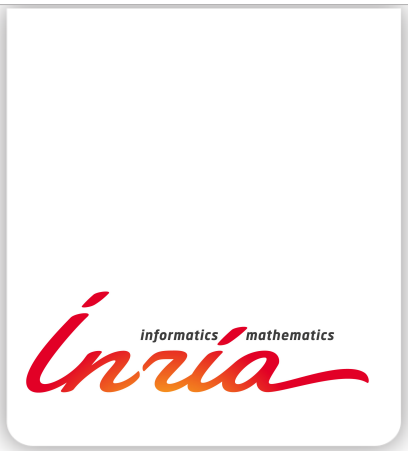


When Amdahl Meets Young/Daly

Aurélien Cavelan, Jiafan Li, Yves Robert, Hongyang Sun

RESEARCH REPORT
N° 8871
February 2016
Project-Team ROMA

ISSN 0249-6399 ISRN INRIA/RR--8871--FR+ENG



When Amdahl Meets Young/Daly

Aurélien Cavelan*†‡, Jiafan Li§, Yves Robert*‡¶,
Hongyang Sun*‡

Project-Team ROMA

Research Report n° 8871 — February 2016 — 21 pages

Abstract: This paper investigates the optimal number of processors to execute a parallel job, whose speedup profile obeys Amdahl's law, on a large-scale platform subject to fail-stop and silent errors. We combine the traditional checkpointing and rollback recovery strategies with verification mechanisms to cope with both error sources. We provide an exact formula to express the execution overhead incurred by a periodic checkpointing pattern of length T and with P processors, and we give first-order approximations for the optimal values T^* and P^* as a function of the individual processor failure rate λ_{ind}. A striking result is that P^* is of the order $\lambda_{\text{ind}}^{-1/4}$ if the checkpointing cost grows linearly with the number of processors, and of the order $\lambda_{\text{ind}}^{-1/3}$ if the checkpointing cost stays bounded for any P. We conduct an extensive set of simulations to support the theoretical study. The results confirm the accuracy of first-order approximation under a wide range of parameter settings.

Key-words: resilience, fail-stop errors, silent errors, optimal checkpointing period, optimal processor allocation, Amdahl's law.

* École Normale Supérieure de Lyon
† INRIA, France
‡ LIP laboratory, CNRS, ENS Lyon, INRIA, University Lyon 1
§ East China Normal University, China
¶ University of Tennessee Knoxville, USA

RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

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

Quand Amdahl rencontre Young/Daly

Résumé : Cet article étudie le nombre optimal de processeurs pour exécuter un travail parallèle dont le profil d'accélération obéit à la loi d'Amdahl, sur une plateforme à grande échelle exposée aux pannes et aux erreurs silencieuses. Nous combinons l'approche traditionnelle de checkpointing/recovery avec des mécanismes de vérification pour faire face aux deux types d'erreurs. Nous fournissons une formule exacte pour mesurer le surcoût du temps d'exécution induit par un motif de checkpoint périodique de longueur T et avec P processeurs, et nous donnons une approximation au premier ordre des valeurs optimales de T^* et P^* en fonction du taux d'erreur individuel d'un processeur λ_{ind}. Un résultat frappant est que P^* est de l'ordre de $\lambda_{\text{ind}}^{-1/4}$ quand le coût de checkpoint croît linéairement avec le nombre de processeurs, et de l'ordre de $\lambda_{\text{ind}}^{-1/3}$ quand le coût de checkpoint reste borné par P. Nous menons une large campagne de simulations pour appuyer l'étude théorique. Les résultats confirmes la précision de l'approximation au premier ordre pour une large gamme de paramètres.

Mots-clés : résilience, erreur fatale, erreur silencieuse, facteur d'accélération, vérification, checkpoint, loi d'Amdahl, schéma de calcul optimal.

1 Introduction

Consider a typical HPC (High Performance Computing) application that will run for days or even weeks on a parallel platform, and whose sequential time is non-negligible. What is the optimal number of processors to execute this application so as to minimize its total execution time? Assume that the application speedup profile obeys *Amdahl's law* [1]: a fraction α of the work is sequential, while the remaining $1-\alpha$ fraction is perfectly parallel. The speedup with P processors is then

$$S(P) = \frac{1}{\alpha + \frac{1-\alpha}{P}} \ . \tag{1}$$

While $S(P)$ is bounded above by $1/\alpha$, it is a strictly increasing function of P, which means that one should enroll as many processors as possible to minimize execution time.

However, this reasoning only holds for error-free execution. With 100,000+ nodes in current petascale platforms, and even more computing resources when entering the exascale era, resilience becomes a challenge [11]. Even if each node provides an individual MTBF (Mean Time Between Failures) of, say, one century, a machine with 100,000 such nodes will encounter a failure every 9 hours on average, which is smaller than the execution time of many HPC applications. Furthermore, a one-century MTBF per node is an optimistic figure, given that each node may be composed of tens or even hundreds of cores. Moreover, several types of errors need to be considered when computing at scale. In addition to the classical fail-stop errors (such as hardware failures), silent errors (or SDC, for Silent Data Corruptions) constitute another threat [29, 39, 27]. This phenomenon is not so well understood, but has been recently identified as one of the major challenges towards exascale [11].

While checkpoint/restart [12, 19, 23] is the de-facto recovery technique for dealing with fail-stop errors, there is no widely adopted general-purpose technique to cope with silent errors. In contrast to a fail-stop error whose detection is immediate, a silent error is identified only when the corrupted data leads to an unusual application behavior. Such a detection latency raises a new challenge: if the error struck before the last checkpoint, and is detected after that checkpoint, then the checkpoint is corrupted and cannot be used for rollback. In order to avoid corrupted checkpoints, an effective approach consists in employing some verification mechanism and combining it with checkpointing. Such a verification mechanism can be general-purpose (e.g., based on replication [21] or even triplication [26]) or application-specific [8, 6, 13, 31].

We address both fail-stop and silent errors by using *verified checkpoints*, which corresponds to performing a verification just before taking each checkpoint. Note that this approach is agnostic of the nature of the verification mechanism. If the verification succeeds, then one can safely store the checkpoint. Otherwise, it means that a silent 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. Of course, if a fail-stop error strikes, we can also safely recover from the last checkpoint, just as in the classical checkpoint and rollback recovery method. We refer to this protocol as the VC protocol, and it basically amounts to replacing the cost C of a checkpoint by the cost $V + C$ of a verification followed by a checkpoint. However, because we deal with two sources of errors, one detected immediately and the other only when we reach the verification, the analysis of the optimal checkpointing strategy is more involved.

This paper shows that on failure-prone platforms, it is no longer true that enrolling more processors will always decrease the (expected) execution time of a parallel application. First, more processors means more failures: if the failure rate of an individual processor is λ_{ind} (and its MTBF is $\mu_{\text{ind}} = 1/\lambda_{\text{ind}}$), then the failure rate for a platform with P processors is $P\lambda_{\text{ind}}$ (and its MTBF is μ_{ind}/P) [23, Proposition 1.2]. Second, the cost of checkpointing may well increase linearly with P [18, 38], because of the synchronization needed among the processors in order to take a coherent snapshot of the global application state. The intuition is that at some point adding more resources will be an overkill, for failures and resilience operations to handle them will become too frequent for the application to make any progress.

These considerations raise the following fundamental question: *What is the optimal number of processors to execute a parallel application on a failure-prone platform?* Surprisingly, this question

has never received a quantitative answer, although some experimental study has been reported [25, 38]. The major contribution of this paper is to answer this question by providing a detailed analysis on the performance of the VC protocol in the presence of both fail-stop and silent errors. In particular, we consider a periodic checkpointing pattern $\textsc{Pattern}(T, P)$, which consists of a work chunk of duration T and executed with P processors, followed by a verification and then by a checkpoint (see Figure 1). We give first-order approximations for the optimal values T^* and P^* as a function of the individual processor failure rate λ_{ind}. A striking result is that, as long as the sequential fraction α of the application is a non-negligible constant, P^* is of the order $\lambda_{\text{ind}}^{-1/4}$ if the checkpointing cost grows linearly with the number of processors, and of the order $\lambda_{\text{ind}}^{-1/3}$ if the checkpointing cost stays bounded for any P. The corresponding values of T^* in these two cases are of the orders $\lambda_{\text{ind}}^{-1/2}$ and $\lambda_{\text{ind}}^{-1/3}$, respectively. The results nicely extend the well-known Young/Daly formula [36, 14] by characterizing the optimal number of resources to enroll. These first-order bounds are well corroborated and validated by our simulation study conducted using real platform parameters.

The main contributions of this paper are the following:

- The derivation of an exact analytical formula for the expected execution time of a pattern in the presence of both fail-stop and silent errors, where fail-stop errors can strike at any time (while silent errors only strike during computations);
- The determination of the optimal pattern length and processor count, up to the first-order term. Given error rates and checkpoint/verification costs, we compute both the optimal pattern length and optimal number of processors to enroll. To the best of our knowledge, this is the first analytical characterization of the optimal degree of parallelism for executing a parallel application whose speedup obeys Amdahl's law;
- An extensive set of simulations with data collected from real platforms. The results confirm the accuracy of the performance model and validity of first-order approximation under a wide range of parameter settings and resilience scenarios.

