Impact of fault prediction on checkpointing strategies

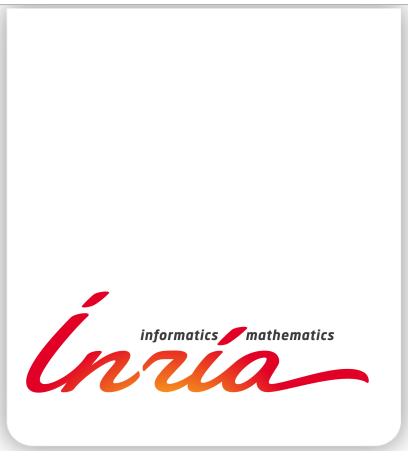
Guillaume Aupy, Yves Robert, Frédéric Vivien, Dounia Zaidouni

Impact of fault prediction on checkpointing strategies

Guillaume Aupy, Yves Robert, Frédéric Vivien, Dounia Zaidouni

RESEARCH REPORT
N° 8023
October 2012
Project-Team ROMA

ISSN 0249-6399 ISRN INRIA/RR--8023--FR+ENG



Impact of fault prediction on checkpointing strategies

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

Project-Team ROMA

Research Report n° 8023 — October 2012 — 26 pages

Abstract: This paper deals with the impact of fault prediction techniques on checkpointing strategies. We extend the classical analysis of Young and Daly in the presence of a fault prediction system, which is characterized by its recall and its precision, and which provides either exact or window-based time predictions. We succeed in deriving the optimal value of the checkpointing period (thereby minimizing the waste of resource usage due to checkpoint overhead) in all scenarios. These results allow to analytically assess the key parameters that impact the performance of fault predictors at very large scale. In addition, the results of this analytical evaluation are nicely corroborated by a comprehensive set of simulations, thereby demonstrating the validity of the model and the accuracy of the results.

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

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

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), et qui fournit des prédictions soit exactes soit avec des fenêtres. 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. Finalement, les résultats de ce papier sont confirmés par tout un ensemble de simulations démontrant ainsi la validité du modèle ainsi que la précision des résultats.

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

1 Introduction

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 (expectation of the) waste of resource usage due to checkpointing is easily approximated as $T_{\text{opt}} = \sqrt{2\mu C}$, or $T_{\text{opt}} = \sqrt{2(\mu + R)C}$ (where R is the duration of the recovery). The former expression is the well-known Young's formula [1], while the latter is due to Daly [2].

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. We deal with two problem instances, one where the predictor system provides exact dates for predicted events, and another where it only provides time windows during which events take place. The key contributions of this paper are the following: (i) The design of several checkpointing policies, their analysis, and a new formula for the checkpointing period that extends Young's and Daly's to take predictions into account; (ii) The analytical characterization of the best policy for each set of parameters; (iii) The validation of the theoretical results via extensive simulations, for both Exponential and Weibull failure distributions; (iv) The demonstration that even a poor predictor can lead to a significant reduction of application execution time; and (v) The demonstration that recall is far more important than precision, hence giving insight into the design of future predictors.

The rest of the paper is organized as follows. We first detail the framework in Section 2. We deal with exact date predictions in Section 3, and with time-window based predictions in Section 4. Section 5 is devoted to simulations. 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. 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 $\mu = \frac{\mu_{ind}}{N}$. Checkpoints are taken at regular intervals, or periods, of length T. We use C, D, and R for the duration of the checkpoint, downtime and recovery (respectively). We must enforce that $C \leq T$, and useful work is done only

during $T - C$ units of time for every period of length T, if no fault occurs. The *waste* due to checkpointing in a fault-free execution is $\text{WASTE} = \frac{C}{T}$. In the following, the *waste* always denote the fraction of time that the platform is not doing useful work.

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. 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 $p = \frac{True_P}{True_P + False_P}$.

2.3 Fault rates

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:

- $\frac{1-r}{\mu} = \frac{1}{\mu_{NP}}$ (here, $1 - r$ is the fraction of faults that are unpredicted);
- $\frac{r}{\mu} = \frac{p}{\mu_P}$ (here, r is the fraction of faults that are predicted, and p is the fraction of fault predictions that are correct);
- $\frac{1}{\mu_e} = \frac{1}{\mu_P} + \frac{1}{\mu_{NP}}$ (here, events are either predicted (true or false), or not).

3 Predictor with exact event dates

In this section, we present an analytical model to assess the impact of prediction on periodic checkpointing strategies. We consider the case where the predictor is able to provide exact prediction dates, and to generate such predictions at least C seconds in advance, so that a checkpoint can indeed be taken before the event (otherwise the prediction cannot be used, because there is not enough time to take proactive actions). We consider the following algorithm:
(1) While no fault prediction is available, checkpoints are taken periodically with period T;
(2) When a fault is predicted, we decide whether to take the prediction into account or not. This decision is randomly taken: with probability q, we trust the predictor and take the prediction into account, and, with probability $1 - q$, we ignore the prediction. If we take the prediction into account, there are two cases. If we have enough time before the prediction date, we take a checkpoint as late as possible, i.e., so that it completes right at the time where the fault is

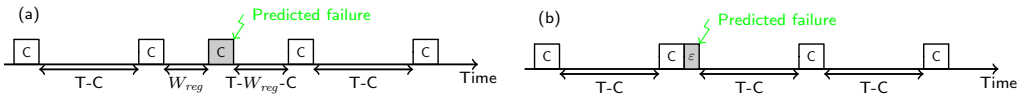


Figure 1: Strategy: (a) Whenever there is enough time to take a checkpoint, the algorithm takes one just before the predicted failure; (b) otherwise, it just executes some extra work.

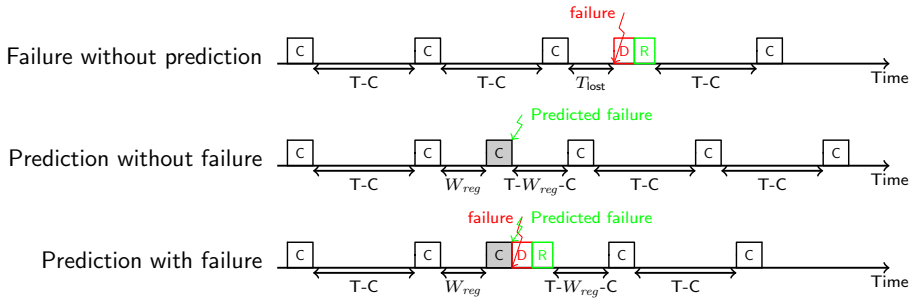


Figure 2: Actions taken when the predictor provides exact dates.

predicted to happen. After the checkpoint, we then complete the execution of the period (see Figure 1(a)). Otherwise, if we do not have enough time to take an extra checkpoint (because we are already checkpointing), then we do some extra work during ε seconds (see Figure 1(b)). We account for this work as idle time in the expression of the waste, to ease the analysis. Our expression of the waste is thus an upper bound.

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, μ, r, and p.

3.1 Computing the waste

Our goal in this section is to compute a formula for the expected waste. Recall that the waste is the fraction of time that the processors do not perform useful computations, either because they are checkpointing, or because a failure has struck. There are four different sources of waste (see Figure 2):
(1) **Checkpoints:** During a fault-free execution, the fraction of resources used in checkpointing is $\frac{C}{T}$.
(2) **Unpredicted faults:** This overhead occurs each time a unpredicted fault strikes, that is, on average, once every μ_{NP} seconds. The time wasted because of the unpredicted fault is then the time elapsed between the last checkpoint and the fault, plus the downtime and the time needed for the recovery. The expectation of the time elapsed between the last checkpoint and the fault is

equal to half the period of checkpoints, because the time where the fault hits the system is independent of the checkpointing algorithm. Finally, the waste due to unpredicted faults is: $\frac{1}{\mu_{NP}}\left[\frac{T}{2}+D+R\right]$.
(3) **Predictions taken into account:** Now we have to compute the execution overhead due to a prediction which we trust (hence we checkpoint just before its date). This overhead occurs each time a prediction is made by the predictor, that is, on average, once every μ_P seconds, and that we decide to trust it, with probability q. If the predicted event is an actual fault, we waste $C+D+R$ seconds: we waste $D+R$ seconds because the predicted event corresponds to an actual fault, and if we have enough time before the prediction date, we waste C seconds because we take an extra checkpoint as late as possible before the prediction date (see Figure 1(a)). Note that if we do not have enough time to take an extra checkpoint (see Figure 1(b)), we overestimate the waste as C seconds. If the predicted event is not an actual fault, we waste C seconds. An actual fault occurs with probability p, and a false prediction is made with probability $(1-p)$. Averaging with these probabilities, we waste an expected amount of $[p(C+D+R)+(1-p)C]$ seconds. Finally, the corresponding overhead is $\frac{1}{\mu_P}q\left[p(C+D+R)+(1-p)C\right]$.
(4) **Ignored predictions:** The final source of waste is for predictions that we do not trust. This overhead occurs each time a prediction is made by the predictor, that is, on average, once every μ_P seconds, and that we decide to trust it, with probability $1-q$. If the predicted event corresponds to an actual fault, we waste $(\frac{T}{2}+D+R)$ seconds (as for an unpredicted fault). Otherwise there is no fault and we took no extra checkpoint, and thus we lose nothing. An actual fault occurs with a probability p. The corresponding overhead is $\frac{1}{\mu_P}(1-q)\left[p(\frac{T}{2}+D+R)+(1-p)0\right]$.
Summing up the overhead over the four different sources, and after simplification, we obtain the following equation for the waste:

$$\textsc{Waste} = \frac{C}{T} + \frac{1}{\mu}\left[(1-rq)\frac{T}{2} + D + R + \frac{qr}{p}C\right] \tag{1}$$

