Checkpointing strategies for parallel jobs

Marin Bougeret, Henri Casanova, Mikael Rabie, Yves Robert, Frédéric Vivien

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Checkpointing strategies for parallel jobs

Marin Bougeret — Henri Casanova — Mikael Rabie — Yves Robert — Frédéric Vivien

N° 7520

January 2011

Distributed and High Performance Computing

ISSN 0249-6399 ISRN INRIA/RR--7520--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
GRENOBLE - RHÔNE-ALPES

Checkpointing strategies for parallel jobs

Marin Bougeret[*] , Henri Casanova[†] , Mikael Rabie[*] , Yves Robert[*‡] , Frédéric Vivien[§]

Theme : Distributed and High Performance Computing
Équipe-Projet GRAAL

Rapport de recherche n° 7520 — January 2011 — 41 pages

Abstract: This work provides a rigorous analysis of checkpointing strategies for sequential and parallel jobs. The objective is to minimize the expected job execution time in an environment that is subject to processor failures. For sequential jobs, we give the optimal solution if failure inter-arrival times are exponentially distributed. To the best of our knowledge, our result in the Exponential case is the first published rigorous proof that periodic checkpointing is optimal. In the general case (i.e., non-exponentially distributed), we give a dynamic programming algorithm that computes an accurate solution. For parallel jobs, we also provide the optimal solution in the Exponential case, which is given for various models of job parallelism and of checkpointing overhead. In the general case, we develop a dynamic programming algorithm to maximize the amount of work completed before the next failure, which provides a good heuristic solution for minimizing the expected execution time. To assess our work, we perform an extensive set of simulation experiments considering both Exponential and Weibull laws, as several studies have shown that Weibull distributions are the most appropriate to model the distribution of failures. These experiments corroborate all our theoretical results. Furthermore, they show that our dynamic programming algorithm far outperforms existing solutions when failure inter-arrival times follow a Weibull distribution.

Key-words: Fault-tolerance, checkpointing, sequential job, parallel job, Weibull

[*] LIP, Ecole Normale Supérieure de Lyon
[†] Univ. of Hawai'i at Mānoa, Honolulu, USA
[‡] Yves Robert is with the Institut Universitaire de France. This work was supported in part by the ANR StochaGrid and RESCUE projects, and by the INRIA-Illinois Joint Laboratory for Petascale Computing.
[§] INRIA, Lyon, France

Centre de recherche INRIA Grenoble – Rhône-Alpes
655, avenue de l'Europe, 38334 Montbonnot Saint Ismier
Téléphone : +33 4 76 61 52 00 — Télécopie +33 4 76 61 52 52

Stratégies de checkpoint pour applications parallèles

Résumé : Nous présentons dans ce travail une analyse rigoureuse des stratégies de checkpoint, pour les applications séquentielles et parallèles. L'objectif est de minimiser l'espérance du temps de complétion d'une application s'exécutant sur des processeurs pouvant être de victimes de pannes. Dans le cas séquentiel, une résolution exacte est proposée lorsque les intervalles inter-pannes sont distribués selon une loi exponentielle. Il semble que ce résultat soit la première preuve rigoureuse de l'optimalité des stratégies périodiques de checkpoint dans ce cadre. Dans le cas général (c'est-à-dire pour des lois quelconques), nous fournissons une programmation dynamique permettant des calculs optimaux, à un quantum de temps fixé près. Dans le cas des applications parallèles, nous étendons le résultat exact (pour le cas exponentiel), et ce pour différents modèles d'applications et de coûts de checkpoints. Pour le cas général, nous proposons une deuxième programmation dynamique (plus rapide) dont l'objectif est de maximiser l'espérance du travail fait avant la prochaine panne, qui s'avère fournir de bonnes approximations pour le problème initial. Nous validons nos résultats grâce à de nombreuses simulations, réalisées pour des lois inter-pannes de distribution exponentielles et de Weibull (cette dernière distribution étant selon de nombreuses études plus appropriée pour modéliser des durées inter-pannes). Ces simulations confirment nos résultats théoriques. De plus, ils apparaît que notre algorithme de programmation dynamique fournit de bien meilleures performances que les solutions existantes pour le cas des lois de Weibull.

Mots-clés : Tolérance aux pannes, checkpoint, tâche séquentielle, tâche parallèle, Weibull

1 Introduction

Resilience is one of the key challenges for post-petascale high-performance computing (HPC) systems [10, 21]. Indeed, failures are increasingly likely to occur during the execution of parallel applications that enroll increasingly large numbers of processors. Even if the MTBF (*mean time between failures*) of a single processor is assumed to be large, say 100 years, a failure would occur every 50 minutes on a machine with 1,000,000 processors. An Exascale machine with 10^9 processors would experience a failure every 30 seconds even if the processor MTBF is one millennium.

Fault-tolerance in the context of HPC applications typically combines redundancy and rollback recovery. Low-overhead redundancy mechanisms detect and correct local faults (e.g., memory errors, arithmetic errors). When such faults cannot be corrected they lead to failures, in which case rollback recovery is used to resume execution from a previously saved fault-free execution state. Rollback recovery implies frequent (usually periodic) *checkpointing* events at which the application state is saved to resilient storage. Checkpointing leads to non-negligible overhead during fault-free execution, and to the loss of recent execution progresses when a failure occurs. Frequent checkpoints cause a higher overhead but also imply a smaller loss when a failure occurs. The design of efficient *checkpointing strategies*, that specify when checkpoints should be taken, is thus key to high performance. Our main contributions are analytical performance models, together with simulation results, that guide the design of such strategies.

In this paper, we target applications, or *jobs*, that execute on one or several *processors*. We use the generic term *processor* to indicate any computing resource, be it a single core, a multi-core array, or even a cluster node. In other words, our work is agnostic to the granularity of the platform. The execution of a job is subject to *failures* that can be of different nature, and whose occurrences obey various probability distribution laws. While in the literature failures are attributed to faults that are either software or hardware, and that can be either transient and unrecoverable, in this work we adopt a unified treatment. When a failure occurs on a processor, this processor experiences a *downtime* period followed by a *recovery* period. Job execution can resume from the last checkpointed state. If the failure is due to a transient fault, typically the case for software faults, the processor is rebooted during its downtime period [14, 8]. If the failure is due to an unrecoverable fault, typically the case for hardware faults, then the downtime corresponds to the time needed either to repair the processor, or to replace it by a spare. In both cases, we assume that the faulty processor has been rejuvenated and becomes a fault-free resource whose lifetime beings at the beginning of the recovery period.

Our objective is to minimize the expectation of the job execution time, or *makespan*. In this context, our novel contributions are as follows. For sequential jobs, we provide the optimal solution for exponentially distributed failure inter-arrival times, and an accurate dynamic programming algorithm in the general case. The optimal solution in the Exponential case, i.e., periodic checkpointing, is widely known in the "folklore" but we were not able to find a rigorous proof in the literature (we prove the result using a novel approach based on a recursive formulation of the problem). The dynamic programming algorithm is completely novel, and is the key to the first accurate solution of the makespan minimization

problem with Weibull laws. In the context of parallel jobs, we provide the optimal solution for the Exponential case in a variety of execution scenarios with different models of job parallelism (embarrassingly parallel jobs, jobs that obey Amdahl's law, typical numerical kernels such as matrix product or LU decomposition), and on the overhead of checkpointing a parallel job (which may or may not depend on the total number of processors in use). In the general case, we explain why current approaches that rejuvenate all processors after a failure are likely not appropriate in practice. Given that minimizing the expected makespan in this case is difficult, we instead provide a dynamic programming algorithm to maximize the amount of work successfully completed before the next failure. This approach turns out to provide a good heuristic solution to the expected makespan minimization problem. In particular, this solution greatly outperforms the existing solutions when failure inter-arrival times follow a Weibull distribution, the most realistic case according to the literature. All our theoretical results are corroborated by an extensive set of experiments.

Sections 2 and 3 give theoretical results for single- and multi-processor jobs, respectively. Section 4 presents the simulation settings and Section 5 the experimental results obtained in simulation. Section 6 discusses related work. Finally, Section 7 concludes the paper with a summary of our findings and a discussion of future directions.

2 Single-processor Jobs

In this section we consider jobs that execute on a single processor. This processor is subject to failures, which occur at time intervals that obey some probabilistic distribution law. We first formalize the model and then define two relevant optimization problems, namely makespan minimization and work maximization. We then seek analytical results for solving both problems.

2.1 Model

Probabilistic model Failures occur on a processor at times $(t_n)_{n\geq 1}$, with $t_n = \sum_{m=1}^{n} X_m$, where the random variables $(X_m)_{m\geq 1}$ are *iid* (independent and identically distributed). We consider the processor from time 0 on, and we do not assume that the failure stochastic process is memoryless. Given a current time t, we simply write $X = X_{n(t)}$, where $n(t) = \min\{n | t_n \geq t\}$, for the current time-interval variable. We use $P_{suc}(\alpha, t^{alv})$ to denote the probability that the processor does not fail for the next α units of time, knowing that the last failure occurred t^{alv} units of time ago. In other words,

$$P_{suc}(\alpha, t^{alv}) = \mathbb{P}(X \geq t^{alv} + \alpha \mid X \geq t^{alv}) \, .$$

Execution model A job must complete $\mathcal{W}$ units of (divisible) work, which can be split arbitrarily into separate chunks. The application state is checkpointed after the execution of every chunk. The definition of chunk sizes is therefore equivalent to the definition of checkpointing dates. We use C to denote the time to perform a checkpoint, D the downtime, and R the recovery time. We define $\alpha(W)$ as the time needed for executing a chunk of size W and

then checkpointing it: $\alpha(W) = W+C$ (without loss of generality, this expression assumes a unit-speed processor). We assume that failures can happen during a recovery or a checkpointing, but not during a downtime (otherwise, there would be no difference between a downtime and a recovery from an algorithmic point of view).

2.2 Problem statement

We focus on two optimization problems defined informally as:

- MAKESPAN: Minimize the expected time needed to execute $\mathcal{W}$ units of work; and
- NEXTFAILURE: Maximize the expected ammount of work completed before the next failure.

Solving MAKESPAN is our ultimate goal. When MAKESPAN cannot be solved exactly, however, a solution to NEXTFAILURE provides a reasonable heuristic solution to MAKESPAN. Solving NEXTFAILURE amounts to optimizing the execution time "failure by failure", selecting the next chunk size as if the next failure were to imply termination of the execution. Intuitively, maximizing the (expected) amount of work executed between two consecutive failures should lead to a good approximation of the solution to MAKESPAN, at least for large job sizes $\mathcal{W}$. Consequently, in this section we formalize both problems and provide solutions in the next two sections.

