Checkpointing algorithms and fault prediction

Guillaume Aupy, Yves Robert, Frédéric Vivien, Dounia Zaidouni

Checkpointing algorithms and fault prediction

Guillaume Aupy, Yves Robert, Frédéric Vivien, Dounia Zaidouni

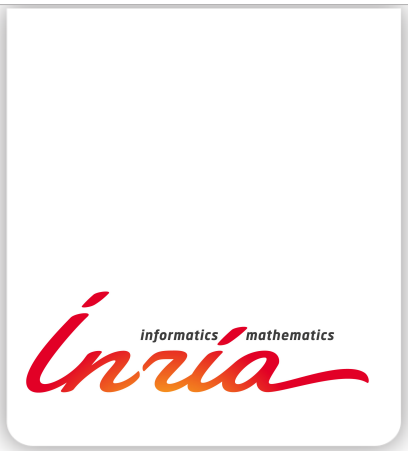
RESEARCH REPORT

N° 8237

February 2013

Project-Team ROMA

ISSN 0249-6399 ISRN INRIA/RR--8237--FR+ENG



Checkpointing algorithms and fault prediction

Guillaume Aupy*, Yves Robert*†‡, Frédéric Vivien§*, Dounia Zaidouni§*

Project-Team ROMA

Research Report n° 8237 — February 2013 — 30 pages

Abstract: This paper deals with the impact of fault prediction techniques on checkpointing strategies. We extend the classical first-order analysis of Young and Daly in the presence of a fault prediction system, characterized by its recall and its precision. In this framework, we provide an optimal algorithm to decide when to take predictions into account, and we derive the optimal value of the checkpointing period. These results allow to analytically assess the key parameters that impact the performance of fault predictors at very large scale.

Key-words: Fault-tolerance, checkpointing, prediction, algorithms, model, exascale

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

RESEARCH CENTRE GRENOBLE – RHÔNE-ALPES

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

Étude de l'impact de la prédiction de fautes sur les stratégies de protocoles de checkpoint

Résumé : Ce travail considère l'impact des techniques de prédiction de fautes sur les stratégies de protocoles de sauvegarde de points de reprise (*checkpoints*) et de redémarrage. Nous étendons l'analyse classique de Young en présence d'un système de prédiction de fautes, qui est caractérisé par son rappel (taux de pannes prévues sur nombre total de pannes) et par sa précision (taux de vraies pannes parmi le nombre total de pannes annoncées). Dans ce travail, nous avons pu obtenir la valeur optimale de la période de checkpoint (minimisant ainsi le gaspillage de l'utilisation des ressources dû au coût de prise de ces points de sauvegarde) dans différents scénarios. Ce papier pose les fondations théoriques pour de futures expériences et une validation du modèle.

Mots-clés : Tolérance aux pannes, checkpoint, prédiction, algorithmes, modèle, exascale

1 Introduction

Nowadays, the most powerful High Performance Computing systems experience about one fault per day [1, 2]. Consider the relative slopes describing the evolution of the reliability of individual components on one side, and the evolution of the number of components on the other side: the reliability of an entire platform is expected to decrease, due to probabilistic amplification, as its number of components increases. Therefore, applications running on large computing systems have to cope with platform faults. There are two main approaches. On the one hand, applications can use fault-tolerance mechanisms such as checkpoint and rollback in order to become resilient. On the other hand, system administrators can try to predict where and when faults will strike. Although considerable research has been devoted to fault predictors [3, 4, 5, 6, 7, 8], no predictor will ever be able to predict every fault. Therefore, fault predictors will have to be used in conjunction with fault-tolerance mechanisms.

In this paper, we assess the impact of fault prediction techniques on checkpointing strategies. We assume to have jobs executing on a platform subject to faults, and we let μ be the mean time between faults (MTBF) of the platform. In the absence of fault prediction, the standard approach is to take periodic checkpoints, each of length C, every period of duration T. In steady-state utilization of the platform, the value T_{opt} of T that minimizes the expected waste of resource usage due to checkpointing is approximated as $T_{\text{opt}} = \sqrt{2\mu C} + C$, or $T_{\text{opt}} = \sqrt{2(\mu + R)C} + C$ (where R is the duration of the recovery). The former expression is the well-known Young's formula [9], while the latter is due to Daly [10].

Now, when some fault prediction mechanism is available, can we compute a better checkpointing period to decrease the expected waste? and to what extent? Critical parameters that characterize a fault prediction system are its recall r, which is the fraction of faults that are indeed predicted, and its precision p, which is the fraction of predictions that are correct (i.e., correspond to actual faults). The major objective of this paper is to refine the expression of the expected waste as a function of these new parameters, and to design efficient checkpointing policies that take predictions into account. The key contributions of this paper are:

- A refined first-order analysis in the absence of fault prediction. It leads to similar performance to Young [9] and Daly [10] when faults follow an Exponential distribution, and to better performance when faults follow a Weibull distribution.
- The extension of this analysis to fault predictions, and the design of a new checkpointing policy that takes optimal decisions on whether to take these predictions into account or to ignore them.
- An extensive set of simulations that corroborates all mathematical derivations, both for Exponential fault distributions, and for (more realistic) Weibull fault distributions.

The rest of the paper is organized as follows. We first detail the framework in Section 2. We revisit Young and Daly's approach in Section 3. We provide an optimal algorithm to account for predictions in Section 4. Section 5 is devoted to simulations. We discuss related work in Section 6. Finally, we provide concluding remarks in Section 7.

2 Framework

2.1 Checkpointing strategy

We consider a *platform* subject to faults. Our work is agnostic of the granularity of the platform, which may consist either of a single processor, or of several processors that work concurrently and use coordinated checkpointing. *Checkpoints* are taken at regular intervals, or periods, of length T. We denote by C the duration of a checkpoint (all checkpoints have same duration). By construction, we must enforce that $C \leq T$. When a fault strikes the platform, the application is lacking some resource for a certain period of time of length D, the *downtime*. The downtime accounts for software rejuvenation (i.e., rebooting [11, 12]) or for the replacement of the failed hardware component by a spare one. Then, the application recovers from the last checkpoint. R denotes the duration of this *recovery* time.

2.2 Fault predictor

A fault predictor is a mechanism that is able to predict that some faults will take place, either at a certain point in time, or within some time-interval window. In this paper, we assume that the predictor is able to provide exact prediction dates, and to generate such predictions early enough so that a *proactive* checkpoint can indeed be taken before the event.

The accuracy of the fault predictor is characterized by two quantities, the *recall* and the *precision*. The recall r is the fraction of faults that are predicted while the precision p is the fraction of fault predictions that are correct. Traditionally, one defines three types of *events*: (i) *True positive* events are faults that the predictor has been able to predict (let $True_P$ be their number); (ii) *False positive* events are fault predictions that did not materialize as actual faults (let $False_P$ be their number); and (iii) *False negative* events are faults that were not predicted (let $False_N$ be their number). With these definitions, we have $r = \frac{True_P}{True_P + False_N}$ and $p = \frac{True_P}{True_P + False_P}$.

Proactive checkpoints may have a different length C_p than regular checkpoints of length C. In fact there are many scenarios. On the one hand, we may well have $C_p > C$ in scenarios where regular checkpoints are taken at time-steps where the application memory footprint is minimal [13]; on the contrary, proactive checkpoints are taken according to predictions that can take place at arbitrary instants. On the other hand, we may have $C_p < C$ in other scenarios [8], e.g., when the prediction is localized to a particular resource subset, hence allowing for a smaller volume of checkpointed data.

To keep full generality, we deal with two checkpoint sizes in this paper: C for periodic checkpoints, and C_p for proactive checkpoints (those taken upon predictions).

In the literature, the *lead time* is the interval between the date at which the prediction is made available, and the actual prediction date. However, we point out that the distribution of these lead times is irrelevant to the problem: either a fault is predicted at least C_p seconds in advance, and then one can checkpoint just in time before the fault, or the prediction is useless! In other words, predictions that come too late should be classified as unpredicted faults whenever they materialize as actual faults, leading to a smaller value of the

predictor recall.

2.3 Fault rates

The key parameter is μ, the mean time between faults (MTBF) of the platform. If the platform is made of N components whose individual MTBF is μ_{ind}, then $\mu = \frac{\mu_{\text{ind}}}{N}$. This result is true regardless of the fault distribution law[1].

In addition to μ, the platform MTBF, let μ_{P} be the mean time between predicted events (both true positive and false positive), and let μ_{NP} be the mean time between unpredicted faults (false negative). Finally, we define the mean time between events as μ_{e} (including all three event types). The relationships between μ, μ_{P}, μ_{NP}, and μ_{e} are the following:

- Rate of unpredicted faults: $\frac{1}{\mu_{\text{NP}}} = \frac{1-r}{\mu}$, since $1-r$ is the fraction of faults that are unpredicted;
- Rate of predicted faults: $\frac{r}{\mu} = \frac{p}{\mu_{\text{P}}}$, since r is the fraction of faults that are predicted, and p is the fraction of fault predictions that are correct;
- Rate of events: $\frac{1}{\mu_{\text{e}}} = \frac{1}{\mu_{\text{P}}} + \frac{1}{\mu_{\text{NP}}}$, since events are either predictions (true or false), or unpredicted faults.

2.4 Objective: waste minimization

The natural objective is to minimize the expectation of the total execution time, *makespan*, of the application. Instead, in order to ease mathematical derivations, we aim at minimizing the *waste*. The waste is the expected percentage of time lost, or "wasted", during the execution. In other words, the *waste* is the fraction of time during which the platform is not doing useful work. This definition was introduced by Wingstrom [14]. Obviously, the lower the waste, the lower the expected makespan, and reciprocally. Hence the two objectives are strongly related and minimizing one of them also minimizes the other.

3 Revisiting Daly's first-order approximation

Young proposed in [9] a "first order approximation to the optimum checkpoint interval". Young's formula was later refined by Daly [10] to take into account the recovery time. We revisit their analysis using the notion of waste.