3.2 Validity of the analysis

Equation (1) is accurate only when two events (an event being a prediction (true or false) or an unpredicted fault) do not take place within the same period. To ensure that this condition is met with a high probability, we bound the length of the period: without predictions, or when predictions are not taken into account, we enforce the condition $T < \alpha\mu$; otherwise, with predictions, we enforce the condition $T < \alpha\mu_e$. Here, α is some tuning parameter chosen as follows. The number of events during a period of length T can be modeled as a Poisson process of parameter $\beta = \frac{T}{\mu}$ (without prediction) or $\beta = \frac{T}{\mu_e}$ (with prediction). 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.

In addition to the previous constraint, we must always enforce the condition $C \leq T$, by construction of the periodic checkpointing policy. Finally, the optimal waste may never exceed 1; when the waste is equal to 1, the application no longer makes any progress.

3.3 Waste minimization

We differentiate twice Equation (1) with respect to T:

$$\textsc{Waste}'(T) = \frac{-C}{T^2} + \frac{1}{\mu}\left[(1-rq)\frac{1}{2}\right]$$

$$\textsc{Waste}''(T) = \frac{2C}{T^3} > 0$$

We obtain that $\textsc{Waste}''(T)$ is strictly positive, hence $\textsc{Waste}(T)$ is a convex function of T and admits a unique minimum on its domain. We also compute $T_{\text{extr}}^{\{q\}}$, the extremum value of T that is the unique zero of the function $\textsc{Waste}'(T)$, as $T_{\text{extr}}^{\{q\}} = \sqrt{\frac{2\mu C}{1-rq}}$. Note that this Equation makes sense even when $1 - rq = 0$. Indeed this would mean that both $r = 1$ and $q = 1$: the predictor predicts every fault, and we take proactive action for each one of them, there should never be any periodic checkpointing! Finally, note that $T_{\text{extr}}^{\{q\}}$ may well not belong to the admissible domain $[C, \alpha\mu_e]$.

The optimal waste $\textsc{Waste}_{\text{opt}}$ is determined via the following case analysis. We rewrite the waste as an affine function of q:

$$\textsc{Waste}(q) = \frac{rq}{\mu}\left(\frac{C}{p} - \frac{T}{2}\right) + \left(\frac{C}{T} + \frac{T}{2\mu} + \frac{D+R}{\mu}\right)$$

For any value of T, we deduce that $\textsc{Waste}(q)$ is minimized either for $q = 0$ or for $q = 1$. This (somewhat unexpected) conclusion is that the predictor should sometimes be always trusted, and sometimes never, but no in-between value for q will do a better job. Thus we need to minimize the two functions $\textsc{Waste}^{\{0\}}$ and $\textsc{Waste}^{\{1\}}$ over the domain of admissible values for T, and to retain the best result.

We have $\textsc{Waste}^{\{0\}}(T) = \frac{C}{T} + \frac{1}{\mu}\left[\frac{T}{2} + D + R\right]$. We recognize here the waste function of Young [1] and write $\textsc{Waste}_Y = \frac{C}{T} + \frac{1}{\mu}\left[\frac{T}{2} + D + R\right]$. The function $\textsc{Waste}_Y(T)$ is a convex function and reaches its minimum for T_{Y} in the interval $[C, \alpha\mu]$:

- If $(C < T_{\text{extr}}^{\{0\}} < \alpha\mu)$: $T_{\text{Y}} = T_{\text{extr}}^{\{0\}} = \sqrt{2\mu C}$
- If $(T_{\text{extr}}^{\{0\}} < C)$: $T_{\text{Y}} = C$
- If $(T_{\text{extr}}^{\{0\}} \geq \alpha\mu)$: $T_{\text{Y}} = \alpha\mu$

Thus, $\textsc{Waste}_Y$ $(= \textsc{Waste}^{\{0\}})$ is minimized for:

$$T_{\text{Y}} = \min\left(\alpha\mu, \max(\sqrt{2\mu C}, C)\right)$$

Similarly, we have: $\textsc{Waste}^{\{1\}}(T) = \frac{C}{T} + \frac{1}{\mu}\left[(1-r)\frac{T}{2} + D + R + \frac{r}{p}C\right]$. The function $\textsc{Waste}^{\{1\}}(T)$ is a convex function and reaches its minimum for T_1 in the interval $[C, \alpha\mu_e]$.

- If $(C < T_{\text{extr}}^{\{1\}} < \alpha\mu_e)$: $T_1 = T_{\text{extr}}^{\{1\}} = \sqrt{\frac{2\mu C}{1-r}}$
- If $(T_{\text{extr}}^{\{1\}} < C)$: $T_1 = C$
- If $(T_{\text{extr}}^{\{1\}} \geq \alpha\mu_e)$: $T_1 = \alpha\mu_e$

Thus, $\text{WASTE}^{\{1\}}$ is minimized for:

$$T_1 = \min\left(\alpha\mu_e, \max(\sqrt{\frac{2\mu C}{1-r}}, C)\right)$$

Finally, the optimal waste is:

$$\text{WASTE}_{\text{opt}} = \min\left(\text{WASTE}_Y(T_{\text{Y}}), \text{WASTE}^{\{1\}}(T_1)\right)$$

3.4 Prediction and preventive migration

In this section, we make a short digression and briefly present an analytical model to assess the impact of prediction and preventive migration on periodic checkpointing strategies. As before, we consider a predictor that is able to predict exactly when faults happen, and to generate these predictions at least C seconds before the event dates.

The idea of migration consists in moving a task for execution on another node, when a fault is predicted to happen on the current node in the near future. Note that the faulty node can later be replaced, in case of a hardware fault, or software rejuvenation can be used in case of a software fault. We consider the following algorithm, which is very similar to that used in Section 3.1:

1. When no fault prediction is available, checkpoints are taken periodically with period T.
2. When a fault is predicted, we decide whether to execute the migration or not. The decision is a random one: with probability q we trust the predictor and do the migration and, with probability 1-q, we ignore the prediction. If we take the prediction into account, we execute the migration as late as possible, so that it completes right at the time when the fault is predicted to happen.

As before, we have four different sources of waste. Summing the overhead of the execution of these different sources, we obtain the following equation for the waste (where M is the duration of a migration):

$$\begin{aligned}\text{WASTE} = &\frac{C}{T} \\ &+ \frac{1}{\mu_{NP}}\left[\frac{T}{2} + D + R\right] \\ &+ \frac{1}{\mu_P} q\left[p(M) + (1-p)M\right] \\ &+ \frac{1}{\mu_P}(1-q)\left[p(\frac{T}{2} + D + R) + (1-p)0\right]\end{aligned}$$

After simplification, we get:

$$\textsc{Waste} = \frac{C}{T} + \frac{1}{\mu}\left[(1-rq)\left(\frac{T}{2}+D+R\right)+\frac{qr}{p}M\right] \tag{3}$$

Equation (3) is very similar to Equation (1), and the minimization of the waste proceeds exactly as in Section 3.3. In a nutshell, $\textsc{Waste}(T)$ is again a convex function and admits a unique minimum over its domain $[C, \alpha\mu_e]$, the unique zero of the derivative has the same value $T_{\text{extr}}^{\{q\}} = \sqrt{\frac{2\mu C}{1-rq}}$, and for any value of T, the waste is minimized for either $q = 0$ or $q = 1$. We conduct the very same case analysis as in Section 3.3.

4 Predictor with a prediction window

In the previous section, we supposed that the predictor was able to predict exactly when faults will strike. Here, we suppose (maybe more realistically) that the predictor gives a *prediction window*, that is an interval of time of length I during which the predicted fault is likely to happen. As before in Section 3: (i) We suppose that we have enough time to checkpoint before the beginning of the prediction window; and (ii) When a prediction is made, we enforce that the scheduling algorithm has the choice either to take or not to take this prediction into account, with probability q.

We start with a description of the strategies that can be used, depending upon the (relative) length I of the prediction window. Let us define two *modes* for the scheduling algorithm:

Regular: This is the mode used when no fault prediction is available, or when a prediction is available but we decide to ignore it (with probability $1-q$). In regular mode, we use periodic checkpointing with period T_{R}. Intuitively, T_{R} corresponds to the checkpointing period T of Section 3.

