HAL
archives-ouvertes.fr

# Checkpointing strategies with prediction windows

Guillaume Aupy, Yves Robert, Frédéric Vivien, Dounia Zaidouni

▶ **To cite this version:**

Guillaume Aupy, Yves Robert, Frédéric Vivien, Dounia Zaidouni. Checkpointing strategies with prediction windows. PRDC - The 19th IEEE Pacific Rim International Symposium on Dependable Computing - 2013, Dec 2013, Vancouver, Canada. IEEE, 2013. <hal-00847622>

**HAL Id: hal-00847622**
**https://hal.inria.fr/hal-00847622**

Submitted on 24 Jul 2013

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Checkpointing strategies with prediction windows

**Regular paper**

Guillaume Aupy[1,3], Yves Robert[1,3,4], Frédéric Vivien[2,3], and Dounia Zaidouni[2,3]
[1]École Normale Supérieure de Lyon, France [2]INRIA, France
[3]LIP laboratory - CNRS, ENS Lyon, INRIA, UCBL, and Université de Lyon, France
[4]University of Tennessee Knoxville, USA

***Abstract*—This paper deals with the impact of fault prediction techniques on checkpointing strategies. We consider fault-prediction systems that do not provide exact prediction dates, but instead time intervals during which faults are predicted to strike. These intervals dramatically complicate the analysis of the checkpointing strategies. We propose a new approach based upon two periodic modes, a regular mode outside prediction windows, and a proactive mode inside prediction windows, whenever the size of these windows is large enough. We are able to compute the best period for any size of the prediction windows, thereby deriving the scheduling strategy that minimizes platform waste. In addition, the results of the analytical study are nicely corroborated by a comprehensive set of simulations, which demonstrate the validity of the model and the accuracy of the approach.**

## I. Introduction

In this paper, we assess the impact of fault prediction techniques on coordinated checkpointing strategies. We assume to have jobs executing on a platform subject to faults, and we let $\mu$ 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 $T_{\text{opt}}$ of $T$ that minimizes the expected waste of resource usage due to failures and checkpointing is easily 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 [1], while the latter is due to Daly [2].

Assume now that some fault prediction system is available. Such a system is characterized by two critical parameters, 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). In the simple case where predictions are exact-date predictions, several recent papers [3], [4] have independently shown that the optimal checkpointing period becomes $T_{\text{opt}} = \sqrt{\frac{2\mu C}{1-r}}$. This latter expression is valid only when $\mu$ is large enough. This expression can be seen as an extension of Young's formula where $\mu$ is replaced by $\frac{\mu}{1-r}$: faults are replaced by non-predicted faults, and the overhead due to false predictions is negligible. A more accurate expression for the optimal checkpointing period is available in [4].

This paper deals with the realistic case (see [5], [6] and Section V) where the predictor system does not provide exact dates for predicted events, but instead provides *prediction windows*. A *prediction window* is a time interval of length $I$ during which the predicted event is likely to happen. Intuitively, one is more at risk during such an interval than in the absence of any prediction, hence the need to checkpoint more frequently. But with which period? Should we take into account all predictions? And what is the size of the prediction window above which it proves worthwhile to use a different (smaller) checkpointing period during the prediction windows? It turns out that the answer to those questions is dramatically more complicated than when using exact-date predictions. Our key contributions are the following: (i) The design of several checkpointing policies that account for the different sizes of prediction windows; (ii) The analytical characterization of the best policy for each set of parameters; and (iii) The validation of the theoretical results via extensive simulations, for both Exponential and Weibull failure distributions.

The rest of the paper is organized as follows. First we detail the framework in Section II. In Section III we describe the new checkpointing policies with prediction windows, and show how to compute the optimal checkpointing periods that minimize the platform waste. Section IV is devoted to simulations. Section V provides a brief overview of related work. Finally, we present concluding remarks in Section VI.

## II. Framework

### A. 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; by construction, we must enforce that $T \geq C$. 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 $\text{WASTE} = \frac{C}{T}$. Here, the *waste* is defined as the fraction of time that the platform is not doing useful work.

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 [7], [8]) 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.

### B. Fault predictor

A fault predictor is a mechanism that is able to predict that some faults will take place, within some time-interval window.

In this paper, we assume that the predictor is able to generate its predictions early enough so that a *proactive* checkpoint can indeed be taken before or during the event. A first proactive checkpoint will typically be taken just before the beginning of the prediction window, and possibly several other ones will be taken inside the prediction window, if its size $I$ is large enough.

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 when the application memory footprint is minimal [9]; in theses scenarios, proactive checkpoints are, on the contrary, taken according to predictions that can take place at arbitrary instants. On the other hand, we may have $C_p < C$ in some other scenarios [10], 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).

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 $\mathit{True}_P$ be their number); (ii) *False positive* events are fault predictions that did not materialize as actual faults (let $\mathit{False}_P$ be their number); and (iii) *False negative* events are faults that were not predicted (let $\mathit{False}_N$ be their number). With these definitions, we have $r = \frac{\mathit{True}_P}{\mathit{True}_P + \mathit{False}_N}$ and $p = \frac{\mathit{True}_P}{\mathit{True}_P + \mathit{False}_P}$.

In the literature, the *lead time* is the interval between the date at which the prediction is made available, and the predicted date of failure (or, more precisely, the beginning of the prediction window). However, because we do not consider pro-active actions with different durations (they all have length $C_p$), we point out that the distribution of these lead times is irrelevant to our problem. Indeed, either we have the time to take a proactive action before the failure strikes or not. Therefore, if a failure strikes less than $C_p$ seconds after the prediction is made available, the prediction was useless. In other words, predicted failures that come too early to enable any proactive action should be classified as unpredicted faults, leading to a smaller value of the predictor recall and to a shorter prediction window. Therefore, in the following, we consider, without loss of generality, that all predictions are made available $C_p$ seconds before the beginning of the prediction window.

### C. Fault rates

The key parameter is $\mu$, the mean time between faults (MTBF) of the platform. If the platform is made of $N$ components whose individual MTBF is $\mu_{\text{ind}}$, then $\mu = \frac{\mu_{\text{ind}}}{N}$. This result is true regardless of the fault distribution law [4]. In addition to $\mu$, the platform MTBF, let $\mu_P$ be the mean time between predicted events (both true positive and false positive), and let $\mu_{NP}$ be the mean time between unpredicted faults (false negative). Finally, we define the mean time between events as $\mu_e$ (including all three event types). The relationships between $\mu$, $\mu_P$, $\mu_{NP}$, and $\mu_e$ are the following:

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

## III. Checkpointing strategies

In this section, we introduce the new checkpointing strategies, and we determine the waste that they induce. We then proceed to computing the optimal period for each strategy.

### A. Description of the different strategies

We consider the following general scheme:

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;
3) If we decide to trust the predictor, we use various strategies, depending upon the length $I$ of the prediction window.

Before describing the different strategies in situation (3), we point out that the rationale for not always trusting the predictor is to avoid taking useless checkpoints too frequently. Indeed, 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. Intuitively, we need a larger proportion of the former cases, i.e., a good precision, for the predictor to be really useful.