For both problems, a solution can be fully defined by a function $f(W, t^{alv})$ that returns the size of the next chunk to execute, given the amount of work that has not yet been executed successfully ($W \leq \mathcal{W}$) and the amount of time elapsed since the last failure (t^{alv}). Indeed, one can derive an entire solution by invoking f at each point of decision (i.e., after each checkpoint or recovery). Our goal is to determine this function for the optimal solution. Assuming that f is given for MAKESPAN, and for a given W, let us denote by $W_1 = f(W, t^{alv})$ the size of the first chunk and let $X^f_{exec}(W, t^{alv})$ be the random variable that quantifies the time needed for executing W units of work. We can now write the following recursion:

$$
\begin{aligned}
&X^f_{exec}(0, t^{alv}) \ = 0 \\
&X^f_{exec}(W, t^{alv}) = \\
&\qquad \begin{cases} \alpha(W_1) + X^f_{exec}(W - W_1, t^{alv} + \alpha(W_1)) \\ \quad \text{if the processor does not fail during,} \\ \quad \text{the next } \alpha(W_1) \text{ units of time} \\ X_{wasted}(\alpha(W_1), t^{alv}, R, D) + X^f_{exec}(W, R) \\ \quad \text{otherwise.} \end{cases}
\end{aligned}
$$

The two cases in the formula are explained as follows:

- If the processor does not fail during the execution and checkpointing of the first chunk (i.e., for $\alpha(W_1)$ time units), there remains to execute a work of size $W - W_1$ and the time since the last failure is $t^{alv} + \alpha(W_1)$;
- If the processor fails before successfully completing the first chunk and its checkpointing, then some additional delays are incurred, as captured by the variable $X_{wasted}(\alpha(W_1), t^{alv}, R, D)$. Time is wasted because of the

execution up to the failure, then because of the downtime and of the recovery. Worse, another failure may happen during the recovery. We provide the (complicated) expression for X_{wasted} in the next section. Regardless, once a successful recovery has been completed, there still remains a work of size W to execute, and the time since the last failure is simply R.

For problem NEXTFAILURE, the random variable $X^f_{work}(W, t^{alv})$ quantifies the amount of work successfully executed before the next failure, where W denotes the remaining work, and t^{alv} the amount of time elapsed since the last failure. Defining again $W_1 = f(W, t^{alv})$, we can write another recursion:

$$
\begin{aligned}
&X^f_{work}(0, t^{alv}) \ = 0 \\
&X^f_{work}(W, t^{alv}) = \\
&\qquad \begin{cases} W_1 + X^f_{work}(W - W_1, t^{alv} + \alpha(W_1)) \\ \quad \text{if the processor does not fail during} \\ \quad \text{the next } \alpha(W_1) \text{ units of time,} \\ 0 \quad \text{otherwise.} \end{cases}
\end{aligned}
$$

This recursion is simpler than the previous one because a failure during the computation of the first chunk means that no work (i.e., no fraction of W) will have been successfully executed before the next failure. We now can define both our optimization problems formally:

- MAKESPAN: find f that minimizes $\mathbb{E}(X^f_{exec}(\mathcal{W}, 0))$,
- NEXTFAILURE: find f that maximizes $\mathbb{E}(X^f_{work}(\mathcal{W}, 0))$.

2.3 The Makespan problem

Before proposing solutions to the MAKESPAN problem, we establish two straightforward propositions. As mentioned earlier, one of the challenges for solving MAKESPAN is the computation of $X_{wasted}(\alpha(W_1), t^{alv}, R, D)$. We rely on the following decomposition:

$$X_{wasted}(\alpha(W_1), t^{alv}, R, D) = X_{lost}(\alpha(W_1), t^{alv}) + X_{rec}(R, D)$$

where

- $X_{lost}(\alpha, t)$ is the amount of time spent computing before a failure, knowing that the next failure occurs before α units of time, and that the last failure has occurred t units of time ago (see Figure 1).
- $X_{rec}(R, D)$ is the amount of time needed by the system to recover from the failure (accounting for the fact that other failures may also occur during recovery).

Recall that $P_{suc}(\alpha, t)$ denotes the probability of successfully computing during α time units, knowing that the last failure occurred t units of time ago. Based on the recursion for X_{exec} given in the previous section, by simply weighting the expectations by the probabilities of occurrence of each of the two cases, we obtain the following proposition:

Proposition 1. *The* MAKESPAN *problem is equivalent to minimizing the following quantity:*

$$\begin{aligned}&\mathbb{E}(X^f_{exec}(\mathcal{W}, t^{alv})) = \\ &P_{suc}(\alpha(W_1), t^{alv})(\alpha(W_1)+\mathbb{E}(X_{exec}(\mathcal{W}–W_1, t^{alv}+\alpha(W_1)))) \\ &+(1-P_{suc}(\alpha(W_1), t^{alv}))\Big(\mathbb{E}(X_{lost}(\alpha(W_1), t^{alv}) \\ &+\mathbb{E}(X_{rec}(R, D)) + \mathbb{E}(X_{exec}(\mathcal{W}, R))\Big).\end{aligned}$$

$\mathbb{E}(X_{rec}(R, D))$ can be computed in terms of $\mathbb{E}(X_{lost}(R, 0))$ in a similar manner. We can compute X_{rec} as follows:

$$X_{rec}(R, D) = \begin{cases} D + R & \text{with probability } P_{suc}(R, 0), \\ D + X_{lost}(R, 0) + X_{rec}(R, D) & \\ \qquad \text{with probability } 1 - P_{suc}(R, 0). & \end{cases}$$

Thus, we obtain

$$\begin{aligned}&\mathbb{E}(X_{rec}(R, D)) = P_{suc}(R, 0)(D + R) \\ &+(1 - P_{suc}(R, 0))\,(D + \mathbb{E}(X_{lost}(R, 0)) + \mathbb{E}(X_{rec}(R, D)))\end{aligned}$$

which leads to the following proposition:

Proposition 2. *$\mathbb{E}(X_{rec}(R, D))$ is given by*

$$\mathbb{E}(X_{rec}(R, D)) = D + R + \frac{1 - P_{suc}(R, 0)}{P_{suc}(R, 0)}(D + \mathbb{E}(X_{lost}(R, 0))).$$

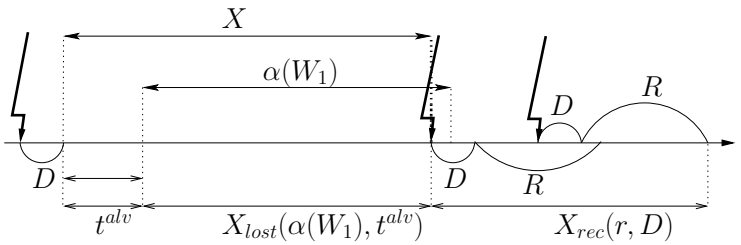


Figure 1: Example of a restart scenario. X is the random variable that describes the time elapsed between two failures.

2.3.1 Results for the Exponential distribution

In this section we assume that the failure inter-arrival times follow an Exponential distribution with parameter λ, i.e., each $X_n = X$ has probability density $f_X(t) = \lambda e^{-\lambda t}dt$ and cumulative density $F_X(t) = 1 - e^{-\lambda t}$ for all $t \geq 0$. The key advantage of the Exponential distribution, exploited time and again in the literature, is its "memoryless" property. The time at which the next failure occurs does not depend on the time at which the last failure occurred. This simplification makes it possible to solve the MAKESPAN problem analytically because, in this case, all possible executions for a given strategy use the same sequence of chunk sizes. The only difference between two different executions is the number of times each chunk is tentatively executed before its successful completion. To

see this, note that size of the first chunk, $W_1 = f(\mathcal{W}, t^{alv})$, does not depend upon t^{alv} (due to the memoryless property). Either its first execution is successful or it fails. If it fails, then the optimal solution consists in retrying a chunk of same size W_1 since there remains $\mathcal{W}$ units of work to be executed and W_1 does not depend on t^{alv}. Once a chunk of size W_1 has been successfully completed, the next attempted chunk is of size $W_2 = f(\mathcal{W} - W_1, t^{alv})$, whose size does not depend on t^{alv}, and for which the same reasoning holds.

Given the above, all executions can be described as sequences of the form $W_1^{(\ell_1)} W_2^{(\ell_2)} \dots W_k^{(\ell_k)} \dots$, where $W^{(\ell)}$ means that a chunk of size W was tentatively executed ℓ times, the first $\ell - 1$ tentatives being unsuccessful and the last one being successful. Because each chunk is executed at least once, the time to execute it is always bounded below by C, the time to take a checkpoint. Say that the optimal strategy uses K successive chunk sizes, with K to be determined. Any execution following this optimal strategy will thus have a makespan as least as large as KC. Hence $\mathbb{E}(X_{exec}^{f}(\mathcal{W}))$, the expectation of the makespan with the optimal strategy, is also greater than or equal to KC.

We first prove that, as might be expected, K is finite. Let us consider a simple, non-optimal, strategy, defined by $f(W, t) = W$. In other words, this strategy executes the whole work $\mathcal{W}$ as a single chunk, repeating its execution until it succeeds. Let us denote by $\mathbb{E}(X_{exec}^{id}(\mathcal{W}, t^{alv}))$ the expected makespan when this strategy is used. It turns out that, because of the Exponentially *iid* assumption, $\mathbb{E}(X_{lost}(W))$ and $\mathbb{E}(X_{rec}(R, D))$ can be computed analytically:

Lemma 1. *With the Exponential distribution:*

$$\mathbb{E}(X_{lost}(W)) = \frac{1}{\lambda} - \frac{W}{e^{\lambda W} - 1} \text{ and}$$
$$\mathbb{E}(X_{rec}(R, D)) = D + R + \frac{1 - e^{-\lambda R}}{e^{-\lambda R}}(D + \mathbb{E}(X_{lost}(R, 0))).$$

Proof.

$$\begin{aligned}
\mathbb{E}(X_{lost}(W)) &= \textstyle\int_0^\infty x\, \mathbb{P}(X = x | X < W) dx \\
&= \tfrac{1}{\mathbb{P}(X<W)} \textstyle\int_0^W x f_X(x) dx \\
&= \tfrac{1}{1-e^{-\lambda W}} \textstyle\int_0^W x \lambda e^{-\lambda x} dx \\
&= \tfrac{1}{1-e^{-\lambda W}} ([-x e^{-\lambda x}]_0^W + \textstyle\int_0^W e^{-\lambda x}) \\
&= \tfrac{1}{1-e^{-\lambda W}} (-W e^{-\lambda W} + \tfrac{1}{\lambda} - \tfrac{e^{-\lambda W}}{\lambda}) \\
&= \tfrac{1}{\lambda} - \tfrac{W}{e^{\lambda W} - 1}
\end{aligned}$$

The formula for $\mathbb{E}(X_{rec}(R, D))$ is directly obtained from Proposition 2 by replacing $P_{suc}(R, 0)$ by $e^{-\lambda R}$. □

Using Proposition 1, we have:

$$\begin{aligned}
\mathbb{E}(X_{exec}^{id}(\mathcal{W})) =& P_{suc}(\alpha(\mathcal{W}))(\alpha(\mathcal{W})) \\
&+ (1 - P_{suc}(\alpha(\mathcal{W}))) \\
&\quad \times \left(\mathbb{E}(X_{lost}(\alpha(\mathcal{W}))) + \mathbb{E}(X_{rec}(R, D)) + \mathbb{E}(X_{exec}^{id}(\mathcal{W}))\right).
\end{aligned}$$

Given that $P_{suc}(\alpha(\mathcal{W})) = e^{-\lambda \alpha(\mathcal{W})}$, we obtain:

$$\begin{aligned}
\mathbb{E}(X_{exec}^{id}(\mathcal{W})) &= \alpha(\mathcal{W}) \\
&\quad + \left(\mathbb{E}(X_{lost}(\alpha(\mathcal{W}))) + \mathbb{E}(X_{rec}(R, D))\right) \frac{1 - e^{-\lambda \alpha(\mathcal{W})}}{e^{-\lambda \alpha(\mathcal{W})}}.
\end{aligned}$$

This last expression shows that $\mathbb{E}(X_{exec}^{id}(\mathcal{W}))$ is finite, implying that the expected makespan for the optimal strategy, $\mathbb{E}(X_{exec}^{f}(\mathcal{W}))$, is also finite. Since it is bounded below by KC, we conclude that K is finite, meaning that the optimal solution uses a bounded number of chunks.