Proactive: This is the mode used when a fault prediction is available and we decide to trust it, a decision taken with probability q. Consider such a trusted prediction made with the prediction window $[t_0, t_0 + I]$. Several strategies can be envisioned:

(1) Instant, for *Instantaneous*– The first strategy is to ignore the time-window and to execute the same algorithm as if the predictor had given an exact date prediction at time t_0. Just as described in Section 3, the algorithm interrupts the current period (of scheduled length T_{R}), checkpoints during the interval $[t_0 - C, t_0]$, and then returns to regular mode: at time t_0, it resumes the work needed to complete the interrupted period of the regular mode.

(2) NoCkptI, for *No checkpoint during prediction window*– The second strategy is intended for a short prediction window: instead of ignoring it, we acknowledge it, but make the decision not to checkpoint during it. As in the first strategy, the algorithm interrupts the current period (of scheduled length T_{R}), and checkpoints during the interval $[t_0 - C, t_0]$. But here, we return to regular mode only at time $t_0 + I$, where we resume the work needed to complete the interrupted period of the regular mode. During the whole length of the time-window, we execute work without checkpointing, at the risk of losing work if a fault indeed strikes. But for a small value of I, it may not be worthwhile to checkpoint during the prediction window (if at all possible, since there is no choice if $I < C$).

(3) WithCkptI, for *With checkpoints during prediction window*– The third strategy is intended for a longer prediction window and assumes that $C \leq I$: the algorithm interrupts the current period (of scheduled length T_{R}), and checkpoints during the interval $[t_0 - C, t_0]$, but now decides to take several checkpoints during the prediction window. The period T_{P} of these checkpoints in proactive mode will presumably be shorter than T_{R}, to take into account the higher fault probability. To simplify the presentation, we use an integer number of periods of length T_{P} within the prediction window. In the following, we analytically compute the optimal number of such periods. But we take at least one period here, hence one checkpoint, which implies $C \leq I$. We return to regular mode either right after the fault strikes within the time window $[t_0, t_0 + I]$, or at time $t_0 + I$ if no actual fault happens within this window. Then, we resume the work needed to complete the interrupted period of the regular mode. The third strategy is the most complex to describe, and the complete behavior of the scheduling algorithm is shown in Algorithm 1.

Note that for all strategies, exactly as in Section 3, we insert some additional work for the particular case where there is not enough time to take a checkpoint before entering proactive mode (because a checkpoint for the regular mode is currently on-going, see Figure 1(b)). We account for this work as idle time in the expression of the waste, to ease the analysis. Our expression of the waste is thus an upper bound.

Algorithm 1: WithCkptI.

if *fault happens* **then**
  After downtime, execute recovery;
  Enter *regular* mode;
if *in* proactive *mode for a time greater than or equal to* I **then**
  Switch to *regular* mode
if *Prediction made with interval* $[t, t + I]$ ***and*** *prediction taken into account* **then**
  Let t_C be the date of the last checkpoint under *regular* mode to start no later than $t - C$;
  if $t_C + C < t - C$ **then** (enough time for an extra checkpoint)
    Take a checkpoint starting at time $t - C$
  else (no time for the extra checkpoint)
    Work in the time interval $[t_C + C, t]$
  $W_{reg} \leftarrow \max(0, t - C - (t_C + C))$;
  Switch to *proactive* mode at time t;
while *in* regular *mode and no predictions are made and no faults happen* **do**
  Work for a time T_{R}-W_{reg}-C and then checkpoint;
  $W_{reg} \leftarrow 0$;
while *in* proactive *mode and no faults happen* **do**
  Work for a time T_{P}-C and then checkpoint;

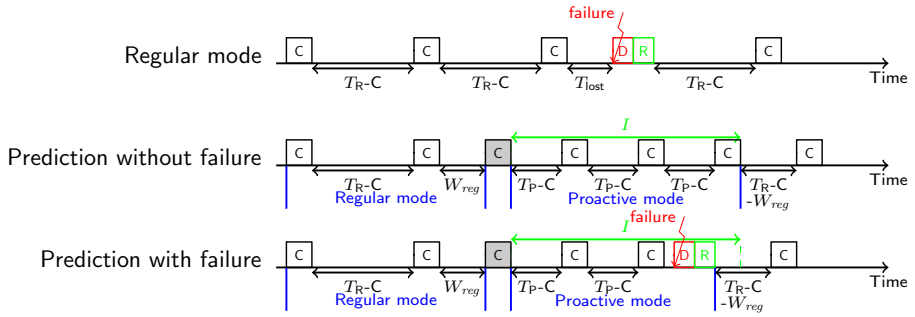


Figure 3: Outline of Algorithm 1 (strategy WithCkptI).

4.1 Waste for strategy WithCkptI

In this section we focus on computing the waste of WithCkptI, the most complex strategy. We first compute the fraction of time spent in the *regular* mode (checkpointing with period T_R) and the fraction of time spent in the *proactive* mode (checkpointing with period T_P). Let I' be the average time spent in the *proactive* mode. When a prediction is made, we may choose to ignore it, which happens with probability $1-q$. In this case, the algorithm stays in regular mode and does not spend any time in the proactive mode. With probability q, we may decide to take the prediction into account. In this case, if the prediction is a false positive event (no actual fault strikes), which happens with probability $1-p$, then the algorithm spends I units of time in the proactive mode. Otherwise, if the prediction is a true positive event (an actual fault hits the system), which happens with probability p, then the algorithm spends an average of $\mathbb{E}_I^{(f)}$ in the proactive mode. Here $\mathbb{E}_I^{(f)}$ is the expectation of the time elapsed between the beginning of the prediction window and the time when a fault happens, knowing that a fault happens in the prediction window. Note that if faults are uniformly distributed across the prediction window, then $\mathbb{E}_I^{(f)} = \frac{I}{2}$. Altogether, we obtain $I' = q\left((1-p)I + p\mathbb{E}_I^{(f)}\right)$. Each time there is a prediction, that is, on the average, every μ_P seconds, the algorithm spends a time I' in the proactive mode. Therefore, Algorithm 1 spends a fraction of time $\frac{I'}{\mu_P}$ in the proactive mode, and a fraction of time $1-\frac{I'}{\mu_P}$ in the regular mode.

As in Section 3, we assume that there is a single event of any type (either a prediction (true or false), or an unpredicted failure) within each interval under study. The condition $T \leq \alpha\mu_e$ then becomes $T_R + I \leq \alpha\mu_e$, since $T_R + I$ is the longest time interval considered in the analysis of Algorithm 1. We now identify the four different sources of waste, and we analyze their respective costs:

(1) **Waste due to periodic checkpointing.** There are two cases, depending upon the mode of Algorithm 1:

(a) **Regular mode.** In this mode, we take periodic checkpoints. We take a checkpoint of size C each time the algorithm has processed work for a time $T_R - C$ in the regular mode. This remains true if, after spending some time in the regular mode, the algorithm switches to the proactive mode, and later switches back to the regular mode. This behavior is enforced by recording the amount of work performed under the regular mode (variable W_{reg}, at line 12 of Algorithm 1), and by taking this value into account at line 15. Given the

fraction of time that Algorithm 1 spends in the regular mode, this source of waste has a total cost of $\left(1 - \frac{I'}{\mu_P}\right) \frac{C}{T_R}$.

(b) **Proactive mode.** In this mode, we take a checkpoint of size C each time the algorithm has processed work for a time $T_P - C$. If no fault happens while the algorithm is in the proactive mode, then the algorithm stays exactly a time I in this mode (thanks to the condition at line 4). The waste due to the periodic checkpointing is then exactly $\frac{C}{T_P}$ (because T_P divides I). If a fault happens while the algorithm is in proactive mode, then, the expectation of the waste due to the periodic checkpointing is upper-bounded by the same quantity $\frac{C}{T_P}$ (this is an over-approximation of the waste in that case). Overall, taking into account the fraction of time Algorithm 1 is in the proactive mode, the cost of this source of waste is $\frac{I'}{\mu_P} \frac{C}{T_P}$.

(2) **Waste incurred when switching to the proactive mode.** Each time we take into account a prediction (which happens with probability q on average every μ_P units of time), we start by doing one preliminary checkpoint if we have the time to do so (line 9). If we do not have the time to take an additional checkpoint, the algorithm do not do any processing for a duration of at most C (line 11). In both cases, the wasted time is at most C and this happens once every $\frac{\mu_P}{q}$. Hence, switching from the regular mode to the proactive one induces a waste of at most $\frac{q}{\mu_P} C$.

(3) **Waste due to predicted faults.** Predicted faults happen with frequency $\frac{p}{\mu_P}$. As we may choose to ignore a prediction, there are still two cases depending on the mode of the algorithm at the time of the fault:

(a) **Regular mode.** If the algorithm is in regular mode when a predicted fault hits, this means that we have chosen to ignore the prediction, a decision taken with probability $(1 - q)$. The time wasted because of the predicted fault is then the time elapsed between the last checkpoint and the fault, plus the downtime and the time needed for the recovery. The expectation of the time elapsed between the last checkpoint and the fault is equal to half the period of checkpoints, because the time where the fault hits the system is independent of the checkpointing algorithm. Therefore, the waste due to predicted faults hitting the system in regular mode is $\frac{p(1-q)}{\mu_P} \left(\frac{T_R}{2} + D + R\right)$.