Now, to describe the strategies used when we trust a prediction (situation (3)), we define two *modes* for the scheduling algorithm. The **Regular** mode is 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 $T_{\text{R}}$. Intuitively, $T_{\text{R}}$ corresponds to the checkpointing period $T$ of Section II-A. The **Proactive** mode is 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 for a prediction window $[t_0, t_0 + I]$. Several strategies can be envisioned:
(1) INSTANT, for *Instantaneous*– The first strategy (see Figure 1) is to ignore the time-window and to execute the same algorithm as if the predictor had given an exact date prediction at time $t_0$. The algorithm interrupts the current period (of scheduled length $T_{\text{R}}$), checkpoints during the interval $[t_0 - C_p, t_0]$, 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 (see Figure 2) 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

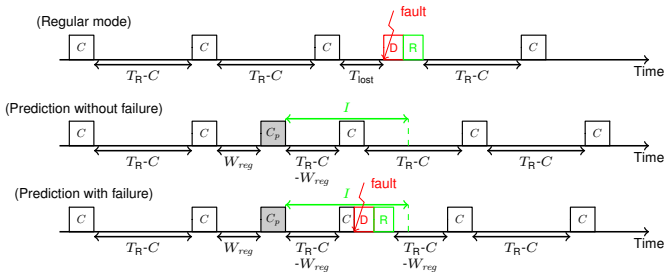



Figure 1: Outline of strategy INSTANT.

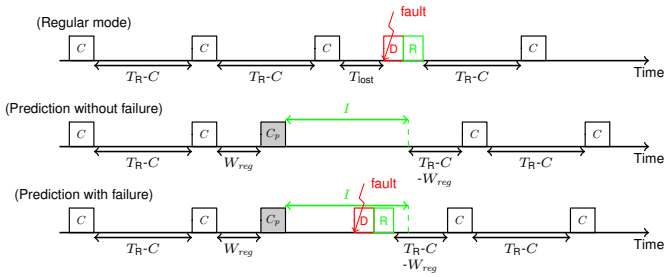



Figure 2: Outline of strategy NOCKPTI.

the first strategy, the algorithm interrupts the current period (of scheduled length $T_{\mathrm{R}}$), and checkpoints during the interval $[t_0 - C_p, t_0]$. 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 $I < C_p$).

(3) WITHCKPTI, for *With checkpoints during prediction window*– The third strategy (see Figure 3) is intended for a longer prediction window and assumes that $C_p \leq I$: the algorithm interrupts the current period (of scheduled length $T_{\mathrm{R}}$), and checkpoints during the interval $[t_0 - C_p, t_0]$, but now also decides to take several checkpoints during the prediction window. The period $T_{\mathrm{P}}$ of these checkpoints in proactive mode will presumably be shorter than $T_{\mathrm{R}}$, to take into account the higher fault probability. In the following, we analytically compute the optimal number of such periods. But we assume that there is at least one period here, hence, that we take at least one checkpoint (in the absence of faults), which implies $C_p \leq I$. 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 corresponding scheduling algorithm is shown in Algorithm 1.

Note that, for all strategies, 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). 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.

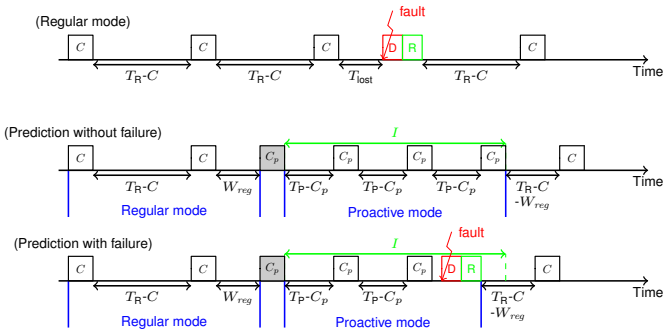



Figure 3: Outline of strategy WITHCKPTI.

## B. *Strategy* WITHCKPTI

In this section we evaluate the execution time under heuristic WITHCKPTI. To do so, we partition the whole execution into time intervals defined by the presence or absence of events. An interval starts and ends with either the completion of a checkpoint or of a recovery (after a failure). To ease the analysis, we make a simplifying hypothesis: we assume that *at most* one event, failure or prediction, occurs within any interval of length $T_{\mathrm{R}} + I + C_p$. In particular, this implies that a prediction or an unpredicted fault always take place during the regular mode.

We list below the four types of intervals, and evaluate their respective average length, together with the average work completed during each of them (see Table I for a summary):

1) **Two consecutive regular checkpoints with no intermediate events.** The time elapsed between the completion of the two checkpoints is exactly $T_{\mathrm{R}}$, and the work done is exactly $T_{\mathrm{R}} - C$.
2) **Unpredicted fault.** Recall that, because of the simplifying hypothesis, the fault happens in regular mode. Because instants where the fault strikes and where the

**Algorithm 1:** WITHCKPTI.

**if** *fault happens* **then**
  After downtime, execute recovery
  Enter *regular* mode
**if** *in* proactive *mode for a time greater than* $I$ **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 $t - C_p$
  **if** $t_C + C < t - C_p$ **then** (enough time for extra checkpoint)
    Take a checkpoint starting at time $t - C_p$
  **else** (no time for the extra checkpoint)
    Work in the time interval $[t_C + C, t]$
  $W_{reg} \leftarrow \max(0, t - C_p - (t_C + C))$
  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 $T_{\mathrm{R}}$-$W_{reg}$-$C$ and then checkpoint
  $W_{reg} \leftarrow 0$
**while** *in* proactive *mode and no faults happen* **do**
  Work for a time $T_{\mathrm{P}}$-$C_p$ and then checkpoint

| Mode | Number of intervals | Time spent | Work done |
|---|---|---|---|
| (1) | $w_1$ | $T_{\text{R}}$ | $T_{\text{R}} - C$ |
| (2) | $w_2 = \frac{\text{TIME}_{\text{Final}}}{\mu_{NP}}$ | $T_{\text{R}}/2 + D + R$ | 0 |
| (3) | $w_3 = \frac{(1-p)\text{TIME}_{\text{Final}}}{\mu_P}$ | $T_{\text{R}} + q(I + C_p)$ | $T_{\text{R}} - C + q(I - \frac{I}{T_{\text{P}}}C_p)$ |
| (4) | $w_4 = \frac{p\text{TIME}_{\text{Final}}}{\mu_P}$ | $q(T_{\text{R}} + \mathbb{E}_I^{(f)} + C_p)$ $+(1-q)T_{\text{R}}/2+D+R$ | $q\left(T_{\text{R}} - C + \left(\frac{\mathbb{E}_I^{(f)}}{T_{\text{P}}} - 1\right)(T_{\text{P}} - C_p)\right)$ |

Table I: Summary of the different interval types for WITHCKPTI.

last checkpoint was taken are independent, on average the fault strikes at time $T_{\text{R}}/2$. A downtime of length $D$ and a recovery of length $R$ occur before the interval completes. There is no work done.

3) **False prediction.** Recall that it happens in regular mode. There are two cases:
   a) **Taken into account.** This happens with probability $q$. The interval lasts $T_{\text{R}}+C_p+I$, since we take a proactive checkpoint and spend the time $I$ in proactive mode. The work done is $(T_{\text{R}} - C) + (I - \frac{I}{T_{\text{P}}}C_p)$.
   b) **Not taken into account.** This happens with probability $1 - q$. The interval lasts $T_{\text{R}}$ and the work done is $T_{\text{R}} - C$.