We can now state the main result of this section:

Theorem 1. *Let $\mathcal{W}$ be the amount of work to execute on a processor whose failure process follows an Exponential law with parameter λ. Let $n_0 = \frac{\lambda\mathcal{W}}{1+\mathbb{W}(-e^{-\lambda C-1})}$ where $\mathbb{W}$ is Lambert W function, defined as $\mathbb{W}(z)e^{\mathbb{W}(z)} = z$. Then the optimal strategy to minimize the expectation of the execution time is to split $\mathcal{W}$ into n^* equal to $\max(1, \lfloor n_0 \rfloor)$ or $\lceil n_0 \rceil$ (whichever leads to the smaller value) same-size chunks. The minimal expectation of the makespan is*

$$\mathbb{E}^*(\mathcal{W}, \lambda, D, C, R) = n^* \left(\frac{1}{\lambda} + \mathbb{E}(X_{rec}(R, D))\right)\left(e^{\lambda(\frac{\mathcal{W}}{n^*}+C)} - 1\right)$$

where $\mathbb{E}(X_{rec}(R, D))$ is given by Lemma 1.

Proof. We already know that the optimal solution uses a bounded number n_f of chunks, where chunk $1 \leq i \leq n_f$ has size $W_i = f(\mathcal{W} - \sum_{j=1}^{i-1} W_j)$. From Proposition 1, we derive, by using the memoryless property of the Exponential distribution, that

$$\begin{aligned}\mathbb{E}(X_{exec}^{f}(\mathcal{W})) = P_{suc}(\alpha(W_1))(\alpha(W_1) + \mathbb{E}(X_{exec}^{f}(\mathcal{W} - W_1)) \\ + (1 - P_{suc}(\alpha(W_1)))\Big(\mathbb{E}(X_{lost}(\alpha(W_1))) \\ + \mathbb{E}(X_{rec}(R, D)) + \mathbb{E}(X_{exec}^{f}(\mathcal{W})\Big).\end{aligned}$$

Let us define $\rho = \mathbb{E}(X_{exec}^{f}(\mathcal{W}))$ and $R' = \mathbb{E}(X_{rec}(R, D))$. Using Lemma 1 we obtain:

$$\begin{aligned}\rho = \alpha(W_1) + \mathbb{E}(W - W_1) \\ + \tfrac{1-P_{suc}(\alpha(W_1))}{P_{suc}(\alpha(W_1))}(\mathbb{E}(X_{lost}(\alpha(W_1))) + R'))\ .\end{aligned}$$

We have $\frac{1-P_{suc}(\alpha(W_1))}{P_{suc}(\alpha(W_1))} = e^{\lambda\alpha(W_i)} - 1$ and, using Lemma 1, $\mathbb{E}(X_{lost}(W)) = \frac{1}{\lambda} - \frac{W}{e^{\lambda W}-1}$. Developing the expression for ρ leads to

$$\rho = \sum_{i=1}^{n_f} \alpha(W_i) + \sum_{i=1}^{n_f} \left(\left(\frac{1}{\lambda} - \frac{\alpha(W_i)}{e^{\lambda\alpha(W_i)}-1} + R'\right)(e^{\lambda\alpha(W_i)} - 1)\right)$$

Terms cancel out and the above simplifies into:

$$\rho = \left(\frac{1}{\lambda} + R'\right) \sum_{i=1}^{n_f} (e^{\lambda\alpha(W_i)} - 1)$$

As $\alpha(W) = W + C$ is a convex function of W, ρ is minimized when all the chunks W_i have the same size $\frac{\mathcal{W}}{n_f}$, in which case $\rho = n_f(\frac{1}{\lambda} + R')(e^{\lambda(\frac{\mathcal{W}}{n_f}+C)} - 1)$. We look for the value of n_f that minimizes $\psi(n_f) = n_f(e^{\lambda(\frac{\mathcal{W}}{n_f}+C)} - 1)$. We call n_0 this value and, differentiating, we must solve $\psi'(n_0) = 0$ where

$$\psi'(n_0) = e^{\lambda(\frac{\mathcal{W}}{n_0}+C)}\left(1 - \frac{\lambda\mathcal{W}}{n_0}\right) - 1. \tag{1}$$

This equation can be rewritten as $ye^y = -e^{-\lambda C-1}$, where $y = \frac{\lambda \mathcal{W}}{n_0} - 1$. The only solution is $y = \mathbb{W}(-e^{-\lambda C-1})$, where $\mathbb{W}$ is Lambert W function. Differentiating again, we easily see that ψ'' is always non-negative. The optimal value is thus obtained by one of the two integers surrounding the zero of ψ', which proves the theorem. □

Remark 1. *Although periodic checkpoints have been widely used in the literature, Theorem 1 is, to the best of our knowledge, the first proof that the optimal deterministic strategy uses a finite number of checkpoints and is periodic. In addition, as a consequence of Proposition 4.4.3 in [20], this strategy can be shown to be optimal among all deterministic and non-deterministic strategies.*

2.3.2 Results for arbitrary distributions

Solving the MAKESPAN problem for arbitrary distributions is difficult because, unlike in the memoryless case, there is no reason for the optimal solution to use a single chunk size. In fact, the optimal solution is very likely to use chunk sizes that depend on additional information (i.e., failure occurrences to date) that becomes available during the execution. Based on Proposition 1, and using the notation shortcut $\mathbb{E}(W, t^{alv}) = \mathbb{E}(X^f_{exec}(W, t^{alv}))$, we can write

$$\mathbb{E}(\mathcal{W}, t^{alv}) = \min_{0<W_1\leq\mathcal{W}} \left(\begin{array}{l} P_{suc}(\alpha(W_1), t^{alv})(\alpha(W_1) \\ +\mathbb{E}(\mathcal{W} - W_1, t^{alv} + \alpha(W_1))) \\ +(1 - P_{suc}(\alpha(W_1), t^{alv}))\times \\ (\mathbb{E}(X_{lost}(\alpha(W_1), t^{alv}))+\mathbb{E}(X_{rec}(R, D))+\mathbb{E}(\mathcal{W}, R)) \end{array} \right)$$

which can be solved via dynamic programming. We introduce a time quantum u, meaning that all chunk sizes W_i are integer multiples of u. This restricts the search for the optimal execution to a finite set of possible executions. The trade-off is that a smaller value of u leads to a more accurate solution, but also to a higher number of states in the algorithm, hence to a higher computing time.

Proposition 3. *Using a time quantum u, Algorithm 1 (*PDMAKESPAN*) computes, for any failure distribution law, an optimal solution to* MAKESPAN *in time $\mathcal{O}(\frac{\mathcal{W}}{u}^3(1+\frac{c}{u})a)$, where a is an upper bound on the time needed to compute $E(X_{lost}(\alpha, t))$, for any α and t.*

Proof. Our goal is to compute $f(W_0, t_0^{alv})$, for any W_0 and t_0^{alv} that are possible during the execution of $\mathcal{W}$. A first attempt would be to design an algorithm A such that $A(x, y)$ computes an optimal solution assuming that the remaining work to process is $W = xu$ and the time since the last failure is $t^{alv} = yu$, with $x \in [|0, \frac{\mathcal{W}}{u}|]$ and $y \in [|0, \delta|]$. To bound δ, we observe that the maximum possible elapsed time without failure occurs when (successfully) executing $\frac{\mathcal{W}}{u}$ chunks of size u, leading to $\delta = \frac{\frac{\mathcal{W}}{u}(u+c)+t_0^{alv}}{u}$. To avoid using the arbitrarily large value t_0^{alv}, we instead introduce a boolean b which is equal to 1 only if a failure has never occurred since the initial state (W_0, t_0^{alv}). We now define the optimal solution as PDMAKESPAN(x, b, y, t_0^{alv}), where the remaining work is $W = xu$ and the last failure occurred at time $t^{alv} = bt_0^{alv} + yu$, with $x \in [|0, \frac{\mathcal{W}}{u}|]$ and $y \in [|0, \frac{W}{u}(1 + \frac{c}{u})|]$. Note that all elements of array *solution* are initialized to *unknown*, and that X_{rec} can be computed using Proposition 2. Thus, the size of the chunk $f(W_0, t_0^{alv})$ is obtained by computing

$snd(\text{PDMakespan}(\frac{W_0}{u}, 1, 0, t_0^{alv}))$ (PDMakespan returns a couple formed by the optimal expectation, and the corresponding optimal chunk size), and the complexity result is immediate. □

Algorithm 1: PDMakespan (x,b,y,t_0^{alv})

```
if x = 0 then
    return 0
if solution[x][b][y] = unknown then
    best ← ∞
    t^alv ← b t_0^alv + yu
    for i = 1 to x do
        α = iu + C
        exp_succ ← first(PDMakespan(x − i, b, y + α/u, t_0^alv))
        exp_fail ← first(PDMakespan(x, 0, R/u, t_0^alv))
        cur ← P_suc(α, t^alv)(α + exp_succ)
              +(1 − P_suc(α, t^alv))( E(X_lost(α, t^alv))
              + E(X_rec(R, D)) + exp_fail )
        if cur < best then
            best ← cur;   chunksize ← i
    solution[x][b][y] ← (best, chunksize)
return solution[x][b][y]
```

Algorithm 1 provides an approximation of the optimal solution to the Makespan problem. We evaluate this approximation experimentally in Section 5, including a direct comparison with the optimal solution in the case of Exponential failures (in which case the optimal can be computed thanks to Theorem 1).

2.4 The NextFailure problem

As for the Makespan problem, we can use the recursion given in Section 2.2 to derive an expression for the expected amount of work successfully computed before the next failure. Denoting $\mathbb{E}(X^f_{work}(W, t^{alv}))$ by $\mathbb{E}(W, t^{alv})$, we can compute the expectation as

$$\mathbb{E}(W, t^{alv}) = P_{suc}(\alpha(W_1), t^{alv})(W_1 + \mathbb{E}(W - W_1, t^{alv} + \alpha(W_1)))$$

Unlike for Makespan, the objective function, to be maximized, can easily be written as a closed-form formula, even for arbitrary distributions. Developing the expression above leads to the following result:

Proposition 4. *The* NextFailure *problem is equivalent to maximizing the following quantity:*

$$\mathbb{E}(\mathcal{W}, 0) = \sum_{i=1}^{n_0} W_i \times \prod_{j=1}^{i} P_{suc}(\alpha(W_j), t_j)$$

where $t_j = \sum_{l=1}^{j-1} \alpha(W_l)$ *is the total time elapsed (without failure) before the start of the execution of chunk* W_l*, and* n_0 *is the (unknown) target number of chunks.*

Unfortunately, there does not seem to be an exact solution to this problem. However, in the case of the Exponential distribution, and if the work were infinite, the solution is as follows:

Proposition 5. *For an exponential distribution, and for $\mathcal{W} = \infty$, the optimal solution is periodic, and the (unique) size of chunk is $W^* = \frac{\mathbb{W}(e^{-\lambda C-1})+1}{\lambda}$.*

Proof. In this case, f no longer depends on the remaining work W or t^{alv}, so f is constant. Thus, we look for a unique size of chunk W that maximizes $\mathbb{E}(\infty)$, the expectation of work done before the first failure when $\mathcal{W} = \infty$, and using only chunks of size W. From the equation $\mathbb{E}(\infty) = \mathcal{P}_{\text{succ}}(\alpha(W))(W + \mathbb{E}(\infty))$, we find that $\mathbb{E}(\infty) = \frac{P_{suc}(\alpha)}{1-P_{suc}(\alpha)}W = \frac{e^{-\lambda(W+C)}}{1-e^{-\lambda(W+C)}}W$. Differentiating $\mathbb{E}_W(\infty)$, we get exactly Equation 1 of Theorem 1 (with n_0 replaced by 1), and thus we solve again $\mathbb{E}'_W(\infty) = 0$ using Lambert $\mathbb{W}$ function to obtain the desired value of W^*. □

Note that the period of Proposition 5 is the same as in Theorem 1 for the Makespan problem, which confirms that NextFailure is a good candidate to approximate Makespan.