(b) **Proactive mode.** If the algorithm is in proactive mode when a fault hits, then we have chosen to take the prediction into account, a decision that is taken with probability q. The time wasted because of the predicted fault is then, in addition to the downtime and the time needed for the recovery, the time elapsed between the last checkpoint and the fault or, if no checkpoint had already been taken in the proactive mode, the time elapsed between the start of the proactive mode and the fault. Here, we can no longer assume that the time the fault hits the system is independent of the checkpointing date. This is because the proactive mode starts exactly at the beginning of the prediction window. Let T_{lost} denote the computation time elapsed between the latest of the beginning of the proactive mode and the last checkpoint, and the fault date. Then the expectation of T_{lost} depends on the distribution of the fault date in the prediction window. However, we know that whatever the distribution, $T_{\text{lost}} \leq T_P$. Therefore we over approximate the waste in that case by $\frac{qp}{\mu_P} (T_P + D + R)$.

(4) **Waste due to unpredicted faults.** There are again two cases, depending

upon the mode of the algorithm at the time the fault hits the system:
(a) **Regular mode.** In this mode the work done is periodically checkpointed with period T_{R}. The time wasted because of an unpredicted fault is then the time elapsed between the last checkpoint and the fault, plus the downtime and the time needed for the recovery. As before, the expectation of this value is $T_{\mathrm{lost}} = \frac{T_{\mathrm{R}}}{2}$. An unexpected fault hits the system once every μ_{NP} seconds on the average. Taking into account the fraction of the time the algorithm is in regular mode, the waste due to unpredicted faults hitting the system in regular mode is $\left(1 - \frac{I'}{\mu_P}\right) \frac{1}{\mu_{NP}} \left(\frac{T_{\mathrm{R}}}{2} + D + R\right)$.
(b) **Proactive mode.** Because of the assumption that a single event takes place within a time-interval, we do not consider the very unlikely case where a unpredicted fault strikes during a prediction window. This amounts to assume that $\frac{I'}{\mu_P} \frac{1}{\mu_{NP}} (T_{\mathrm{P}} + D + R)$ is negligible.

We gather the expressions of the six different types of waste and simplify to obtain the formula of the overall waste:

$$\begin{aligned}\textsc{Waste}_{\textsc{WithCkptI}} = \quad & \left(\left(1 - \frac{I'}{\mu_P}\right)\frac{1}{T_{\mathrm{R}}} + \frac{I'}{\mu_P}\frac{1}{T_{\mathrm{P}}} + \frac{q}{\mu_P}\right) C + \frac{p(1-q)}{\mu_P}\frac{T_{\mathrm{R}}}{2} \\ & + \frac{pq}{\mu_P} T_{\mathrm{P}} + \left(1 - \frac{I'}{\mu_P}\right)\frac{1}{\mu_{NP}}\frac{T_{\mathrm{R}}}{2} \\ & + \left(\frac{p}{\mu_P} + \left(1 - \frac{I'}{\mu_P}\right)\frac{1}{\mu_{NP}}\right)(D + R) \end{aligned} \tag{4}$$

4.2 Waste of the other strategies

The waste of the first strategy (*Instantaneous*) is very close to the one given in Equation (1). The difference lies in T_{lost}, the expectation of the work lost when a fault is predicted and the prediction is taken into account. When a prediction is taken into account and the predicted event is an actual fault, the waste in Equation (1) was $\frac{qp}{\mu_P}(C + D + R)$. Because the prediction was exact, T_{lost} was equal to 0. However in our new Equation, the waste for this part is now $\frac{qp}{\mu_P}(C + T_{\mathrm{lost}} + D + R)$. On average, the fault occurs after a time $\mathbb{E}_I^{(f)}$. However, because we do not know the relation between $\mathbb{E}_I^{(f)}$ and T_{R}, then T_{lost} has expectation $\frac{T_{\mathrm{R}}}{2}$ if $\frac{T_{\mathrm{R}}}{2} \leq \mathbb{E}_I^{(f)}$. The new waste is then:

$$\textsc{Waste}_{\textsc{Instant}} = \frac{C}{T_{\mathrm{R}}} + \frac{1}{\mu}\left[(1 - rq)\frac{T_{\mathrm{R}}}{2} + D + R + \frac{qr}{p}C + qr \min\left(\mathbb{E}_I^{(f)}, \frac{T_{\mathrm{R}}}{2}\right)\right] \tag{5}$$

As for the second strategy (*No checkpoint during prediction window*), we do no longer incur the waste due to checkpointing in proactive mode as we no longer checkpoint in proactive mode. Furthermore, the value of T_{lost} in proactive mode becomes $\mathbb{E}_I^{(f)}$ instead of T_{P}. Consequently, the total waste when there is no checkpoint during the proactive mode is:

$$\textsc{Waste}_{\text{noCkpt}} = \left(1 - \frac{I'}{\mu_P}\right)\frac{C}{T_{\text{R}}} + \frac{q}{\mu_P}C + \frac{p(1-q)}{\mu_P}\left(\frac{T_{\text{R}}}{2} + D + R\right) + \frac{pq}{\mu_P}\left(\mathbb{E}_I^{(f)} + D + R\right) + \left(1 - \frac{I'}{\mu_P}\right)\frac{1}{\mu_{NP}}\left(\frac{T_{\text{R}}}{2} + D + R\right)$$

which we rewrite as

$$\textsc{Waste}_{\text{NoCkptI}} = \left(\left(1 - \frac{I'}{\mu_P}\right)\frac{1}{T_{\text{R}}} + \frac{q}{\mu_P}\right)C + \frac{p(1-q)}{\mu_P}\frac{T_{\text{R}}}{2} + \frac{pq}{\mu_P}\mathbb{E}_I^{(f)} + \left(1 - \frac{I'}{\mu_P}\right)\frac{1}{\mu_{NP}}\frac{T_{\text{R}}}{2} + \left(\frac{p}{\mu_P} + \left(1 - \frac{I'}{\mu_P}\right)\frac{1}{\mu_{NP}}\right)(D+R) \tag{6}$$

Note that when $I = 0$, Instant and NoCkptI are identical. Indeed, we have $\mathbb{E}_I^{(f)} = 0$ if $I = 0$, and we check that Equations (5) and (6) are identical in that case.

4.3 Waste minimization

In this section we aim at minimizing the waste of the three strategies, and then we find conditions to characterize which one is better. Recall that :

$$I' = q\left((1-p)I + p\mathbb{E}_I^{(f)}\right)$$

WithCkptI. In order to compute the optimal value for T_{P}, let us find the portion of the waste that depends on T_{P}:

$$\textsc{Waste}_{T_{\text{P}}} = \frac{rq}{\mu}\left(\frac{(1-p)I + p\mathbb{E}_I^{(f)}}{p}\frac{C}{T_{\text{P}}} + T_{\text{P}}\right)$$

As we can see, the optimal value for T_{P} is independent from q, but also from μ. The optimal value for T_{P} is thus:

$$T_{\text{P}}^{\text{extr}} = \sqrt{\frac{(1-p)I + p\mathbb{E}_I^{(f)}}{p}C} \tag{7}$$

However, for our algorithm to be correct, we want $\frac{I}{T_{\text{P}}} \in \mathbb{N}$ (the interval I is partitioned in k intervals of length T_{P}, for some integer k). We choose $T_{\text{P}}^{\text{opt}}$ equal to either $\frac{I}{\left\lfloor \frac{I}{T_{\text{P}}^{\text{extr}}} \right\rfloor}$ or $\frac{I}{\left\lfloor \frac{I}{T_{\text{P}}^{\text{extr}}} \right\rfloor + 1}$, depending on the value that minimizes $\textsc{Waste}_{T_{\text{P}}}$. Note that we also have the constraint $T_{\text{P}}^{\text{opt}} \geq C$, hence if both values are lower than C, then $T_{\text{P}}^{\text{opt}} = C$.

Now that we know that $T_{\text{P}}^{\text{opt}}$ is independent from both q and T_{R}, we can see the waste in Equation (4) as a function of two variables. One can see from Equation (4) that the waste is an affine function of q. This means that the

minimum is always reached for either $q = 0$ or $q = 1$. We now consider the two functions $\text{WASTE}_{\text{withCkpt}\{q=0\}}$ and $\text{WASTE}_{\text{withCkpt}\{q=1\}}$ in order to minimize them with respect to T_{R}. First we have:

$$\text{WASTE}_{\text{withCkpt}\{q=0\}} = \frac{C}{T_{\text{R}}} + \frac{1}{\mu}\left(\frac{T_{\text{R}}}{2} + D + R\right) \tag{8}$$

As expected, this is exactly the equation without prediction, the study of the optimal solution has been done in Section 3, it is minimized when $T_{\text{R}}^{\text{opt}_0} = \min\left(\alpha\mu_e - I, \max\left(\sqrt{2C\mu}, C\right)\right)$.