   Considering both cases with their probabilities, the average time spent is equal to: $q(T_{\text{R}} + C_p + I) + (1 - q)T_{\text{R}} = T_{\text{R}} + q(C_p + I)$. The average work done is: $q(T_{\text{R}} - C + I - \frac{I}{T_{\text{P}}}C_p) + (1 - q)(T_{\text{R}} - Cr) = T_{\text{R}} - C + q(I - \frac{I}{T_{\text{P}}}C_p)$.
4) **True prediction.** Recall that it happens in regular mode. There are two cases:
   a) **Taken into account.** Let $\mathbb{E}_I^{(f)}$ be the average time at which a fault occurs within the prediction window (the time at which the fault strikes is certainly correlated to the starting time of the prediction window; $\mathbb{E}_I^{(f)}$ may not be equal to $I/2$). Up to time $\mathbb{E}_I^{(f)}$, we work and checkpoint in proactive mode, with period $T_{\text{P}}$. In addition, we take a proactive checkpoint right before the start of the prediction window. Then we spend the time $\mathbb{E}_I^{(f)}$ in proactive mode, and we have a downtime and a recovery. Hence, such an interval lasts $T_{\text{R}}+C_p+\mathbb{E}_I^{(f)}+D+R$ on average. The total work done during the interval is $T_{\text{R}} - C + x(T_{\text{P}} - C_p)$ where $x$ is the expectation of the number of proactive checkpoints successfully taken during the prediction window. Here, $x \approx \frac{\mathbb{E}_I^{(f)}}{T_{\text{P}}} - 1$.
   b) **Not taken into account.** On average the fault occurs at time $T_{\text{R}}/2$. The time interval has duration $T_{\text{R}}/2+D+R$, and there is no work done.

   Overall the time spent is $q(T_{\text{R}} + C_p + \mathbb{E}_I^{(f)} + D + R) + (1 - q)(T_{\text{R}}/2 + D + R)$, and the work done is $q(T_{\text{R}} - C + (\frac{\mathbb{E}_I^{(f)}}{T_{\text{P}}} - 1)(T_{\text{P}} - C_p)) + (1 - q)0$.

We want to estimate the total execution time, $\text{TIME}_{\text{Final}}$. So far, we have evaluated the length, and the work done, for each of the interval types. We now estimate the expectation of the number of intervals of each type. Consider the intervals defined by an event whose mean time between occurrences is $\nu$. On average, during a time $T$, there will be $T/\nu$ such intervals. Due to the simplifying hypothesis, intervals of different types never overlap. Table I presents the estimation of the number of intervals of each type.

To estimate the time spent within intervals of a given type, we multiply the expectation of the number of intervals of that type by the expectation of the time spent in each of them. Of course, multiplying expectations is correct only if the corresponding random variables are independent. Nevertheless, we hope that this will lead us to a good approximation of the expected execution time. We will assess the quality of the approximation through simulations in Section IV. We have:

$$\text{TIME}_{\text{Final}} = w_1 \times T_{\text{R}} + w_2\left(\frac{T_{\text{R}}}{2} + D + R\right) + w_3\left(T_{\text{R}} + q(I + C_p)\right) + w_4\left(q(T_{\text{R}} + \mathbb{E}_I^{(f)} + C_p) + (1 - q)\frac{T_{\text{R}}}{2} + D + R\right) \quad (1)$$

We use the same line of reasoning to compute the overall amount of work done, that must be equal, by definition, to $\text{TIME}_{\text{base}}$, the execution time of the application without any overhead:

$$\text{TIME}_{\text{base}} = w_1(T_{\text{R}} - C) + w_2 \times 0 + w_3\left(T_{\text{R}} - C + q\left(I - \frac{I}{T_{\text{P}}}C_p\right)\right) + w_4\left(q\left(T_{\text{R}} - C + \left(\frac{\mathbb{E}_I^{(f)}}{T_{\text{P}}} - 1\right)(T_{\text{P}} - C_p)\right)\right) \quad (2)$$

This equation gives the value of $w_1$ as a function of the other parameters. Looking at Equations (1) and (2), and at the values of $w_2$, $w_3$, and $w_4$, we remark that $\text{TIME}_{\text{Final}}$ can be rewritten as a function of $q$ as follows: $\text{TIME}_{\text{Final}} = \alpha\text{TIME}_{\text{base}} + \beta\text{TIME}_{\text{Final}} + q\gamma\text{TIME}_{\text{Final}}$, that is $\text{TIME}_{\text{Final}} = \frac{\alpha}{1-\beta-q\gamma}\text{TIME}_{\text{base}}$, where neither $\alpha$, nor $\beta$, nor $\gamma$ depend on $q$. The derivative of $\text{TIME}_{\text{Final}}$ with respect to $q$ has constant sign. Hence, in an optimal solution, either $q = 0$ or $q = 1$. 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 can now focus on the two functions $\text{TIME}_{\text{Final}}$, the one when $q = 0$ ($\text{TIME}_{\text{Final}}^{\{0\}}$), and the one when $q = 1$ ($\text{TIME}_{\text{Final}}^{\{1\}}$).

When $q = 0$, from Table I and Equations (1) and (2), we derive that

$$\left(1 - \frac{C}{T_{\text{R}}}\right)\left(1 - \frac{T_{\text{R}}/2 + D + R}{\mu}\right)\text{TIME}_{\text{Final}}^{\{0\}} = \text{TIME}_{\text{base}} \quad (3)$$

This is exactly the equation from [4] in the case of exact-date predictions that are never taken into account (a good sanity check!). When $q = 1$, we have:

$$\begin{aligned}\text{TIME}_{\text{Final}}^{\{1\}} = {} & \text{TIME}_{\text{base}}\frac{T_{\text{R}}}{T_{\text{R}} - C} \\ & - \frac{\text{TIME}_{\text{Final}}^{\{1\}}}{\mu_P}\frac{T_{\text{R}}}{T_{\text{R}} - C}\left((T_{\text{R}} - C) + (1-p)\left(I - \frac{I}{T_{\text{P}}}C_p\right) + p\left(\frac{\mathbb{E}_I^{(f)}}{T_{\text{P}}} - 1\right)(T_{\text{P}} - C_p)\right) \\ & + \frac{\text{TIME}_{\text{Final}}^{\{1\}}}{\mu_{NP}}\left(\frac{T_{\text{R}}}{2} + D + R\right) + \frac{(1-p)\text{TIME}_{\text{Final}}^{\{1\}}}{\mu_P}(T_{\text{R}} + I + C_p) \\ & + \frac{p\text{TIME}_{\text{Final}}^{\{1\}}}{\mu_P}\left(T_{\text{R}} + C_p + \mathbb{E}_I^{(f)} + D + R\right)\end{aligned}$$

Finally, after some rewriting, the waste (defined as $\text{WASTE} = \frac{\text{TIME}_{\text{Final}} - \text{TIME}_{\text{base}}}{\text{TIME}_{\text{Final}}}$) becomes

$$\text{WASTE} = 1 - \frac{r}{p\mu}\left(1 - \frac{C_p}{T_\text{P}}\right)\left((1-p)I + p\left(\mathbb{E}_I^{(f)} - T_\text{P}\right)\right) - \left(1 - \frac{C}{T_\text{R}}\right)\left(1 - \frac{1}{p\mu}\left(p(D+R) + rC_p + (1-r)p\frac{T_\text{R}}{2} + r\left((1-p)I + p\mathbb{E}_I^{(f)}\right)\right)\right) \quad (4)$$

*Waste minimization:* When $q = 0$, the optimal period can readily be computed from Equation (3) and we derive that the optimal period is $\sqrt{2(\mu - (D+R))C}$. This defines a periodic policy we call RFO, for Refined First-Order approximation. We now minimize the waste of the strategy where $q = 1$. In order to compute the optimal value for $T_\text{P}$, we identify the fraction of the waste in Equation (4) that depends on $T_\text{P}$. We can rewrite Equation (4) as:

$$\text{WASTE}^{\{1\}} = \alpha + \frac{r}{p\mu}\left(\left((1-p)\,I + p\mathbb{E}_I^{(f)}\right)\frac{C_p}{T_\text{P}} + pT_\text{P}\right)$$

where $\alpha$ does not depend on $T_\text{P}$. The waste is thus minimized when $T_\text{P}$ is equal to $T_\text{P}^{\text{extr}} = \sqrt{\frac{\left((1-p)I + p\mathbb{E}_I^{(f)}\right)C_p}{p}}$. Note that we always have to enforce that $T_\text{P}$ is larger than $C_p$ and does not exceed $I$. Therefore, the optimal period $T_\text{P}^{\text{opt}}$ is defined as follows: $T_\text{P}^{\text{opt}} = \min\{I,\ \max\{C_p,\ T_\text{P}^{\text{extr}}\}\}$. The rounding only occurs for extreme cases.