Let $\textsc{Time}_{\text{base}}$ be the base time of the application without any overhead (neither checkpoints nor faults). First, assume a *fault-free* execution of the application with periodic checkpointing. In such an environment, during each period of length T we take a checkpoint, which lasts for a time C, and only $T - C$ units of work are executed. Let $\textsc{Time}_{\text{FF}}$ be the execution time of the application in this setting. Following most works in the literature, we also take a checkpoint at the end of the execution. The fault-free execution time $\textsc{Time}_{\text{FF}}$ is equal to the time needed to execute the whole application, $\textsc{Time}_{\text{base}}$, plus the time taken by the checkpoints:

$$\textsc{Time}_{\text{FF}} = \textsc{Time}_{\text{base}} + N_{\text{ckpt}} C \tag{1}$$

[1]For the sake of completeness, we provide a proof of this widely-used result in A. To the best of our knowledge, no proof has been published in the literature yet.

where N_{ckpt} is the number of checkpoints taken. We have

$$N_{\text{ckpt}} = \left\lceil \frac{\text{TIME}_{\text{base}}}{T-C} \right\rceil \approx \frac{\text{TIME}_{\text{base}}}{T-C}$$

When discarding the ceiling function, we assume that the execution time is very large with respect to the period or, symmetrically, that there are many periods during the execution. Plugging back the (approximated) value $N_{\text{ckpt}} = \frac{\text{TIME}_{\text{base}}}{T-C}$, we derive that

$$\text{TIME}_{\text{FF}} = \frac{\text{TIME}_{\text{base}}}{T-C} C \tag{2}$$

The waste due to checkpointing in a fault-free execution, WASTE_{FF}, is defined as the fraction of the execution time that does not contribute to the progress of the application:

$$\text{WASTE}_{\text{FF}} = \frac{\text{TIME}_{\text{FF}} - \text{TIME}_{\text{base}}}{\text{TIME}_{\text{FF}}} \qquad \Leftrightarrow \qquad \left(1-\text{WASTE}_{\text{FF}}\right)\text{TIME}_{\text{FF}} = \text{TIME}_{\text{base}} \tag{3}$$

Combining Equations (2) and (3), we get:

$$\text{WASTE}_{\text{FF}} = \frac{C}{T} \tag{4}$$

Now, let $\text{TIME}_{\text{final}}$ denote the expected execution time of the application in the presence of faults. This execution time can be divided into two parts: (i) the execution of "chunks" of work of size $T - C$ followed by their checkpoint; and (ii) the time lost due to the faults. This decomposition is illustrated by Figure 1. The first part of the execution time is equal to TIME_{FF}. Let N_{faults} be the number of faults occurring during the execution, and let T_{lost} be the average time lost per fault. Then,

$$\text{TIME}_{\text{final}} = \text{TIME}_{\text{FF}} + N_{\text{faults}} \times T_{\text{lost}} \tag{5}$$

On average, during a time $\text{TIME}_{\text{final}}$, $N_{\text{faults}} = \frac{\text{TIME}_{\text{final}}}{\mu}$ faults happen. We need to estimate T_{lost}. The instants at which periods begin and at which faults strike are independent. Therefore, the expected time elapsed between the completion of the last checkpoint and a fault is $\frac{T}{2}$ for all distribution laws, regardless of their particular shape. We conclude that $T_{\text{lost}} = \frac{T}{2} + D + R$, because after each fault there is a downtime and a recovery. This leads to:

$$\text{TIME}_{\text{final}} = \text{TIME}_{\text{FF}} + \frac{\text{TIME}_{\text{final}}}{\mu} \times \left(D + R + \frac{T}{2} \right)$$

Let $\text{WASTE}_{\text{fault}}$ be the fraction of the total execution time that is lost because of faults:

$$\text{WASTE}_{\text{fault}} = \frac{\text{TIME}_{\text{final}} - \text{TIME}_{\text{FF}}}{\text{TIME}_{\text{final}}} \qquad \Leftrightarrow \qquad \left(1 - \text{WASTE}_{\text{fault}}\right) \text{TIME}_{\text{final}} = \text{TIME}_{\text{FF}} \tag{6}$$

We derive:

$$\text{WASTE}_{\text{fault}} = \frac{1}{\mu} \left(D + R + \frac{T}{2} \right). \tag{7}$$

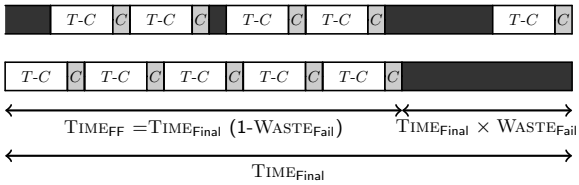


Figure 1: An execution (top), and its re-ordering (bottom), to illustrate both sources of waste. Blackened intervals correspond to work destroyed by faults, downtimes, and recoveries.

In [10], Daly uses the expression

$$\textsc{Time}_{\text{final}} = \big(1 + \textsc{Waste}_{\text{fault}}\big)\textsc{Time}_{\text{FF}} \tag{8}$$

instead of Equation (6), which leads him to his well-known first-order formula

$$T = \sqrt{2(\mu + (D + R))C} + C \tag{9}$$

Figure 1 explains why Equation (8) is not correct and should be replaced by Equation (6). Indeed, the expected number of faults depends on the final time, not on the time for a fault-free execution. We point out that Young [9] also used Equation (8), but with $D = R = 0$. Equation (6) can be rewritten $\textsc{Time}_{\text{final}} = \textsc{Time}_{\text{FF}}/\left(1 - \textsc{Waste}_{\text{fault}}\right)$. Therefore, using Equation (8) instead of Equation (6), in fact, is equivalent to write $\frac{1}{1-\textsc{Waste}_{\text{fault}}} \approx 1 + \textsc{Waste}_{\text{fault}}$ which is indeed a first-order approximation if $\textsc{Waste}_{\text{fault}} \ll 1$.

Now, let WASTE denote the total waste:

$$\textsc{Waste} = \frac{\textsc{Time}_{\text{final}} - \textsc{Time}_{\text{base}}}{\textsc{Time}_{\text{final}}} \tag{10}$$

Therefore

$$\textsc{Waste} = 1 - \frac{\textsc{Time}_{\text{base}}}{\textsc{Time}_{\text{final}}} = 1 - \frac{\textsc{Time}_{\text{base}}}{\textsc{Time}_{\text{FF}}} \frac{\textsc{Time}_{\text{FF}}}{\textsc{Time}_{\text{final}}} = 1 - (1 - \textsc{Waste}_{\text{FF}})(1 - \textsc{Waste}_{\text{fault}}).$$

Altogether, we derive the final result:

$$\begin{aligned} \textsc{Waste} &= \textsc{Waste}_{\text{FF}} + \textsc{Waste}_{\text{fault}} - \textsc{Waste}_{\text{FF}}\textsc{Waste}_{\text{fault}} && (11) \\ &= \frac{C}{T} + \left(1 - \frac{C}{T}\right)\frac{1}{\mu}\left(D + R + \frac{T}{2}\right) && (12) \end{aligned}$$

We obtain $\textsc{Waste} = \frac{u}{T} + v + wT$ where $u = C\big(1 - \frac{D+R}{\mu}\big)$, $v = \frac{D+R-C/2}{\mu}$, and $w = \frac{1}{2\mu}$. Thus WASTE is minimized for $T = \sqrt{\frac{u}{w}}$. The Refined First-Order (RFO) formula for the optimal period is thus:

$$T_{\text{RFO}} = \sqrt{2(\mu - (D + R))C} \tag{13}$$

It is interesting to point out why Equation (13) is a first-order approximation, even for large jobs. Indeed, there are several restrictions to enforce for the approach to be valid:

- We have stated that the expected number of faults during execution is $N_{\text{faults}} = \frac{\text{TIME}_{\text{final}}}{\mu}$, and that the expected time lost due to a fault is $T_{\text{lost}} = \frac{T}{2}$. Both statements are true individually, but the expectation of a product is the product of the expectations only if the random variables are independent, which is not the case here because $\text{TIME}_{\text{final}}$ depends upon the failure inter-arrival times.
- In Equation (4), we have to enforce $C \leq T$ to have $\text{WASTE}_{\text{FF}} \leq 1$
- In Equation (7), we have to enforce $D + R \leq \mu$ and to bound T in order to have $\text{WASTE}_{\text{fault}} \leq 1$. Intuitively, we need μ to be large enough for Equation (7) to make sense. However, regardless of the value of the individual MTBF μ_{ind}, there is always a threshold in the number of components N above which the platform MTBF $\mu = \frac{\mu_{\text{ind}}}{N}$ becomes too small for Equation (7) to be valid.
- Equation (7) is accurate only when two or more faults do not take place within the same period. Although unlikely when μ is large in front of T, the possible occurrence of many faults during the same period cannot be eliminated.

To ensure that the latter condition (at most a single fault per period) is met with a high probability, we cap the length of the period: we enforce the condition $T \leq \alpha\mu$, where α is some tuning parameter chosen as follows. The number of faults during a period of length T can be modeled as a Poisson process of parameter $\beta = \frac{T}{\mu}$. The probability of having $k \geq 0$ faults is $P(X = k) = \frac{\beta^k}{k!}e^{-\beta}$, where X is the number of faults. Hence the probability of having two or more faults is $\pi = P(X \geq 2) = 1 - (P(X = 0) + P(X = 1)) = 1 - (1 + \beta)e^{-\beta}$. If we assume $\alpha = 0.27$ then $\pi \leq 0.03$, hence a valid approximation when bounding the period range accordingly. Indeed, with such a conservative value for α, we have overlapping faults for only 3% of the checkpointing segments in average, so that the model is quite reliable. For consistency, we also enforce the same type of bound on the checkpoint time, and on the downtime and recovery: $C \leq \alpha\mu$ and $D+R \leq \alpha\mu$. However, enforcing these constraints may lead to use a sub-optimal period: it may well be the case that the optimal period $\sqrt{2(\mu - (D + R))C}$ of Equation (13) does not belong to the admissible interval $[C, \alpha\mu]$. In that case, the waste is minimized for one of the bounds of the admissible interval: this is because, as seen from Equation (12), the waste is a convex function of the period.

We conclude this discussion on a positive note. While capping the period, and enforcing a lower bound on the MTBF, is mandatory for mathematical rigor, simulations (see Section 5 for both Exponential and Weibull distributions) show that actual job executions can always use the value from Equation (13), accounting for multiple faults whenever they occur by re-executing the work until success. The first-order model turns out to be surprisingly robust!

To the best of our knowledge, despite all the limitations above, there is no better approach to estimate the waste due to checkpointing when dealing with arbitrary fault distributions. However, assuming that faults obey an Exponential distribution, it is possible to use the memory-less property of this distribution to provide more accurate results. A second-order approximation when faults obey an Exponential distribution is given in Daly [10, Equation (20)] as $\text{TIME}_{\text{final}} = \mu e^{R/\mu}(e^{\frac{T}{\mu}} - 1)\frac{\text{TIME}_{\text{base}}}{T-C}$. In fact, in that case, the exact value of $\text{TIME}_{\text{final}}$ is provided in [15, 16] as $\text{TIME}_{\text{final}} = (\mu + D)e^{R/\mu}(e^{\frac{T}{\mu}} - 1)\frac{\text{TIME}_{\text{base}}}{T-C}$,

and the optimal period is then $\frac{1+\mathbb{L}(-e^{-\frac{C}{\mu}-1})}{\mu}$ where $\mathbb{L}$, the Lambert function, is defined as $\mathbb{L}(z)e^{\mathbb{L}(z)} = z$.