However, in practice the work is always finite, and there is then no reason for the size of the next chunk to not depend on the amount of work that remains to execute. Fortunately, just as for the Makespan problem, the above recursive definition of $\mathbb{E}(\mathcal{W}, t^{alv})$ naturally leads to a dynamic programming algorithm. Here the dynamic programming scheme can be simplified because the size of the i-th chunk is only needed when no failure has occurred during the execution of the first $i-1$ chunks, regardless of the value of the t^{alv} parameter. More formally:

Proposition 6. *Using a time quantum u, Algorithm 2 (*PDNextFailure*) computes an optimal solution to* NextFailure *in time $\mathcal{O}(\frac{\mathcal{W}}{u}^3)$ for any failure distribution law.*

Proof. Our goal is to compute $f(W_0, t_0^{alv})$, for any W_0 and t_0^{alv} that are possible during the execution of $\mathcal{W}$. We define PDNextFailure(x, n, t_0^{alv}) as the optimal solution for time quantum u, where $W = xu$ is the remaining work, n is the number of chunks already computed successfully, and t_0^{alv} is the amount of time elapsed since last failure. Notice that x and n are in $[|0, \frac{\mathcal{W}}{u}|]$. Given x and n, the last failure necessarily occurred $t_0^{alv} + (\mathcal{W} - xu) + nC$ units of time ago. Finally, we suppose that all elements of array *solution* are initialized to *unknown*. The size of the chunk $f(W_0, t_0^{alv})$ is obtained by computing snd(PDNextFailure$(\frac{W_0}{u}, 0, t_0^{alv})$) (as PDMakespan, PDNextFailure returns a couple), and the complexity result is immediate. □

3 Multi-processor jobs

In this section, we study the Makespan problem for parallel jobs. Let $\mathcal{W}$ be the total job size, i.e., the amount of work to be processed. We assume parallel moldable jobs, meaning that jobs can execute using any number of processors, p. Given this assumption, we consider the following relevant scenarios for checkpointing/recovery overheads and for parallel execution times:

Algorithm 2: PDNextFailure (x,n,t_0^{alv})

```
if x = 0 then
    return 0
if solution[x][n] = unknown then
    best ← ∞
    t^alv ← t_0^alv + (W − xu) + nC
    for i = 1 to x do
        α = iu + C
        work = first(PDNextFailure(x − i, n + 1, t_0^alv))
        cur ← P_suc(α, t^alv) × (iu + work)
        if cur < best then
            best ← cur;   chunksize ← i
    solution[x][n] ← (best, chunksize)
return solution[x][n]
```

Checkpointing/recovery overheads – Checkpoints are synchronized over all processors. We use $C(p)$ and $R(p)$ to denote the time for saving a checkpoint and for recovering from a checkpoint on p processors, respectively (we assume that the downtime D does not depend on p). Assuming that the application's memory footprint is V bytes, with each processor holding V/p bytes, we consider two scenarios:

- Proportional overhead: $C(p) \propto V/p$, which is representative of cases in which the bandwidth of the outgoing communication link of each processor is the I/O bottleneck.
- Constant overhead: $C(p) \propto V$, which is representative of cases in which the incoming bandwidth of the resilient storage system is the I/O bottleneck.

Parallel work – Let $\mathcal{W}' = \beta(\mathcal{W}, p)$ be the time required for a failure-free execution on p processors. We use three classical models:

- Embarrassingly parallel jobs: $\mathcal{W}' = \mathcal{W}/p$.
- Amdahl parallel jobs: $\mathcal{W}' = \mathcal{W}/p + \gamma\mathcal{W}$. As in Amdahl's law [1], $\gamma < 1$ is the fraction of the work that is inherently sequential.
- Numerical kernels: $\mathcal{W}' = \mathcal{W}/p + \gamma\mathcal{W}^{2/3}/\sqrt{p}$. This is representative of a matrix product or a LU/QR factorization of size N on a 2D-processor grid, where $\mathcal{W} = O(N^3)$. In the algorithm in [3], $p = q^2$ and each processor receives $2q$ blocks of size N^2/q^2. γ is the communication-to-computation ratio of the platform.

We assume that the parallel job is tightly coupled, meaning that all p processors communicate continuously and thus operate synchronously throughout the job execution. These processors execute the same amount of work $\mathcal{W}'$ in parallel, chunk by chunk. The total time (on one processor) to execute a chunk of size W, and then checkpointing it, is defined as $\alpha(W, p) = W + C(p)$. For the Makespan problem, we aim at computing a function f such that $f(W, t_1^{alv}, \ldots, t_p^{alv})$ is the size of the next chunk that should be executed on every processor, given that the remaining amount of work is $W \leq \mathcal{W}'$ and for a given system state $(t_1^{alv}, \ldots, t_p^{alv})$, where t_i^{alv} denotes the time elapsed since the last failure of the i-th processor. We assume that the failure distribution laws of all processors are *iid*.

An important remark on rejuvenation – Two options are possible for recovering after a failure. Assume that the first processor, say P_1, fails at time t (during the computation of a chunk, or during checkpoint/recovery). A first option that can be found in the literature [5, 25] is to refresh (for instance by rebooting in the case of software failure) *all* processors together along with P_1 from time t to $t+D$. All processors then start the recovery, from time $t+D$ to $t+D+R(p)$. Since all processors are rejuvenated, the time elapsed since the last failure is $t_i^{alv} = R(p)$ for each processor P_i. In the second option, *only* P_1 is rejuvenated, and the other processors are kept idle from time t to $t+D$ (and then all processors recover simultaneously). In this option, $t_1^{alv} = R(p)$ and all t_i^{alv}, $i \neq 1$, are increased by $D+R(p)$.

Both options above coincide for Exponentially distributed failure inter-arrival times, due to the memoryless property, but they are different for other failure laws. Consider a platform with p processors that experience *iid* failures according to a Weibull distribution with scale parameter λ and shape parameter k (whose cumulative density is $F(t) = 1 - e^{-\frac{t}{\lambda^k}}$). Define a *platform* failure as the occurrence of a failure for any of the processors. When rejuvenating all processors after each failure, platform failures are distributed according to a Weibull distribution with scale parameter $\frac{\lambda}{p^{1/k}}$ and shape parameter k. The MTBF for the platform is thus $\frac{\lambda}{p^{1/k}}\Gamma(1+\frac{1}{k}) = \frac{MTBF}{p^{1/k}}$, where $MTBF$ is the processor-level MTBF. When rejuvenating only the processor that failed, the platform MTBF is simply $\frac{MTBF}{p}$. Thus, if $k < 1$, rejuvenating all processors after a failure leads to a lower platform MTBF than rejuvenating only the processor that failed. This is shown on an example in Figure 2, which plots the MTBF of a platform vs. the number of processors. This behavior is easily explained: for a Weibull distribution with shape parameter $k < 1$, the probability $P(X > t+\alpha | X < t)$ strictly increases with t. In other words, a processor is less likely to fail the longer it remains in a fault-free state. It turns out that failure inter-arrival times for real-life systems are effectively modeled using Weibull distributions whose shape parameter are strictly lower than 1 [11, 23]. The overall conclusion is then that rejuvenating all processors after a failure, albeit used in the literature, is not appropriate for large-scale platforms. In the rest of this paper we assume that after a failure only the failed processor is rejuvenated.

3.1 Exponential distribution

In the case of the Exponential distribution, due to the memoryless property, the p processors used for a job can be conceptually aggregated into a virtual "macro-processor" with the following characteristics:

- Failures follow an Exponential law of parameter $\lambda' = p\lambda$;
- The amount of work to execute is $\mathcal{W}' = \beta(\mathcal{W}, p)$; and
- The checkpoint and recovery overheads are $C(p)$ and $R(p)$, respectively.

A direct application of Theorem 1 yields the optimal solution of the MAKESPAN problem for multi-processor jobs, and the optimal expectation of the makespan is given by $\mathbb{E}^*(\beta(\mathcal{W}, p), p\lambda, D, C(p), R(p))$.

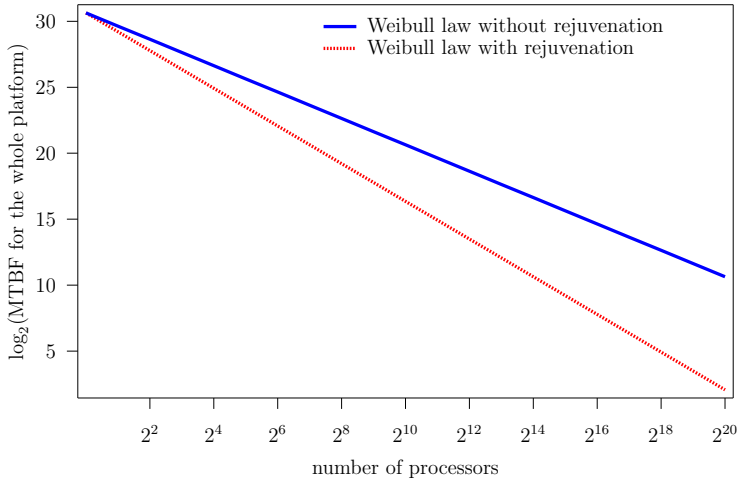


Figure 2: Impact of the two rejunevation options on the platform MTBF for a Weibull law with shape parameter 0.70 and a processor-level MTBF of 53 years.