Next we have:

$$\begin{aligned}\text{WASTE}_{\text{withCkpt}\{q=1\}} = &\left(1 - \frac{r\left((1-p)I + p\mathbb{E}_I^{(f)}\right)}{p\mu}\right)\left(\frac{C}{T_{\text{R}}} + \frac{1-r}{\mu}\frac{T_{\text{R}}}{2}\right)\\ &+ \frac{r}{\mu}\left(\frac{\left((1-p)I + p\mathbb{E}_I^{(f)}\right)}{p}\frac{C}{T_{\text{P}}^{\text{opt}}} + T_{\text{P}}^{\text{opt}}\right) + \frac{r}{p\mu}C\\ &+ \left(\frac{r}{\mu} + \left(1 - \frac{r\left((1-p)I + p\mathbb{E}_I^{(f)}\right)}{p\mu}\right)\frac{1-r}{\mu}\right)(D+R)\end{aligned} \tag{9}$$

This equation is minimized when

$$T_{\text{R}}^{\text{opt}_1} = \sqrt{\frac{2\mu C}{(1-r)}}$$

One can remark that this value is equal to the result without intervals (Section 3). Actually, the only impact of the prediction interval I is the moment when we should take a pre-emptive action. Note that when $r = 0$ (this means that there is no prediction), we have $T_{\text{R}}^{\text{opt}_1} = T_{\text{R}}^{\text{opt}_0}$, and we retrieve Young's formula [1].

Finally, we know that the waste is defined for $C \leq T_{\text{R}} \leq \alpha\mu_e - I$. Hence, if $T_{\text{R}}^{\text{opt}_1} \notin [C, \alpha\mu_e - I]$, this solution is not satisfiable. However Equation (9) is convex, so the optimal solution is C if $T_{\text{R}}^{\text{opt}_1} < C$, and $\alpha\mu_e - I$ if $T_{\text{R}}^{\text{opt}_1} > \alpha\mu_e$. Hence, when $q = 1$, the optimal solution should be

$$\min\left(\alpha\mu_e - I, \max\left(\sqrt{\frac{2\mu C}{(1-r)}}, C\right)\right). \tag{10}$$

Instant. The derivation is similar . The optimal value for q is either 0 or 1, thus we consider $\text{WASTE}_{\text{INSTANT}}^{\{0\}} = \text{WASTE}_Y$ and $\text{WASTE}_{\text{INSTANT}}^{\{1\}}$. If $\mathbb{E}_I^{(f)} > \frac{T_{\text{R}}}{2}$, then $\text{WASTE}_{\text{INSTANT}}^{\{0\}} < \text{WASTE}_{\text{INSTANT}}^{\{1\}}$, so we can assume $\min(\mathbb{E}_I^{(f)}, \frac{T_{\text{R}}}{2}) = \mathbb{E}_I^{(f)}$. Then we derive that $\text{WASTE}_{\text{INSTANT}}^{\{1\}}$ is minimized for $T_{\text{R}}^{\text{opt}_1}$ as before.

NoCkptI. One can see that Equation (6) and Equation (4) only differ by the quantity :

$$\frac{qr}{\mu}\left(\frac{(1-p)I + p\mathbb{E}_I^{(f)}}{p}\frac{C}{T_{\text{P}}^{\text{opt}}} + T_{\text{P}}^{\text{opt}} - \mathbb{E}_I^{(f)}\right)$$

This value is linear in q and a constant with regards to T_{R}. Hence the minimization is almost the same.

Once again we can see that the optimal value for q is either 0 or 1. We can consider the two functions $\textsc{Waste}_{\mathrm{noCkpt}\{q=0\}}$ and $\textsc{Waste}_{\mathrm{noCkpt}\{q=1\}}$. We remark that $\textsc{Waste}_{\mathrm{noCkpt}\{q=0\}} = \textsc{Waste}_{\mathrm{withCkpt}\{q=0\}}$, and hence that the study has already been done. As for $\textsc{Waste}_{\mathrm{noCkpt}\{q=1\}}$, it is also minimized when $T_{\mathrm{R}}^{\mathrm{opt}} = \sqrt{\frac{2\mu C}{(1-r)}}$.

Finally, the last step of this study is identical to the previous minimization, and the optimal solution when $q = 1$ is defined by :

$$T_{\mathrm{R}}^{\mathrm{opt}_1} = \min\left(\alpha\mu_e - I, \max\left(\sqrt{\frac{2\mu C}{(1-r)}}, C\right)\right)$$

Summary. Finally in this section, we consider the waste for the two algorithms that take the prediction window into account (the one that does not checkpoint during the prediction window, and the one that checkpoints during the prediction window), and try to find conditions of dominance of one strategy over the other. Since the equation of the waste is identical when $q = 0$, let us consider the case when $q = 1$. We have seen that:

$$\begin{aligned}(\textsc{Waste}_{\mathrm{withCkpt}\{q=1\}} - \textsc{Waste}_{\mathrm{noCkpt}\{q=1\}}) = &\frac{r\left((1-p)I + p\mathbb{E}_I^{(f)}\right)}{p\mu}\frac{C}{T_{\mathrm{P}}^{\mathrm{opt}}} \\ &+ \frac{r}{\mu}\left(T_{\mathrm{P}}^{\mathrm{opt}} - \mathbb{E}_I^{(f)}\right) \qquad (11)\end{aligned}$$

We want to know when Equation (11) is nonnegative (meaning that it is beneficial not to take any checkpoints during proactive mode). We know that this value is minimized when $T_{\mathrm{P}}^{\mathrm{extr}} = \sqrt{\frac{(1-p)I + p\mathbb{E}_I^{(f)}}{p}C}$ (Equation (7)), then a sufficient condition would be to study the equation :

$$\textsc{Waste}_{\mathrm{withCkpt}\{q=1\}} - \textsc{Waste}_{\mathrm{noCkpt}\{q=1\}} \geq 0$$

with $T_{\mathrm{P}}^{\mathrm{extr}}$ instead of $T_{\mathrm{P}}^{\mathrm{opt}}$. That is:

$$\begin{aligned}&\frac{r(1-p)I + p\mathbb{E}_I^{(f)}}{p\mu}\frac{C}{\sqrt{\frac{(1-p)I + p\mathbb{E}_I^{(f)}}{p}C}} + \frac{r}{\mu}\left(\sqrt{\frac{(1-p)I + p\mathbb{E}_I^{(f)}}{p}C} - \mathbb{E}_I^{(f)}\right) \geq 0 \\ \Leftrightarrow\ &2\sqrt{\frac{(1-p)I + p\mathbb{E}_I^{(f)}}{p}C} \geq \mathbb{E}_I^{(f)2} \qquad (12)\end{aligned}$$

Consequently, we can say that if Equation (12) is matched, then $\textsc{Waste}_{\mathrm{noCkpt}} \leq \textsc{Waste}$, the algorithm where we do not checkpoint during the proactive mode has a better solution than Algorithm 1. For example, if we assume that faults strike uniformly during the prediction window $[t_0, t_0 + I]$, in other words, if

$0 \leq x \leq I$, the probability that the fault occurs in the interval $[t_0, t_0 + x]$ is $\frac{x}{I}$, then $\mathbb{E}_I^{(f)} = \frac{I}{2}$, and our condition becomes

$$I \leq 16 \frac{1 - p/2}{p} C.$$

We can now finish our study by saying that in order to find the optimal solution, one should compute both optimal solutions for $q = 0$ and $q = 1$, for both algorithms, and choose the one that minimizes the waste, as was done in Section 3, except when Equation (12) is valid, then we can focus on the computation of the waste of the algorithms that does not checkpoint during proactive mode.

5 Simulation results

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 [3, 4]. We have $C = R = 10mn$, and $D = 1mn$. The individual (processor) MTBF $\mu_{ind} = 125$ 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 $\mu = 4,000mn$ (about 1.5 day) down to $\mu = 125mn$ (about 2 hours). For instance the Jaguar platform, with $N = 45,208$ processors, is reported to experience about one failure per day [5], which leads to $\mu_{ind} = \frac{45,208}{365} \approx 125$ years.

We have analytically computed the optimal value of the waste for each strategy (using the formulas of Section 4.3) using a computer algebra software. In order to check the accuracy of our model, we have compared the results with those from simulations using a fault generator. Our simulation engine generates a random trace of failures, parameterized either by an Exponential failure distribution or by a Weibull distribution law with shape parameter 0.5 and 0.7; Exponential failures are widely used for theoretical studies, while Weibull failures are representative of the behavior of real-world platforms [6, 7, 8]. With probability r, we decide if a failure is predicted or not. In both cases, the distribution is scaled so that its expectation corresponds to the platform MTBF μ. Then the simulation engine generates another random trace of false predictions (whose distribution is identical to the first trace or a uniform distributions). This second distribution is scaled so that its expectation is $\frac{p\mu}{r(1-p)}$, the inter-arrival time of false predictions. Finally, both traces are merged to derive the final trace with all events. Each value reported for the simulations is the average of 100 randomly generated experiments.