The rest of the paper is organized as follows. Section 2 briefly discusses the related work. Section 3 introduces the models and notations. Section 4 presents all our analytical results, followed by the presentation of the simulation results in Section 5. Finally, Section 6 provides concluding remarks and hints for future directions.

2 Related Work

Checkpointing. The most commonly deployed strategy to cope with fail-stop errors is checkpointing, in which processes periodically save their states, so that computation can be resumed from that point when some failure disrupts the execution. Checkpointing strategies are numerous, ranging from fully coordinated checkpointing [12] to uncoordinated checkpointing and recovery with message logging [19]. Despite a very broad applicability, these fault-tolerance methods suffer from the intrinsic limitation that both protection and recovery generate an I/O workload, which grows with failure probability and becomes unsustainable at large scale [20, 9] (even with optimizations such as diskless or incremental checkpointing [30]). To reduce the checkpointing overhead, many authors have proposed multi-level checkpointing protocols, which combine global disk checkpointing with local or in-memory checkpointing [34, 33, 27, 4, 15, 5].

The cost of checkpointing clearly depends upon the protocol and storage type, hence we adopt a quite general formula to account for checkpoint overhead in this paper. We let $C_P = a + b/P + cP$ to model the time to save a checkpoint on P processors. Here, $a + b/P$ represents the I/O overhead to write the application's memory footprint M to the storage system. For in-memory checkpointing [37, 16], $a + b/P$ is a communication time with latency a and $b/P = M/(\tau_{net}P)$, where τ_{net} is the network bandwidth (each processor stores M/P data items). For coordinated checkpointing to stable storage, there are two cases: if the storage system's bandwidth is the I/O

bottleneck, $a = \beta + M/\tau_{io}$ and $b = 0$, where β is a start-up time and τ_{io} is the I/O bandwidth; otherwise, if the network is the I/O bottleneck, we retrieve the same formula as for in-memory checkpointing. Finally, cP represents the message passing overhead that grows linearly with the number of processor, in order for all processors to reach a global consistent state [18, 38].

Silent error detection. Considerable efforts have been directed at verification techniques to reveal silent errors. A perfect verification is often only achievable with expensive techniques, such as process replication [21, 28] or redundancy [26, 17]. Application-specific information can be very useful in decreasing the verification cost. Algorithm-based fault tolerance (ABFT) [24, 8, 32] is a well-known technique to detect errors in linear algebra kernels using checksums. Various techniques have been proposed in other application domains. Benson et al. [6] compared a higher-order scheme with a lower-order one to detect errors in the numerical analysis of ODEs. Chen [13] uses orthogonality checks for Krylov-based sparse solvers. Sao and Vuduc [31] investigate self-stabilizing corrections after error detection in the conjugate gradient method. Heroux and Hoemmen [22] design a fault-tolerant GMRES capable of converging despite silent errors. Bronevetsky and de Supinski [10] provide a comparative study of detection costs for iterative methods. Recently, detectors based on data analytics, using use interpolation techniques, such as time series prediction and spatial multivariate interpolation, have also been proposed as verification mechanisms [7, 2, 3]. Altogether, there is a wide range of available detectors, and our approach is agnostic of the nature of verification mechanism used for silent errors.

Resilience and speedup. Several authors have investigated the optimal number of processors to enroll when running a parallel application on a failure-prone platform. Zheng et al. [38] address this problem for fail-stop errors and provide a formula to compute the speedup of an application obeying Amdhal's law and running with P processors. Also for fail-stop errors, Jin et al. [25] use an iterative relaxation procedure to compute the optimal number of resources for a perfectly parallel job. These two important works are the most closely related to ours. In comparison, the main differences with our work are: (i) we account for both fail-stop and silent errors (instead of only fail-stop errors); (ii) we consider several relevant scenarios for checkpointing costs (instead of only linearly growing costs); (iii) we analytically characterize both the optimal number of processors and optimal checkpointing period as a function of the individual processor failure rate, the speedup profile and the checkpoint/verification cost (instead of using numerical procedures); and (iv) our formulas are exact up to first-order term and account for errors in checkpointing. Our first-order approximation formulas (see Theorems 2 and 3 below) are the first quantitative assessments of the best degree of parallelism that should be deployed.

3 Models and Notations

This section presents the analytical models for evaluating the performance of resilience algorithms. Table 1 summarizes the list of main notations used in the paper.

Failure model. We incorporate both hardware faults and silent data corruptions, which are also known as *fail-stop errors* and *silent errors* in the literature. Since the two types of errors are caused by different sources on realistic systems, we assume that they are independent and that both arrivals follow *exponential* distributions based on the standard assumption. Let $\lambda_{\text{ind}} = 1/\mu_{\text{ind}}$ denote the reciprocal of the MTBF μ_{ind} of each individual processor by accounting for both types of errors, and suppose f fraction of the total number of errors are fail-stop and the remaining $s = 1 - f$ fraction are silent. Then, the arrival rates of fail-stop and silent errors when using P processors are given by $\lambda_P^f = f\lambda_{\text{ind}}P$ and $\lambda_P^s = s\lambda_{\text{ind}}P$, respectively [23]. Thus, the probability of encountering at least one fail-stop error during a computation of time T is $q_P^f(T) = 1 - e^{-\lambda_P^f T}$ and that of encountering at least one silent error during the same computation is $q_P^s(T) = 1 - e^{-\lambda_P^s T}$.

Application model. We consider HPC applications that are long-lasting even when executed on a large number of processors. Suppose an application has a total amount of computation (or work) W_{total} and a speedup function $S(P)$ when executed on P processors without considering failures.

Table 1: List of Notations.

Application parameters	
$\textsc{Pattern}(T, P)$	Periodic checkpointing pattern
T	Length (or period) of pattern
P	Number of allocated processors
$S(P)$	Speedup function w/o failure
$H(P) = 1/S(P)$	Execution overhead w/o failure
$\mathbb{E}(\textsc{Pattern})$ or $\mathbb{E}(T, P)$	Expected exec. time of a pattern
$\mathbb{S}(\textsc{Pattern})$ or $\mathbb{S}(T, P)$	Expected speedup of a pattern
$\mathbb{H}(\textsc{Pattern})$ or $\mathbb{H}(T, P)$	Expected exec. overhead of a pattern
Resilience parameters	
$\lambda_{\text{ind}} = 1/\mu_{\text{ind}}$	Error rate of an individual processor
$\lambda_P^f = f\lambda_{\text{ind}}P$	Fail-stop error rate on P processors
$\lambda_P^s = s\lambda_{\text{ind}}P$	Silent error rate on P processors
$C_P = a + b/P + cP$	Checkpointing cost on P processors
$R_P = a + b/P + cP$	Recovery cost on P processors
$V_P = v + u/P$	Verification cost on P processors
D	Downtime after a fail-stop error

In this paper, we consider the speedup function obeying Amdahl's law as given in Equation (1). For convenience, we define $H(P) = \frac{1}{S(P)} = \alpha + \frac{1-\alpha}{P}$ to be the execution overhead of the application, where α denotes the fraction of the application that is inherently sequential and cannot be parallelized. The makespan W_{final} of the application in an error-free execution is therefore given by $W_{\text{final}} = \frac{W_{\text{total}}}{S(P)} = H(P)W_{\text{total}}$.

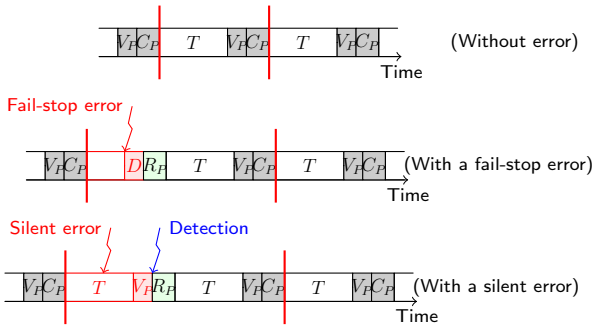


Figure 1: Illustration of a resilience protocol using periodic checkpointing patterns (highlighted in red). The first figure shows the execution of a pattern without any error. The second figure shows that the execution is stopped immediately when a fail-stop error strikes, in which case the pattern is re-executed after a downtime and a recovery. The third figure shows that the execution continues when a silent error strikes, till the error is detected by the verification at the end. The pattern is then re-executed after a recovery.

Resilience model. To enforce resilience, a standard protocol is by checkpointing the status of the application periodically, thus creating periodic checkpointing *patterns* as illustrated in Figure 1. To cope with silent errors, an additional error detection (or verification) mechanism is performed just before taking each checkpoint [35, 5]. If a fail-stop error strikes inside a pattern, the computation is interrupted immediately, while a silent error, if strikes, is only detected at the end of the pattern by the verification. In both cases, we roll back to the beginning of the pattern and recover from the last checkpoint, thus avoiding restarting the application from scratch. Note that a fail-stop error could strike after a silent error within the same pattern but before the verification is reached. In this case, the silent error is masked by the fail-stop error and need not be detected, since a recovery is nevertheless required.