To assess the accuracy of the different first order approximations, we compare the periods defined by Young's formula [9], Daly's formula [10], and Equation (13), to the optimal period, in the case of an Exponential distribution. Results are reported in Table 1. To establish these results, we use the same parameters as in Section 5: $C = R = 600$ s, $D = 60$ s, and $\mu_{\text{ind}} = 125$ years. Furthermore, to compute the optimal period, for each platform size we choose the application size so that $\text{TIME}_{\text{base}} = 2$ hours. One can observe in Table 1 that the relative error for Daly's period is slightly larger than the one for Young's period. In turn, the absolute value of the relative error for Young's period is slightly larger than the one for RFO. More importantly, when Young's and Daly's formulas overestimate the period, RFO underestimates it. Table 1 does not allow us to assess whether these differences are actually significant. However we also report in Section 5.2 some simulations that show that Equation (13) leads to smaller execution times for Weibull distributions than both classical formulas (Tables 3 and 4).

N	μ	Young		Daly		RFO		Optimal
2^{10}	3849609	68567	(0.5 %)	68573	(0.5 %)	67961	(-0.4 %)	68240
2^{11}	1924805	48660	(0.7 %)	48668	(0.7 %)	48052	(-0.6 %)	48320
2^{12}	962402	34584	(1.2 %)	34595	(1.2 %)	33972	(-0.6 %)	34189
2^{13}	481201	24630	(1.6 %)	24646	(1.7 %)	24014	(-0.9 %)	24231
2^{14}	240601	17592	(2.3 %)	17615	(2.5 %)	16968	(-1.3 %)	17194
2^{15}	120300	12615	(3.2 %)	12648	(3.5 %)	11982	(-1.9 %)	12218
2^{16}	60150	9096	(4.5 %)	9142	(5.1 %)	8449	(-2.9 %)	8701
2^{17}	30075	6608	(6.3 %)	6673	(7.4 %)	5941	(-4.4 %)	6214
2^{18}	15038	4848	(8.8 %)	4940	(10.8 %)	4154	(-6.8 %)	4458
2^{19}	7519	3604	(12.0 %)	3733	(16.0 %)	2869	(-10.8 %)	3218

Table 1: Comparing periods produced by the different approximations with optimal value. Beside each period, we report its relative deviation to the optimal. Each value is expressed in seconds.

4 Taking predictions into accounts

In this section, we present an analytical model to assess the impact of predictions on periodic checkpointing strategies. As already mentioned, we consider the case where the predictor is able to provide exact prediction dates, and to generate such predictions at least C_p seconds in advance, so that a proactive checkpoint of length C_p can indeed be taken before the event.

For the sake of clarity, we start with a simple algorithm (Section 4.1) which we refine in Section 4.2. We then compute the value of the period that minimizes the waste in Section 4.3.

4.1 Simple algorithm

In this section, we consider the following algorithm:

- While no fault prediction is available, checkpoints are taken periodically with period T;
- When a fault is predicted, there are two cases: either there is the possibility to take a proactive checkpoint, or there is not enough time to do so, because we are already checkpointing (see Figures 2(b) and 2(c)). In the latter case, there is no other choice than ignoring the prediction. In the former case, we still have the possibility to ignore the prediction, but we may also decide to trust it: in fact the decision is randomly taken. With probability q, we trust the predictor and take the prediction into account (see Figures 2(f) and 2(g)) ,and, with probability $1 - q$, we ignore the prediction (see Figures 2(d) and 2(e));
- If we take the prediction into account, we take a proactive checkpoint (of length C_p) as late as possible, i.e., so that it completes right at the time when the fault is predicted to happen. After this checkpoint, we complete the execution of the period (see Figures 2(f) and 2(g));
- If we ignore the prediction, either by necessity (not enough time to take an extra checkpoint, see Figures 2(b) and 2(c)), or or by choice (with probability $1-q$, Figures 2(d) and 2(e)), we finish the current period and start a new one.

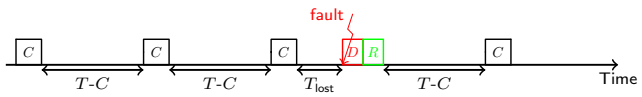
The rationale for not always trusting the predictor is to avoid taking useless checkpoints too frequently. Intuitively, the precision p of the predictor must be above a given threshold for its usage to be worthwhile. In other words, if we decide to checkpoint just before a predicted event, either we will save time by avoiding a costly re-execution if the event does correspond to an actual fault, or we will lose time by unduly performing an extra checkpoint. We need a larger proportion of the former cases, i.e., a good precision, for the predictor to be really useful. The following analysis will determine the optimal value of q as a function of the parameters C, C_p, μ, r, and p.

We could refine the approach by taking into account the amount of work already done in the current period when deciding whether to trust the predictor or not. Intuitively, the more work already done, the more important to save it, hence the more worthwhile to trust the predictor. We design such a refined strategy in Section 4.2. Right now, we analyze a simpler algorithm where we decide to trust or not to trust the predictor, independently of the amount of work done so far within the period.

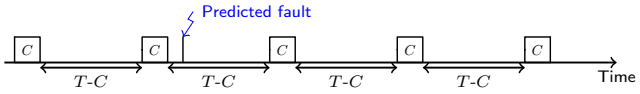
We analyze the algorithm in order to compute a formula for the expected waste, just as in Equation (12). While the value of WASTE_{FF} is unchanged ($\text{WASTE}_{\text{FF}} = \frac{C}{T}$), the value of $\text{WASTE}_{\text{fault}}$ is modified because of predictions. As illustrated in Figure 2, there are many different scenarios that contribute to $\text{WASTE}_{\text{fault}}$ that can be sorted into three categories:

(1) **Unpredicted faults:** This overhead occurs each time an unpredicted fault strikes, that is, on average, once every μ_{NP} seconds. Just as in Equation (7), the corresponding waste is $\frac{1}{\mu_{\text{NP}}}\left[\frac{T}{2} + D + R\right]$.

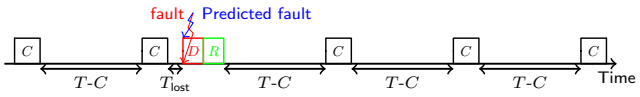
(2) **Predictions not taken into account:** The second source of waste is for predictions that are ignored. This overhead occurs in two different scenarios. First, if we do not have time to take a proactive checkpoint, we have an overhead



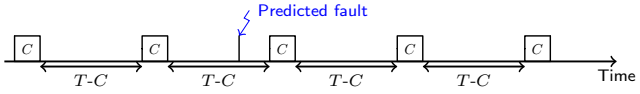
(a) Unpredicted fault



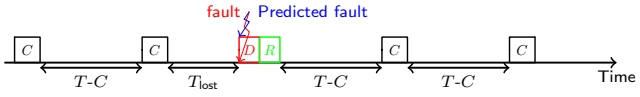
(b) Prediction cannot be taken into account - no actual fault



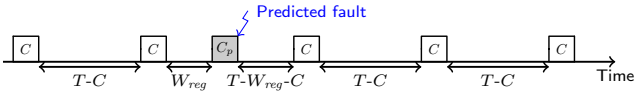
(c) Prediction cannot be taken into account - with actual fault



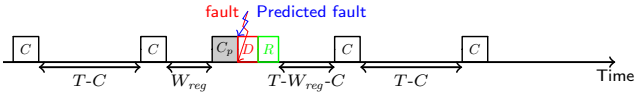
(d) Prediction not taken into account by choice - no actual fault



(e) Prediction not taken into account by choice - with actual fault



(f) Prediction taken into account - no actual fault



(g) Prediction taken into account - with actual fault

Figure 2: Actions taken for the different event types.

if and only the prediction is an actual fault. This case happens with probability p. We then lose a time $t + D + R$ if the predicted fault happens a time t after the completion of the last periodic checkpoint. The expected time lost is thus

$$T^1_{\text{lost}} = \frac{1}{T} \int_0^{C_p} (p(t + D + R) + (1 - p)0)\, dt$$

Then, if we do have time to take a proactive checkpoint but still decide to ignore the prediction, we also have an overhead if and only the prediction is an actual fault, but the expected time lost is now weighted by the probability $(1 - q)$:

$$T^2_{\text{lost}} = (1 - q)\frac{1}{T} \int_{C_p}^{T} (p(t + D + R) + (1 - p)0)\, dt$$

(3) **Predictions taken into account:** We now compute the overhead due to a prediction which we trust (hence we checkpoint just before its date). If the prediction is an actual fault, we lose $C_p + D + R$ seconds, but if it is not, we lose the unnecessary extra checkpoint time C_p. The expected time lost is now weighted by the probability q and becomes

$$T^3_{\text{lost}} = q\frac{1}{T} \int_{C_p}^{T} (p(C_p + D + R) + (1 - p)C_p)\, dt$$

We derive the final value of $\text{WASTE}_{\text{fault}}$:

$$\text{WASTE}_{\text{fault}} = \frac{1}{\mu_{\text{NP}}} \left[\frac{T}{2} + D + R\right] + \frac{1}{\mu_{\text{P}}} \left[T^1_{\text{lost}} + T^2_{\text{lost}} + T^3_{\text{lost}}\right]$$

This final expression comes from the disjunction of all possibles cases, using the Law of Total Probability [17, p.23]: the waste comes either from non-predicted faults or from predictions; in the latter case, we have analyzed the three possible sub-cases and weighted them with their respective probabilities. After simplifications, we obtain

$$\text{WASTE}_{\text{fault}} = \frac{1}{\mu} \left((1 - rq)\frac{T}{2} + D + R + \frac{qr}{p} C_p - \frac{qrC_p^2}{pT}(1 - p/2) \right) \qquad (14)$$

We could now plug this expression back into Equation (11) to compute the value of T that minimizes the total waste. Instead, we move on to describing the refined algorithm, and we minimize the waste for the refined strategy, since it always induces a smaller waste.

4.2 Refined algorithm

In this section, we refine the approach and consider different trust strategies, depending upon the time in the period where the prediction takes place. Intuitively, the later in the period, the more likely we are inclined to trust the predictor, because the amount of work that we could lose gets larger and larger. As before, we cannot take into account a fault predicted to happen less than C_p units of time after the beginning of the period. Therefore, we focus on what happens in the period after time C_p. Formally, we now divide the interval $[C_p, T]$

into n intervals $[\beta_i; \beta_{i+1}]$ for $i \in \{0, \cdots, n-1\}$, where $\beta_0 = C_p$ and $\beta_n = T$. For each interval $[\beta_i; \beta_{i+1}]$, we trust the predictor with probability q_i. We aim at determining the values of n, β_i, and q_i that minimize the waste. As mentioned before, intuition tells us that the q_i values should be non-decreasing. We prove below a somewhat unexpected theorem: in the optimal strategy, there is either one or two different q_i values, and these values are 0 or 1. This means that we should *never* trust the predictor in the beginning of a period, and always trust it in the end of the period, without any intermediate behavior in between.

We formally express this striking result below. Let $\beta_{\text{lim}} = \frac{C_p}{p}$. The optimal strategy is provided by Theorem 1 below. We first prove the following proposition:

Proposition 1. *The values of β_i and q_i that minimize the waste satisfy the following conditions:*
(i) For all i such that $\beta_{i+1} \leq \beta_{lim}$, $q_i = 0$.
(ii) For all i such that $\beta_i \geq \beta_{lim}$, $q_i = 1$.