3.2 Arbitrary distributions

For arbitrary distributions, i.e., distributions without the memoryless property, we cannot simply extend the dynamic programming approach of PDMakespan. Indeed, it would imply memorizing the evolution of the time elapsed since last failure under all the possible failure scenarios for each processor, leading to a number of states exponential in p. The idea behind the NextFailure problem now clearly appears: there is no need (when solving this problem using dynamic programming) to keep as parameters of the recursive calls the time elapsed since the last failure, as the t^{alv}'s variables of all processors evolve identically (recursive calls only concern cases where no failure occurs). Therefore, we rely on the solution to the NextFailure problem as a heuristic for computing a solution to the Makespan problem: each time a decision must be taken (after a checkpoint or a recovery), we call $f(W, t_1^{alv}, \ldots, t_p^{alv})$ to compute the size of the next chunk. The goal is to find a function f maximizing $\mathbb{E}(X_{work}^f(W, t_1^{alv}, \ldots, t_p^{alv}))$ where $X_{work}^f(0, t_1^{alv}, \ldots, t_p^{alv}) = 0$ and

$$X_{work}^f(W, t_1^{alv}, \ldots, t_p^{alv}) = \begin{cases} W_1 + X_{work}^f(W - W_1, t_1^{alv} + \alpha(W_1, p), \ldots, t_p^{alv} + \alpha(W_1, p)) \\ \quad \text{if none of the processors fails during a time } \alpha(W_1, p) \\ 0 \quad \text{otherwise.} \end{cases}$$

Using an obvious adaptation of PDNextFailure, which computes the probability of success $P_{suc}(\alpha, t_1^{alv}, \ldots, t_p^{alv}) = \prod_{i=1}^{p} \mathbb{P}(X \geq \alpha + t_i^{alv} | X \geq t_i^{alv})$, we get:

Proposition 7. *Using a time quantum u,* PDNextFailure *computes an optimal solution to* NextFailure *with p processors in time $O(p\frac{\mathcal{W}}{u}^3)$, and this for any failure distribution law.*

Even if a linear dependency in p (that comes from the computation of P_{suc}) seems to be a minimal price to pay when addressing the p processor case, the

previous complexity is still not satisfactory. Indeed, typical execution platforms that are in the scope of this paper (as Jaguar [4] or Exascale platforms) consist of tens of thousands of processors, making the previous algorithm unusable (remember that this algorithm will be invoked after each failure). We are thus interested in further reducing the complexity of PDNextFailure.

We have already recalled that failure inter-arrival times for real-life systems are best modeled using Weibull distributions whose shape parameter are strictly lower than 1 [11, 23]. With such a distribution, the processors that most influence the value of $P_{suc}(\alpha, t_1^{alv}, \ldots, t_p^{alv})$ are those that failed most recently. Therefore, in PDNextFailure, instead of considering the exact state fo the system $s = \{t_1^{alv}, \ldots, t_p^{alv}\}$ we heuristically replaced it by a state s' composed of the 10 lowest t_i^{alv} values, and of $p-10$ times the maximum of the t_i^{alv}'s. $P_{suc}(s')$ can then be evaluated in constant time (through an exponentiation). This approximation could seem to be rather crude. We studied its precision in the settings of our Petascale simulations of Section 5.2.2 by evaluating the relative error incurred when computing the probability using the approximated state s' rather than the exact one s, and this for chunks of size 2^{-i} times the platform-MTBF, with $i \in \{0..6\}$. Its turns out that the largest the size of the chunk, the worst the approximation. Over the whole execution of a job in the settings of Section 5.2.2 (i.e., for 45,208 processors), the worst relative error was of 2.82% for a chunk of duration one platform-MTBF. In practice, the chunks considered by PDNextFailure are far smaller and the evaluation of their probability of success is thus significantly better.

4 Simulation framework

In this section we detail our simulation methodology. The source code and all simulation results are publicly available at: `http://graal.ens-lyon.fr/~fvivien/checkpoint`.

4.1 Heuristics

We present simulation results for the following checkpointing policies ($MTBF$ denotes the mean time between failures of the whole platform):

- Young is the periodic checkpointing policy of period $\sqrt{2 \times C \times MTBF}$ defined in [27].
- DalyLow is the first order approximation defined by Daly in [9]. This is a periodic policy of period: $\sqrt{2 \times C \times (MTBF + D + R)}$.
- DalyHigh is the high order approximation defined by Daly in [9]. This is a periodic policy of period $(2\xi^2 + 1 + \mathbb{W}(-e^{-(2\xi^2+1)})) \times MTBF - C$ where $\xi = \sqrt{C/(2MTBF)}$.
- Liu is the non periodic policy defined in [17].
- OptExp is the periodic checkpointing policy whose period is defined in Theorem 1.
- PDMakespan is our dynamic programming algorithm minimizing the expectation of the makespan.

	p	D	C,R	$MTBF$	$\mathcal{W}$
Peta	45208	60 s	600 s	53 y	1000 y
Exa	2^{20}	60 s	600 s	500 y	10000 y

Table 1: Parameters used in the simulations (C, R and D chosen according to [12, 7]) . The first line corresponds to our Petascale platforms, and the second to the Exascale ones.

- PDNextFailure is our dynamic programming algorithm at maximizing the expectation of the amount of work completed before the next failure occurs.

We also consider the two following references to assess the absolute performance of the above heuristics:

- PeriodVariation is a numerical search for the optimal period: the period computed by OptExp is multiplied (respectively divided) by $1 + 0.05 \times i$ with $i \in \{1, ..., 180\}$, or by 1.1^j with $j \in \{1, ..., 60\}$. BestPeriod denotes the performance of the periodic policy using the best period found by PeriodVariation.
- LowerBound is the omniscient algorithm that knows when the next failure will happen and checkpoints just in time, i.e., C time-steps before this event. The makespan of LowerBound is thus an absolute (but in practice unattainable) lower bound on the execution time achievable by any policy.

Note that DalyLow and DalyHigh in this list compute the checkpointing period based solely on the MTBF, which comes from the implicit assumption that failures are exponentially distributed. For the sake of completeness we nevertheless include them in all our simulations, simply using the MTBF value even when failures follow a Weibull distribution.

4.2 Platforms

To choose failure distribution parameters that are representative of realistic systems, we use failure data from the Jaguar platform. Over a five hundred days period, the average number of failures/day for Jaguar is 2.33 [18, 2]. Consequently, we compute the processor MTBF as $MTBF = \frac{p}{2.33*365} \approx 53$ years, where $p = 45,208$ is the number of processors of the Jaguar platform. We then compute the parameters of Exponential and Weibull distributions so that they lead to this MTBF value. For the Exponential distribution we thus use $\lambda = \frac{1}{MTBF}$. The Weibull distribution requires two parameters k and λ. Following [23], we set $k = 0.7$ and compute $\lambda = MTBF/\Gamma(1 + 1/k)$. We consider two platform scenarios as detailed in Table 1, corresponding to Petascale and Exascale platforms. In each case, we determined a job size $\mathcal{W}$ corresponding to a job using the whole platform for a significant amount of time, namely ≈ 8 days (in the absence of failures) for the Petascale platforms and ≈ 3.5 days for the Exascale ones.

4.3 Methodology

Failure trace generation. Given a p-processor platform, a failure trace is a set of failure dates for each processor over a fixed time horizon h. For one-processor platforms, h is set to 1 year. In all the other cases, h is set to 10 years. Given a failure distribution f for the failure inter-arrival times at a processor, for each processor we generated a trace via independent sampling until the target time horizon is reached. Finally, for simulations where the only varying parameter is the number of processor $a \leq p \leq b$, we generate a trace for b processors. Then, the traces for p processors are simply the first p traces. This ensures that simulation results are coherent when varying p.

Performance Metric. We compare the heuristics using average makespan degradation, defined as follows. Given an experimental scenario (i.e., parameter values for failure distribution and platform configuration), we generate a set $X = \{tr_1, \ldots, tr_{1000}\}$ of $1,000$ traces. For each trace tr_i and each heuristic j we compute the achieved makespan, $res_{(i,j)}$. The makespan degradation for heuristic j on trace tr_i is defined as $v_{(i,j)} = res_{(i,j)}/min_j\{res_{(i,j)}\}$. Finally, we compute the average degradation for heuristic j as $av_j = \sum_{i=1}^{1000} v_{(i,j)}/1000$.

Standard deviations were generally small (especially for large platform simulations, as we can see in Table 6), and thus are not plotted on figures.

Application models. We consider 6 different application models in which $\beta(\mathcal{W}, p)$ is equal to either $\frac{\mathcal{W}}{p}$ (embarrassingly parallel scenario), $\frac{\mathcal{W}}{p} + \gamma\mathcal{W}$ with $\gamma \in \{10^{-4}, 10^{-6}\}$, or $\frac{\mathcal{W}}{p} + \gamma\frac{\mathcal{W}^{2/3}}{\sqrt{p}}$ with $\gamma \in \{0.1, 1, 10\}$. Our exhaustive simulations show that the general conclusions drawn from the results depend neither on the application profile nor on the checkpointing policy. Therefore, for the sake of readability we only report in Section 5 results on embarrassingly parallel applications executed with platform-independent checkpoint (i.e., $C(p)$ is independent of p) and recovery costs, and we refer each time the reader to the appropriate part of the Appendix to find graphs for other combinations of an application profile and a checkpointing policy. Note that models for which LowerBound is not capable to successfully compute $\mathcal{W}$ within the fixed time horizon are not presented in the Appendix (this if for instance the case for Petascale platforms and an Amdahl law with $\gamma = 10^{-2}$). Finally, comparisons between the different application profiles, but for a fixed heuristic (the best one, depending on the distribution used) are presented in the Appendix, Section C.

5 Simulation results

5.1 With one-processor

For a single processor, we cannot target a MTBF of 53 years, as we would need a job to run for 200 years or so to need a few checkpoints. Hence we study scenarios with smaller values of the MTBF, from one hour to one week. The main goal is to assess the quality of all the heuristics listed in Section 4 and to prepare for the study of large-scale platforms.

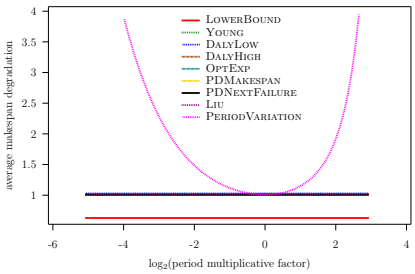


Figure 3: Evaluation of the different heuristics on a single processor with Exponential failures (MTBF = 1 hour).

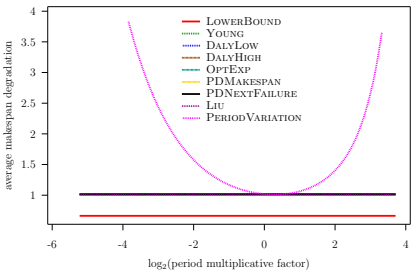


Figure 4: Evaluation of the different heuristics on a single processor with Weibull failures (MTBF = 1 hour).

5.1.1 Exponential failures

Figure 3 plots average makespan degradation for the different heuristics. The horizontal axis corresponds to the period used by PeriodVariation. Given that all other heuristics compute a specific period, they appear as horizontal lines on the figure. A first observation is that the checkpointing period used by the well-known Young, DalyLow and DalyHigh heuristics is indeed close to optimal. While this result seems widely accepted, we are not aware of previously published simulation studies that have demonstrated this claim. Table 2 shows numerical results for different MTBF values. Not surprisingly, our exact optimal solution is slightly better than all other solutions and matches the result obtained with the best period for PeriodVariation.

Recall that, as explained in Section 2.4, solving the NextFailure problem, using PDNextFailure, should provide a reasonable solution to the Makespan problem. While it is not possible to compare PDNextFailure to PDMakespan for large platforms (because the complexity of PDMakespan is exponential in p), we can compare them in the one-processor case. Results in Table 2 confirm that, at least in the one-processor case, PDNextFailure leads to solutions that are close to that computed by PDMakespan. We notice that Liu is more dispersed than other heuristics.