Formally, we characterize a periodic checkpointing pattern, denoted as $\textsc{Pattern}(T, P)$, by the following two parameters.

- T: length (or period) of the pattern, i.e., amount of time to do useful computation before taking each checkpoint;
- P: number of processors allocated to the application.

As discussed in Section 2, we let $C_P = a + b/P + cP$ denote the time to save a checkpoint on P processors. The recovery cost is assumed to be the same as the checkpointing cost, i.e., $R_P = C_P$, because it involves the same I/O operations. To perform a verification, we assume the use of application-specific error detection techniques (as detailed in Section 2). Since a verification is only done in memory, its cost can be modeled as $V_P = v + u/P$. Here, v is a start-up overhead, and u/P is the time needed to verify the application data distributed across P processors. Finally, a constant downtime D is required after each fail-stop error in order to replace or repair a failed processor.

In our analysis, fail-stop errors can strike at any time during the execution of an application, including verifications, checkpointing and recoveries. However, silent errors can only strike the computations, since otherwise they cannot be detected. Hence, we assume that I/O operations and verifications are protected from silent errors (e.g., by using expensive redundancy or replication techniques). Finally, no error of any kind can strike during downtime.

Optimization objective. The objective is to minimize the expected total execution time (or makespan) of an application. Since the application is divided into periodic checkpointing patterns defined by $\textsc{Pattern}(T, P)$, the amount of work done in a pattern is $W_{\text{pattern}} = T \cdot S(P)$. For long-lasting applications, the total number of patterns in the application can be approximated as $\frac{W_{\text{total}}}{W_{\text{pattern}}} = \frac{W_{\text{total}}}{T \cdot S(P)}$. Let $\mathbb{E}(\textsc{Pattern})$ denote the expected execution time of the pattern. The expected makespan $\mathbb{E}(W_{\text{final}})$ of the application is then given by $\mathbb{E}(W_{\text{final}}) \approx \mathbb{E}(\textsc{Pattern}) \frac{W_{\text{total}}}{T \cdot S(P)}$. Now, define $\mathbb{S}(\textsc{Pattern}) = \frac{T \cdot S(P)}{\mathbb{E}(\textsc{Pattern})}$ to be the expected speedup of the pattern, and define $\mathbb{H}(\textsc{Pattern}) = \frac{1}{\mathbb{S}(\textsc{Pattern})} = \frac{\mathbb{E}(\textsc{Pattern})}{T} H(P)$ to be the expected execution overhead of the pattern. The expected makespan of the application can therefore be written as $\mathbb{E}(W_{\text{final}}) \approx \frac{W_{\text{total}}}{\mathbb{S}(\textsc{Pattern})} = \mathbb{H}(\textsc{Pattern}) W_{\text{total}}$. We observe that the optimal expected makespan is obtained by maximizing the expected speedup or minimizing the expected execution overhead of a periodic checkpointing pattern. In the next section, we will focus on such a pattern $\textsc{Pattern}(T, P)$, and find its optimal length T and processor count P.

4 Optimal Periodic Checkpointing Pattern

In this section, we analytically determine the optimal periodic checkpointing pattern using first-order approximation, and derive explicit formulas for the checkpointing period T and processor allocation P. We validate the first-order solution in Section 5 by showing its close proximity to the optimal numerical solution.

4.1 Expected Execution Time of a Pattern

We start by computing the expected execution time of a pattern when the parameters T and P are given.

Proposition 1. *The expected execution time of a given pattern* $\textsc{Pattern}(T, P)$ *is*

$$\mathbb{E}(\textsc{Pattern}) = \left(\frac{1}{\lambda_P^f} + D\right)\left(e^{\lambda_P^f C_P}\left(1 - e^{\lambda_P^s T}\right) + e^{\lambda_P^f R_P}\left(e^{\lambda_P^f (C_P + T + V_P) + \lambda_P^s T} - 1\right)\right). \tag{2}$$

Proof. To successfully execute a pattern PATTERN(T, P), we need to complete the pattern length T, the verification V_P and the checkpoint C_P. Hence, according to the linearity of expectation, we have

$$\begin{aligned}\mathbb{E}(\text{PATTERN}) &= \mathbb{E}(T + V_P + C_P) \\ &= \mathbb{E}(T + V_P) + \mathbb{E}(C_P) \ , \end{aligned} \tag{3}$$

We first compute $\mathbb{E}(C_P)$, the expected time to successfully store a checkpoint subject to fail-stop errors. During checkpointing, there is a probability $q_P^f(C_P)$ that a fail-stop error strikes. If that happens, we need to perform a recovery from the last checkpoint after a downtime, and then re-execute both T and V_P before re-executing C_P again. If there is no error, we just need to pay the checkpointing cost C_P. Therefore, we can express $\mathbb{E}(C_P)$ as

$$\begin{aligned}\mathbb{E}(C_P) = q_P^f(C_P) \left(\mathbb{E}^{\text{lost}}(C_P) + D + \mathbb{E}(R_P) + \mathbb{E}(T + V_P)\right) \\ + \left(1 - q_P^f(C_P)\right) C_P \ , \end{aligned} \tag{4}$$

where $\mathbb{E}(R_P)$ denotes the expected time to perform a recovery, and $\mathbb{E}^{\text{lost}}(C_P)$ denotes the expected time lost executing C_P if a fail-stop error strikes. More generally, we can define $\mathbb{E}^{\text{lost}}(W)$ to be the expected time lost for any execution of length W, and it can be computed as follows:

$$\mathbb{E}^{\text{lost}}(W) = \int_0^\infty t\mathbb{P}(X = t | X < W)dt = \frac{\int_0^W t\lambda_P^f e^{-\lambda_P^f t} dt}{\mathbb{P}(X < W)} \ ,$$

where $\mathbb{P}(X = t)$ denotes the probability that a fail-stop error strikes exactly at time t. By definition, we have $\mathbb{P}(X < W) = q_P^f(W) = 1 - e^{-\lambda_P^f W}$. Integrating by parts, we get

$$\mathbb{E}^{\text{lost}}(W) = \frac{1}{\lambda_P^f} - \frac{W}{e^{\lambda_P^f W} - 1} \ . \tag{5}$$

Now, substituting $q_P^f(C_P)$ and $\mathbb{E}^{\text{lost}}(C_P)$ into Equation (4), we can get

$$\mathbb{E}(C_P) = \left(e^{\lambda_P^f C_P} - 1\right) \left(\frac{1}{\lambda_P^f} + D + \mathbb{E}(R_P) + \mathbb{E}(T + V_P)\right) \ .$$

Now, we compute $\mathbb{E}(R_P)$, the expected time to successfully perform a recovery subject to fail-stop errors. Unlike checkpointing, a recovery is always done at the beginning of a pattern. With probability $q_P^f(R_P)$, it fails due to a fail-stop error and we have to try again after a downtime. Otherwise, we just need to pay the recovery cost R_P. Therefore, we have

$$\begin{aligned}\mathbb{E}(R_P) = q_P^f(R_P) \left(\mathbb{E}^{\text{lost}}(R_P) + D + \mathbb{E}(R_P)\right) \\ + \left(1 - q_P^f(R_P)\right) R_P \ , \end{aligned}$$

which leads to

$$\mathbb{E}(R_P) = \left(\frac{1}{\lambda_P^f} + D\right) \left(e^{\lambda_P^f R_P} - 1\right) \ .$$

In order to compute $\mathbb{E}(\text{PATTERN})$, and according to Equation (3), we need to compute $\mathbb{E}(T + V_P)$. Recall that fail-stop errors can strike at any time during the execution, while silent errors only strike the computations. When a fail-stop error strikes, which happens with probability $q_P^f(T+V_P)$, we do not need to account for silent errors, since the application is stopped immediately and we need to re-execute $T + V_P$ anyway, following a downtime and a recovery. Otherwise, with probability $1 - q_P^f(T + V_P)$, there is no fail-stop error, and only in this case, we check for silent errors. With probability $q_P^s(T)$, a silent error strikes (and is detected by the verification), and we

need to perform a recovery and re-execute $T+V_P$. Otherwise, the execution is complete. Overall, we have

$$\begin{aligned}\mathbb{E}(T+V_P) = q_P^f(T+V_P)\left(\mathbb{E}^{\text{lost}}(T+V_P) + D + \mathbb{E}(R_P) + \mathbb{E}(T+V_P)\right) \\ + \Big(1-q_P^f(T+V_P)\Big)\Big(T+V_P+q_P^s(T)\big(\mathbb{E}(R_P)+\mathbb{E}(T+V_P)\big)\Big).\end{aligned}$$