Proof. First we compute the waste with the refined algorithm, using Equation (11). The formula for $\text{WASTE}_{\text{fault}}$ is similar to Equation (14) on each interval:

$$\begin{aligned}\text{WASTE} = \frac{C}{T} + \left(1 - \frac{C}{T}\right) \Bigg[\frac{1}{\mu_{\text{NP}}} \left(\frac{T}{2} + D + R \right) \\ + \frac{1}{\mu_{\text{P}}} \sum_{i=0}^{n-1} \left(q_i \int_{\beta_i}^{\beta_{i+1}} \frac{(p(C_p + D + R) + (1-p)C_p)}{T} dt \right. \\ \left. + \ (1 - q_i) \int_{\beta_i}^{\beta_{i+1}} \frac{p(t + D + R)}{T} dt \right) \Bigg]\end{aligned}$$

Now, consider a fixed value of i and express the value of WASTE as a function of q_i:

$$\text{WASTE} = K + \left(1 - \frac{C}{T}\right) \frac{q_i}{\mu_{\text{P}}} \int_{\beta_i}^{\beta_{i+1}} \left(\frac{C_p}{T} - \frac{pt}{T} \right) dt$$

where K does not depend on q_i. From the sign of the function to be integrated, one sees that WASTE is minimized when $q_i = 0$ if $\beta_{i+1} \leq \beta_{\text{lim}} = \frac{C_p}{p}$, and when $q_i = 1$ if $\beta_i \geq \beta_{\text{lim}}$. □

Theorem 1. *The optimal algorithm takes proactive actions if and only if the prediction falls in the interval $[\beta_{lim}, T]$.*

Proof. From Proposition 1, the values for q_i are optimally defined for every i but one: we do not know the optimal value if there exists i_0 such that $\beta_{i_0} < \beta_{\text{lim}} < \beta_{i_0+1}$. Then let us consider the waste where q_{i_0} is replaced by $q_{i_0}^{(1)}$ on $[\beta_{i_0}, \beta_{\text{lim}}]$ and by $q_{i_0}^{(2)}$ on $[\beta_{\text{lim}}, \beta_{i_0+1}]$. The new waste is necessarily smaller than the one with only q_{i_0}, since we relaxed the constraint. We know from Proposition 1 that the optimal solution is then to have $q_{i_0}^{(1)} = 0$ and $q_{i_0}^{(2)} = 1$. □

Let us now compute the value of the waste with the optimal algorithm. There are two cases, depending upon whether $T \leq \beta_{\text{lim}}$ or not. For values of T smaller than β_{lim}, Theorem 1 shows that the optimal algorithm never takes any proactive action; in that case the waste is given by Equation (12) in Section 3. For values of T larger than $\beta_{\text{lim}} = \frac{C_p}{p}$, we compute the waste due to predictions as

$$\frac{1}{\mu_{\text{P}}}\frac{1}{T}\left(\int_0^{C_p/p} p(t+D+R)dt + \int_{C_p/p}^{T} (p(C_p+D+R)+(1-p)C_p)dt\right)$$
$$= \frac{r}{p\mu}\left(p(D+R)+C_p-\frac{C_p^2}{2pT}\right)$$

Indeed, in accordance with Theorem 1, no prediction is taken into account in the interval $[0, \frac{C_p}{p}]$, while all predictions are taken into account in the interval $[\frac{C_p}{p}, T]$. Adding the waste due to unpredicted faults, namely $\frac{1}{\mu_{\text{NP}}}\left[\frac{T}{2}+D+R\right]$, we derive

$$\textsc{Waste}_{\text{fault}} = \frac{1}{\mu}\left((1-r)\frac{T}{2} + \frac{r}{p}C_p\left(1-\frac{1}{2p}\frac{C_p}{T}\right)+D+R\right).$$

Plugging this value into Equation (11), we obtain the total waste when $\frac{C_p}{p} \leq T$:

$$\begin{aligned}\textsc{Waste} &= \frac{C}{T} + \frac{1}{\mu}\left((1-r)\frac{T}{2} + \frac{r}{p}C_p\left(1-\frac{1}{2p}\frac{C_p}{T}\right)+D+R\right)\left(1-\frac{C}{T}\right)\\ &= \frac{rCC_p^2}{2p^2}\frac{1}{\mu T^2} + \left(\mu C - \frac{rC_p^2}{2p^2} - C\left(\frac{rC_p}{p}+D+R\right)\right)\frac{1}{\mu T} + \frac{1-r}{2\mu}T\\ &\quad + \frac{-(1-r)\frac{C}{2}+\frac{rC_p}{p}+D+R}{\mu}\end{aligned}$$

Altogether, the expression for the total waste becomes:

$$\begin{cases}\textsc{Waste}_1(T) = \frac{C\left(1-\frac{D+R}{\mu}\right)}{T} + \frac{D+R-C/2}{\mu} + \frac{1}{2\mu}T & \text{if } \frac{C_p}{p} \geq T\\ \\ \textsc{Waste}_2(T) = \frac{rCC_p^2}{2\mu p^2}\frac{1}{T^2} + \frac{\left(C\left(1-\frac{\frac{rC_p}{p}+D+R}{\mu}\right)-\frac{rC_p^2}{2\mu p^2}\right)}{T} + \frac{-(1-r)\frac{C}{2}+\frac{rC_p}{p}+D+R}{\mu} + \frac{1-r}{2\mu}T & \text{if } \frac{C_p}{p} \leq T\end{cases} \quad (15)$$

One can check that when $r = 0$ (no error predicted, hence no proactive action in the algorithm), then $\textsc{Waste}_1$ and $\textsc{Waste}_2$ coincide. We also check that both values coincide for $T = \frac{C_p}{p}$. We show how to minimize the waste in Equation (15) in Section 4.3.

4.3 Waste minimization

In this section we focus on minimizing the waste in Equation (15). Recall that, by construction, we always have to enforce the constraint $T \geq C$. First consider the case where $C \leq \frac{C_p}{p}$. On the interval $T \in [C, \frac{C_p}{p}]$, we retrieve the optimal

value found in Section 3, and derive that WASTE_1, the waste when predictions are not taken into account, is minimized for

$$T_{\text{NoPred}} = \max\left(C, \min\left(T_{\text{RFO}}, \frac{C_p}{p}\right)\right) \tag{16}$$

Indeed, the optimal value should belong to the interval $[C, \frac{C_p}{p}]$, and the function WASTE_1 is convex: if the extremal solution $\sqrt{2(\mu-(D+R))C}$ does not belong to this interval, then the optimal value is one of the bounds of the interval.

On the interval $T \in \left[\frac{C_p}{p}, +\infty\right)$, we find the optimal solution by differentiating twice WASTE_2 with respect to T. Writing $\text{WASTE}_2(T) = \frac{u}{T^2} + \frac{v}{T} + w + xT$ for simplicity, we obtain $\text{WASTE}_2''(T) = \frac{2}{T^3}\left(\frac{3u}{T} + v\right)$. Here, a key parameter is the sign of :

$$v = \left(C\left(1 - \frac{\frac{rC_p}{p} + D + R}{\mu}\right) - \frac{rC_p^2}{2\mu p^2}\right)$$

We detail the case $v \geq 0$ in the following, because it is the most frequent with realistic parameter sets; we do have $v \geq 0$ for all the whole range of simulations in Section 5. For the sake of completeness, we will briefly discuss the case $v < 0$ in the comments below.

When $v \geq 0$, we have $\text{WASTE}_2''(T) \geq 0$, so that WASTE_2 is convex on the interval $\left[\frac{C_p}{p}, +\infty\right)$ and admits a unique minimum T_{extr}. Note that T_{extr} can be computed either numerically or using Cardano's method, since it is the unique real root of a polynomial of degree 3. The optimal solution on $\left[\frac{C_p}{p}, +\infty\right)$ is then: $T_{\text{Pred}} = \max\left(T_{\text{extr}}, \frac{C_p}{p}\right)$.

It remains to consider the case where $\frac{C_p}{p} < C$. In fact, it suffices to add the constraint that the value of T_{Pred} should be greater than C, that is:

$$T_{\text{Pred}} = \max\left(C, \max\left(T_{\text{extr}}, \frac{C_p}{p}\right)\right) \tag{17}$$

Finally, the optimal solution for the waste is given by the minimum of the following two values:

$$\begin{cases} \frac{C\left(1-\frac{D+R}{\mu}\right)}{T_{\text{NoPred}}} + \frac{D+R-C/2}{\mu} + \frac{1}{2\mu}T_{\text{NoPred}} \\ \\ \frac{rCC_p^2}{2\mu p^2}\frac{1}{T_{\text{Pred}}^2} + \frac{\left(C\left(1-\frac{\frac{rC_p}{p}+D+R}{\mu}\right)-\frac{rC_p^2}{2\mu p^2}\right)}{T_{\text{Pred}}} + \frac{-(1-r)\frac{C}{2}+\frac{rC_p}{p}+D+R}{\mu} + \frac{1-r}{2\mu}T_{\text{Pred}} \end{cases}$$

We make a few observations:

- Just as for Equation (13) in Section 3, mathematical rigor calls for capping the values of D, R, C, C_p and T in front of the MTBF. The only difference is that we should replace μ by μ_e: this is to account for the occurrence rate of all events, be they unpredicted faults or predictions.
- While the expression of the waste looks complicated, the numerical value of the optimal period can easily be computed in all cases. We have dealt

with the case $v \geq 0$, where v is the coefficient of $1/T$ in $\text{WASTE}_2(T) = \frac{u}{T^2} + \frac{v}{T} + w + xT$. When $v < 0$ we only needs to compute all the nonnegative real roots of a polynomial of degree 3, and check which one leads to the best value. More precisely, these root(s) partition the admissible interval $\left[\frac{C_p}{p}, +\infty\right)$ into several sub-intervals, and the optimal value is either a root or a sub-interval bound.

- In many practical situations, when μ is large enough, we can dramatically simplify the expression of $\text{WASTE}_2(T)$: we have $T = O(\sqrt{\mu})$, the term $\frac{u}{T^2}$ becomes negligible, checkpoint parameters become negligible in front of μ, and we derive the approximated value $\sqrt{\frac{2\mu C}{1-r}}$. This value can be seen as an extension of Equation (13) giving T_{RFO}, where μ is replaced by $\frac{\mu}{1-r}$: faults are replaced by non-predicted faults, and the overhead due to false predictions is negligible. As a word of caution, recall that this conclusion is valid only when μ is very large in front of all other parameters.

5 Simulation results

We start by presenting the simulation framework (Section 5.1). Then we report results using the characteristics of two fault predictors from the literature (Section 5.2). Finally, we assess the respective impact of the two key parameters of the predictor, its recall and its precision, on checkpointing strategies (Section 5.3).