In the simulations, we compare up to ten checkpointing strategies. Here is the list:

• YOUNG is the periodic checkpointing strategy of period $T_{\text{extr}}^{\{0\}} = \sqrt{2\mu C}$ given in [1]. Note that Daly's formula [2] leads to the same results.

• EXACTPREDICTION is derived from the strategy Section 3 (with exact prediction dates). However, in the simulations, we always take prediction into account and use an uncapped period $T_{\text{extr}}^{\{1\}} = \sqrt{\frac{2\mu C}{1-r}}$ instead of $T_1 = \min(\alpha\mu_e - I, \max(C, T_{\text{extr}}^{\{1\}}))$.

• Similarly, INSTANT, NOCKPTI and WITHCKPTI are the three strategies described in Section 4, with the same modification: we always take prediction into account and use an uncapped period $T_{\text{extr}}^{\{1\}}$ instead of T_1 in regular mode.
• 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_{R}. This latter period is computed via a brute-force numerical search for the optimal period.

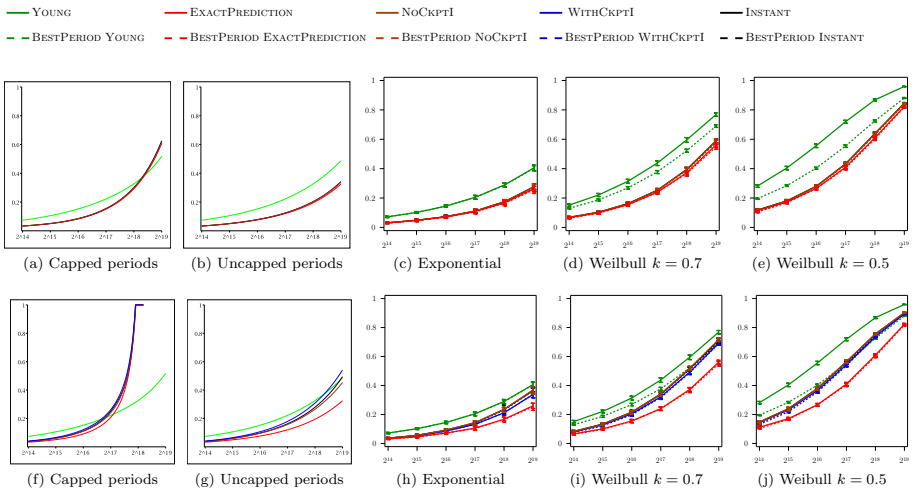
The rationale for modifying the strategies described in the previous sections is of course to better assess the impact of prediction. For the computer algebra plots, in addition to the waste with the *capped periods* given in Section 4.3, i.e., with $T_0 = T_{\text{Y}} = \min(\alpha\mu, \max(C, T_{\text{extr}}^{\{0\}}))$, and $T_1 = \min(\alpha\mu_e - I, \max(C, T_{\text{extr}}^{\{1\}}))$, we also report the waste obtained for the *uncapped periods*, i.e., using $T_0 = T_{\text{extr}}^{\{0\}}$ without prediction and $T_1 = T_{\text{extr}}^{\{1\}} = \sqrt{\dfrac{2\mu C}{1-r}}$ with prediction. The objective is twofold: (i) Assess whether the validity of the model can be extended; and (ii) Provide an exact match with the simulations, which mimic a real-life execution and do allow for an arbitrary number of faults per period.

5.1 Predictors from the literature

We first experiment with two predictors from the literature: one accurate predictor with high recall and precision [9], namely with $p = 0.82$ and $r = 0.85$, and another predictor with more limited recall and precision [10], namely with $p = 0.4$ and $r = 0.7$. In both cases, we use two different time-windows, $I = 300s$ and $I = 3,000s$. The former value does not allow for checkpointing within the prediction window, while the latter values allow for several checkpoints. Note that we always compare the results with EXACTPREDICTION, the strategy that assumes exact prediction dates. Figures 4 and 6 show the average waste degradation of the ten heuristics for both predictors, 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 $\mu = \mu_{ind}/N$, because it is more natural to see the waste increase with larger platforms; however, 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 that the prediction is always useful for the whole set of parameters under study! The second observation is the good correspondence between analytical results and simulations in Figures 4 and 6 (compare subfigures (a) and (b) with (c), (d) and (e), and subfigures (f) and (g) with (h), (i) and (j)). This shows the validity of the model for the whole range of distributions (Exponential and both Weibull shapes). More precisely: (i) The capped model overestimates the waste for large platforms (or small MTBFs), in particular for large values of I (see Figures 4(f) and 6(f)), but this was the price to pay for mathematical rigor; (ii) The uncapped model is accurate for the whole range of the study. Another striking result is that all strategies taking prediction into account have the same waste as their BESTPERIOD counterpart, which demonstrates that our formula $T_{\text{extr}}^{\{1\}} = \sqrt{\frac{2\mu C}{1-r}}$ is indeed the best possible checkpointing period in regular mode.

Unsurprisingly, EXACTPREDICTION is better than the heuristics that use a time window instead of exact prediction dates, especially with a high number



(a) Capped periods (b) Uncapped periods (c) Exponential (d) Weibull $k = 0.7$ (e) Weibull $k = 0.5$

(f) Capped periods (g) Uncapped periods (h) Exponential (i) Weibull $k = 0.7$ (j) Weibull $k = 0.5$

Figure 4: Waste for the different heuristics, with $p = 0.82, r = 0.85, I = 300$s (first row) or $I = 3,000$s (second row) and with a trace of false predictions parametrized by a distribution identical to the distribution of the trace of failures.

of processors. However, interval based heuristics achieve close results when $I = 300s$, or when $I = 3,000s$ and a small number of processors ($N < 2^{16}$).

In order to compare the heuristics without prediction to those with prediction, we report job execution times in Table 2. For the strategies with prediction, we compute the gain (expressed in percentage) over Young, the reference strategy without prediction. For $I = 300s$, the three strategies are identical. But for $I = 3,000s$, WithCkptI has often better results. First, with $p = 0.85$ and $r = 0.82$ and $I = 3,000s$, we save 25% of the total time with $N = 2^{19}$, and 14% with $N = 2^{16}$ using strategy WithCkptI. With $I = 300s$, we save up to 44% with $N = 2^{19}$, and 18% with $N = 2^{16}$ using any strategy (though NoCkptI is slightly better than Instant). Then, with $p = 0.4$ and $r = 0.7$, we still save 32% of the execution time when $I = 300s$ and $N = 2^{19}$, and 13% with $N = 2^{16}$. The gain gets smaller with $I = 3,000s$, but remains non negligible since we can save up to 9.7% with $N = 2^{19}$, and 7.6% with $N = 2^{16}$. Unexpectedly in this last case, the strategy that is the most efficient is Instant and not WithCkptI.

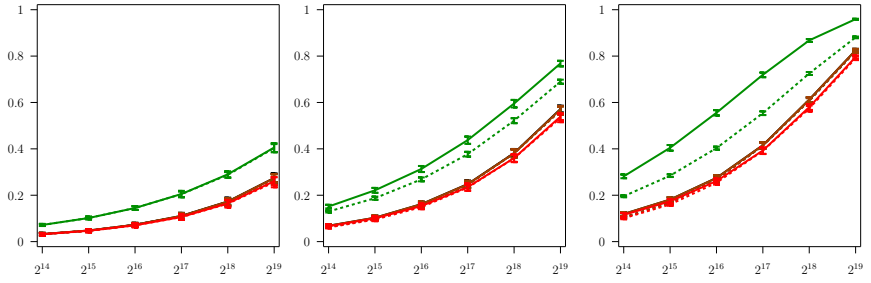
We observe that the size of the prediction-window I plays an important role too: we have better results for $I = 300$ and $(p, r) = (0.4, 0.7)$, than for $I = 3000$ and $(p, r) = (0.82, 0.85)$.

In Table 2, we report the job execution times for Weibull distributions with $k = 0.5$.

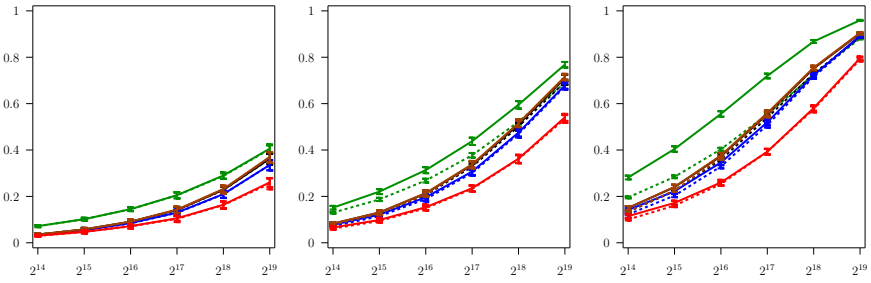
For $I = 300s$, the three strategies are identical. But for $I = 3,000s$, WithCkptI has often better results. First, with $p = 0.85$ and $r = 0.82$ and $I = 3,000s$, we save 61% of the total time with $N = 2^{19}$, and 30% with $N = 2^{16}$ using strategy WithCkptI.