5.1.2 Weibull failures

Following the same approach as in the exponential case, we compared Young, DalyLow, DalyHigh, Liu (which handles Weibull distributions), and PeriodVariation to PDNextFailure and PDMakespan. Contrarily to the exponential case, the optimal checkpoint policy may be non-periodic (to the best of our knowledge, this question is still open), making the comparison intriguing. Figure 4 shows that all the heuristics lead to approximately the same optimal result. This implies that, in the one-processor case, we can safely use Young, DalyLow, and DalyHigh, which use only the failure MTBF, even for Weibull failures. This result does not hold for multi-processor platforms (see Section 5.2.2). Numerical results in Table 3 show that, just as in the Exponen-

Heuristics	MTBF					
	1 hour		1 day		1 week	
	avg	std	avg	std	avg	std
LowerBound	0.62851	0.01336	0.90673	0.01254	0.97870	0.01430
Young	1.01747	0.01053	1.01558	0.00925	1.02317	0.00936
DalyLow	1.02802	0.01199	1.01598	0.00940	1.02323	0.00936
DalyHigh	1.00748	0.00627	1.01560	0.00881	1.02333	0.00949
Liu	1.01735	0.01378	1.05438	0.03725	1.20756	0.16730
BestPeriod	1.00743	0.00630	1.01547	0.00872	1.02257	0.00922
OptExp	1.00743	0.00630	1.01547	0.00872	1.02257	0.00922
PDMakespan	1.00782	0.00644	1.01556	0.01029	1.03415	0.01977
PDNextFailure	1.00790	0.00602	1.01653	0.00884	1.02789	0.01361

Table 2: Degradation from best for a single processor and failures following an exponential law.

tial case, PDNextFailure leads to solutions that are close to that computed by PDMakespan.

Heuristics	MTBF					
	1 hour		1 day		1 week	
	avg	std	avg	std	avg	std
LowerBound	0.66348	0.01384	0.91006	0.01717	0.97612	0.01655
Young	1.00980	0.00690	1.01637	0.00821	1.02291	0.00969
DalyLow	1.01181	0.00811	1.01661	0.00944	1.02296	0.00968
DalyHigh	1.01740	0.00850	1.01620	0.00934	1.02284	0.00962
Liu	1.01051	0.01135	1.07003	0.05755	1.19306	0.17194
BestPeriod	1.00990	0.00700	1.01591	0.00949	1.02234	0.00956
OptExp	1.01734	0.00847	1.01641	0.00932	1.02234	0.00956
PDMakespan	1.00738	0.00740	1.01533	0.01020	1.03451	0.02134
PDNextFailure	1.01356	0.00818	1.01648	0.00922	1.02713	0.01321

Table 3: Degradation from best for a single processor and failures following a Weibull law.

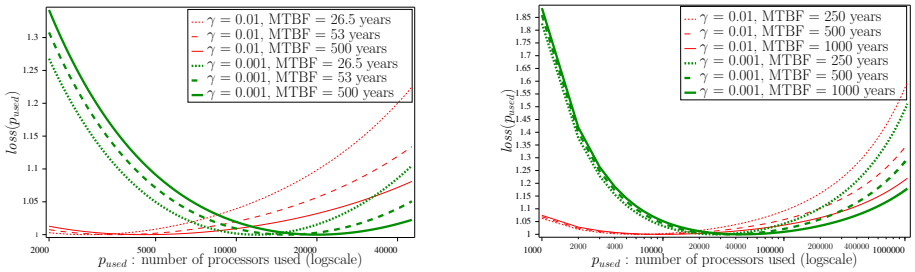
5.2 Petascale platforms

5.2.1 Exponential failures

Optimal number of processors As shown in Section 3.1, the optimal expected makespan when using $p_{used} \leq p$ processors is given by

$$\rho(p_{used}) = \mathbb{E}^*(\beta(\mathcal{W}, p_{used}), p_{used}\lambda, D, C(p_{used}), R(p_{used})).$$

Given the different models presented in Section 3, we aim at determining if using the entire platform to execute the job always is the best solution. We define the *loss* as the ratio $loss(p_{used}) = \frac{\rho(p_{used})}{\rho(p^*)}$ of the expected execution time with p_{used} processors over the minimal execution time, obtained with p^* processors. To



a) 45208 processors, $\mathcal{W} = 1000$ years b) 2^{20} processors, $\mathcal{W} = 10000$ years

Figure 5: $loss(p_{used})$ vs p_{used} for constant checkpoint and recovery overhead, and job following Amdahl law. In all scenarios, $D = 60$ s, $R = C = 600$ s.

	$\gamma = 0.01$	$\gamma = 0.001$	$\gamma = 0.0001$
Peta	$p^* = 3570$ $loss(p) = 1.13$	$p^* = 15480$ $loss(p) = 1.05$	$p^* = p$ $loss(p) = 1$
Exa	$p^* = 7820$ $loss(p) = 1.35$	$p^* = 35060$ $loss(p) = 1.30$	$p^* = 45200$ $loss(p) = 1.18$

Table 4: Optimal number of processors and loss when using the whole platform with p processors

determine p^* we use a brute force search implemented with Matlab, computing the Lambert W function using [22].

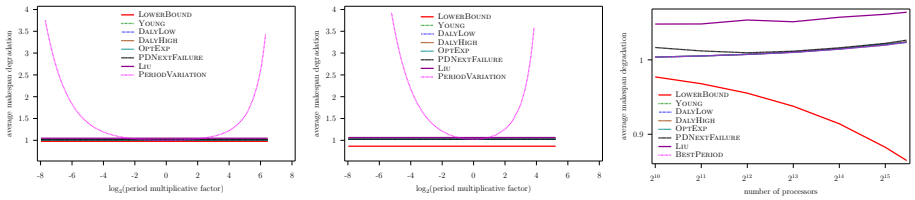
For the scenario with constant checkpoint and recovery costs, and jobs obeying Amdahl's law with sequential fraction γ, we report loss values for a Petascale and an Exascale platform in Figure 5 (case a)), with different values for γ and $MTBF$. As expected, we observe that p^* decreases when γ increases (using more processors for a job having a large sequential fraction leads to more failures without much benefit in execution time), and that p^* decreases when the $MTBF$ decreases (when p_{used} approaches p, the time saved using p_{used} processors becomes small compared to the idle time due to failures). We obtain a similarly shaped curve when using data corresponding to an Exascale platform (see Figure 5, b)), and key values are reported in Table 4.

The results for other application scenarios or with proportional checkpoint and recovery costs lead to a different conclusion. According to the extreme cases studied in Table 5 (where the MTBF has intentionally been shortened, and D, C, R increased), we conclude that the loss when using the whole platform in the other models becomes negligible, while it reaches up to 35% in the above study. In conclusion, and not suprisingly, the optimal number of processors to use to run an application depends on the application and checkpointing models, and may not always be the maximum available number of processors.

Comparison of the different heuristics Figure 6 *a*) and *b*) shows that for both small and large platforms, the different approximations (Young, DalyLow and DalyHigh, Liu, PDNextFailure) compute an almost optimal solution (OptExp). In *c*) we see that PDNextFailure behaves satisfactorily: its average makespan degradation is less than 1.02 for $2^{10} \leq p \leq 2^{12}$, and be-

Application model	Checkpt/Recov Model	$\mathcal{W}$	p	γ	D	MTBF	$loss(p)$
$\frac{\mathcal{W}}{p}+\gamma\frac{\mathcal{W}}{\sqrt{p}}$	constant	1000 y	45208	10	80	$\frac{53}{2}$ y	1
	constant	10000 y	2^{20}	10	150	$\frac{500}{3}$ y	1.014
$\frac{\mathcal{W}}{p}+\gamma\mathcal{W}$	proportional	1000 y	45208	0.01	80	$\frac{53}{2}$ y	1.026
	proportional	10000 y	2^{20}	0.01	150	$\frac{500}{3}$ y	1.026

Table 5: Cases where $loss(p)$ is negligible, even using extreme values of MTBF of D, C, R. Notice that $C = R = 10D$.



a) 1024 processors b) 45208 processors c)

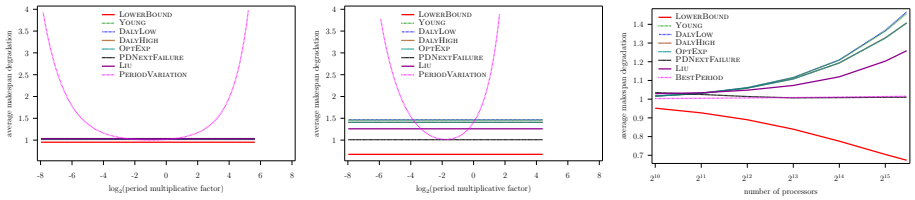
Figure 6: Evaluation of the different heuristics on a Petascale platform with Exponential failures, using embarrassingly parallel job, and constant overhead model

comes extremely small for $p \geq 2^{12}$. Results for other application models are presented in the Appendix, in Section A.1.

Impact of sub-optimal checkpoint intervals Finally, we provide an answer to [13], where the authors study the impact of sub-optimal checkpoint intervals on the *application efficiency*, defined as the ratio between useful computing time and total execution time. Based on simulations (parametrized using a model for a 1926 node cluster), the authors conclude that "underestimating an application's optimal checkpoint interval is generally more severe than overestimation". In Figure 8, for various platform sizes, and for several values of the factor x, we compute the makespan $Mult(x)$ obtained when multiplying the best period, as computed by OptExp, by x (overestimation), the makespan $Div(x)$ obtained when dividing this best period by x (underestimation), and we plot the average (over 1000 scenarios) of the ratio $\frac{Mult(x)}{Div(x)} \times 100$ (which we call the percentage of degradation). Contrarily to the conclusion of [13], we see that the previous quantity is always positive, and that significant values are possible, even for a small multiplicative factor x. For instance, we see that for $p = 45208$ and a factor $x = 5$, it is in average 10% worse to overestimate the period than to underestimate it.

5.2.2 Weibull failures

A key contribution of this paper is the comparison between PDNextFailure and all published heuristics for the Makespan problem on large platforms whose failure obey Weibull distribution laws. Existing heuristics provided good solutions for one-processor jobs (Section 5.1), and Figure 7 *a*) shows that this remains true up to 1024 processors. However, it is no longer the case when



a) 1024 processors b) 45208 processors c)

Figure 7: Evaluation of the different heuristics on a Petascale platform with Weibull failures.

increasing the number of processors (see Figures 7, *b*) and *c*)). For example with 45208 processors, Young, DalyLow and DalyHigh are approximately 40% worse than PDNextFailure, and even Liu (which handles Weibull laws) is 20% worse than PDNextFailure. Furthermore, the optimal solution is far from periodic. For instance, throughout a complete execution with 45208 processors, PDNextFailure changed the size of inter-checkpoint times which evolved from 740 seconds at the beginning, up to 2220 seconds at the end. We conclude that our dynamic programming approach significantly improves all known results for the Makespan problem with large platforms. Results for other application models are presented in the Appendix, in Section A.2.

There is a price to pay for PDNextFailure in that its execution is not instantaneous, contrarily to Young, DalyLow, DalyHigh, or OptExp. However, this price is quite small, as the optimized version of PDNextFailure takes a few seconds on a standard laptop. To better quantify the overhead incurred by the use of PDNextFailure, we point out that its execution time never represents more than 0.14% of the job execution time (over all simulated instances), and hence is completely negligible in front of the huge improvement it brings for the application execution.

The number of failures encountered during the execution of PDNextFailure obviously depends on the amount of computation required by the application. For a job running around 17 days on a Petsacale platform, we encountered 396.6 failures on the average, with a worst case of 474 failures. This gives an idea on the number of spare ressources required to run such an application (circa 1% of the total machine size).

