpha_i W$ be the amount of work before the i-th checkpoint, for $1 \leq i \leq p$ (and with $\sum_{i=1}^{p} \alpha_i = 1$). Figure 5 presents an example with $p = 3$. What is the gain due to the presence of the $p-1$ intermediate checkpoints? If an error strikes before the first checkpoint, which happens with probability α_1, there is no gain, because we always rollback from the beginning of the pattern. This is true regardless of the number and repartition of the q verifications in the pattern. If an error strikes after the first checkpoint and before the second one, which happens with probability α_2, we do have a gain: instead of rolling back to the beginning of the pattern, we rollback only to the first checkpoint, which saves $\alpha_1 W$ units of re-executed work. Again, this is true regardless of the number and repartition of the q verifications in the pattern. For the general case, if an error strikes after the $(i-1)$-th checkpoint and before the i-th one, which happens with probability α_i, the gain is $\sum_{j=1}^{i-1} \alpha_j W$. We derive that

$$f_{\text{re}}^{(1)} - f_{\text{re}}^{(p)} = \sum_{i=1}^{p} \left(\alpha_i \sum_{j=1}^{i-1} \alpha_j \right).$$

Similarly to the proof of Theorem 1, we have

$$\sum_{i=1}^{p} \left(\alpha_i \sum_{j=1}^{i-1} \alpha_j \right) = \frac{1}{2} \left(\left(\sum_{i=1}^{p} \alpha_i \right)^2 - \sum_{i=1}^{p} \alpha_i^2 \right) = \frac{1}{2} \left(1 - \sum_{i=1}^{p} \alpha_i^2 \right)$$

and by convexity, the difference $f_{\text{re}}^{(1)} - f_{\text{re}}^{(p)}$ is maximal when $\alpha_i = 1/p$ for all i. In that latter case, $f_{\text{re}}^{(1)} - f_{\text{re}}^{(p)} = \sum_{i=1}^{p} (i-1)/p^2 = (p-1)/p^2$. This result shows that the checkpoints should be equipartitioned in the pattern, regardless of the location of the verifications.

To conclude the proof, we now use Theorem 1: to minimize the value of $f_{\text{re}}^{(1)}$, we should equipartition the verifications too. In that case, we have $f_{\text{re}}^{(1)} = \frac{q+1}{2q}$ and $f_{\text{re}}^{(p)} = \frac{q+1}{2q} - \frac{p-1}{2p} = \frac{q+p}{2pq}$, which concludes the proof. □

Theorem 2 shows that BalancedAlgorithm is the optimal pattern with p checkpoints and q verifications when μ is large. An important consequence of this result is that we never need to keep more than two checkpoints in memory when $p \leq q$, because it is optimal to regularly interleave checkpoints and verifications.

6.3. Choosing the optimal pattern

In this section, we outline a simple procedure to determine the best pattern. We start with the following result:

Theorem 3. *Assume that μ is large in front of C, R and V, and that $\sqrt{\frac{V}{C}}$ is a rational number $\frac{u}{v}$, where u and v are relatively prime. Then the optimal patten S_{opt} is obtained with the* BalancedAlgorithm*, using $p = u$ checkpoints, $q = v$ verifications, and pq equal-size chunks of total length $\sqrt{\frac{2pq(pC+qV)\mu}{p+q}}$.*

We prove this theorem before discussing the case where $\sqrt{\frac{V}{C}}$ is not a rational number.

Proof. Assume that $V = \gamma C$, where $\gamma = \frac{u^2}{v^2}$, with u and v relatively prime integers. Then, the product $o_{\text{ff}} f_{\text{re}}$ can be expressed as

$$o_{\text{ff}} f_{\text{re}} = \frac{p+q}{2pq}(pC + qV) = C \times \frac{p+q}{2}\left(\frac{1}{q} + \frac{\gamma}{p}\right).$$

Therefore, given a value of C and a value of V, i.e., given γ, the goal is to minimize the function $\frac{p+q}{2}\left(\frac{1}{q} + \frac{\gamma}{p}\right)$ with $1 \leq p \leq q$, and p, q taking integer values.