With $I = 300s$, we save up to 74% with $N = 2^{19}$, and 38% with $N = 2^{16}$ using any strategy (though NoCkptI is slightly better than Instant).

Then, with $p = 0.4$ and $r = 0.7$, we still save 66% of the execution time when $I = 300s$ and $N = 2^{19}$, and 33% with $N = 2^{16}$. The gain gets smaller with

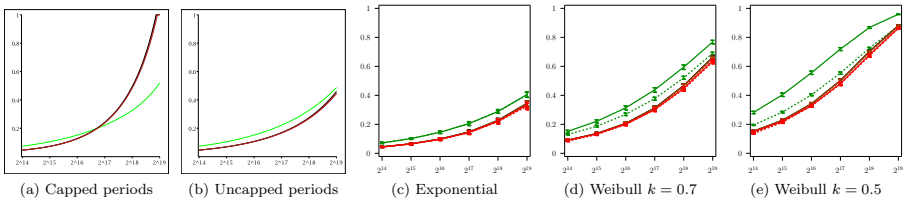


(a) Exponential (b) Weibull $k = 0.7$ (c) Weibull $k = 0.5$

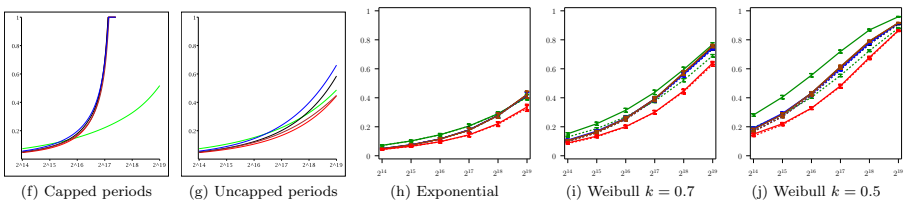


(d) Exponential (e) Weibull $k = 0.7$ (f) Weibull $k = 0.5$

Figure 5: Waste for the different heuristics, with $p = 0.82, r = 0.85, I = 300$s (first row) or $I = 3,000$s (second row) and with a trace of false predictions parametrized by a uniform distribution.

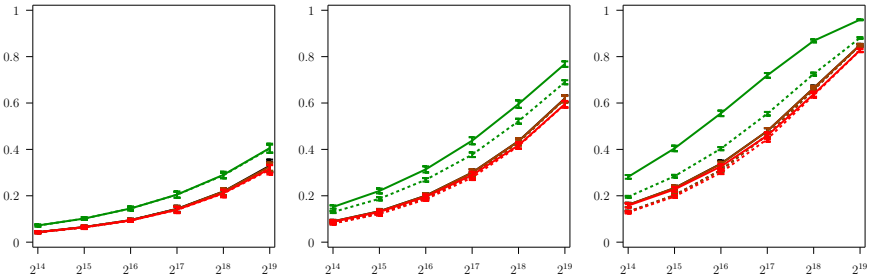


(a) Capped periods (b) Uncapped periods (c) Exponential (d) Weibull $k = 0.7$ (e) Weibull $k = 0.5$

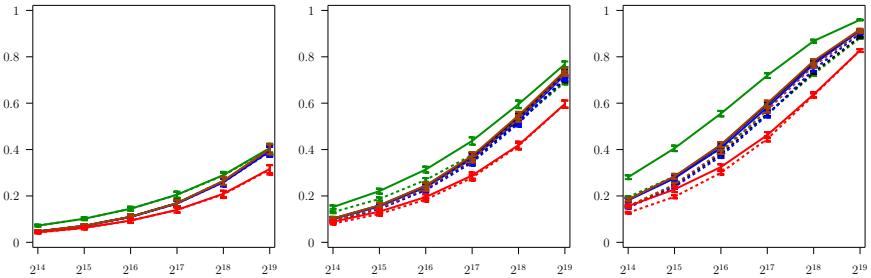


(f) Capped periods (g) Uncapped periods (h) Exponential (i) Weibull $k = 0.7$ (j) Weibull $k = 0.5$

Figure 6: Waste for the different heuristics, with $p = 0.4, r = 0.7, I = 300$s (first row) or $I = 3,000$s (second row) and with a trace of false predictions parametrized by a distribution identical to the distribution of the trace of failures.



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



(d) Exponential (e) Weibull $k = 0.7$ (f) Weibull $k = 0.5$

Figure 7: Waste for the different heuristics, with $p = 0.4, r = 0.7, I = 300$s (first row) or $I = 3,000$s (second row) and with a trace of false predictions parametrized by a uniform distribution.

$I = 3,000s$, but we can save up to 52% with $N = 2^{19}$, and 22% with $N = 2^{16}$.

Using a Weibull failure distribution with shape parameter 0.5, we observe that the gain due to prediction is twice larger than the gain computed with a Weibull failure distribution with shape parameter 0.7. We can conclude the same remark from Figures 4(e), 4(j), 6(e) and 6(j).

We also performed simulations with a trace of false predictions parametrized by a uniform distribution and we observe that the result (Figures 5 and 7) are similar to the result (Figures 4 and 6) with simulations with a trace of false predictions parametrized by a distribution identical to the distribution of the trace of failures.

5.2 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, and we let the other vary. We choose two platforms, a smaller one with $N = 2^{16}$ processors (or a MTBF $\mu = 1,000mn$) and the other with $N = 2^{19}$ processors (or a MTBF $\mu = 125mn$). In both cases, we use a prediction-window of size $I = 300s$, and a Weibull failure distribution with shape parameter $k = 0.7$ (we have similar results (Figures 9 and 11) for $k = 0.5$).

In Figure 8, 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

$I = 300$	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	81.2	30.1
EXACTPREDICTION	65.9 (19%)	15.9 (47%)	69.7 (14%)	19.3 (36%)
NOCKPTI	66.5 (18%)	16.9 (44%)	70.3 (13%)	20.5 (32%)
INSTANT	66.5 (18%)	17.0 (44%)	70.3 (13%)	20.7 (31%)

$I = 3,000$	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	81.2	30.1	81.2	30.1
EXACTPREDICTION	66.0 (19%)	15.9 (47%)	69.8 (14%)	19.3 (36%)
NOCKPTI	71.1 (12%)	24.6 (18%)	75.2 (7.3%)	28.9 (4.0%)
WITHCKPTI	70.0 (14%)	22.6 (25%)	75.4 (7.1%)	27.2 (9.7%)
INSTANT	71.2 (12%)	24.2 (20%)	75.0 (7.6%)	28.3 (6.0%)

Table 1: Comparing job execution times for a Weibull distribution ($k = 0.7$), and reporting the gain when comparing to YOUNG.

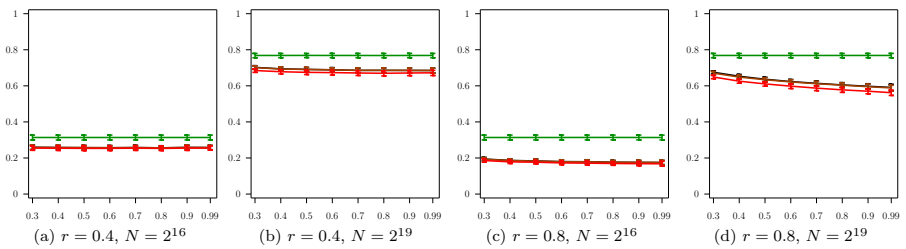
$I = 300$	Execution time (in days) ($p = 0.82, r = 0.85$)		Execution time (in days) ($p = 0.4, r = 0.7$)	
	2^{16} procs	2^{19} procs	2^{16} procs	2^{19} procs
YOUNG	125.4	171.8	125.5	171.7
EXACTPREDICTION	75.8 (40%)	39.4 (77%)	82.9 (34%)	51.8(70%)
NOCKPTI	77.3 (38%)	44.8 (74%)	84.6 (33%)	58.2 (66%)
INSTANT	77.4 (38%)	45.1 (74%)	84.7 (33%)	59.1 (66%)

$I = 3,000$	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	125.4	171.9	125.4	172.0
EXACTPREDICTION	76.1 (39%)	39.4 (77%)	83.0 (34%)	51.7 (70%)
NOCKPTI	90.0 (28%)	71.8 (58%)	98.3 (22%)	84.5 (51%)
WITHCKPTI	87.8 (30%)	66.6 (61%)	98.0 (22%)	82.2 (52%)
INSTANT	89.8 (28%)	70.9 (59%)	98.2 (22%)	83.2 (52%)

Table 2: Comparing job execution times for a Weibull distribution ($k = 0.5$), and reporting the gain when comparing to YOUNG.