Plugging $q_P^f(T+V_P)$, $q_P^s(T)$ and $\mathbb{E}^{\text{lost}}(T+V_P)$ into the above equation, and solving for $\mathbb{E}(T+V_P)$, we can get

$$\begin{aligned}\mathbb{E}(T+V_P) = e^{\lambda_P^s T}\big(e^{\lambda_P^f(T+V_P)}-1\big)\left(\frac{1}{\lambda_P^f}+D\right) \\ + e^{\lambda_P^s(T+V_P)}\big(T+V_P\big) \\ + \big(e^{\lambda_P^f(T+V_P)+\lambda_P^s(T)}-1\big)\mathbb{E}(R_P)\ .\end{aligned}$$

Finally, plugging $\mathbb{E}(T+V_P)$, $\mathbb{E}(C_P)$ and $\mathbb{E}(R_P)$ back into Equation (3), we find that

$$\begin{aligned}\mathbb{E}(\textsc{Pattern}) = \left(\frac{1}{\lambda_P^f}+D\right)\Bigg(e^{\lambda_P^f C_P+\lambda_P^s T}\big(e^{\lambda_P^f(T+V_P)}-1\big) \\ + e^{\lambda_P^f C_P}-1+\big(e^{\lambda_P^f R_P}-1\big)\Big(e^{\lambda_P^f(T+V_P+C_P)+\lambda_P^s(T)}-1\Big)\Bigg),\end{aligned}$$

which simplifies to the expression shown in Equation (2). □

To find the optimal pattern, one needs to search for values of T and P that minimize the expected execution overhead $\mathbb{H}(\textsc{Pattern})$ of a pattern based on the expected execution time $\mathbb{E}(\textsc{Pattern})$ computed above. However, due to the complex expression given by Equation (2), an analytical solution is difficult to find, and one has to rely on numerical methods to approximate the optimal solution. In the following, we will use first-order approximation to derive explicit formulas for the optimal checkpointing period and processor allocation. The simulation results in Section 5 show that first-order approximation offers very close estimates to the optimal solution.

4.2 Limitation of First-Order Approximation

Before deriving the optimal pattern parameters, we first investigate the limitation of first-order approximation by bounding the maximum orders of T and P that can be approximated by the approach. Suppose P and T satisfy

$$P = \Theta(\lambda_{\text{ind}}^{-x}) \text{ and } T = \Theta(\lambda_{\text{ind}}^{-y})\ ,$$

where $x, y > 0$. Since $\lambda_P C_P = \lambda_{\text{ind}} P(a+b/P+cP)$ and $\lambda_P V_P = \lambda_{\text{ind}} P(v+u/P)$, let $\lambda_P(C_P+V_P) = \lambda_{\text{ind}} P(d+h/P+cP)$ with $d = a+v$ and $h = b+u$. Hence, we have $\lambda_P(C_P+V_P) = \Theta\left(\lambda_{\text{ind}}^{\epsilon}\right)$, where

$$\epsilon = \begin{cases} 1-2x & \text{if } c \neq 0 \\ 1-x & \text{if } c = 0 \text{ and } d \neq 0 \\ 1 & \text{if } c = 0 \text{ and } d = 0 \end{cases},$$

and $\lambda_P T = \Theta(\lambda_{\text{ind}}^{1-x-y})$. Therefore, in order to accurately estimate $e^{\lambda_P C_P}$, $e^{\lambda_P V_P}$ and $e^{\lambda_P T}$ using Taylor series expansion, we need $\epsilon > 0$ and $1-x-y > 0$, which translates to

$$x < \delta, \text{ where } \delta = \begin{cases} 1/2 & \text{if } c \neq 0 \\ 1 & \text{if } c = 0 \end{cases}, \tag{6}$$

$$y < 1-x\ . \tag{7}$$

Inequalities (6) and (7) specify, respectively, the maximum order on the number of processors and, for a fixed processor count, the maximum order on the checkpointing period. The first-order results obtained within these bounds offer valid approximation to the optimal solution as long as the MTBF $\mu_{\text{ind}} = 1/\lambda_{\text{ind}}$ of an individual processor is sufficiently large (e.g., in the order of years, which is true for modern processors). Beyond these bounds, unfortunately, the first-order analysis will no longer be applicable.

4.3 Optimal Checkpointing Period for Fixed Processor Count

In this section, we derive the optimal checkpointing period when the application is run with a fixed number of processors. The result extends the classical formula given by Young [36] and Daly [14] for fail-stop errors only.

Theorem 1. *Given a processor allocation $P = \Theta(\lambda_{\text{ind}}^{-x})$ with $x < \delta$ as shown in Inequality (6), the optimal checkpointing period of a pattern is*

$$T_P^* = \sqrt{\frac{V_P + C_P}{\frac{\lambda_P^f}{2} + \lambda_P^s}} \ . \tag{8}$$

The expected execution overhead (ignoring lower-order terms) in this case is given by

$$\mathbb{H}\left(T_P^*, P\right) = H(P)\left(1 + 2\sqrt{\left(\frac{\lambda_P^f}{2} + \lambda_P^s\right)\left(V_P + C_P\right)}\right) . \tag{9}$$

Proof. For a fixed $P = \Theta(\lambda_{\text{ind}}^{-x})$ with $x < \delta$, we can consider C_P, R_P, V_P as constants, which are smaller in magnitude compared to the platform MTBFs $1/\lambda_P^f$ and $1/\lambda_P^s$. Applying Taylor series to expand $e^z = 1 + z + \frac{z^2}{2}$ up to the second-order term, we rewrite the expected execution time $\mathbb{E}(\textsc{Pattern})$ of a pattern (Equation (2)) as follows (ignoring lower-order terms):

$$\begin{aligned}
\mathbb{E}(\textsc{Pattern}) = T + V_P + C_P &+ \left(\frac{\lambda_P^f}{2} + \lambda_P^s\right) T^2 \\
&+ \lambda_P^f T\left(V_P + C_P + R_P + D\right) + \lambda_P^s T\left(V_P + R_P\right) \\
&+ \lambda_P^f C_P \left(\frac{C_P}{2} + R_P + V_P + D\right) \\
&+ \lambda_P^f V_P \left(V_P + R_P + D\right) \ .
\end{aligned}$$

The expected execution overhead of the pattern can then be computed as

$$\mathbb{H}(T, P) = H(P)\left(\frac{\left(V_P + C_P\right)\left(1 + O\left(\lambda_{\text{ind}}^{\epsilon'}\right)\right)}{T} + \left(\frac{\lambda_P^f}{2} + \lambda_P^s\right) T + 1 + O\left(\lambda_{\text{ind}}^{\epsilon'}\right)\right) ,$$

where $\epsilon' = 1 - 2x$ if $c \neq 0$ and $\epsilon' = 1 - x$ otherwise. Since $P = \Theta(\lambda_{\text{ind}}^{-x})$ is fixed and $x < \delta$, we have $\epsilon' > 0$ and hence the term $O\left(\lambda_{\text{ind}}^{\epsilon'}\right)$ becomes negligible (in front of 1) when λ_{ind} is sufficiently small (e.g., tends to 0). Given a processor allocation P, the optimal expected overhead is achieved by setting

$$\frac{\partial \mathbb{H}(T, P)}{\partial T} = H(P)\left(-\frac{V_P + C_P}{T^2} + \frac{\lambda_P^f}{2} + \lambda_P^s\right) = 0 \ ,$$

which gives rise to the optimal checkpointing period T_P^* as shown in Equation (8). Now, substituting T_P^* back into $\mathbb{H}(T, P)$, we obtain the expected execution overhead as shown in Equation (9). □

Theorem 1 shows that, for a given processor count $P = \Theta(\lambda_{\text{ind}}^{-x})$ with $x < \delta$ as specified by Inequality (6), the optimal checkpointing period satisfies $T_P^* = O(\lambda_{\text{ind}}^{-y})$, where

$$y = \begin{cases} 1/2 & \text{if } c \neq 0 \\ (1-x)/2 & \text{if } c = 0 \text{ and } d \neq 0 \\ 1/2 - x & \text{if } c = 0 \text{ and } d = 0 \end{cases}.$$

In all cases, we get $y < 1 - x$ as specified by Inequality (7), thus validating the accuracy of the first-order approximation.

Note that, in the case of $c = 0$ and $d = 0$, we also need $x < 1/2$ in order to have $y > 0$. This additional constraint on the order of P is required to derive the first-order approximation for the optimal checkpointing period as given in Equation (8).

4.4 Optimal Processor Allocation and Pattern Parameters

We now optimize the number of allocated processors to an application. We discuss different cases based on the characteristic of the error-free overhead $H(P)$, as well as on the scalability of checkpointing and verification costs, which have the general form $C_P = a + \frac{b}{P} + cP$ and $V_P = v + \frac{u}{P}$. In the following analysis, we assume that all the parameters a, b, c, v, u and the sequential fraction α are constants and independent of the error rate λ_{ind}.