Let $p = \lambda \times q$. Then we aim at minimizing

$$\frac{1+\lambda}{2}\left(1 + \frac{\gamma}{\lambda}\right) = \frac{\lambda}{2} + \frac{\gamma}{2\lambda} + \frac{1+\gamma}{2},$$

and we obtain $\lambda_{opt} = \sqrt{\gamma} = \sqrt{\frac{V}{C}} = \frac{u}{v}$. Hence the best pattern is that returned by the BalancedAlgorithm with $p = u$ checkpoints and $q = v$ verifications. This pattern uses pq equal-size chunks whose total length is given by Equation (7), hence the result. □

For instance, for $V = 4$ and $C = 9$, we obtain $\lambda_{opt} = \sqrt{\frac{V}{C}} = \frac{2}{3}$, and a balanced pattern with $p = 2$ and $q = 3$ is optimal. This pattern will have 6 equal-size chunks whose total length is $\sqrt{\frac{12(2C+3V)\mu}{5}} = 6\sqrt{2\mu}$. However, if $V = C = 9$, then $\lambda_{opt} = 1$ and the best solution is the base algorithm with $p = q = 1$ and a single chunk of size $\sqrt{(C+V)\mu} = \sqrt{13\mu}$.

In some cases, $\lambda_{opt} = \sqrt{\frac{V}{C}}$ may not be a rational number, and we need to find good approximations of p and q in order to minimize the asymptotical waste. A solution is to try all reasonable values of q, say from 1 to 50, and to compute the asymptotic waste achieved with $p_1 = \lfloor \lambda_{opt} \times q \rfloor$ and $p_2 = \lceil \lambda_{opt} \times q \rceil$, hence testing at most 100 configurations (p, q). One can even further constrain the value of q if the capping of S is exceeded (see discussion in Section 4.1). Altogether, we can compute the best pattern with $q \leq 50$ in constant time.

7. Simulation results

The BalancedAlgorithm has been implemented and simulations have been conducted in Maple for a wide range of scenarios. We discuss these scenarios in Section 7.1, which is devoted to describing the simulation framework. In Section 7.2, we present the results, outlining the gain achieved by BalancedAlgorithm over the base case (where $p = q = 1$) , and deriving main conclusions from these simulations.

The Maple sheet is publicly available at [26], and users are invited to instantiate the model with their preferred parameters.

7.1. Simulation framework

This section provides information about the parameters used for instantiating the performance model for the BalancedAlgorithm. We have chosen realistic parameters that depict large-scale platforms. In the first instance, we consider three main scenarios for three different values of $C = R = \{600, 300, 100\}$ seconds. In each scenario, we use five different platform MTBFs and six values for the ratio $\gamma = V/C$ (where $V \leq C$). In the second instance, we fix $C = 600$ and report the values p_{opt} and q_{opt}, the number of checkpoints and verifications which allows us to produce an optimal pattern S_{opt}. We also report the optimal waste, the waste due to base algorithm, and the gain (in %) achieved. For a component MTBF of 100 years, we have considered five different platforms with the total number of nodes = $\{10^2, 10^3, 10^4, 10^5, 10^6\}$, which correspond to a platform MTBF μ ranging from $\mu = 100 \times 365 \times 24/100 = 8760$ hours (365 days) down to $\mu = 3153.6$ sec ($\approx$ 52 min). For each μ, results are reported for 13 different $\gamma = V/C$ ratios, where $\gamma = 0.025, 0.05, 0.075, 0.1, 0.2, \ldots, 1$. Since λ_{opt} (or $\sqrt{\gamma}$) may not be rational, in order to find integral values of p_{opt} and q_{opt}, we test all possible configurations such that $1 \leq p \leq q \leq 10$ to obtain p and q with the minimal waste. We used Maple to analytically compute the waste and to plot the gain achieved over the base case.

7.2. Results and analysis

Based on the above framework, in this section we report the results through six plots in Figure 6 and a table of values in Table 1. Regarding the accuracy of our model and the computation of waste values, we make the following statements:

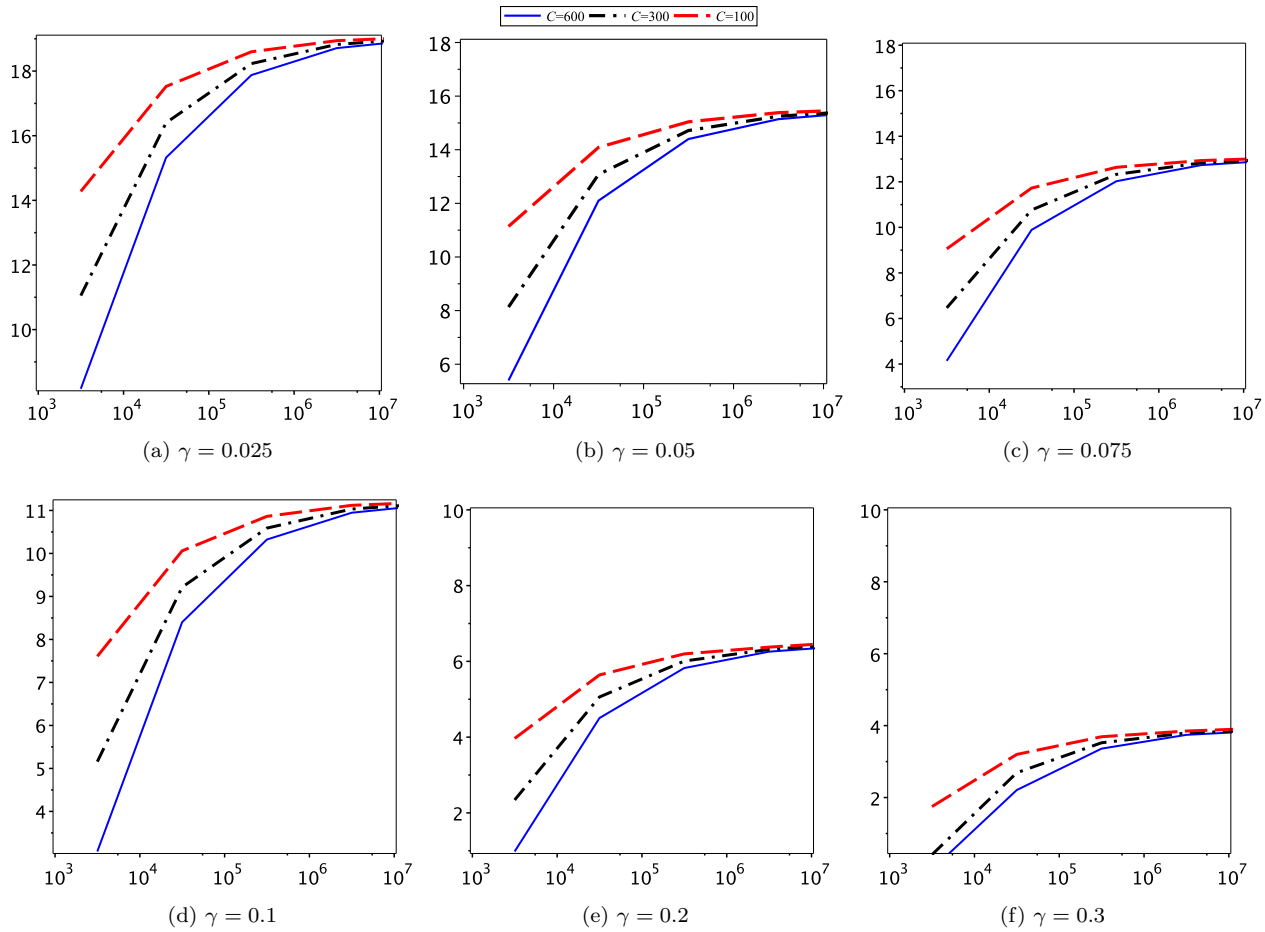
1. For the platform with 10^6 nodes when $C = 600$, S_{opt} lies in the interval $[pC + qV,\ 0.6\mu]$. So in this case our model is accurate for 88% of the checkpointing segments. For $C = 300$ and 100, S_{opt} is less than 0.4μ (94% accuracy) and 0.2μ (98% accuracy) respectively.
2. For the platform with 10^5 nodes, S_{opt} always lies in the interval $[pC+qV,\ 0.2\mu]$. For all other platforms, $S_{\text{opt}} \leq 0.1\mu$.

Figure 6 provides an overall view of the performance of the BalancedAlgorithm and gives an insight into the behavior of this algorithm for different values of the resilience parameters C, R and V. Table 1 provides, for $C = 600$, gain values for γ ranging from 0.025 to 1 with the best pattern and the corresponding waste when μ is fixed. Each plot, representative of a certain V/C ratio, reports gain of the optimal pattern over the base case for $C = 600$, 300 and 100 as a function of the platform MTBF μ (plotted on a log scale). Plots show that the maximum gain achieved is 19.05% for $\gamma = 0.025$ when $C = 100$ for a platform having 100 nodes. In every plot, it can be observed that gain drops as the number of nodes increases. This is to be expected, since an increase in failure-prone processing elements would reduce the mean time to failure. This shortens S_{opt} roughly by one-third with every ten-fold increase in nodes as $S_{\text{opt}} \propto 1/p\sqrt{N}$, where N is the number of nodes and p is the number of checkpoints on the optimal pattern. Thus, the optimal pattern reduces to the base case pattern in such large configurations.