5.1 Simulation framework

In order to validate our model, we have instantiated it with several scenarios. The experiments use parameters that are representative of current and forthcoming large-scale platforms [18, 19]. We take $C = R = 10$ min, and $D = 1$ min. For the proactive checkpoints we consider three scenarios where proactive checkpoints are (i) exactly as expensive as periodic ones ($C_p = C$), (ii) ten times cheaper ($C_p = 0.1C$), and (iii) two times more expensive ($C_p = 2C$). The individual (processor) MTBF is $\mu_{\text{ind}} = 125$ years, and the total number of processors N varies from $N = 16,384$ to $N = 524,288$, so that the platform MTBF μ varies from $\mu = 4,010$ min (about 2.8 days) down to $\mu = 125$ min (about 2 hours). For instance the Jaguar platform, with $N = 45,208$ processors, is reported to experience about one fault per day [1], which leads to $\mu_{\text{ind}} = \frac{45,208}{365} \approx 125$ years. The application size is set to $\text{TIME}_{\text{base}} = 10,000$ years/N.

We used Maple to analytically compute and plot the optimal value of the waste for both the algorithm taking predictions into account, OPTIMALPREDICTION, and for the algorithm ignoring them, RFO. In order to check the accuracy of our model, we have compared these results with results obtained with a discrete-event simulator. The simulation engine generates a random trace of faults, parameterized either by an Exponential fault distribution or by Weibull distribution laws with shape parameter 0.5 or 0.7. Note that Exponential faults are widely used for theoretical studies, while Weibull faults are representative of the behavior of real-world platforms [20, 21, 22]. In both cases, the distribution is scaled so that its expectation corresponds to the platform MTBF μ. With probability r, we decide if a fault is predicted or not. The simulation engine also

generates a random trace of false predictions, whose distribution is identical to that of the first trace (in B, we also consider the case where false predictions are generated according to a uniform distribution; results are quite similar). This second distribution is scaled so that its expectation is equal to $\frac{\mu_P}{1-p} = \frac{p\mu}{r(1-p)}$, the inter-arrival time of false predictions. Finally, both traces are merged to produce the final trace including all events (true predictions, false predictions, and non predicted faults). Each reported value is the average over 100 randomly generated instances.

In the simulations, we compare four checkpointing strategies:

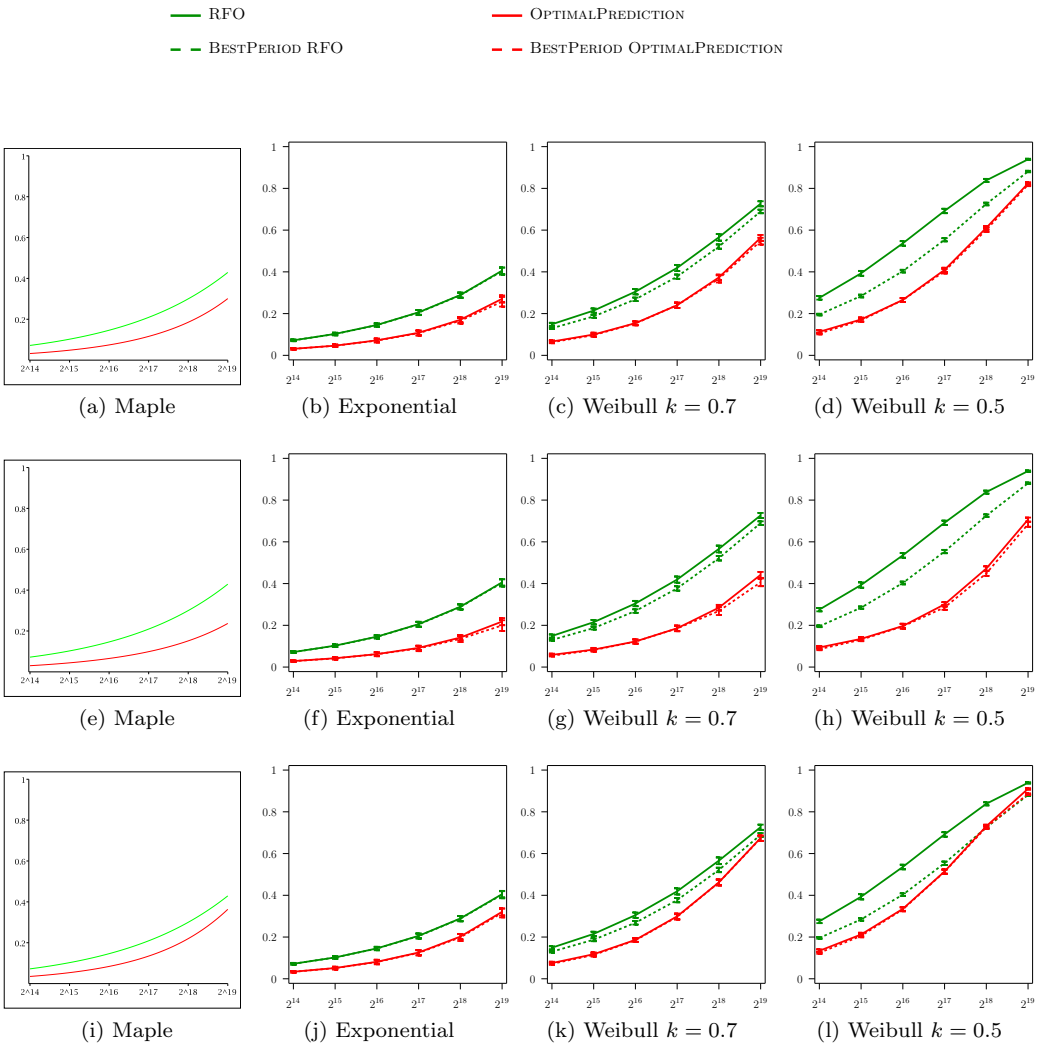
- RFO is the checkpointing strategy of period $T = \sqrt{2(\mu - (D+R))C}$ (see Section 3).
- OptimalPrediction is the refined algorithm described in Section 4.2.
- To assess the quality of each strategy, we compare it with its BestPeriod counterpart, defined as the same strategy but using the best possible period T. This latter period is computed via a brute-force numerical search for the optimal period (each tested period is evaluated on 100 randomly generated traces, and the period achieving the best average performance is elected as the "best period").

5.2 Predictors from the literature

We first experiment with two predictors from the literature: one accurate predictor with high recall and precision [7], namely with $p = 0.82$ and $r = 0.85$, and another predictor with intermediate recall and precision [8], namely with $p = 0.4$ and $r = 0.7$. Figures 3 and 4 show the average waste degradation for the two checkpointing policies, and for their BestPeriod counterparts, for both predictors. The waste is reported as a function of the number of processors N. We draw the plots as a function of the number of processors N rather than of the platform MTBF $\mu = \mu_{\text{ind}}/N$, because it is more natural to see the waste increase with larger platforms. However, recall that this work is agnostic of the granularity of the processors and intrinsically focuses on the impact of the MTBF on the waste.

The first observation is the very good correspondence between analytical results and simulations in Figures 3 and 4. In particular, the Maple plots and the simulations for Exponentially distributed faults are very similar. This shows the validity of the model and of its analysis. Another striking result is that OptimalPrediction has the same waste as its BestPeriod counterpart even for Weibull fault distributions, which demonstrates that our period T_{Pred} is indeed the best possible checkpointing period. These conclusions are valid regardless of the cost ratio of periodic and proactive checkpoints.

The second observation is that the prediction is useful for the vast majority of the set of parameters under study! However, when proactive checkpoints are cheaper than periodic ones, the benefits of fault prediction are increased. On the contrary, when proactive checkpoints are more expensive than periodic ones, the benefits of fault prediction are greatly reduced. One can even observe that the waste with prediction is not better than without prediction in the following scenario: $C_p = 2C$, and using the limited-quality predictor ($p = 0.4$, $r = 0.7$) with 2^{19} processors, see Figures 4(i),(j),(k) and (l).

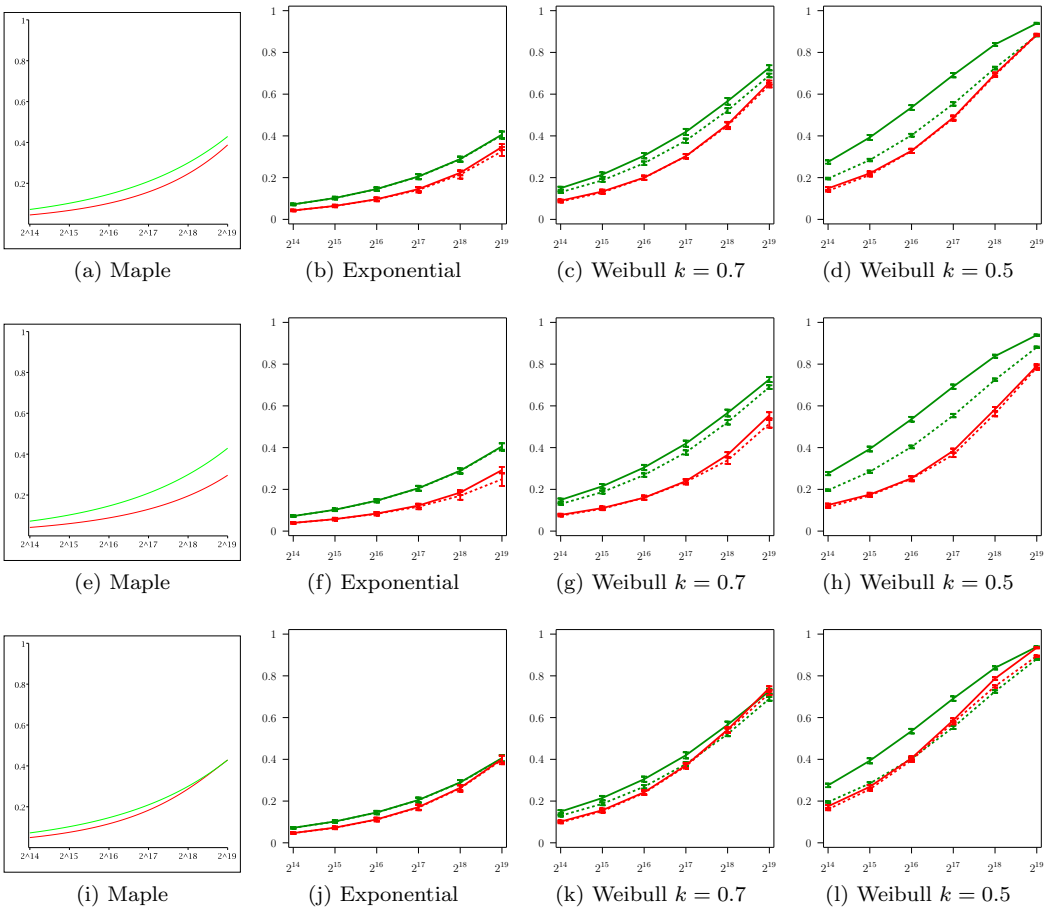


(a) Maple (b) Exponential (c) Weibull $k = 0.7$ (d) Weibull $k = 0.5$

(e) Maple (f) Exponential (g) Weibull $k = 0.7$ (h) Weibull $k = 0.5$

(i) Maple (j) Exponential (k) Weibull $k = 0.7$ (l) Weibull $k = 0.5$

Figure 3: Waste (y-axis) for the different heuristics as a function of the platform size (x-axis), with $p = 0.82$, $r = 0.85$, $C_p = C$ (first row), $C_p = 0.1C$ (second row), or $C_p = 2C$ (third row) and with a trace of false predictions parametrized by a distribution identical to the distribution of the trace of failures.