Heuristics	Average degradation	Standard deviation
LowerBound	0.67358	0.01039
Young	1.45825	0.06264
DalyLow	1.46605	0.06320
DalyHigh	1.40643	0.05745
Liu	1.25895	0.02060
OptExp	1.40784	0.05870
PDNextFailure	1.01013	0.00938

Table 6: Average degradation from best and standard deviation for a 45208 processor platform, failures following a Weibull law, embarrassingly parrallel job and fixed checkpoint/recovery costs

5.3 Exascale platforms

Simulations for Exascale platforms corroborate well our findings on Petascale platforms, and this for both Exponential (Figure 9) and Weibull (Figure 10) distributions. The major differences are found for Weibull distribution, as depicted on Figure 10. Indeed, the superiority of PDNextFailure over pre-existing heuristics is even more important. Results for other application models are presented in the Appendix, in Section B.1 (for exponential law), and in Section B.2 (for Weibull law). Finally, we point out that on the largest platforms, Liu does not succeed to complete the execution of the job within our 10-year time horizon h, while PDNextFailure leads to a runtime of 27 days.

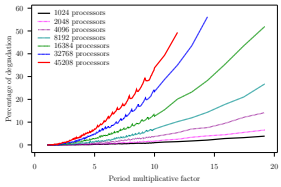


Figure 8: Overestimation vs. underestimation of the period.

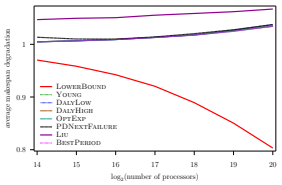


Figure 9: Relative performance of the heuristics for platforms containing up to 1048576 processors (Exponential failures).

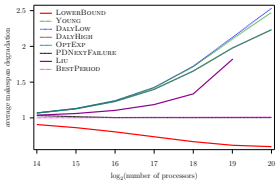


Figure 10: Relative performance of the heuristics for platforms containing up to 1048576 processors (Weibull failures).

6 Related work

In [9], Daly addressed the problem of the periodic checkpointing of applications executed on a platform where failures inter-arrival times are exponentially distributed. This study comprises checkpointing and recovery overheads (but no downtimes), and allows failures to happen during recoveries. Daly proposed two estimates of the optimal period. The lower order estimate is a generalization of Young's approximation [27], which takes recovery overheads into account. The higher order estimate is ill-formed as it relies on an equation that sums up non-independent probabilities (Equation (13) in [9]). This study of period estimation is extended in [13] with the question of the impact of sub-optimal periods on application efficiency. The authors of [13] conclude that it is preferable to overestimate the optimal period rather than to underestimate it (which is the exact opposite of our own conclusion in Section 3.1) .

In [5], Bouguerra et al. study the design of an optimal checkpointing policy when failures can occur during checkpointing and recovery, with checkpointing and recovery overheads depending upon the application progress. They show that the optimal checkpointing policy is periodic when checkpointing and recovery overheads are constant, and when failure inter-arrival times follow either an Exponential or a Weibull law. They also give formulas to compute the optimal period in both cases. Their results, however, rely on the un-stated assumption

that the failure laws are rejuvenated after each failure and after each checkpoint. The work in [25] suffers from the same problem.

In [26], the authors claim to use an "optimal checkpoint restart model [for] Weibull's and Exponential distributions" that they have designed in another paper referenced as [1] in [26]. However, this latter paper is not available, and we were unable to compare with this solution. Moreover, as explained in [26] the "optimal" solution in [1] is found using the assumption that checkpoint is periodic (even for Weibull law). In addition, the authors of [26] partially address the question of the optimal number of processors for parallel jobs, presenting experiments for four MPI applications, using a non-optimal policy, and for up to 35 processors. Our approach here is completely different since we target large-scale platforms up to tens of thousands of processors and rely on generic application models for deriving optimal solutions.

In this work, we solve the NextFailure problem to obtain heuristic solutions to the Makespan problem in the case of multi-processor jobs. The NextFailure problem has been studied by many authors in the literature, often for single-processor jobs. Maximizing the expected work successfully completed before the first failure is equivalent to minimizing the expected wasted time before the first failure, which is itself a classical problem. Some authors propose analytical resolution using a "checkpointing frequency function", for both infinite (see [16, 17]) and finite time horizons (see [19]). However, these works use approximations, as for example assuming that the expected failure occurrence is exactly halfway between two checkpointing events, which does not hold for general failure distributions. Approaches that do not rely on a checkpointing frequency function are used in [24, 15], but for infinite time horizons. Finally, authors in [6] address a problem that is related to NextFailure, where checkpoints can only be scheduled between (indivisible) jobs. The proposed dynamic programming algorithm is close to PDNextFailure, however the rejuvenation of failure laws is assumed after each checkpoint, leading to the problems described in Section 3.

7 Conclusion

In this paper, we have addressed the Makespan problem, i.e., scheduling checkpoints for minimizing the execution time of sequential and parallel jobs on large-scale and failure-prone platforms. An auxiliary problem, NextFailure, was introduced as an approximation of Makespan. Particular care has been taken to define these problems in a general setting, and with full rigor. For exponential distributions, we have provided a complete analytical solution of the Makespan problem together with an assessment of the quality of the NextFailure approximation. We have also designed dynamic programming solutions for both problems, that hold for any failure distribution law.

We have also obtained new interesting results through comprehensive simulations. For the exponential distribution, our exact formula allows us to determine the optimal number of processors to be used (depending on application and checkpoint models), and to provide an answer to previous work on the impact of sub-optimal period choices. For the Weibull distribution, we have demonstrated the importance of using the "single rejuvenation" model. With this model, we

have shown that our dynamic programming algorithm leads to dramatically more efficient executions than all existing algorithms (with a decrease in the application makespan between 19.7 and 30.7% for Petascale platforms, and of at least 55% for the largest simulated Exascale platforms). Because of the omnipresence of Weibull laws to model real-life failures, and owing to our various application and checkpoint scenarios, we strongly believe that the dynamic programming approach provides a key step for the effective use of next-generation platforms.

Because of the huge cost incurred by the power consumed by large-scale platforms (not to speak of environmental concerns), future work will be devoted to the design of checkpointing strategies having the ability to trade-off between a shorter execution time and a reduced energy consumption.

References

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

[2] L. Bautista Gomez, A. Nukada, N. Maruyama, F. Cappello, and S. Matsuoka. Transparent low-overhead checkpoint for GPU-accelerated clusters. https://wiki.ncsa.illinois.edu/download/attachments/17630761/INRIA-UIUC-WS4-lbautista.pdf?version=1&modificationDate=1290470402000.

[3] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. *ScaLAPACK Users' Guide*. SIAM, 1997.

[4] A.S. Bland, R.A. Kendall, D.B. Kothe, J.H. Rogers, and G.M. Shipman. Jaguar: The World's Most Powerful Computer. In *GUC'2009*, 2009.

[5] Mohamed-Slim Bouguerra, Thierry Gautier, Denis Trystram, and Jean-Marc Vincent. A flexible checkpoint/restart model in distributed systems. In *PPAM*, volume 6067 of *LNCS*, pages 206–215, 2010.

[6] Mohamed Slim Bouguerra, Denis Trystram, and Frédéric Wagner. An optimal algorithm for scheduling checkpoints with variable costs. Research report, INRIA, 2010. Available at http://hal.archives-ouvertes.fr/inria-00558861/en/.

[7] Franck Cappello, Henri Casanova, and Yves Robert. Checkpointing vs. migration for post-petascale supercomputers. In *ICPP'2010*. IEEE Computer Society Press, 2010.

[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.*, 45(2):311–332, 2001.

[9] J. T. Daly. A higher order estimate of the optimum checkpoint interval for restart dumps. *Future Generation Computer Systems*, 22(3):303–312, 2004.

[10] Jack Dongarra, Pete Beckman, Patrick Aerts, Frank Cappello, Thomas Lippert, Satoshi Matsuoka, Paul Messina, Terry Moore, Rick Stevens, Anne Trefethen, and Mateo Valero. The international exascale software project: a call to cooperative action by the global high-performance community. *Int. J. High Perform. Comput. Appl.*, 23(4):309–322, 2009.

[11] T. Heath, R. P. Martin, and T. D. Nguyen. Improving cluster availability using workstation validation. *SIGMETRICS Perf. Eval. Rev.*, 30(1):217–227, 2002.

[12] J.C.Y. Ho, C.L. Wang, and F.C.M. Lau. Scalable group-based checkpoint/restart for large-scale message-passing systems. In *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, pages 1–12. IEEE, 2008.

[13] W.M. Jones, J.T. Daly, and N. DeBardeleben. Impact of sub-optimal checkpoint intervals on application efficiency in computational clusters. In *HPDC'10*, pages 276–279. ACM, 2010.

[14] Nick Kolettis and N. Dudley Fulton. Software rejuvenation: Analysis, module and applications. In *FTCS '95*, page 381, Washington, DC, USA, 1995. IEEE CS.

[15] P. L'Ecuyer and J. Malenfant. Computing optimal checkpointing strategies for rollback and recovery systems. *IEEE Transactions on computers*, 37(4):491–496, 2002.

[16] Y. Ling, J. Mi, and X. Lin. A variational calculus approach to optimal checkpoint placement. *IEEE Transactions on computers*, pages 699–708, 2001.

[17] Y. Liu, R. Nassar, C. Leangsuksun, N. Naksinehaboon, M. Paun, and SL Scott. An optimal checkpoint/restart model for a large scale high performance computing system. In *IPDPS 2008*, pages 1–9. IEEE, 2008.

[18] E. Meneses. Clustering Parallel Applications to Enhance Message Logging Protocols. https://wiki.ncsa.illinois.edu/download/attachments/17630761/INRIA-UIUC-WS4-emenese.pdf?version=1&modificationDate=1290466786000.

[19] T. Ozaki, T. Dohi, H. Okamura, and N. Kaio. Distribution-free checkpoint placement algorithms based on min-max principle. *IEEE TDSC*, pages 130–140, 2006.

[20] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, 2005.

[21] Vivek Sarkar and others. Exascale software study: Software challenges in extreme scale systems, 2009. White paper available at: http://users.ece.gatech.edu/mrichard/ExascaleComputingStudyReports/ECSS%20report%20101909.pdf.

[22] Nicol N. Schraudolph. http://www.cs.toronto.edu/~dross/code/LambertW.m.

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

[24] A.N. Tantawi and M. Ruschitzka. Performance analysis of checkpointing strategies. *ACM (TOCS)*, 2(2):123–144, 1984.

[25] S. Toueg and O. Babaoglu. On the optimum checkpoint selection problem. *SIAM J. Computing*, 13(3):630–649, 1984.

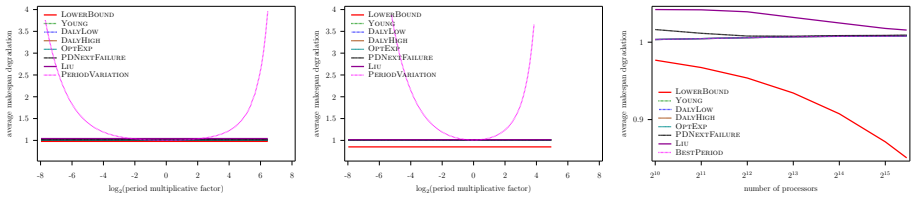
[26] K. Venkatesh. Analysis of Dependencies of Checkpoint Cost and Checkpoint Interval of Fault Tolerant MPI Applications. *Analysis*, 2(08):2690–2697, 2010.