In Table 1, it can also be observed that an increase in the cost of verification leads the gain to drop drastically, as quickly as $\gamma \geq 0.3$ for a platform with 10^6 nodes and as slowly as $\gamma \geq 0.8$ for a platform with 10^2 nodes. This corroborates with Equation (8). An increase in the cost of verification, as well as in the number of verifications and checkpoints, leads to a larger value of o_{ff}. The BalancedAlgorithm, by construction, distributes q/p verifications every checkpoint. We see that this results into 6 verifications in the best case giving $f_{\text{re}} = 0.58$, and 1 verification in the worst case (base case) giving $f_{\text{re}} = 0.1$. Since o_{ff} as well as f_{re} increases, waste increases and optimal pattern approaches the base case.

8. Conclusion

In this paper, we revisit traditional checkpointing strategies in the context of silent data corruption errors. These are latent errors that cannot be detected immediately because they are identified only when the corrupted data is activated. Strategies by Young and Daly [11, 12] cannot be relied upon because they assume instantaneous error detection. Due to occurrence of silent errors, the checkpoint taken during the



(a) $\gamma = 0.025$ (b) $\gamma = 0.05$ (c) $\gamma = 0.075$

(d) $\gamma = 0.1$ (e) $\gamma = 0.2$ (f) $\gamma = 0.3$

Figure 6: Gain (in % on y-axis) achieved, for optimal values of p and q over the base case $p = q = 1$ when $C = 600$ (10 minutes), $C = 300$ (5 minutes) and $C = 100$ (≈ 1.6 minutes) as a function of the platform MTBF μ (on log scale on x-axis) for six different V/C ratios.

computation may not be used to recover as it might itself be a corrupted checkpoint. In order to safely recover from a checkpoint, we must ensure that the error occurred after this checkpoint. This can be done with an error detection mechanism that periodically performs a verification. To incorporate this, we devised an algorithm, called BalancedAlgorithm, coupling verification and checkpointing so that verification checks whether there is an error or not. If there is no error, then the previous checkpoint can be marked as valid. Otherwise, we recover from the previous checkpoint; if this was not a validated checkpoint, then we add a verification just after recovery to check whether this checkpoint is valid, and go back to the preceding checkpoint if it is not valid. The algorithm produces a repetitive periodic pattern of equally-spaced checkpoints and equally-spaced verifications, interleaved with work segments. Periodicity is governed by the number of checkpoints and verifications that can be executed within each pattern. When there are more verifications than checkpoints, this pattern obviates the need to keep more than two checkpoints in memory.

We have also presented a performance model to assess the efficiency of any checkpoint/verification pattern. This model provides an expression for the exact computation of the optimal pattern as well as the optimal waste for any combination of checkpoints and verifications. We establish the fact that when the MTBF μ is large in front of all resilience parameters R, C and V, the optimal waste is obtained by minimizing the

product of two negatively correlated variables o_{ff} (fault-free overhead) and f_{re} (re-executed fraction of work due to failures). The performance model has been applied to analyze the generic BalancedAlgorithm. For fixed values of p and q, and hence with $o_{\text{ff}} = pC + qV$, we have proved that f_{re} is minimized when the pattern has pq same-size intervals and when the checkpoints and verifications are equally spaced among these intervals. Thus, we conclude that BalancedAlgorithm produces an optimal pattern with p checkpoints and q verifications. Finally, we demonstrate how to choose the optimal pattern for different verification and checkpointing costs. We found that a pattern with p checkpoints and q verifications is optimal when $p/q = \sqrt{V/C}$. Therefore, if $V = C$, it is worth doing a verification immediately followed by checkpoint only at the end of the pattern, that is, obtaining a base pattern with $p = q = 1$.

We have instantiated the performance model with realistic parameters, and we have compared the BalancedAlgorithm with the case $p = q = 1$. Simulation results show a maximum gain of up to 19% and findings corroborate the theoretical analysis. Overall, we have analytically determined the best balance of verifications between checkpoints so as to optimize platform throughput for given costs of R, C and V, we proved that the balanced algorithm produces an optimal pattern, and we evaluated this algorithm for multiple scenarios.

We focused in this work on cases where $V \leq C$, hence with more verifications than checkpoints, but all results hold in the symmetrical case where $C \leq V$. However, we did not detail these results because they would imply to keep more than two checkpoints in memory, which may be difficult.