(a) Maple (b) Exponential (c) Weibull $k = 0.7$ (d) Weibull $k = 0.5$

(e) Maple (f) Exponential (g) Weibull $k = 0.7$ (h) Weibull $k = 0.5$

(i) Maple (j) Exponential (k) Weibull $k = 0.7$ (l) Weibull $k = 0.5$

Figure 4: Waste (y-axis) for the different heuristics as a function of the platform size (x-axis), with $p = 0.4$, $r = 0.7$, $C_p = C$ (first row), $C_p = 0.1C$ (second row), or $C_p = 2C$ (third row) and with a trace of false predictions parametrized by a distribution identical to the distribution of the trace of failures.

In order to compare the heuristics without prediction to those with prediction, we report job execution times when fault distribution follows either an Exponential distribution law (Table 2), or a Weibull distribution law (Table 3 for shape parameter $k = 0.7$ and Table 4 for $k = 0.5$).

We compute the gain (expressed in percentage) achieved by OptimalPrediction over RFO. We also add in these tables the execution times obtained when using the expression of T given by Young [9] and Daly [10] (denoted respectively as Young and Daly) to assess whether T_{RFO} is a better approximation. Recall that these three approaches do not use any predictor, which explains why the numbers are identical on both sides of the tables.

As a general trend, we observe that the gains due to predictions are more important when the distribution law is further apart from an Exponential distribution. Indeed, the largest gains are when the fault distribution follows a Weibull law of parameter 0.5. Using OptimalPrediction in conjunction with a "good" fault predictor gains up to 66% when there is a large number of processors (2^{19}). The gain is still of 37% with 2^{16} processors. Using a predictor with limited recall and precision, OptimalPrediction can still decrease the execution time by 47% with 2^{19} processors, and 31% with 2^{16} processors. In all tested cases, the decrease of the execution times is significant. Gains are less important with Weibull laws of shape parameter $k = 0.7$, however they are still reaching a minimum of 13% with 2^{16} processors, and up to 38% with 2^{19} processors. Finally, gains are further reduced with an Exponential law. They are still reaching at least 5% with 2^{16} processors, and up to 19% with 2^{19} processors.

Coming back to the case without fault prediction, it is striking to observe in Tables 3 and 4 that job execution time increases together with the number for processors (from $N = 2^{16}$ to $N = 2^{19}$) if the checkpointing period is Daly or Young. On the contrary, job execution time (rightfully) decreases when using RFO. However, the expressions of T given by Young, Daly and RFO are identical for Exponential distributions (Table 2). This confirms the analytical evaluation of Table 1 in Section 3. Altogether, the main (striking) conclusion is that RFO should be preferred to both classical approaches for Weibull distributions.

$C_p = C$	Execution time (in days) ($p = 0.82$, $r = 0.85$)		Execution time (in days) ($p = 0.4$, $r = 0.7$)	
	2^{16} procs	2^{19} procs	2^{16} procs	2^{19} procs
Young	65.2	11.7	65.2	11.7
Daly	65.2	11.8	65.2	11.8
RFO	65.2	11.7	65.2	11.7
OptimalPrediction	60.0 (8%)	9.5 (19%)	61.7 (5%)	10.7 (8%)

Table 2: Job execution times for an Exponential distribution, and gains due to the fault predictor (with respect to the performance of RFO).

5.3 Recall vs. precision

In this section, we assess the impact of the two key parameters of the predictor, its recall r and its precision p. To this purpose, we conduct simulations where one parameter is fixed while the other varies. We choose two platforms, a smaller

$C_p = C$	Execution time (in days) ($p = 0.82$, $r = 0.85$)		Execution time (in days) ($p = 0.4$, $r = 0.7$)	
	2^{16} procs	2^{19} procs	2^{16} procs	2^{19} procs
YOUNG	81.3	30.1	81.3	30.1
DALY	81.4	31.0	81.4	31.0
RFO	80.3	25.5	80.3	25.5
OPTIMALPREDICTION	65.9 (18%)	15.9 (38%)	69.7 (13%)	20.2 (21%)

Table 3: Job execution times for a Weibull distribution with shape parameter $k = 0.7$, and gains due to the fault predictor (with respect to the performance of RFO).

$C_p = C$	Execution time (in days) ($p = 0.82$, $r = 0.85$)		Execution time (in days) ($p = 0.4$, $r = 0.7$)	
	2^{16} procs	2^{19} procs	2^{16} procs	2^{19} procs
YOUNG	125.5	171.8	125.5	171.8
DALY	125.8	184.7	125.8	184.7
RFO	120.2	114.8	120.2	114.8
OPTIMALPREDICTION	75.9 (37%)	39.5 (66%)	83.0 (31%)	60.8 (47%)

Table 4: Job execution times for a Weibull distribution with shape parameter $k = 0.5$, and gains due to the fault predictor (with respect to the performance of RFO).

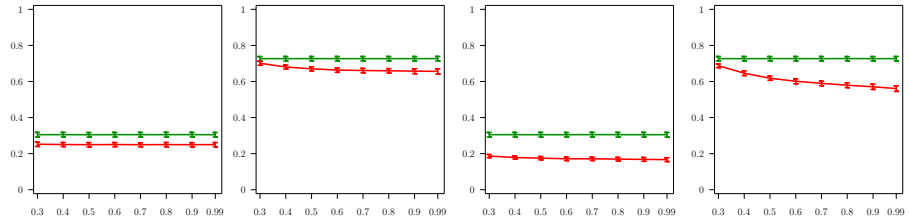
one with $N = 2^{16}$ processors (or a MTBF $\mu = 1,000\ min$) and a larger one with $N = 2^{19}$ processors (or a MTBF $\mu = 125\ min$). In both cases we study the impact of the predictor characteristics assuming a Weibull fault distribution with shape parameter 0.5 or 0.7.

In Figures 5 and 6, we fix the value of r (either $r = 0.4$ or $r = 0.8$) and we let p vary from 0.3 to 0.99. In the four plots, we observe that the precision has a minor impact on the waste, whether it is with a Weibull distribution of shape parameter 0.7 (Figure 5), or a Weibull distribution of shape parameter 0.5 (Figure 6). In Figures 7 and 8, we conduct the converse experiment and fix the value of p (either $p = 0.4$ or $p = 0.8$), letting r vary from 0.3 to 0.99. Here we observe that increasing the recall significantly improves performance.

Altogether we conclude that it is more important (for the design of future predictors) to focus on improving the recall r rather than the precision p, and our results can help quantify this statement. We provide an intuitive explanation as follows: unpredicted faults prove very harmful and heavily increase the waste, while unduly checkpointing due to false predictions turns out to induce a smaller overhead.

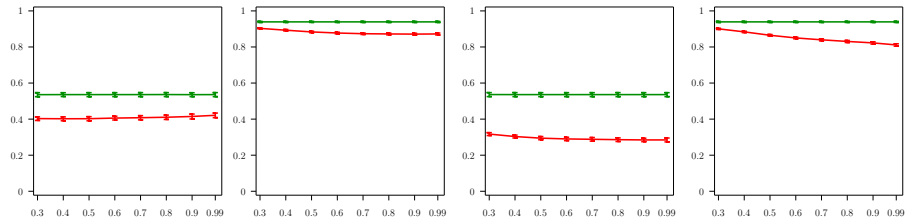
6 Related work

Considerable research has been devoted to fault prediction, using very different models (system log analysis [7], event-driven approach [4, 7, 8], support vector machines [6, 3], nearest neighbors [6], etc). In this section we give a brief overview of existing predictors, focusing on their characteristics rather than on the methods of prediction. For the sake of clarity, we sum up the characteristics



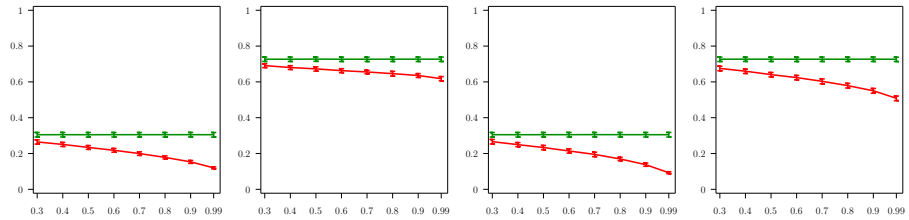
(a) $r = 0.4$, $N = 2^{16}$ (b) $r = 0.4$, $N = 2^{19}$ (c) $r = 0.8$, $N = 2^{16}$ (d) $r = 0.8$, $N = 2^{19}$

Figure 5: Waste (y-axis) as a function of the precision (x-axis) for a fixed recall ($r = 0.4$ and $r = 0.8$) and for a Weibull distribution of faults (with shape parameter $k = 0.7$).



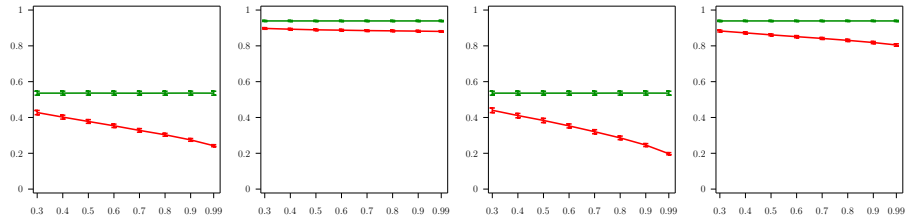
(a) $r = 0.4$, $N = 2^{16}$ (b) $r = 0.4$, $N = 2^{19}$ (c) $r = 0.8$, $N = 2^{16}$ (d) $r = 0.8$, $N = 2^{19}$

Figure 6: Waste (y-axis) as a function of the precision (x-axis) for a fixed recall ($r = 0.4$ and $r = 0.8$) and for a Weibull distribution of faults (with shape parameter $k = 0.5$).



(a) $p = 0.4$, $N = 2^{16}$ (b) $p = 0.4$, $N = 2^{19}$ (c) $p = 0.8$, $N = 2^{16}$ (d) $p = 0.8$, $N = 2^{19}$

Figure 7: Waste (y-axis) as a function of the recall (x-axis) for a fixed precision ($p = 0.4$ and $p = 0.8$) and for a Weibull distribution (k=0.7).