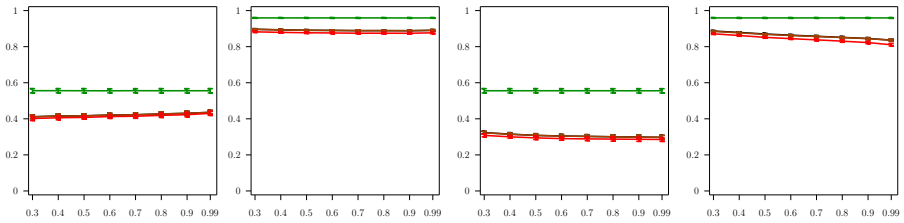
impact on the waste. In Figure 10, we conduct the opposite experiment and fix the value of p (either $p = 0.4$ or $p = 0.8$), letting r vary from 0.3 to 0.99. Here we observe that increasing the recall can significantly improve the performance.

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 failures prove very harmful and heavily increase the waste, while unduly checkpointing due to false predictions turns out to induce a smaller overhead.



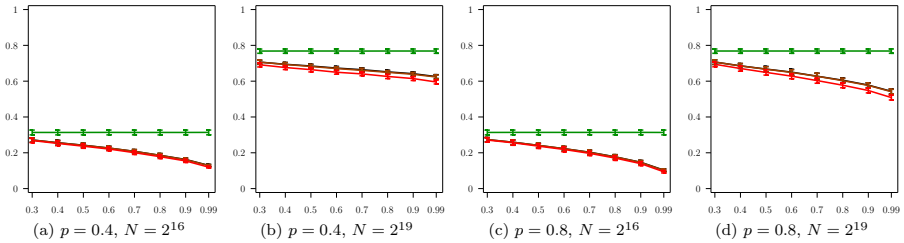
(a) $r = 0.4$, $N = 2^{16}$ (b) $r = 0.4$, $N = 2^{19}$ (c) $r = 0.8$, $N = 2^{16}$ (d) $r = 0.8$, $N = 2^{19}$

Figure 8: Impact of the precision for a fixed recall ($r = 0.4$ and $r = 0.8$) and for a Weibull distribution (k=0.7).



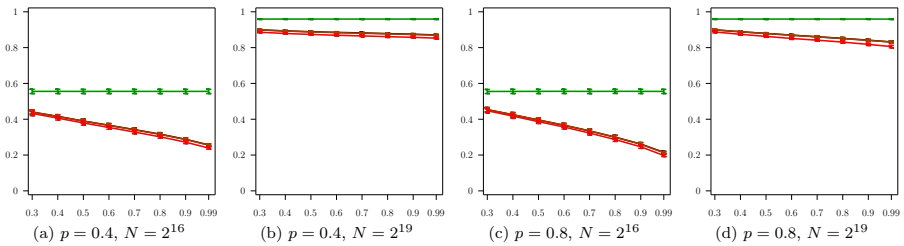
(a) $r = 0.4$, $N = 2^{16}$ (b) $r = 0.4$, $N = 2^{19}$ (c) $r = 0.8$, $N = 2^{16}$ (d) $r = 0.8$, $N = 2^{19}$

Figure 9: Impact of the precision for a fixed recall ($r = 0.4$ and $r = 0.8$) and for a Weibull distribution (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 10: Impact of the recall 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 11: Impact of the recall for a fixed precision ($p = 0.4$ and $p = 0.8$) and for a Weibull distribution (k=0.5).

Paper	Lead Time	Precision	Recall	Prediction Window
[10]	300 s	40 %	70%	-
[10]	600 s	35 %	60%	-
[9]	2h	64.8 %	65.2%	yes (size unknown)
[9]	0 min	82.3 %	85.4 %	yes (size unknown)
[11]	32 s	93 %	43 %	-
[13]	NA	70 %	75 %	-
[12]	NA	20 %	30 %	1h
[12]	NA	30 %	75 %	4h
[12]	NA	40 %	90 %	6h
[12]	NA	50 %	30 %	6h
[12]	NA	60 %	85%	12h

Table 3: Comparative study of different parameters returned by some predictors.

6 Related work

Considerable research has been conducted on fault prediction using different models (system log analysis [9], event-driven approach [11, 9, 10], support vector machines [12, 13]), nearest neighbors [12], ...). In this section we give a brief overview of the results obtained by predictors. We focus on their results rather than on their methods of prediction.

The authors of [10] introduce the *lead time*, that is the time between the prediction and the actual fault. This time should be sufficient to take proactive actions. They are also able to give the location of the fault. While this has a negative impact on the precision (see the low value of p in Table 3), they state that it has a positive impact on the checkpointing time (from 1500 seconds to 120 seconds). The authors of [9] also consider a lead time, and introduce a *prediction window* when the predicted fault should happen. The authors of [12] study the impact of different prediction techniques with different prediction window sizes. They also consider a lead time, but do not state its value. These two latter studies motivate the work of Section 4, even though [9] does not provide the size of their prediction window.

Unfortunately, much of the work done on prediction does not provide information that could be really useful for the design of efficient algorithms. These informations are those stated above, namely the lead time and the size of the prediction window, but other information that could be useful would be: (i) the distribution of the faults in the prediction window; (ii) the precision as a function of the recall (see our analysis); and (iii) the precision and recall as functions of the prediction window (what happens with a larger prediction window).

While many studies on fault prediction focus on the conception of the predictor, most of them consider that the proactive action should simply be a checkpoint or a migration right in time before the fault. However, in their paper [14], Li et al. consider the mathematical problem to determine when and how to migrate. In order to be able to use migration, they stated that at every time, 2% of the resources are available. This allowed them to conceive a Knapsack-based heuristic. Thanks to their algorithm, they were able to save

30% of the execution time compared to an heuristic that does not take the reliability into account, with a precision and recall of 70%, and with a maximum load of 0.7.

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 prediction, either with exact dates or window-based, on checkpointing strategies. We have designed several algorithms that decide when to trust these predictions, and when it is worth taking preventive checkpoints. We have introduced an analytical model to capture the waste incurred by each strategy, and provided the optimal solution to the corresponding optimization problems. We have been able to derive some striking conclusions:

• The model is quite accurate and its validity goes beyond the conservative assumption that requires capping checkpointing periods to diminish the probability of having several faults within the same period;

• A unified formula for the optimal checkpointing period is $\sqrt{\frac{2\mu C}{1-rq}}$, which unifies both cases with and without prediction, and nicely extends the work of Young and Daly to account for prediction;

• The simulations fully validate the model, and show that: (i) A significant gain is induced by using predictions, even for mid-range values of recall and precision; and (ii) The best period (found by brute-force search) is always very close to the one predicted by the model and given by the previous unified formula; this holds true both for Exponential and Weibull failure distributions;

• The recall has more impact on the waste than the precision: *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.

References

[1] J. W. Young, "A first order approximation to the optimum checkpoint interval," *Comm. of the ACM*, vol. 17, no. 9, pp. 530–531, 1974.

[2] J. T. Daly, "A higher order estimate of the optimum checkpoint interval for restart dumps," *FGCS*, vol. 22, no. 3, pp. 303–312, 2004.

[3] F. Cappello, H. Casanova, and Y. Robert, "Preventive migration vs. preventive checkpointing for extreme scale supercomputers," *Parallel Processing Letters*, vol. 21, no. 2, pp. 111–132, 2011.

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

[5] G. Zheng, X. Ni, and L. Kale, "A scalable double in-memory checkpoint and restart scheme towards exascale," in *Dependable Systems and Networks Workshops (DSN-W)*, 2012.

[6] T. Heath, R. P. Martin, and T. D. Nguyen, "Improving cluster availability using workstation validation," *SIGMETRICS Perf. Eval. Rev.*, vol. 30, no. 1, 2002.

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

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

[9] L. Yu, Z. Zheng, Z. Lan, and 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.

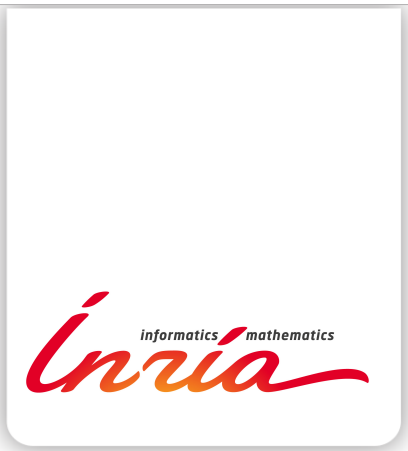
[10] Z. Zheng, Z. Lan, R. Gupta, S. Coghlan, and P. Beckman, "A practical failure prediction with location and lead time for blue gene/p," in *Dependable Systems and Networks Workshops (DSN-W)*, 2010, pp. 15–22.

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

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

[13] E. W. Fulp, G. A. Fink, and 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.

[14] S. Fu and C.-Z. Xu, "Exploring event correlation for failure prediction in coalitions of clusters," in *Proceedings of SC '07*. ACM, 2007, pp. 41:1–41:12.



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