4.4.1 $H(P) = \alpha + \frac{1-\alpha}{P}$ and $C_P = cP + o(P)$, $\alpha, c \neq 0$

This case corresponds to the application having a constant sequential fraction and a checkpointing cost that grows linearly with the number of processors (the verification cost has no impact in this scenario).

Theorem 2. *Suppose the application has a constant sequential fraction $\alpha > 0$, and a checkpointing cost $C_P = cP + o(P)$. The optimal number of processors and the corresponding optimal checkpointing period of a pattern are*

$$P^* = \left(\frac{1}{c\left(\frac{f}{2} + s\right)\lambda_{\text{ind}}} \right)^{1/4} \left(\frac{1-\alpha}{2\alpha} \right)^{1/2}, \tag{10}$$

$$T^* = \left(\frac{c}{\left(\frac{f}{2} + s\right)\lambda_{\text{ind}}} \right)^{1/2}. \tag{11}$$

The expected execution overhead (ignoring lower-order terms) in this case is

$$\mathbb{H}(T^*, P^*) = \alpha + 2\left(4\alpha^2(1-\alpha)^2 c \left(\frac{f}{2} + s\right)\lambda_{\text{ind}} \right)^{1/4}. \tag{12}$$

Proof. Substituting $H(P) = \alpha + \frac{1-\alpha}{P}$ into Equation (9) and applying $C_P + V_P = cP + o(P)$, we can get the expected execution overhead as follows:

$$\mathbb{H}\left(T_P^*, P\right) = \alpha + 2\alpha P \sqrt{c\left(\frac{f}{2} + s\right)\lambda_{\text{ind}}} + \frac{1-\alpha}{P} + o\left(\lambda_{\text{ind}}^{1/2} P\right).$$

The above overhead is minimized when setting

$$\frac{\partial \mathbb{H}\left(T_P^*, P\right)}{\partial P} = 2\alpha \sqrt{c\left(\frac{f}{2} + s\right)\lambda_{\text{ind}}} - \frac{1-\alpha}{P^2} + o\left(\lambda_{\text{ind}}^{1/2}\right) = 0.$$

Keeping only the dominating term, the equation above leads to the optimal processor allocation P^* as shown in Equation (10). Now, substituting P^* back into T_P^* and $\mathbb{H}\left(T_P^*, P\right)$ and simplifying, we obtain the optimal checkpointing period T^* and optimal expected execution overhead $\mathbb{H}(T^*, P^*)$ as shown in Equations (11) and (12), respectively. □

4.4.2 $H(P) = \alpha + \frac{1-\alpha}{P}$ and $C_P + V_P = d + o(1)$, $\alpha, d \neq 0$

This case corresponds to the application having a constant sequential fraction, and a constant checkpointing (and verification) cost.

Theorem 3. *Suppose the application has a constant sequential fraction $\alpha > 0$, and a checkpointing and verification cost $C_P + V_P = d + o(1)$. The optimal number of processors and the corresponding optimal checkpointing period of a pattern are*

$$P^* = \left(\frac{1}{d\left(\frac{f}{2}+s\right)\lambda_{\text{ind}}}\right)^{1/3}\left(\frac{1-\alpha}{\alpha}\right)^{2/3}, \tag{13}$$

$$T^* = \left(\frac{d^2}{\left(\frac{f}{2}+s\right)\lambda_{\text{ind}}}\right)^{1/3}\left(\frac{\alpha}{1-\alpha}\right)^{1/3}. \tag{14}$$

The expected execution overhead (ignoring lower-order terms) in this case is

$$\mathbb{H}(T^*, P^*) = \alpha + 3\left(\alpha^2(1-\alpha)d\left(\frac{f}{2}+s\right)\lambda_{\text{ind}}\right)^{1/3}. \tag{15}$$

Proof. When $H(P) = \alpha + \frac{1-\alpha}{P}$ and $C_P + V_P = d + o(1)$, we can get from Equation (9) the expected execution overhead as follows:

$$\mathbb{H}\left(T_P^*, P\right) = \alpha + 2\alpha\sqrt{d\left(\frac{f}{2}+s\right)\lambda_{\text{ind}}P} + \frac{1-\alpha}{P} + o\left(\lambda_{\text{ind}}^{1/2}P^{1/2}\right).$$

Again, the overhead is minimized by setting $\frac{\partial \mathbb{H}(T_P^*, P)}{\partial P} = 0$, which gives

$$\alpha\sqrt{d\left(\frac{f}{2}+s\right)\frac{\lambda_{\text{ind}}}{P}} - \frac{1-\alpha}{P^2} + o\left(\lambda_{\text{ind}}^{1/2}P^{-1/2}\right) = 0.$$

Solving the equation above while focusing on the dominating term gives the optimal processor allocation P^* as shown in Equation (13). Substituting P^* back into T_P^* and $\mathbb{H}\left(T_P^*, P\right)$, we get the optimal T^* and $\mathbb{H}(T^*, P^*)$ as shown in Equations (14) and (15). □

4.4.3 $H(P) = \alpha + \frac{1-\alpha}{P}$ and $C_P + V_P = \frac{h}{P}$, $\alpha, h \neq 0$

This case corresponds to the application having a constant sequential fraction, and a checkpointing (and verification) cost that decreases linearly with the number of processors.

Recall in this case that the number of processors satisfies $P = \Theta(\lambda_{\text{ind}}^{-x})$ with $x < 1/2$ for the first-order approximation to be valid. Subject to this bound, the expected execution overhead as shown in Equation (9) becomes

$$\mathbb{H}(T_P^*, P) = \left(\alpha + \frac{1-\alpha}{P}\right)\left(1 + 2\sqrt{h\left(\frac{f}{2}+s\right)\lambda_{\text{ind}}}\right),$$

which decreases monotonically as the number of allocated processors P increases up to the order of $\lambda_{\text{ind}}^{-1/2}$. Asymptotically, the overhead satisfies $\mathbb{H}(T_P^*, P) = \alpha + \Theta\left(\lambda_{\text{ind}}^x\right)$ for $x < 1/2$.

Hence, it is better to enroll as many processors as possible in this case, as long as P is within the approximation bound of $O(\lambda_{\text{ind}}^{-1/2})$. Intuitively, the costs of both checkpointing and verification reduce with the processor count, which enables to place both resilience operators more frequently with smaller checkpointing period to compensate for the increased error rate. Numerical simulations conducted in Section 5 show that the optimal number of processors P^* is nevertheless bounded in this case with a value beyond $O(\lambda_{\text{ind}}^{-1/2})$.

4.4.4 $H(P) = \frac{1}{P}$

In this case, the application has a perfectly linear speedup function. Again, the expected execution overhead decreases monotonically with the number of allocated processors, and the following gives the expression for $\mathbb{H}(T_P^*, P)$ under different cases (with lower-order terms ignored):

$$\mathbb{H}(T_P^*, P) = \begin{cases} \frac{1}{P} + 2\sqrt{c\left(\frac{f}{2} + s\right)\lambda_{\text{ind}}} & \text{if } c \neq 0 \\ \frac{1}{P} + 2\sqrt{d\left(\frac{f}{2} + s\right)\frac{\lambda_{\text{ind}}}{P}} & \text{if } c = 0,\ d \neq 0 \\ \frac{1}{P}\left(1 + 2\sqrt{h\left(\frac{f}{2} + s\right)\lambda_{\text{ind}}}\right) & \text{if } c = 0,\ d = 0 \end{cases}.$$

In all the cases above, the overhead is asymptotically bounded by $\Theta\left(\lambda_{\text{ind}}^{x}\right)$ for $x < 1/2$, except in the second case (i.e., $c = 0$, $d \neq 0$) where $x < 1$. Numerical simulations conducted in Section 5 show that the optimal processor count P^* happens around $x = 1/2$ and $x = 1$ for case 1 and case 2, respectively, whereas it is unbounded for the last case, due to the combination of diminishing resilience cost and perfect application speedup.

4.5 Discussions

Consider a (standard) application that is not perfectly parallel (i.e., $\alpha \neq 0$). Theorems 2 and 3 show the impact of the checkpointing cost on the optimal degree of parallelism. When this cost grows linearly with P (e.g., with coordinated checkpointing on stable storage [12]), Theorem 2 states that the optimal number of processors is $P^* = \Theta(\lambda_{\text{ind}}^{-1/4})$. In that case, the optimal period has length $T^* = \Theta(\lambda_{\text{ind}}^{-1/2})$. But when this cost remains bounded (e.g., with in-memory checkpointing [37]), then Theorem 3 shows that the optimal solution has both increased parallelism $P^* = \Theta(\lambda_{\text{ind}}^{-1/3})$ and smaller period $T^* = \Theta(\lambda_{\text{ind}}^{-1/3})$. These two cases represent most practical checkpointing protocols implemented in today's fault-tolerant systems. To the best of our knowledge, the results are the first to analytically establish the relationship between P^* and T^* as a function of the resource MTBF $\mu_{\text{ind}} = 1/\lambda_{\text{ind}}$.