In order to compute the optimal value for $T_\text{R}$, we identify the fraction of the waste in Equation (4) that depends on $T_\text{R}$. We can rewrite Equation (4) as:

$$\text{WASTE}^{\{1\}} = \beta + \frac{1-r}{\mu}\frac{T_\text{R}}{2} + \frac{C}{T_\text{R}}\left(1 - \frac{1}{p\mu}\left(p(D+R) + r\left(C_p + (1-p)\,I + p\mathbb{E}_I^{(f)}\right)\right)\right) \quad (5)$$

where $\beta$ does not depend on $T_\text{R}$ because $T_\text{P}^{\text{opt}}$ does not depend on $T_\text{R}$. Therefore, $\text{WASTE}^{\{1\}}$ is minimized when $T_\text{R}$ is equal to

$$T_\text{R}^{\text{extr}} = \sqrt{\frac{2C\left(p\mu - \left(p(D+R) + r\left(C_p + \left((1-p)I + p\mathbb{E}_I^{(f)}\right)\right)\right)\right)}{p(1-r)}} \quad (6)$$

Recall that we must always enforce that $T_\text{R}^{\text{opt}}$ is always greater than $C$. Also, note that when $r = 0$, we do obtain the same period as without a predictor. Finally, if we assume that, on average, fault strikes at the middle of the prediction window, i.e., $\mathbb{E}_I^{(f)} = \frac{I}{2}$, we obtain simplified values:

$$T_\text{P}^{\text{extr}} = \sqrt{\frac{(2-p)IC_p}{p}} \text{ and}$$

$$T_\text{R}^{\text{extr}} = \sqrt{\frac{2C\left(p\mu - \left(p(D+R) + r\left(C_p + \left(1 - \frac{p}{2}\right)I\right)\right)\right)}{p(1-r)}}$$

| Mode | Number of intervals | Time spent | Work done |
|---|---|---|---|
| (1) | $w_1$ | $T_\text{R}$ | $T_\text{R} - C$ |
| (2) | $w_2 = \frac{\text{TIME}_{\text{Final}}}{\mu_{NP}}$ | $T_\text{R}/2 + D + R$ | 0 |
| (3) | $w_3 = \frac{(1-p)\text{TIME}_{\text{Final}}}{\mu_P}$ | $T_\text{R} + q(I + C_p)$ | $T_\text{R} - C + qI$ |
| (4) | $w_4 = \frac{p\text{TIME}_{\text{Final}}}{\mu_P}$ | $q(T_\text{R} + \mathbb{E}_I^{(f)} + C_p)$ $+(1-q)T_\text{R}/2 + D + R$ | $q\,(T_\text{R} - C)$ |

Table II: Summary of the different interval types for NOCKPTI.

| Mode | Number of intervals | Time spent | Work done |
|---|---|---|---|
| (1) | $w_1$ | $T_\text{R}$ | $T_\text{R} - C$ |
| (2) | $w_2 = \frac{\text{TIME}_{\text{Final}}}{\mu_{NP}}$ | $T_\text{R}/2 + D + R$ | 0 |
| (3) | $w_3 = \frac{(1-p)\text{TIME}_{\text{Final}}}{\mu_P}$ | $T_\text{R} + qC_p$ | $T_\text{R} - C$ |
| (4) | $w_4 = \frac{p\text{TIME}_{\text{Final}}}{\mu_P}$ | $q(T_\text{R} + \mathbb{E}_I^{(f)} + C_p)$ $+(1-q)T_\text{R}/2 + D + R$ | $q\,(T_\text{R} - C)$ |

Table III: Summary of the different interval types for INSTANT.

### C. Strategy NOCKPTI

In this section we evaluate the execution time under heuristic NOCKPTI. Due to lack of space, we only summarize results and refer to [11] for details. The analysis is similar to that for WITHCKPTI. Table II provides the estimation of the number of intervals of each type. As for WITHCKPTI, one shows that in an optimal solution, either $q = 0$ or $q = 1$. When $q = 0$, we derive that

$$\left(1 - \frac{C}{T_\text{R}}\right)\left(1 - \frac{T_\text{R}/2 + D + R}{\mu}\right)\text{TIME}_{\text{Final}}^{\{0\}} = \text{TIME}_{\text{base}} \quad (7)$$

This is exactly the equation from [4] in the case of exact-date predictions that are never taken into account, what we had already retrieved with WITHCKPTI (same sanity check!). When $q = 1$, we derive that:

$$\text{WASTE} = 1 - \frac{r}{p\mu}(1-p)I - \left(1 - \frac{C}{T_\text{R}}\right) \times \left(1 - \frac{1}{p\mu}\left(p(D+R) + rC_p + (1-r)p\frac{T_\text{R}}{2} + r\left((1-p)I + p\mathbb{E}_I^{(f)}\right)\right)\right) \quad (8)$$

*Waste minimization:* The waste is minimized as follows:

- When $q = 0$, the optimal value for $T_\text{R}$ is the same as the one we computed for WITHCKPTI in the case $q = 0$.
- When $q = 1$, the value of $T_\text{R}$ that minimizes the waste is $T_\text{R}^{\text{extr}}$, the value given by Equation (6).

### D. Strategy INSTANT

In this section we evaluate the execution time under heuristic NOCKPTI. Due to lack of space, we only summarize results and refer to [11] for details. The analysis is similar to the previous ones. Table III provides the estimation of the number of intervals of each type. As before, one shows that in an optimal solution, either $q = 0$ or $q = 1$. When $q = 0$, we derive, once again, that

$$\left(1 - \frac{C}{T_\text{R}}\right)\left(1 - \frac{T_\text{R}/2 + D + R}{\mu}\right)\text{TIME}_{\text{Final}}^{\{0\}} = \text{TIME}_{\text{base}} \quad (9)$$

This is exactly the equation from [4] in the case of exact-date predictions that are never taken into account, what we

had already remarked with WITHCKPTI and NOCKPTI (yet another good sanity check!). When $q = 1$, we obtain

$$\text{WASTE} = 1 - \left(1 - \frac{C}{T_{\text{R}}}\right) \times \left(1 - \frac{1}{p\mu}\left(p(D+R) + rC_p + (1-r)p\frac{T_{\text{R}}}{2} + pr\mathbb{E}_I^{(f)}\right)\right) \quad (10)$$

*Waste minimization:* The waste is minimized as follows:

- When $q = 0$, the optimal value for $T_{\text{R}}$ is the same as for WITHCKPTI and for NOCKPTI in the case $q = 0$.
- When $q = 1$, the optimal value for $T_{\text{R}}$ is

$$T_{\text{R}}^{\text{extr}} = \sqrt{\frac{2C\left(p\mu - \left(p(D+R) + rC_p + pr\mathbb{E}_I^{(f)}\right)\right)}{p(1-r)}}$$

Again, recall that we must always enforce that $T_{\text{R}}^{\text{opt}}$ is always greater than $C$. Finally, if we assume that, on average, fault strikes at the middle of the prediction window, i.e., $\mathbb{E}_I^{(f)} = \frac{I}{2}$, we have:

$$T_{\text{R}}^{\text{extr}} = \sqrt{\frac{2C\left(p\mu - \left(p(D+R) + rC_p + pr\frac{I}{2}\right)\right)}{p(1-r)}}.$$

## IV. SIMULATION RESULTS

An experimental validation of the models at targeted scale would require running a large application several times, for each checkpointing strategy, for each fault predictor, and for each platform size. This would require a prohibitive amount of computational hours. Furthermore, some of the targeted platform sizes currently exist only as reasonable projections. Therefore, we resort to simulations. We present the simulation framework in Section IV-A. Then we report results using the characteristics of two fault predictors (Section IV-B). Additional figures and data results are available in [11].

### A. Simulation framework

In order to validate the model, we have instantiated it with several scenarios. The simulations use parameters that are representative of current and forthcoming large-scale platforms [12], [13]. We take $C = R = 600$ seconds, and $D = 60$ seconds. We consider three scenarios where proactive checkpoints are (i) exactly as expensive as periodic checkpoints ($C_p = C$); (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 $N = 2^{16} = 16,384$ to $N = 2^{19} = 524,288$, so that the platform MTBF $\mu$ 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 have experienced about one fault per day [14], 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 use Maple to analytically compute and plot the optimal value of the waste for the three prediction-aware policies, INSTANT, NOCKPTI, and WITHCKPTI, for the prediction-ignoring policy RFO (corresponding to the case $q = 0$), and for the reference heuristic DALY (Daly's [2] periodic policy). In order to check the accuracy of our model, we have compared the analytical 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 [15], [16], [17]. In both cases, the distribution is scaled so that its expectation corresponds to the platform MTBF $\mu$. 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 (results are similar when false predictions follow a uniform distribution [11]). 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). The source code of the fault-simulator and the raw simulation results are freely available [18].Each reported value is the average over 100 randomly generated instances.

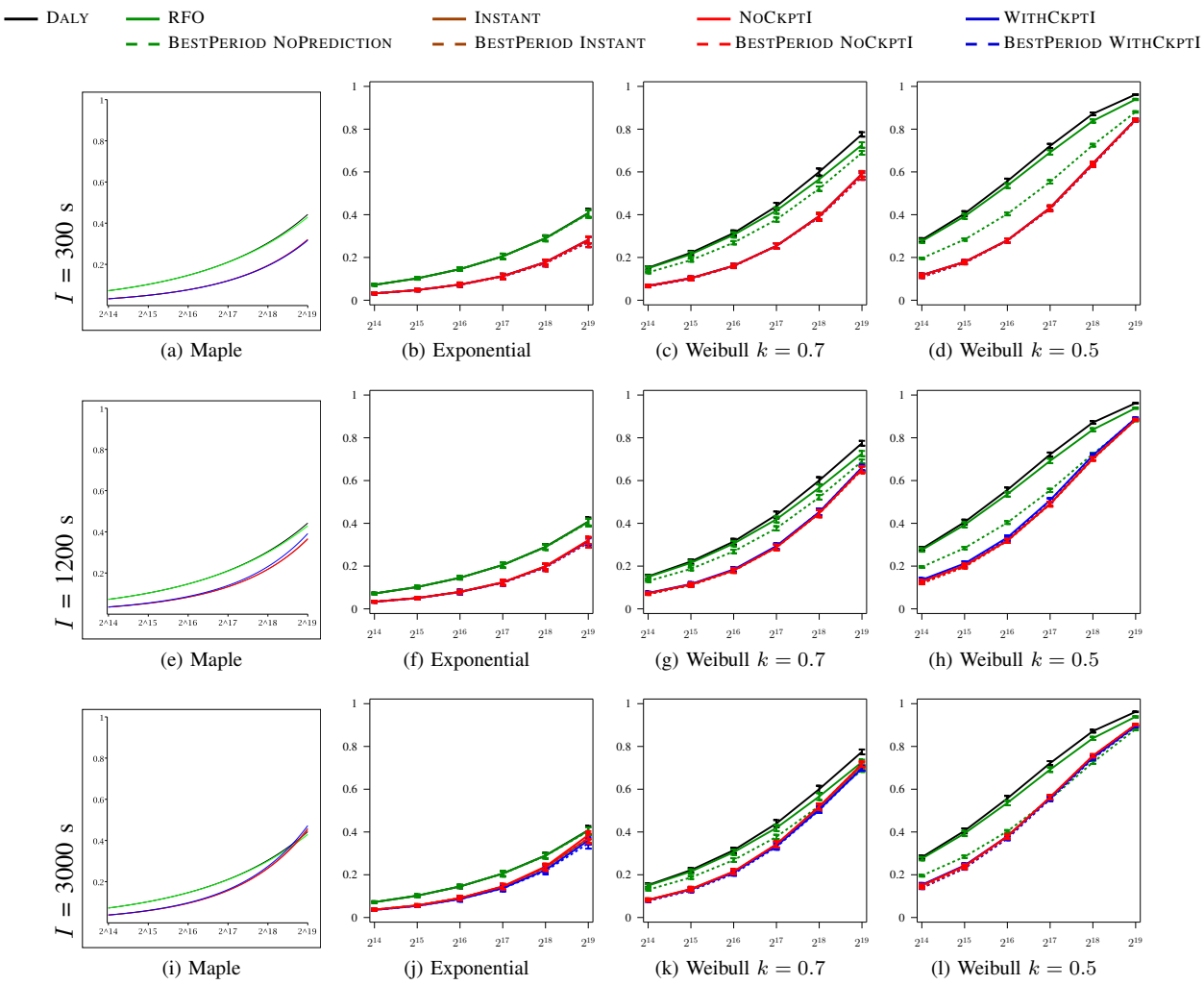
In the simulations, we compare the five checkpointing strategies listed above. To assess the quality of each strategy, we compare it with its BESTPERIOD counterpart, defined as the same strategy but using the best possible period $T_{\text{R}}$. This latter period is computed via a brute-force numerical search for the optimal period. Altogether, there are four BESTPERIOD heuristics, one for each of the three variants with prediction, and one for the case where we ignore predictions, which corresponds to both DALY and RFO. Altogether we have a rich set of nine heuristics, which enables us to comprehensively assess the actual quality of the proposed strategies. Note that for computer algebra plots, obviously we do not need BESTPERIOD heuristics, since each period is already chosen optimally from the equations.

We experiment with two predictors from the literature: one accurate predictor with high recall and precision [5], 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 five different prediction windows, of size $I = 300$, 600, 900, 1200, and 3000 seconds. 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.

### B. Analysis of the results

We start with a preliminary remark: when the graphs for INSTANT and WITHCKPTI cannot be seen in the figures, this is because their performance is identical to that of NOCKPTI, and their respective graphs are superposed.