(a) $p = 0.4$, $N = 2^{16}$ (b) $p = 0.4$, $N = 2^{19}$ (c) $p = 0.8$, $N = 2^{16}$ (d) $p = 0.8$, $N = 2^{19}$

Figure 8: Waste (y-axis) as a function of the recall (x-axis) for a fixed precision ($p = 0.4$ and $p = 0.8$) and for a Weibull distribution (k=0.5).

Paper	Lead Time	Precision	Recall
[8]	300 s	40 %	70 %
[8]	600 s	35 %	60 %
[7]	2h	64.8 %	65.2 %
[7]	0 min	82.3 %	85.4 %
[4]	32 s	93 %	43 %
[5]	10s	92 %	40 %
[5]	60s	92 %	20 %
[5]	600s	92 %	3 %
[3]	NA	70 %	75 %
[6]	NA	20 %	30 %
[6]	NA	30 %	75 %
[6]	NA	40 %	90 %
[6]	NA	50 %	30 %
[6]	NA	60 %	85 %

Table 5: Comparative study of different parameters returned by some predictors.

of the different fault predictors encountered in Table 5.

The authors of [8] introduce the *lead time*, that is the duration between the time the prediction is made and the time the predicted fault is supposed to happen. This time should be sufficiently large to enable proactive actions. As already mentioned, the distribution of lead times is irrelevant. Indeed, only predictions whose lead time is greater than C_p, the time to take a proactive checkpoint, are meaningful. Predictions whose lead time is smaller than C_p, whenever they materialize as actual faults, should be classified as unpredicted faults; the predictor recall should be decreased accordingly.

The predictor of [8] is also able to locate where the predicted fault is supposed to strike. This additional characteristics has a negative impact on the precision (because a fault happening at the predicted time but not on the predicted location is classified as a non predicted fault; see the low value of p in Table 5). The authors of [8] state that fault localization has a positive impact on proactive checkpointing time in their context: instead of a full checkpoint costing $1,500$ seconds they can take a partial checkpoint costing only 12 seconds. This led us to introduce a different cost C_p for proactive checkpoints, that can be smaller than the cost C of regular checkpoints. Gainaru et al. [5] also stated that fault-localization could help decrease the checkpointing time. Their predictor also gives information on fault localization. They studied the impact of different lead times on the recall of their predictor. Papers [7] and [6] also considered lead times.

Most studies on fault prediction state that a proactive action must be taken right before the predicted fault, be it a checkpoint or a migration. However, we have shown in this paper that it is beneficial to ignore some predictions, namely when the predicted fault is announced to happen less than $\frac{C_p}{p}$ seconds after the last periodic checkpoint).

Gainaru et al. [5] studied the impact of prediction on the checkpointing period. Their computation of the total waste is not fully accurate and they do not provide any minimization analysis. Instead, they only propose to use Young's formula, replacing the MTBF by the mean-time of unpredicted faults.

They do not question whether all predictions should be taken into account.

Li et al. [23] considered the mathematical problem of when and how to migrate. In order to be able to use migration, they assumed that at any time 2% of the resources are available as spares. This allows them to conceive a Knapsack-based heuristic. Thanks to their algorithm, they were able to save 30% of the execution time compared to a heuristic that does not take the prediction into account, with a precision and recall of 70%, and with a maximum load of 0.7. In our study we do not consider that we have a batch of spare resources. We assume that after a downtime the resources that failed are once again available.

Note that some authors [7, 6] do not consider that their predictors predict the exact time of the fault. On the contrary, they consider a "prediction window" which is the time interval in which the predicted is supposed to occur. Because most papers focus on prediction windows of negligible length, we did not consider prediction windows in this study.

Finally, to the best of our knowledge, this work is the first to focus on the mathematical aspect of fault prediction, and to provide a model and a detailed analysis of the waste due to all three types of events (true and false predictions and unpredicted failures).

7 Conclusion

In this work we have studied the impact of fault prediction on periodic checkpointing. We started by revisiting the first-order approach by Young and Daly. We have performed a refined analysis leading to a better checkpointing period: T_{PRED} is slightly closer to the optimal period for Exponential distributions (the only case where the optimal is known), and leads to smaller execution times for Weibull distributions (as shown in Section 5.2).

Then we have extended the analysis to include fault predictions. We have established analytical conditions stating whether a fault prediction should be taken into account or not. More importantly, we have proven that the optimal approach is to never trust the predictor in the beginning of a regular period, and to always trust it in the end of the period; the cross-over point $\frac{C_p}{p}$ depends on the time to take a proactive checkpoint and on the precision of the predictor. This striking result is somewhat unexpected, as one might have envisioned more trust regimes, with several intermediate trust levels smoothly evolving from a "never trust" policy to an "always trust" one.

Through an extensive set of simulations involving faults following either an Exponential distribution law or a Weibull one, we have established the accuracy of the model, of its analysis, and of the predicted period (in the presence of a fault predictor). These simulations also show that even a not-so-good fault predictor can lead to quite a significant decrease in the application execution time. We have also shown that the most important characteristic of a fault predictor is its recall (the percentage of actually predicted faults) rather than its precision (the percentage of predictions that actually correspond to faults): *better safe than sorry*, or better prepare for a false event than miss an actual failure!

Altogether, the analytical model and the comprehensive results provided in this work enable to fully assess the impact of fault prediction on optimal checkpointing strategies. Future work will be devoted to refine the assessment

of the usefulness of prediction with trace-based failure and prediction logs from current large-scale supercomputers.

Acknowledgments. The authors are with Université de Lyon, France. Y. Robert is with the Institut Universitaire de France. This work was supported in part by the ANR *RESCUE* project. Finally, the authors would like to thank Thomas Hérault for useful comments.

References

[1] G. Zheng, X. Ni, L. Kale, A scalable double in-memory checkpoint and restart scheme towards exascale, in: Dependable Systems and Networks Workshops (DSN-W), 2012. doi:10.1109/DSNW.2012.6264677.

[2] K. Sato, A. Moody, K. Mohror, T. Gamblin, B. R. de Supinski, N. Maruyama, S. Matsuoka, Design and modeling of a non-blocking checkpointing system, in: SC'12 (the 2012 International Conference for High Performance Computing, Networking, Storage and Analysis), 2012.

[3] E. W. Fulp, G. A. Fink, J. N. Haack, Predicting computer system failures using support vector machines, in: Proceedings of the First USENIX conference on Analysis of system logs, USENIX Association, 2008.

[4] A. Gainaru, F. Cappello, W. Kramer, Taming of the shrew: Modeling the normal and faulty behavior of large-scale hpc systems, in: Proc. IPDPS'12, 2012.

[5] A. Gainaru, F. Cappello, W. Kramer, M. Snir, Fault prediction under the microscope - a closer look into hpc systems, in: SC'12 (the 2012 International Conference for High Performance Computing, Networking, Storage and Analysis), 2012.

[6] Y. Liang, Y. Zhang, H. Xiong, R. K. Sahoo, Failure prediction in ibm bluegene/l event logs, in: ICDM, 2007, pp. 583–588.

[7] L. Yu, Z. Zheng, Z. Lan, S. Coghlan, Practical online failure prediction for blue gene/p: Period-based vs event-driven, in: Dependable Systems and Networks Workshops (DSN-W), 2011, pp. 259–264. doi:10.1109/DSNW.2011.5958823.

[8] Z. Zheng, Z. Lan, R. Gupta, S. Coghlan, P. Beckman, A practical failure prediction with location and lead time for blue gene/p, in: Dependable Systems and Networks Workshops (DSN-W), 2010, pp. 15–22. doi:10.1109/DSNW.2010.5542627.

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

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

[11] N. Kolettis, N. D. Fulton, Software rejuvenation: Analysis, module and applications, in: FTCS '95, IEEE CS, Washington, DC, USA, 1995, p. 381.

[12] V. Castelli, R. E. Harper, P. Heidelberger, S. W. Hunter, K. S. Trivedi, K. Vaidyanathan, W. P. Zeggert, Proactive management of software aging, IBM J. Res. Dev. 45 (2) (2001) 311–332.

[13] J. Hong, S. Kim, Y. Cho, H. Yeom, T. Park, On the choice of checkpoint interval using memory usage profile and adaptive time series analysis, in: Proc. Pacific Rim Int. Symp. on Dependable Computing, IEEE Computer Society, 2001.

[14] J. Wingstrom, Overcoming The Difficulties Created By The Volatile Nature Of Desktop Grids Through Understanding, Prediction And Redundancy, Ph.D. thesis, University of Hawai'i at Manoa (2009).

[15] M. Bougeret, H. Casanova, M. Rabie, Y. Robert, F. Vivien, Checkpointing strategies for parallel jobs, in: Proceedings of SC'11, 2011.

[16] Y. Robert, F. Vivien, D. Zaidouni, On the complexity of scheduling checkpoints for computational workflowss, in: FTXS'2012, the Workshop on Fault-Tolerance for HPC at Extreme Scale, in conjunction with the 42nd Annual IEEE/IFIP Int. Conf. on Dependable Systems and Networks (DSN 2012), IEEE Computer Society Press, 2012.

[17] M. Mitzenmacher, E. Upfal, Probability and Computing: Randomized Algorithms and Probabilistic Analysis, Cambridge University Press, 2005.

[18] F. Cappello, H. Casanova, Y. Robert, Preventive migration vs. preventive checkpointing for extreme scale supercomputers, Parallel Processing Letters 21 (2) (2011) 111–132.

[19] K. Ferreira, J. Stearley, J. H. I. Laros, R. Oldfield, K. Pedretti, R. Brightwell, R. Riesen, P. G. Bridges, D. Arnold, Evaluating the Viability of Process Replication Reliability for Exascale Systems, in: Proceedings of the 2011 ACM/IEEE Conf. on Supercomputing, 2011.

[20] T. Heath, R. P. Martin, T. D. Nguyen, Improving cluster availability using workstation validation, SIGMETRICS Perf. Eval. Rev. 30 (1).

[21] B. Schroeder, G. A. Gibson, A large-scale study of failures in high-performance computing systems, in: Proc. of DSN, 2006, pp. 249–258.

[22] E. Heien, D. Kondo, A. Gainaru, D. LaPine, B. Kramer, F. Cappello, Modeling and tolerating heterogeneous failures in large parallel systems, in: Proc. ACM/IEEE Supercomputing'11, ACM Press, 2011.

[23] Y. Li, Z. Lan, P. Gujrati, X. Sun, Fault-aware runtime strategies for high-performance computing, Parallel and Distributed Systems, IEEE Transactions on 20 (4) (2009) 460–473.

[24] S. M. Ross, Introduction to Probability Models, Tenth Edition, Academic Press, 2009.