Finally, we point out that when both checkpointing and verification costs reduce with P (which is rarely the case in practice), first-order approximation has its limitation and can no longer be used to derive the optimal number of processors and optimal checkpointing period. In this case, one can resort to higher-order approximations or numerical methods to compute the optimal pattern parameters, which are still bounded due to the sequential fraction.

5 Experiments

In this section, we conduct simulations to support the analytical study and to demonstrate the accuracy of first-order approximation under different parameter settings and resilience scenarios. The simulation code is publicly available for download at `http://perso.ens-lyon.fr/aurelien.cavelan/simu.zip`.

5.1 Simulation Settings

We consider four real platforms that were used to evaluate the Scalable Checkpoint/Restart (SCR) library [27]. Measurements of the platform parameters are provided, including error rates of different sources and various checkpointing costs on a specified number of processors (where each processor has a dual quad-core chip). Following [5], the verification cost is set to be the same as that of an in-memory checkpoint, assuming the entire memory footprint needs to be inspected in order to accurately detect silent errors. Table 2 presents the main parameters of the four platforms. The downtime is set as one hour, i.e., $D = 3600s$ (a repair-based restoration value, see the discussion in Section 5.2.5), and the sequential fraction of the application is set to be $\alpha = 0.1$. These values as well as the individual error rate λ_{ind} will be varied in the simulations to assess their impacts on the performance of the optimal pattern.

We envision six resilience scenarios, as shown in Table 3, depending on the scalability of the checkpointing and verification overheads discussed in Section 3. Altogether, these scenarios cover a wide range of resilience protocols, represented by different checkpointing mechanisms and error detection algorithms. For each scenario, we can compute the resilience parameters (i.e., a, b, c, v, u) based on the values of C_P and V_P as well as the number of processors given in Table 2, and then project the corresponding overheads on any number of processors. The optimal pattern under each scenario can be derived using the first-order analysis presented in Section 4. Specifically, for a constant $\alpha > 0$, scenarios 1 and 2 correspond to case 1 $\big(C_P = cP + o(P)\big)$, scenarios 3, 4 and 5 correspond to case 2 $\big(C_P + V_P = d + o(1)\big)$, and scenario 6 corresponds to case 3 $\big(C_P + V_P = h/P\big)$. To assess the accuracy of the first-order approximation, we also compare the performance of the first-order solution with that of the optimal solution obtained using numerical methods such as the one considered in [25].

Once the pattern parameters are determined, fail-stop and silent errors are injected into the simulator as two independent Poisson processes according to the error rates shown in Table 2. The result of each experiment is obtained by averaging over 500 simulation runs, each of which lasts at least 500 patterns. The expected execution overhead of the pattern is computed as the average ratio of the application's execution time with faults and its fault-free execution time.

5.2 Simulation Results

5.2.1 Performance of optimal patterns in different scenarios

Figure 2 shows the performance of the optimal patterns in different resilience scenarios when the sequential fraction of the application is fixed at $\alpha = 0.1$. We can see that, on all the four platforms, the first-order solution provides a very good approximation to the optimal solution in terms of both checkpointing period and processor allocation, under the first four scenarios (the most realistic ones in practical systems). The execution overheads (≈ 0.11) predicted by the first-order formulas (Theorems 2 and 3) are almost identical to the optimal overheads and the ones obtained by simulations. The results confirm the validity of first-order approximations in these scenarios.

In scenario 5, the resilience cost is dominated by the verification overhead, which although is a constant has a relatively small value. This significantly increases the optimal processor count and

Table 2: Platform parameters.

Platform	Hera	Atlas	Coastal	Coastal SSD
λ_{ind}	1.69e-8	1.62e-8	2.34e-9	2.34e-9
f	0.2188	0.0625	0.1667	0.1667
s	0.7812	0.9375	0.8333	0.8333
P	512	1024	2048	2048
C_P	$300s$	$439s$	$1051s$	$2500s$
V_P	$15.4s$	$9.1s$	$4.5s$	$180s$

Table 3: Different resilience scenarios.

Scenario	1	2	3	4	5	6
C_P, R_P	cP	cP	a	a	b/P	b/P
V_P	v	u/P	v	u/P	v	u/P

hence the corresponding error rates, thus compromising the accuracy of the first-order approximation (up to 5% in execution overhead), since the lower-order terms start to become non-negligible. In fact, due to the small constant overhead, scenario 5 closely resembles scenario 6, in which case first-order analysis can no longer predict the optimal pattern parameters within the approximation limit (thus only the results of numerical methods are shown). Figure 2 shows that the optimal pattern parameters in scenario 6 are indeed in the same orders as those of scenario 5 but with higher processor counts and smaller checkpointing periods.

5.2.2 Impact of processor allocation

We study how the number of allocated processors impacts the optimal checkpointing period and the resulting execution overhead under different resilience scenarios. Figure 3 shows the simulation results for the Hera platform (the results are similar for the other platforms). Since the resilience overhead is dominated by the checkpointing cost, the pattern behaviors are mainly influenced by the form of C_P, as demonstrated by the almost overlapping curves between the scenarios that share the same C_P values. In all scenarios, the checkpointing period decreases with the number of processors (Figure 3(a)), which is needed to compensate for the increased error rates. The execution overhead, on the other hand, first improves with the number of processors due to increased parallelism and then degrades due to more errors striking (Figure 3(b)). The optimal processor counts, as we have seen in Figure 2, tend to be higher for scenarios in which the checkpointing cost C_P does not increase (or even decreases) with P. Figure 3(c) shows the difference in execution overhead between the first-order solution and the optimal numerical solution. The difference, for the concerned range of processors, is always within 0.2%, validating once again the accuracy of first-order approximation.

5.2.3 Impact of sequential fraction α

Figure 4 shows the impact of the sequential fraction α on the performance of the optimal patterns in scenarios 1, 3 and 5 (from now on, scenarios 2, 4 and 6 are ignored because they have similar performance as scenarios 1, 3 and 5, respectively). We can see that, as α decreases, more processors are enrolled so that the application can benefit from Amdahl's law to achieve lower execution overheads (or equivalently higher speedups). The checkpointing periods, on the other hand, decrease with α due to increased processor count, except in scenario 1 where the optimal period barely changes with the number of processors (see Theorem 1 and Figure 3(a)). As more processors are used, the first-order approximation of P^* starts to deviate from the optimal value, but the first-order overhead H^*, as shown in Figure 4(c), remains in close proximity to the optimal overhead up to $\alpha = 0.0001$. Also, compared to the other scenarios, scenario 5 starts to show a better overhead as α becomes smaller, because of its smaller checkpointing cost. However, even when $\alpha = 0$, the optimal processor allocation is upper bounded by 10^6 in all three scenarios with an overhead strictly above 10^{-5}. This is in clear contrast to the error-free execution, where an infinity number of processors can be used to achieve (nearly) null overhead.

5.2.4 Impact of error rate λ_{ind}

This experiment assesses the impact of the individual error rate λ_{ind} on the performance of the optimal pattern, in particular on the asymptotic behaviors of the processor allocations and checkpointing periods under different scenarios. Figure 5 shows that, as the processors become more reliable (i.e., as λ_{ind} decreases), the optimal pattern is able to both accommodate more processors