A future research direction would be to consider different kinds of applications, such as graphs of computational tasks where checkpoints and verifications could be taken at the end of a task. It would then not be possible to regularly interleave checkpoints and verifications anymore, and new strategies to decide where to place checkpoints and verifications would have to be designed.

Another possible extension would be to deal with several verifications mechanisms, each with a different cost and a different recall. The *recall* is defined as the fraction of errors that the verification mechanism can detect. In this paper, we have assumed a single verification mechanism with a perfect recall equal to 1. Dealing with imperfect mechanisms is quite an interesting question. A high-recall mechanism is likely to have a high cost, hence trade-offs would have to be identified in order to select which mechanism to use, and how frequently. The corresponding optimization problem seems very challenging.

Acknowledgments

This work was supported in part by the ANR *RESCUE* project. Y. Robert is with the Institut Universitaire de France. The authors would like to thank Dr. Vivek Kumar Singh at Text Analytics Laboratory of South Asian University and Ms Hema N. at JIIT for providing their computing facility to conduct simulations.

References

[1] 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) (2009) 309–322.

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

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

[4] 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) (1994) 553–557.

[5] J. F. Ziegler, H. W. Curtis, H. P. Muhlfeld, C. J. Montrose, B. Chin, IBM Experiments in Soft Fails in Computer Electronics, IBM J. Res. Dev. 40 (1) (1996) 3–18.

[6] J. Ziegler, H. Muhlfeld, C. Montrose, H. Curtis, T. O'Gorman, J. Ross, Accelerated testing for cosmic soft-error rate, IBM J. Res. Dev. 40 (1) (1996) 51–72.

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

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

[9] G. Lu, Z. Zheng, A. A. Chien, When is multi-version checkpointing needed, in: 3rd Workshop for Fault-tolerance at Extreme Scale (FTXS), ACM Press, 2013, `https://sites.google.com/site/uchicagolssg/lssg/research/gvr`.

[10] G. Aupy, A. Benoit, T. Hérault, Y. Robert, F. Vivien, D. Zaidouni, On the combination of silent error detection and checkpointing, in: PRDC 2013, the 19th IEEE Pacific Rim International Symposium on Dependable Computing, IEEE Computer Society Press, 2013.

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

[12] J. T. Daly, A higher order estimate of the optimum checkpoint interval for restart dumps, FGCS 22 (3) (2004) 303–312.

[13] Z. Chen, Online-ABFT: An Online Algorithm Based Fault Tolerance Scheme for Soft Error Detection in Iterative Methods, in: Proc. 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP '13, ACM, 2013, pp. 167–176.

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

[15] D. Fiala, F. Mueller, C. Engelmann, R. Riesen, K. Ferreira, R. Brightwell, Detection and correction of silent data corruption for large-scale high-performance computing, in: Proc. of the ACM/IEEE SC Int. Conf., SC '12, IEEE Computer Society Press, 2012.

[16] X. Ni, E. Meneses, N. Jain, L. V. Kalé, ACR: Automatic Checkpoint/Restart for Soft and Hard Error Protection, in: Proc. Int. Conf. High Performance Computing, Networking, Storage and Analysis, SC '13, ACM, 2013.

[17] A. A. Hwang, I. A. Stefanovici, 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) (2012) 111–122.

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

[19] G. Bosilca, R. Delmas, J. Dongarra, J. Langou, Algorithm-based fault tolerance applied to high performance computing, J. Parallel and Distributed Computing 69 (4) (2009) 410 –416.

[20] M. Shantharam, S. Srinivasmurthy, P. Raghavan, Fault tolerant preconditioned conjugate gradient for sparse linear system solution, in: Proc. ICS '12, ACM, 2012.

[21] A. R. Benson, S. Schmit, R. Schreiber, Silent error detection in numerical time-stepping schemes., CoRR abs/1312.2674.

[22] P. Sao, R. Vuduc, Self-stabilizing iterative solvers, in: Proc. ScalA '13, ACM, 2013.

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

[24] G. Bronevetsky, B. de Supinski, Soft error vulnerability of iterative linear algebra methods, in: Proc. 22nd Int. Conf. on Supercomputing, ICS '08, ACM, 2008, pp. 155–164.

[25] J. Elliott, K. Kharbas, D. Fiala, F. Mueller, K. Ferreira, C. Engelmann, Combining partial redundancy and checkpointing for HPC, in: Proc. ICDCS '12, IEEE Computer Society, 2012.