We first compare the analytical results, plotted by the Maple curves, to the simulations results. As shown in Figure 4, there is a good correspondence between the analytical curves and the simulations, especially those using an Exponential distribution of failures. However, the larger the platform (or

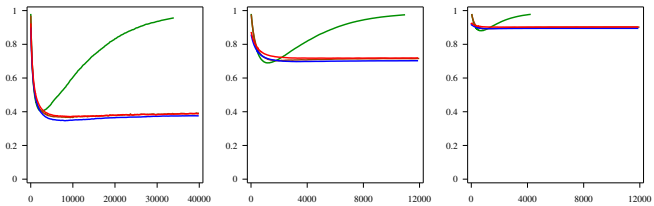



(a) Maple (b) Exponential (c) Weibull $k = 0.7$ (d) Weibull $k = 0.5$

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

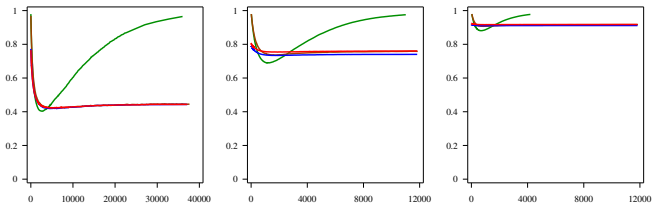
(i) Maple (j) Exponential (k) Weibull $k = 0.7$ (l) Weibull $k = 0.5$

Figure 4: Waste as a function of number $N$ of processors, when $p = 0.82$, $r = 0.85$, $C_p = C$.



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

Figure 5: Waste as function of the period $T_{\mathrm{R}}$, with $p = 0.82$, $r = 0.85$, $C_p = C$, $I = 3000s$, and a platform of $2^{19}$ processors.



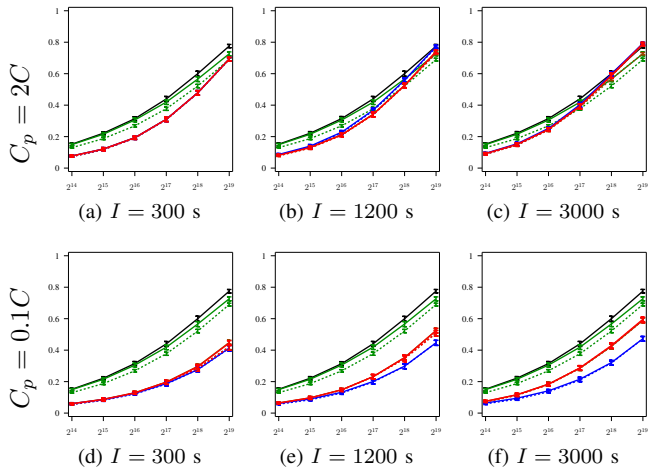
(a) Exponential (b) Weibull $k = 0.7$ (c) Weibull $k = 0.5$

Figure 6: Waste as function of the period $T_{\mathrm{R}}$, with $p = 0.4$, $r = 0.7$, $C_p = C$, $I = 3000s$, and a platform of $2^{19}$ processors.

the smaller the MTBF), the less realistic our assumption that no two events happen during an interval of length $T_{\mathrm{R}} + I + C_p$, and the analytical models become less accurate for prediction-aware heuristics. Therefore, the analytical results are overly pessimistic in the most failure-prone platforms. Also, recall that an exponential law is a Weibull law of shape parameter 1. Therefore, the further the distribution of failures is from an exponential law, the larger the difference between analytical results and simulated ones. However, in all cases, the analytical results are able to predict the general *trends*.

A second assessment of the quality of our analysis comes from the BESTPERIOD variants of our heuristics. When predictions are not taken into account, DALY, and to a lesser extent RFO, are not close to the optimal period given by BESTPERIOD (a similar observation was made in [19]). This gap increases when the distribution is further apart from an Exponential distribution. However, prediction-aware heuristics are very close to BESTPERIOD in almost all configurations.

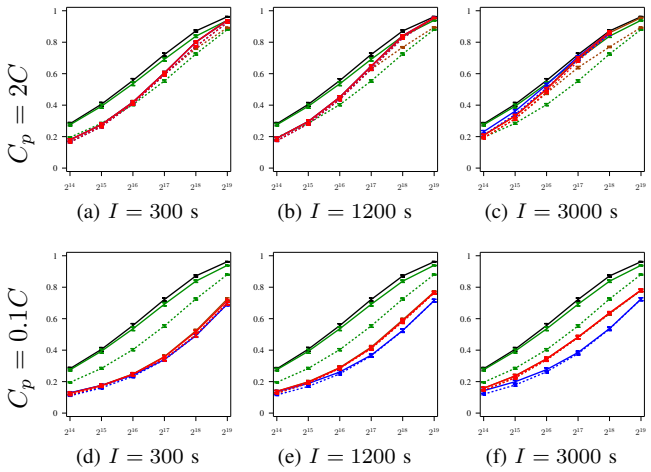
To better understand why close-to-optimal periods are obtained by prediction-aware heuristics (while this is not the




(a) $I = 300$ s (b) $I = 1200$ s (c) $I = 3000$ s

(d) $I = 300$ s (e) $I = 1200$ s (f) $I = 3000$ s

Figure 7: Waste with $p = 0.82$, $r = 0.85$, and Weibull law of parameter 0.7.




(a) $I = 300$ s (b) $I = 1200$ s (c) $I = 3000$ s

(d) $I = 300$ s (e) $I = 1200$ s (f) $I = 3000$ s

Figure 8: Waste with $p = 0.4$, $r = 0.7$, and Weibull law of parameter 0.5.

case without predictions), we plot the waste as a function of the period $T_{\mathrm{R}}$ for RFO and the prediction-aware heuristics. On Figure 5 and 6, one can see that, whatever the configuration, periodic checkpointing policies (ignoring predictions) have well-defined global optimum. (One should nevertheless remark that the performance is almost constant in the neighborhood of the optimal period, which explains why policies using different periods can obtain in practice similar performance, as in [20].) For prediction-aware heuristics, however, the behavior is quite different and two scenarios are possible. In the first one, once the optimum is reached, the waste very slowly increases to reach an asymptotic value which is close to the optimum waste (e.g., when the platform MTBF is large and failures follow an exponential distribution). Therefore, any period chosen close to the optimal one, or greater than it, will deliver good quality performance. In the second scenario, the waste decreases until the period becomes larger than the application size, and the waste stays constant. In other words, in these configurations, periodic checkpointing is unnecessary, only proactive actions matter! This striking result can be explained as follows: a significant fraction of the failures are predicted, and thus taken care of, by proactive checkpoints. The impact of unpredicted failures is mitigated by the proactive measures taken for false predictions.

Figure 7 and 8 presents a comparison of the checkpointing strategies for different values of $C_p$ and $I$. When the prediction window $I$ is shorter than the duration $C_p$ of a proactive checkpoint (i.e., when $I = 300$ s and $C_p \geq C = 600$ s), there is no difference between NoCkptI and WithCkptI. When $I$ is small but greater than $C_p$ (say, when $I$ is around $2C_p$), WithCkptI spends most of the prediction window taking a proactive checkpoint and NoCkptI is more efficient. When $I$ becomes "large" with respect to $C_p$, WithCkptI can become more efficient than NoCkptI, but becomes significantly more efficient only if the proactive checkpoints are significantly shorter than regular ones (see also Table IV). Instant can hardly be seen in the graphs as its performance is most of the time equivalent to that of NoCkptI.