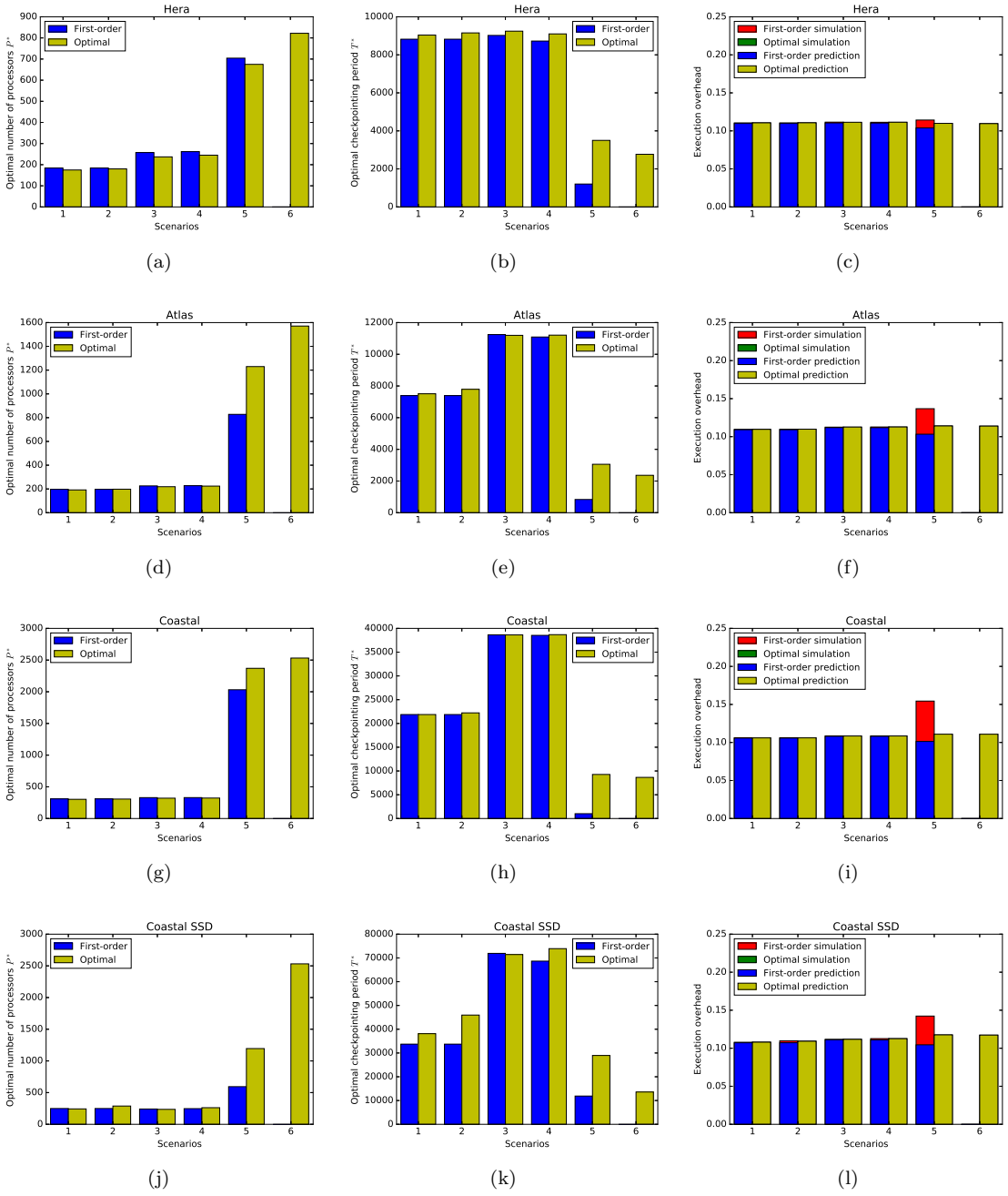
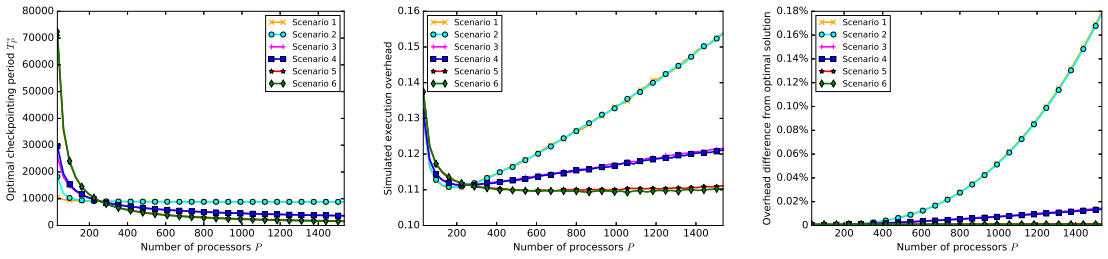
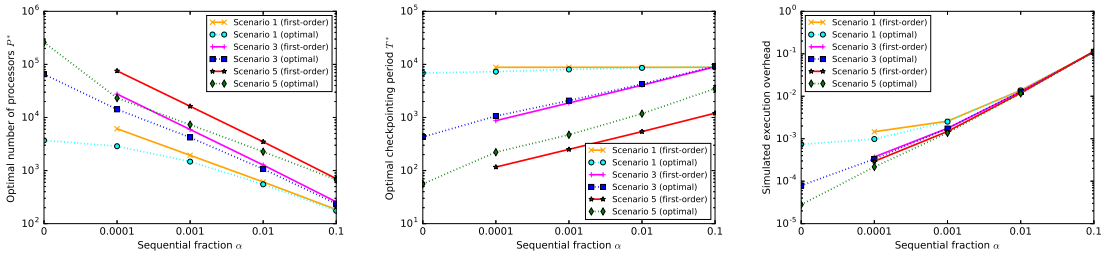


Figure 2: Performance of the optimal pattern in different resilience scenarios on four platforms when the sequential fraction is fixed at $\alpha = 0.1$.



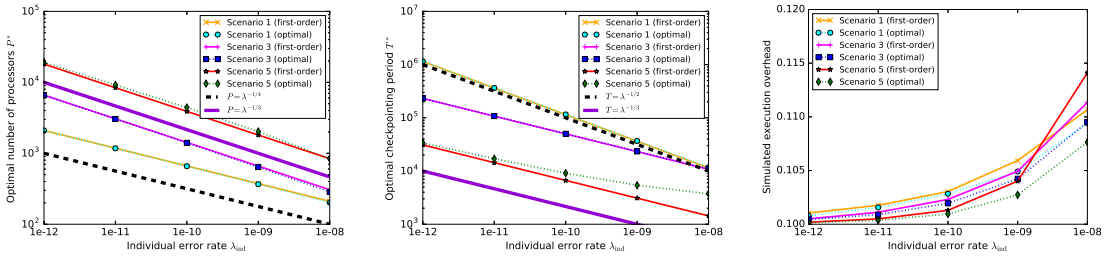
(a) (b) (c)

Figure 3: Optimal checkpointing period T_P^* (from Theorem 1) and simulated execution overhead for different number of processors P on platform Hera.



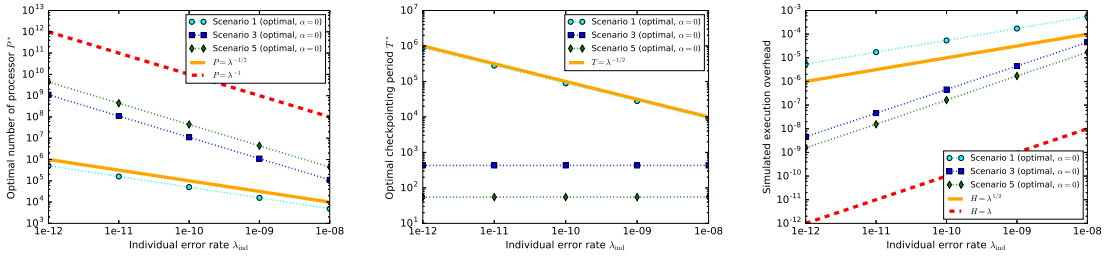
(a) (b) (c)

Figure 4: Optimal checkpointing period T^* and number of processors P^* (from Theorems 2 and 3, and from numerical solution), as well as the simulated execution overhead under different sequential fraction α on platform Hera.



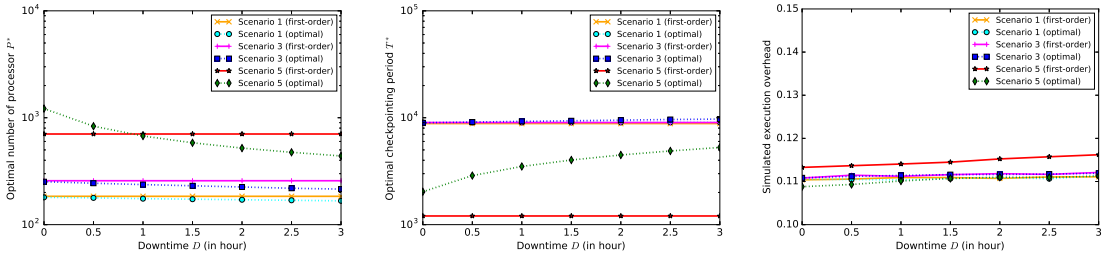
(a) (b) (c)

Figure 5: Optimal checkpointing period T^* and number of processors P^* (from Theorems 2 and 3, and from numerical solution), as well as the simulated execution overhead under different values of λ_{ind} and when $\alpha = 0.1$ on platform Hera.



(a) (b) (c)

Figure 6: Optimal checkpointing period T^* and number of processors P^* (from numerical solution), as well as the simulated execution overhead under different values of λ_{ind} and when $\alpha = 0$ on platform Hera.



(a) (b) (c)

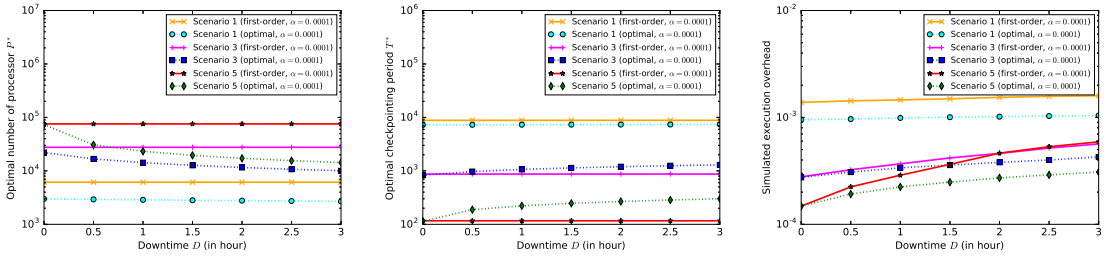
Figure 7: Optimal checkpointing period T^* and number of processors P^* (from Theorems 2 and 3, and from numerical solution), as well as the simulated execution overhead under different downtime D and when $\alpha = 0.1$ on platform Hera.

and use larger checkpointing periods. The results confirm our analytical study that P^* and T^* are in the orders of $\lambda_{\text{ind}}^{-1/4}$ and $\lambda_{\text{ind}}^{-1/2}$, respectively, under scenario 1, and are both in the order of $\lambda_{\text{ind}}^{-1/3}$ under scenarios 3 and 5. Moreover, the first-order approximation becomes more accurate with decreased λ_{ind}, and the execution overheads tend to the theoretical lower bound of 0.1 for all three scenarios.