A

For the sake of completeness, we provide a proof of the following result:

Proposition 2. *Consider a platform comprising N components, and assume that the inter-arrival times of the faults on the components are independent and identically distributed random variables that follow an arbitrary probability law whose expectation is μ_{ind}. Then the expectation of the inter-arrival times of the faults on the whole platform is $\mu = \frac{\mu_{ind}}{N}$.*

Proof. Consider first a single component, say component number q. Let X_i, $i \geq 0$ denote the IID random variables for fault inter-arrival times on that component, with $\mathbb{E}(X_i) = \mu_{\text{ind}}$. Consider a fixed time bound F. Let $n_q(F)$ be the number of faults on the component until time F is exceeded. In other words, the $(n_q(F)-1)$-th fault is the last one to happen strictly before time F, and the $n_q(F)$-th fault is the first to happen at time F or after. By definition of $n_q(F)$, we have

$$\sum_{i=1}^{n_q(F)-1} X_i \leq F \leq \sum_{i=1}^{n_q(F)} X_i$$

Using Wald's equation [24, p. 486], with $n_q(F)$ as a stopping criterion, we derive:

$$(\mathbb{E}(n_q(F)) - 1)\mu_{\text{ind}} \leq F \leq \mathbb{E}(n_q(F))\,\mu_{\text{ind}}$$

and we obtain:

$$\lim_{F\to+\infty} \frac{\mathbb{E}(n_q(F))}{F} = \frac{1}{\mu_{\text{ind}}} \tag{18}$$

Consider now the whole platform, and let Y_i, $i \geq 0$ denote the IID random variables for fault inter-arrival times on the platform, with $\mathbb{E}(Y_i) = \mu$. Consider a fixed time bound F as before. Let $n(F)$ be the number of faults on the whole platform until time F is exceeded. With the same reasoning for the whole platform as for a single component, we derive:

$$\lim_{F\to+\infty} \frac{\mathbb{E}(n(F))}{F} = \frac{1}{\mu} \tag{19}$$

Now let $m_q(F)$ be the number of these faults that strike component number q. Of course we have $n(F) = \sum_{q=1}^{N} m_q(F)$. By definition, except for the component hit by the last failure, $m_q(F)+1$ is the number of failures on component q until time F is exceeded, hence $n_q(F) = m_q(F) + 1$ (and this number is $m_q(F) = n_q(F)$ on the component hit by the last failure). From Equation (18) again, we have for each component q:

$$\lim_{F\to+\infty} \frac{\mathbb{E}(m_q(F))}{F} = \frac{1}{\mu_{\text{ind}}}$$

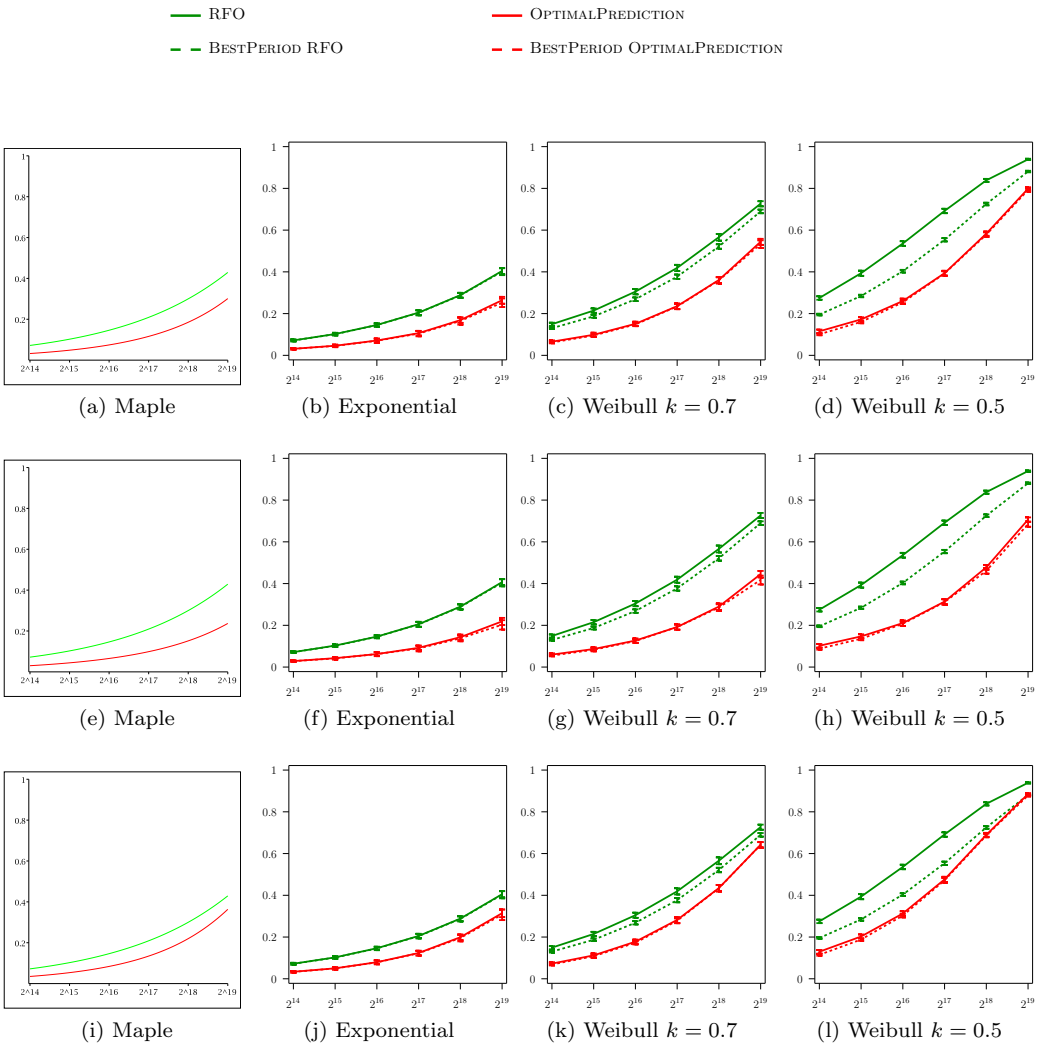
Since $n(F) = \sum_{q=1}^{N} m_q(F)$, we also have:

$$\lim_{F\to+\infty} \frac{\mathbb{E}(n(F))}{F} = \frac{N}{\mu_{\text{ind}}} \tag{20}$$

Equations (19) and (20) lead to the result. □

B

In this section, we provide results when false predictions are generated according to a uniform distribution.

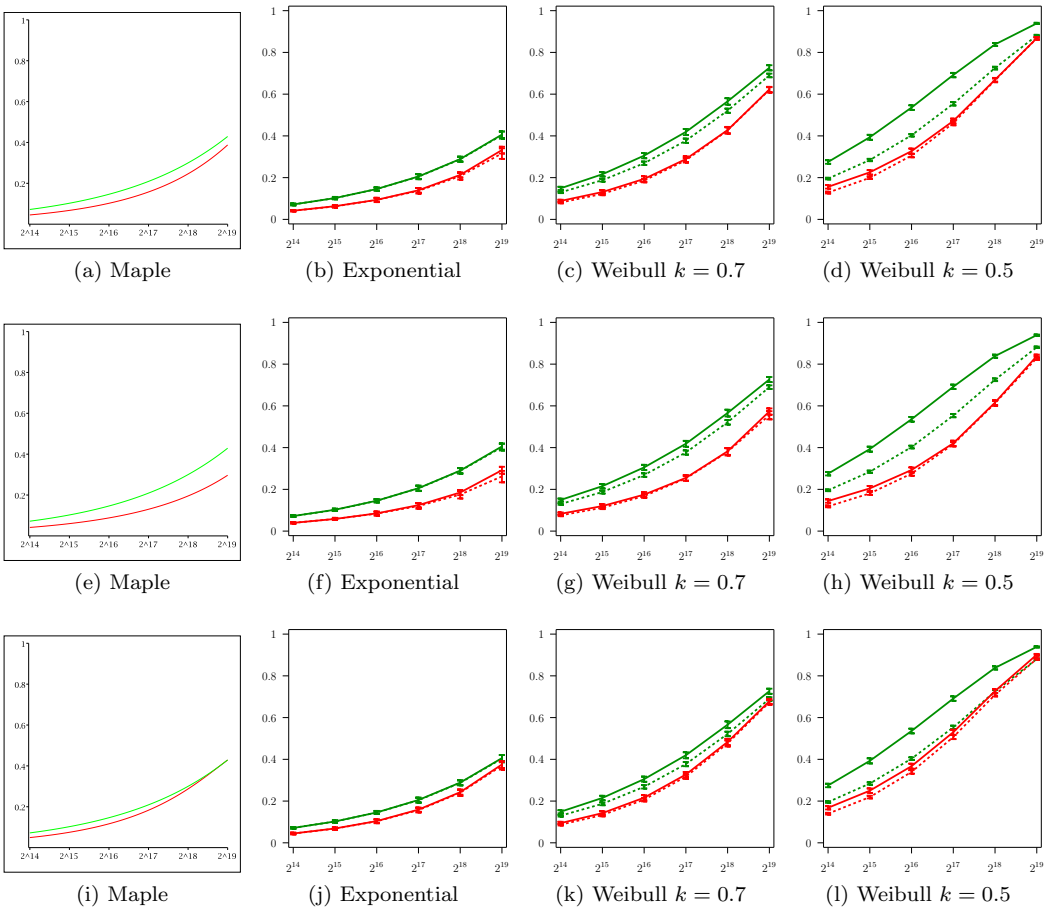


(a) Maple (b) Exponential (c) Weibull $k = 0.7$ (d) Weibull $k = 0.5$

(e) Maple (f) Exponential (g) Weibull $k = 0.7$ (h) Weibull $k = 0.5$

(i) Maple (j) Exponential (k) Weibull $k = 0.7$ (l) Weibull $k = 0.5$

Figure 9: Waste (y-axis) for the different heuristics as a function of the platform size (x-axis), with $p = 0.82$, $r = 0.85$, $C_p = C$ (first row), $C_p = 0.1C$ (second row), or $C_p = 2C$ (third row) and with a trace of false predictions parametrized by a uniform distribution.

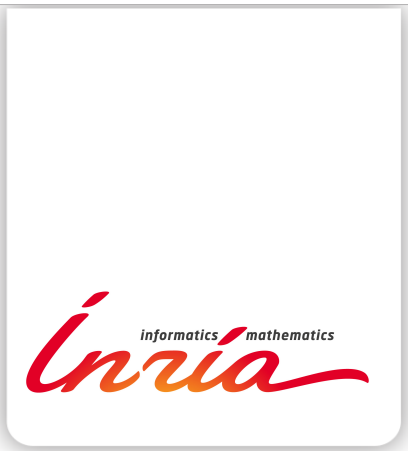


(a) Maple (b) Exponential (c) Weibull $k = 0.7$ (d) Weibull $k = 0.5$

(e) Maple (f) Exponential (g) Weibull $k = 0.7$ (h) Weibull $k = 0.5$

(i) Maple (j) Exponential (k) Weibull $k = 0.7$ (l) Weibull $k = 0.5$

Figure 10: Waste (y-axis) for the different heuristics as a function of the platform size (x-axis), with $p = 0.4$, $r = 0.7$, $C_p = C$ (first row), $C_p = 0.1C$ (second row), or $C_p = 2C$ (third row) and with a trace of false predictions parametrized by a uniform distribution..



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