As expected, the smaller the prediction window, the more efficient the prediction-aware heuristics. Also, the smaller the number of processors (or the larger the platform MTBF), the larger the impact of the size of the prediction window. A surprising result is that taking prediction into account is not always beneficial! The analytical results predict that prediction-aware heuristics would achieve worse performance than periodic policies in our settings, as soon as the platform includes $2^{18}$ processors. In simulations, results are not so extreme. For the largest platforms considered, using predictions has almost no impact on performance. But when the prediction window is very large, taking predictions into account can indeed be detrimental. These observations can be explained as follows. When the platform includes $2^{19}$ processors, the platform MTBF is equal to 7500s. Therefore, any interval of duration 3000s has a 40% chance to include a failure: a prediction window of 3000s is not very informative, unless the precision and recall of the predictor are almost equal to 1 (which is never the case in practice). Because the predictor brings almost no knowledge, trusting it may be detrimental. When comparing the performance of, say, NoCkptI for the two predictors, one can see that when failures follow a Weibull distribution with shape parameter $k = 0.7$, $I = 600$s, and $N = 2^{18}$, NoCkptI achieves better performance than RFO when $r = 0.85$ and $p = 0.82$, but worse when $p = 0.4$ and $r = 0.7$. The latter predictor generates more false predictions —each one inducing an unnecessary proactive checkpoint— and misses more actual failures —each one destroying some work. The drawbacks of trusting the predictor outweigh the advantages. If failures are few and apart, almost any predictor will be beneficial. When the platform MTBF is small with respect to the cost of proactive checkpoints, only almost perfect predictors will be worth using. For each set of predictor characteristics, there is a threshold for the platform MTBF under which predictions will be useless or detrimental, but above which predictions will be beneficial.

In order to compare the impact of the heuristics ignoring predictions to those using them, we report job execution times in Table IV when failures follow a Weibull law of parameter 0.7. For each setting, the best performance is presented in bold if it is achieved by a prediction-aware heuristics. For the strategies with prediction, we compute the gain (expressed in percentage) over Daly, the reference strategy without predic-

| | $I = 300$ s | | $I = 1200$ s | | $I = 3000$ s | |
|---|---|---|---|---|---|---|
| | $2^{16}$ procs | $2^{19}$ procs | $2^{16}$ procs | $2^{19}$ procs | $2^{16}$ procs | $2^{19}$ procs |
| DALY | 81.3 | 31.0 | 81.3 | 31.0 | 81.3 | 31.0 |
| RFO | 80.2 (1%) | 25.5 (18%) | 80.2 (1%) | 25.5 (18%) | 80.2 (1%) | 25.5 (18%) |
| $p = 0.82$, $r = 0.85$ | | | | | | |
| INSTANT | 66.5 (18%) | **17.0** (45%) | 68.0 (16%) | 20.3 (34%) | 70.9 (13%) | 24.1 (22%) |
| NOCKPTI | **66.4** (18%) | **17.0** (45%) | **67.9** (16%) | **20.2** (35%) | 71.0 (13%) | 24.7 (20%) |
| WITHCKPTI | **66.4** (18%) | **17.0** (45%) | 68.3 (16%) | 20.6 (33%) | **70.6** (13%) | **23.1** (25%) |
| $p = 0.4$, $r = 0.7$ | | | | | | |
| INSTANT | 70.3 (13%) | 20.9 (33%) | 72.0 (11%) | 24.6 (21%) | **75.0** (8%) | 27.7 (11%) |
| NOCKPTI | **70.2** (14%) | **20.6** (33%) | **71.8** (12%) | **24.2** (22%) | **75.0** (8%) | 28.7 (7%) |
| WITHCKPTI | **70.2** (14%) | **20.6** (33%) | 73.6 (9%) | 25.5 (18%) | 75.1 (8%) | 26.6 (14%) |

Table IV: Job execution times (in days) under the different checkpointing policies, when failures follow a Weibull distribution of shape parameter 0.7. Gains are reported with respect to DALY.

| | $I = 300$ s | | $I = 1200$ s | | $I = 3000$ s | |
|---|---|---|---|---|---|---|
| | $2^{16}$ procs | $2^{19}$ procs | $2^{16}$ procs | $2^{19}$ procs | $2^{16}$ procs | $2^{19}$ procs |
| DALY | 125.7 | 185.0 | 125.7 | 185.0 | 125.7 | 185.0 |
| RFO | 120.1 (4%) | 114.8 (38%) | 120.1 (4%) | 114.8 (38%) | 120.1 (4%) | 114.8 (38%) |
| $p = 0.82$, $r = 0.85$ | | | | | | |
| INSTANT | **77.4** (38%) | 45.2 (76%) | 82.0 (35%) | 60.8 (67%) | **89.7** (29%) | 70.6 (62%) |
| NOCKPTI | **77.4** (38%) | **44.9** (76%) | **81.8** (35%) | **60.7** (67%) | 90.0 (28%) | 71.5 (61%) |
| WITHCKPTI | **77.4** (38%) | **44.9** (76%) | 83.6 (33%) | 64.4 (65%) | 89.8 (29%) | **66.2** (64%) |
| $p = 0.4$, $r = 0.7$ | | | | | | |
| INSTANT | 84.5 (33%) | 59.6 (68%) | 89.4 (29%) | 76.6 (58%) | **97.7** (22%) | 81.9 (56%) |
| NOCKPTI | **84.4** (33%) | **58.3** (68%) | **89.1** (29%) | 76.8 (58%) | 97.9 (22%) | 83.7 (55%) |
| WITHCKPTI | **84.4** (33%) | **58.3** (68%) | 93.8 (25%) | **75.4** (59%) | 97.8 (22%) | **77.7** (58%) |

Table V: Job execution times (in days) under the different checkpointing policies, when failures follow a Weibull distribution of shape parameter 0.5. Gains are reported with respect to DALY.

tion. We first remark that RFO achieves lower makespans than DALY with gains ranging from 1% with $2^{16}$ processors to 18% with $2^{19}$ processors. Overall, the gain due to the predictions decreases when the size of the prediction window increases, and increases with the platform size. This gain is obviously closely related to the characteristics of the predictor. When $I = 300$s, the three prediction-aware strategies are identical. When $I$ increases, NOCKPTI achieves slightly better results than INSTANT. For low values of $I$, WITHCKPTI is the worst prediction-aware heuristics. But when $I$ becomes large and if the predictor is efficient, then WITHCKPTI becomes the heuristics of choice ($I = 3000$s, $p = 0.82$, and $r = 0.85$). The reductions in the application executions times due to the predictor can be very significant. With $p = 0.85$ and $r = 0.82$ and $I = 3000$s, we save 25% of the total time with $N = 2^{19}$, and 13% with $N = 2^{16}$ using strategy WITHCKPTI. With $I = 300$s, we save up to 45% 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 33% of the execution time when $I = 300$s and $N = 2^{19}$, and 14% with $N = 2^{16}$. The gain gets smaller with $I = 3000$s and $N = 2^{16}$ but remains non negligible since we can save 8%. When $I = 3000$s and $N = 2^{19}$, however, the best solution is to ignore predictions and simply use RFO (we fall-back to the case $q = 0$). If we now consider a Weibull law with shape parameter 0.5 instead of 0.7 (see Table V), keeping all other parameters identical ($I = 3000$s, $N = 2^{19}$, $p = 0.4$ and $r = 0.7$), then the heuristics of choice is WITHCKPTI and the gain with respect to DALY is 57.9%.