Figure 6 further shows the behaviors of the optimal patterns when the application is perfectly parallel (i.e., $\alpha = 0$). Recall that this case does not admit a solution under first-order approximation. Numerical analysis suggests that, under scenario 1, the optimal solution satisfies $P^* \approx \Theta(\lambda_{\text{ind}}^{-1/2})$, $T^* \approx \Theta(\lambda_{\text{ind}}^{-1/2})$, and $H^* \approx \Theta(\lambda_{\text{ind}}^{1/2})$, and under scenarios 3 and 5, we have $P^* \approx \Theta(\lambda_{\text{ind}}^{-1})$, $T^* \approx O(1)$, and $H^* \approx \Theta(\lambda_{\text{ind}})$.

5.2.5 Impact of downtime D

Finally, we evaluate the impact of downtime D on the pattern performance. Depending on if repair-based or replacement-based (migration to a spare processor) restoration is used, downtime can range from a few minutes to several hours [25]. In this experiment, we vary the downtime from 0 to 3 hours. Figures 7 and 8 show the simulation results when $\alpha = 0.1$ and $\alpha = 0.0001$, respectively. Since D does not appear in the formulas of P^* and T^* (given in Theorems 2 and 3) due to the use of first-order approximation, the optimal pattern obtained by the first-order analysis does not vary with D, while the optimal processor count obtained by the numerical solution decreases with increased downtime. This shows that the optimal pattern parameters are indeed influenced by the downtime. However, the simulated execution overheads in both cases



(a) (b) (c)

Figure 8: Optimal checkpointing period T^* and number of processors P^* (from Theorems 2 and 3, and from numerical solution), as well as the simulated execution overhead under different downtime D and when $\alpha = 0.0001$ on platform Hera.

stay close for the first-order solution and the optimal solution, because even a 3-hour downtime is nevertheless much smaller compared to the platform MTBF (in the order of days).

6 Conclusion

In this paper, we considered the optimal processor allocation problem for executing a parallel job on a large-scale platform subject to fail-stop and silent errors. We have provided the exact expression for the expected execution time of a pattern, and closed-form first-order approximation formulas to compute the optimal checkpointing period T^* and optimal number of processors P^*. These formulas are functions of several parameters: the individual processor failure rate λ_{ind}, the sequential fraction of the application α, as well as the checkpointing and verification costs C_P and V_P. For the latter costs, we have envisioned a comprehensive set of scenarios that are representative of the most important fault-tolerant protocols. To the best of our knowledge, these results are the first that analytically establish the relationship between P^* and T^* as a function of the resource MTBF $\mu_{\text{ind}} = 1/\lambda_{\text{ind}}$, and they offer new insights into the relationships of Amdhal's law and the Young/Daly approximation formula. Also, they provide the first (and direct) characterization of the optimal number of resources to enroll, with given error rates, resilience costs and application speedup profile. We have conducted an extensive set of simulations to support the theoretical study, whose outcome confirms the accuracy of first-order approximation under a wide range of parameter settings.

Further work will be devoted to exploring jobs with different speedup profiles, weak vs. strong scalability issues, and multi-level resilience protocols. All these questions raise important but challenging optimization problems. On a global perspective, we strongly believe that going beyond simulations and providing analytical solutions to these problems would be a major step to understanding the potential and limits of parallelism at extreme scale and in failure-prone environments.

Acknowledgments

This research was funded in part by the European project SCoRPiO, by the LABEX MILYON (ANR-10- LABX-0070) of Université de Lyon, within the program "Investissements d'Avenir" (ANR-11-IDEX-0007) operated by the French National Research Agency (ANR), and by the PIA ELCI project. Jiafan Li was supported by the JoRISS program jointly operated by ECNU Shanghai and ENS Lyon. Yves Robert is with Institut Universitaire de France.

References

[1] G. Amdahl. The validity of the single processor approach to achieving large scale computing capabilities. In *AFIPS Conference Proceedings*, volume 30, pages 483–485. AFIPS Press, 1967.

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

[3] L. Bautista Gomez and F. Cappello. Detecting and correcting data corruption in stencil applications through multivariate interpolation. In *Proc. FTS'15*, 2015.

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

[5] A. Benoit, A. Cavelan, Y. Robert, and H. Sun. Optimal resilience patterns to cope with fail-stop and silent errors. In *IPDPS*, 2016.

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

[7] E. Berrocal, L. Bautista-Gomez, S. Di, Z. Lan, and F. Cappello. Lightweight silent data corruption detection based on runtime data analysis for HPC applications. In *Proc. HPDC'15*, 2015.

[8] 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.

[9] G. Bosilca et al. Unified model for assessing checkpointing protocols at extreme-scale. *Concurrency and Computation: Practice and Experience*, 26(17):2772–2791, 2014.

[10] G. Bronevetsky and B. de Supinski. Soft error vulnerability of iterative linear algebra methods. In *Proc. ICS'08*, 2008.

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

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

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

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

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

[16] J. Dongarra, T. Hérault, and Y. Robert. Performance and reliability trade-offs for the double checkpointing algorithm. *Int. J. of Networking and Computing*, 4(1):23–41, 2014.

[17] J. Elliott, K. Kharbas, D. Fiala, F. Mueller, K. Ferreira, and C. Engelmann. Combining partial redundancy and checkpointing for HPC. In *Proc. ICDCS'12*, pages 615–626, 2012.

[18] E. Elnozahy and J. Plank. Checkpointing for peta-scale systems: a look into the future of practical rollback-recovery. *IEEE Trans. Dependable and Secure Computing*, 1(2):97–108, 2004.

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

[20] K. Ferreira, J. Stearley, J. H. I. Laros, R. Oldfield, K. Pedretti, R. Brightwell, R. Riesen, P. G. Bridges, and D. Arnold. Evaluating the Viability of Process Replication Reliability for Exascale Systems. In *Proc. SC'11*, pages 44:1–44:12, 2011.

[21] D. Fiala, F. Mueller, C. Engelmann, R. Riesen, K. Ferreira, and R. Brightwell. Detection and correction of silent data corruption for large-scale high-performance computing. In *Proc. SC'12*, page 78, 2012.

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

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

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

[25] H. Jin, Y. Chen, H. Zhu, and X.-H. Sun. Optimizing HPC fault-tolerant environment: An analytical approach. In *Proc. ICPP'10*, 2010.

[26] 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.

[27] A. Moody, G. Bronevetsky, K. Mohror, and B. R. d. Supinski. Design, modeling, and evaluation of a scalable multi-level checkpointing system. In *Proc. SC'10*, pages 1–11, 2010.

[28] X. Ni, E. Meneses, N. Jain, and L. V. Kalé. ACR: Automatic checkpoint/restart for soft and hard error protection. In *Proc. SC'13*, 2013.

[29] 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.

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

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

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

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

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

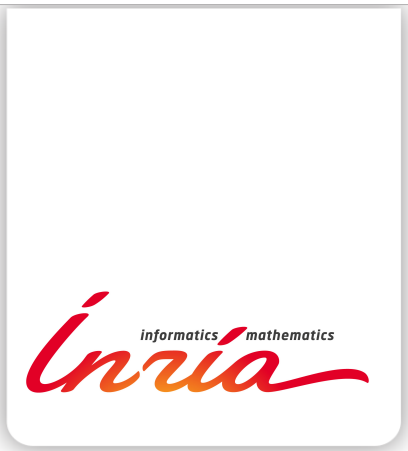
[35] P. M. Widener, K. B. Ferreira, S. Levy, and N. Fabian. Canaries in a coal mine: Using application-level checkpoints to detect memory failures. In *Euro-Par'15: Parallel Processing Workshops*. Springer LNCS 9523, 2015.

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

[37] G. Zheng, L. Shi, and L. V. Kale. FTC-Charm++: an in-memory checkpoint-based fault tolerant runtime for Charm++ and MPI. In *Proc. CLUSTER'04*, pages 93–103, 2004.

[38] Z. Zheng, L. Yu, and Z. Lan. Reliability-aware speedup models for parallel applications with coordinated checkpointing/restart. *IEEE Trans. Computers*, 64(5):1402–1415, 2015.

[39] 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.



RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

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

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

ISSN 0249-6399