[26] Maple sheets for the experiments, `http://graal.ens-lyon.fr/~yrobert/silent-errors/`.

$C = 600$					
	γ	p_{opt}, q_{opt}	$\text{WASTE}_{\text{opt}}$	$\text{WASTE}_{\text{base}}$	GAIN (in %)
$\mu = 100y/10^2$	0.025	1, 6	0.007140	0.008812	18.97
	0.05	2, 9	0.007543	0.008919	15.4
	0.075	2, 7	0.007853	0.009024	13
	0.1	1, 3	0.008111	0.009129	11.15
	0.2	4, 9	0.008922	0.009534	6.42
	0.3	1, 2	0.009538	0.009922	3.9
	0.4	2, 3	0.010062	0.010295	2.27
	0.5	2, 3	0.010522	0.010656	1.25
	0.6	3, 4	0.010940	0.011004	0.58
	0.7	5, 6	0.011326	0.011342	0.14
	0.8	1, 1	0.011670	0.011670	0
	0.9	1, 1	0.011989	0.011989	0
	1.0	1, 1	0.012299	0.012299	0
$\mu = 100y/10^3$	0.025	1, 6	0.022545	0.027734	18.7
	0.05	1, 4	0.023818	0.028068	15.1
	0.075	1, 4	0.024782	0.028398	12.7
	0.1	1, 3	0.025579	0.028724	10.94
	0.2	1, 2	0.028115	0.029992	6.3
	0.3	1, 2	0.030038	0.031207	3.7
	0.4	2, 3	0.031738	0.032375	1.96
	0.5	2, 3	0.033185	0.033501	0.94
	0.6	3, 4	0.034532	0.034591	0.17
	0.7	1, 1	0.035645	0.035645	0
	0.8	1, 1	0.036669	0.036669	0
	0.9	1, 1	0.037664	0.037664	0
	1.0	1, 1	0.038633	0.038633	0
$\mu = 100y/10^4$	0.025	1, 6	0.070931	0.086370	17.9
	0.05	1, 4	0.074809	0.087393	14.4
	0.075	1, 4	0.077774	0.088404	12
	0.1	1, 3	0.080173	0.089402	10.3
	0.2	1, 2	0.087848	0.093281	5.8
	0.3	1, 2	0.093734	0.096992	3.4
	0.4	1, 2	0.099244	0.100557	1.3
	0.5	1, 1	0.103990	0.103990	0
	0.6	1, 1	0.107303	0.107303	0
	0.7	1, 1	0.110509	0.110509	0
	0.8	1, 1	0.113616	0.113616	0
	0.9	1, 1	0.116633	0.116633	0
	1.0	1, 1	0.119567	0.119567	0
$\mu = 100y/10^5$	0.025	1, 6	0.219985	0.259794	15.3
	0.05	1, 4	0.230920	0.262704	12.1
	0.075	1, 3	0.239313	0.265573	9.9
	0.1	1, 3	0.245857	0.268405	8.4
	0.2	1, 2	0.266787	0.279368	4.5
	0.3	1, 2	0.283405	0.289805	2.2
	0.4	1, 2	0.298839	0.299776	0.31
	0.5	1, 1	0.309330	0.309330	0
	0.6	1, 1	0.318508	0.318508	0
	0.7	1, 1	0.327345	0.327345	0
	0.8	1, 1	0.335870	0.335870	0
	0.9	1, 1	0.344110	0.344110	0
	1.0	1, 1	0.352085	0.352085	0
$\mu = 100y/10^6$	0.025	1, 6	0.631979	0.688195	8.2
	0.05	1, 4	0.6567004	0.694144	5.4
	0.075	1, 3	0.670993	0.699967	4.1
	0.1	1, 2	0.684016	0.705668	3.1
	0.2	1, 2	0.720191	0.727326	0.98
	0.3	1, 1	0.747322	0.747322	0
	0.4	1, 1	0.765844	0.765844	0
	0.5	1, 1	0.783046	0.783046	0
	0.6	1, 1	0.799061	0.799061	0
	0.7	1, 1	0.813996	0.813996	0
	0.8	1, 1	0.827946	0.827946	0
	0.9	1, 1	0.840992	0.840992	0
	1.0	1, 1	0.853205	0.853205	0

Table 1: Optimal values of p and q and the corresponding waste with gain over the base case when C is fixed at 600 and γ varies from 0.025 to 1.0 for five different values of μ.