## V. RELATED WORK

Considerable research has been conducted on fault prediction using different models (system log analysis [5], event-driven approach [21], [5], [10], support vector machines [6], [22]), nearest neighbors [6], etc.). 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.

| Paper | Lead Time | Precision | Recall | Prediction Window |
|---|---|---|---|---|
| [10] | 300 s | 40 % | 70% | - |
| [10] | 600 s | 35 % | 60% | - |
| [5] | 2h | 64.8 % | 65.2% | yes (size unknown) |
| [5] | 0 min | 82.3 % | 85.4 % | yes (size unknown) |
| [21] | 32 s | 93 % | 43 % | - |
| [22] | NA | 70 % | 75 % | - |
| [6] | NA | 20 % | 30 % | 1h |
| [6] | NA | 30 % | 75 % | 4h |
| [6] | NA | 40 % | 90 % | 6h |
| [6] | NA | 50 % | 30 % | 6h |
| [6] | NA | 60 % | 85% | 12h |

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

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 VI), they state that it has a positive impact on the checkpointing time (from 1500 seconds to 120 seconds). The authors of [5] also consider a lead time, and introduce a *prediction window* indicating when the predicted fault should happen. The authors of [6] 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 this work, even though [5] 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. Missing information includes the lead time and the size of the prediction window. Other information that could be useful would be: (i) the distribution of the faults in the prediction window; and (ii) the precision and recall as functions of the size 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 [23], Li et al. consider the mathematical problem to determine when and how to migrate.

In the simpler case where predictions are exact-date predictions, Gainaru et al [3] and Bouguerra et al. [24] have shown that the optimal checkpointing period becomes $T_{\text{opt}} = \sqrt{\frac{2\mu C}{1-r}}$, but their analysis is valid only if $\mu$ is very large in front of the other parameters. Our previous work [4] has refined the results of [3], focusing on a more accurate analysis of fault prediction with exact dates, and providing a detailed study on the impact of recall and precision on the waste. As shown in Section III, the analysis of the waste is dramatically more complicated when using prediction windows than when using exact-date predictions. To the best of our knowledge, this work is the first to provide a model and a detailed analysis of the waste for fault prediction with prediction windows.

## VI. Conclusion

In this work, we have studied the impact of prediction windows on checkpointing strategies. The importance of good prediction techniques is increasing with the advent of exascale platforms, for which current checkpointing techniques will not provide efficient solutions any longer [25], [26]. We have designed several heuristics that decide whether to trust predictions or not, when it is worth taking preventive checkpoints, and at which rate. We have been able to derive a comprehensive set of results and conclusions:

- We have introduced an analytical model to capture the waste incurred by each strategy, and provided a closed-form formula for each optimization problem, giving the optimal solution. Contrarily to the cases without prediction, or with exact-date predictions, the computation of the waste requires a sophisticated analysis of the various events, including the time spent in the regular and proactive modes.
- The simulations fully validate the model, and the brute-force computation of the optimal period guarantees that our prediction-aware strategies are always very close to the optimal. This holds true both for Exponential and Weibull failure distributions.
- The model is quite accurate and its validity goes beyond the conservative assumption that requires a single event per time interval; even more surprising, the accuracy of the model for prediction-aware strategies is much better than for the case without predictions, where DALY can be far from the optimal period in the case of Weibull failure distributions [19].
- Both the analytical computations and the simulations enable us to characterize when prediction is useful, and which strategy performs better, given the key parameters of the system: recall $r$, precision $p$, size of the prediction window $I$, size of proactive checkpoints $C_p$ versus regular checkpoints $C$, and platform MTBF $\mu$.

Altogether, the analytical model and the comprehensive results provided in this work enable to fully assess the impact of fault prediction with time-windows on (optimal) checkpointing strategies. Future work will be devoted to refine the assessment of the usefulness of prediction with trace-based failures 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.

## 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] A. Gainaru, F. Cappello, W. Kramer, and M. Snir, "Fault prediction under the microscope - a closer look into HPC systems," in *SC'12*, 2012.

[4] G. Aupy, Y. Robert, F. Vivien, and D. Zaidouni, "Checkpointing algorithms and fault prediction," INRIA, Research report RR-8237, February 2013.

[5] L. Yu, Z. Zheng, Z. Lan, and S. Coghlan, "Practical online failure prediction for Blue Gene/P: Period-based vs event-driven," in *DSN-Workshops*, 2011.

[6] Y. Liang, Y. Zhang, H. Xiong, and R. K. Sahoo, "Failure prediction in IBM BlueGene/L event logs," in *ICDM*, 2007, pp. 583–588.

[7] N. Kolettis and N. D. Fulton, "Software rejuvenation: Analysis, module and applications," in *FTCS '95*. IEEE CS, 1995, p. 381.

[8] V. Castelli, R. E. Harper, P. Heidelberger, S. W. Hunter, K. S. Trivedi, K. Vaidyanathan, and W. P. Zeggert, "Proactive management of software aging," *IBM J. Res. Dev.*, vol. 45, no. 2, pp. 311–332, 2001.

[9] J. Hong, S. Kim, Y. Cho, H. Yeom, and 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.

[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 Syst. and Networks Workshops (DSN-W)*, 2010.

[11] G. Aupy, Y. Robert, F. Vivien, and D. Zaidouni, "Checkpointing strategies with prediction windows," INRIA, Research report RR-8239, February 2013.

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

[13] 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 *SC'11*, 2011.

[14] G. Zheng, X. Ni, and L. Kale, "A scalable double in-memory checkpoint and restart scheme towards exascale," in *Dependable Syst. and Networks Workshops*, 2012.

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

[16] 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.

[17] E. Heien, D. Kondo, A. Gainaru, D. LaPine, B. Kramer, and F. Cappello, "Modeling and tolerating heterogeneous failures in large parallel systems," in *SC'11*, 2011.

[18] "Raw data and simulator source code," http://graal.ens-lyon.fr/~fvivien/DATA/predictionwindow/.

[19] M. Bougeret, H. Casanova, M. Rabie, Y. Robert, and F. Vivien, "Checkpointing strategies for parallel jobs," in *SC'11*, 2011.

[20] M.-S. Bouguerra, T. Gautier, D. Trystram, and J.-M. Vincent, "A flexible checkpoint/restart model in distributed systems," in *PPAM*, ser. LNCS, vol. 6067, 2010, pp. 206–215. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-14390-8_22

[21] 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.

[22] 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.

[23] Y. Li, Z. Lan, P. Gujrati, and X. Sun, "Fault-aware runtime strategies for high-performance computing," *IEEE TPDS*, vol. 20, no. 4, pp. 460–473, 2009.

[24] M. Bouguerra, A. Gainaru, L. Gomez, F. Cappello, S. Matsuoka, and N. Maruyama, "Improving the computing efficiency of HPC systems using a combination of proactive and preventive checkpointing," in *Proceedings of the 27th International Parallel & Distributed Processing Symposium (IPDPS'13)*. IEEE, May 2013, pp. 501–512.

[25] F. Cappello, A. Geist, B. Gropp, L. Kale, B. Kramer, and M. Snir, "Toward Exascale Resilience," *Int. Journal of High Performance Computing Applications*, vol. 23, no. 4, pp. 374–388, 2009.

[26] J. Dongarra, P. Beckman, P. Aerts, F. Cappello, T. Lippert, S. Matsuoka, P. Messina, T. Moore, R. Stevens, A. Trefethen, and M. Valero, "The international exascale software project: a call to cooperative action by the global high-performance community," *Int. J. High Perform. Comput. Appl.*, vol. 23, no. 4, pp. 309–322, 2009.