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

A Appendix : Results for Petascale platforms

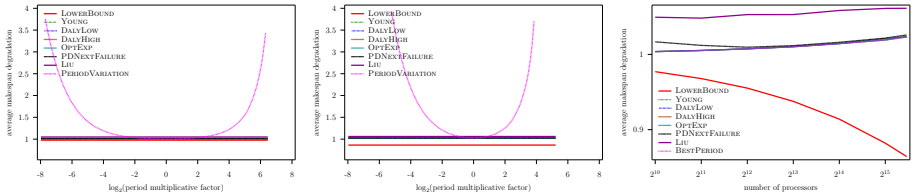
A.1 Exponential law

A.1.1 Fixed checkpoint and recovery cost



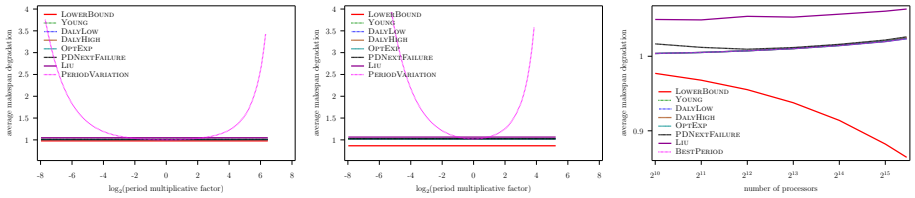
a) 1024 processors b) 45208 processors c)

Figure 11: Evaluation of the different heuristics on a Petascale platform with Exponential failures, using Amdahl law with $\gamma = 10^{-4}$, and constant overhead model



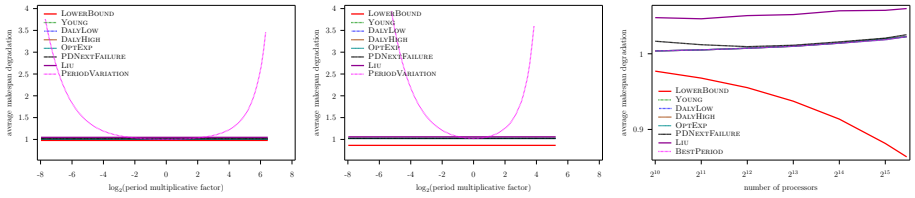
a) 1024 processors b) 45208 processors c)

Figure 12: Evaluation of the different heuristics on a Petascale platform with Exponential failures, using Amdahl law with $\gamma = 10^{-6}$, and constant overhead model



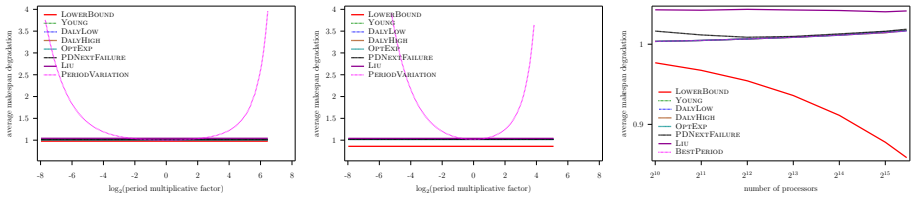
a) 1024 processors b) 45208 processors c)

Figure 13: Evaluation of the different heuristics on a Petascale platform with Exponential failures, using Numerical Kernel law with $\gamma = 0.1$, and constant overhead model



a) 1024 processors b) 45208 processors c)

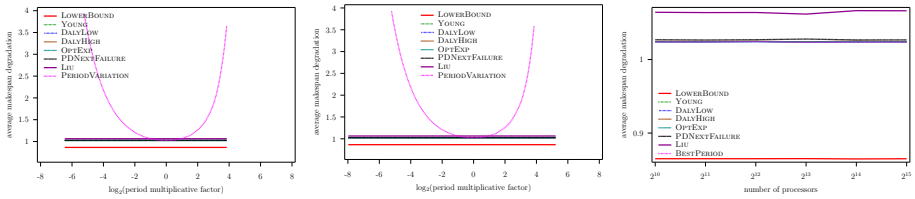
Figure 14: Evaluation of the different heuristics on a Petascale platform with Exponential failures, using Numerical Kernel law with $\gamma = 1$, and constant overhead model



a) 1024 processors b) 45208 processors c)

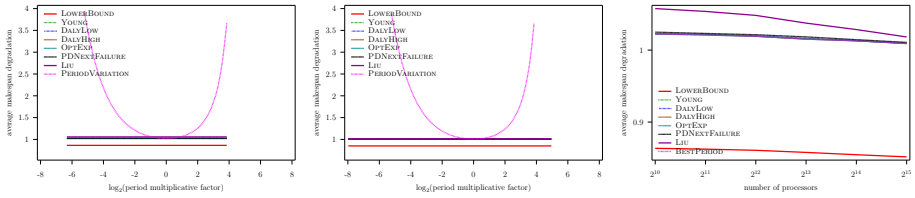
Figure 15: Evaluation of the different heuristics on a Petascale platform with Exponential failures, using Numerical Kernel law with $\gamma = 10$, and constant overhead model

A.1.2 Variable checkpoint and recovery cost



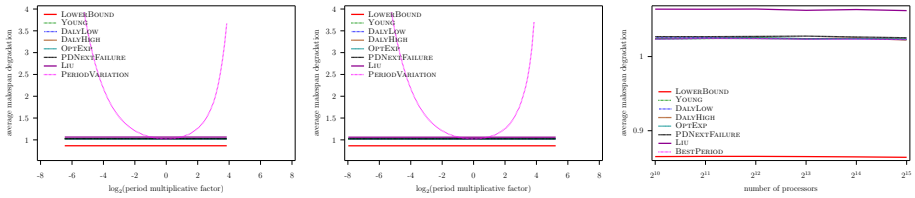
a) 1024 processors b) 45208 processors c)

Figure 16: Evaluation of the different heuristics on a Petascale platform with Exponential failures, using embarrassingly parallel job, and variable overhead model ($C(p) = 600\frac{45208}{p}$)



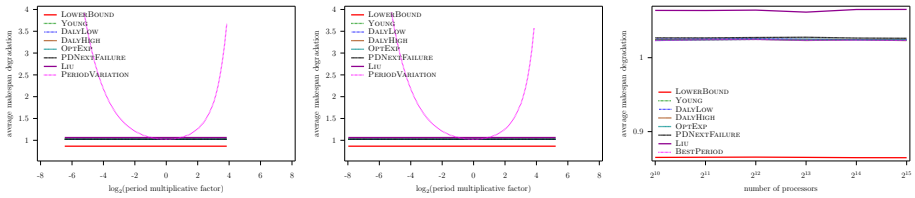
a) 1024 processors b) 45208 processors c)

Figure 17: Evaluation of the different heuristics on a Petascale platform with Exponential failures, using Amdahl law with $\gamma = 10^{-4}$, and variable overhead model ($C(p) = 600\frac{45208}{p}$)



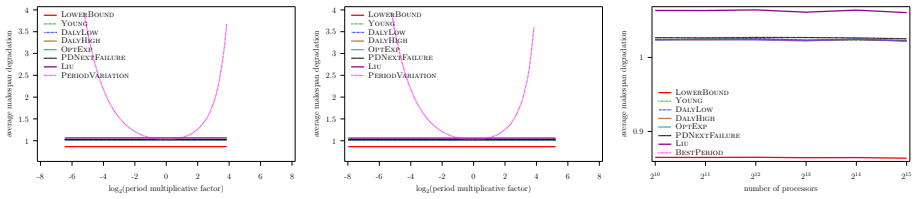
a) 1024 processors b) 45208 processors c)

Figure 18: Evaluation of the different heuristics on a Petascale platform with Exponential failures, using Amdahl law with $\gamma = 10^{-6}$, and variable overhead model ($C(p) = 600\frac{45208}{p}$)



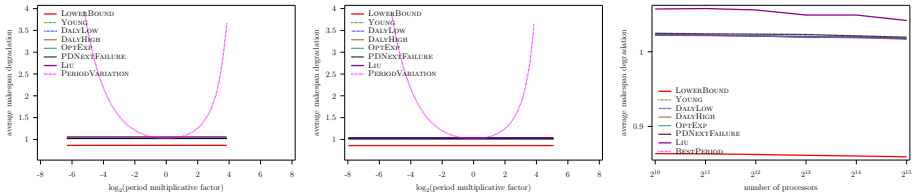
a) 1024 processors b) 45208 processors c)

Figure 19: Evaluation of the different heuristics on a Petascale platform with Exponential failures, using Numerical Kernel law with $\gamma = 0.1$, and variable overhead model ($C(p) = 600\frac{45208}{p}$)



a) 1024 processors b) 45208 processors c)

Figure 20: Evaluation of the different heuristics on a Petascale platform with Exponential failures, using Numerical Kernel law with $\gamma = 1$, and variable overhead model ($C(p) = 600\frac{45208}{p}$)

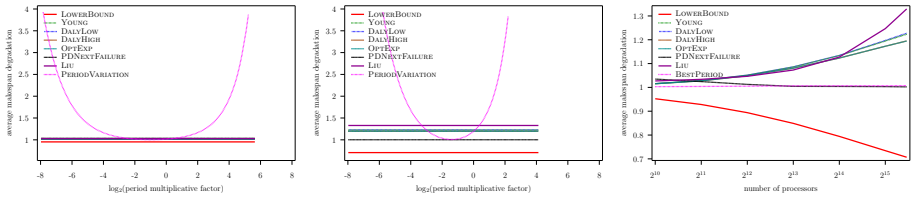


a) 1024 processors b) 45208 processors c)

Figure 21: Evaluation of the different heuristics on a Petascale platform with Exponential failures, using Numerical Kernel law with $\gamma = 10$, and variable overhead model ($C(p) = 600\frac{45208}{p}$)

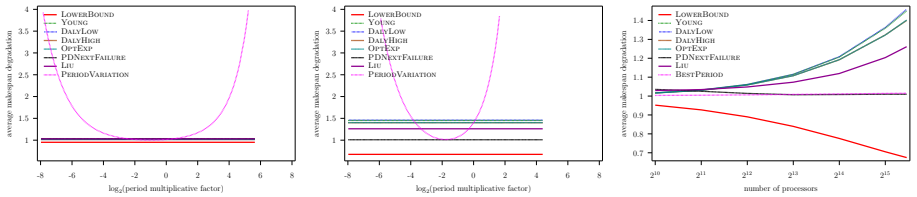
A.2 Weibull law

A.2.1 Fixed checkpoint and recovery cost



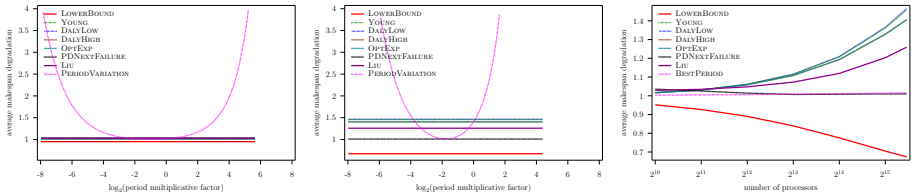
a) 1024 processors b) 45208 processors c)

Figure 22: Evaluation of the different heuristics on a Petascale platform with Weibull failures, using Amdahl law with $\gamma = 10^{-4}$, and constant overhead model



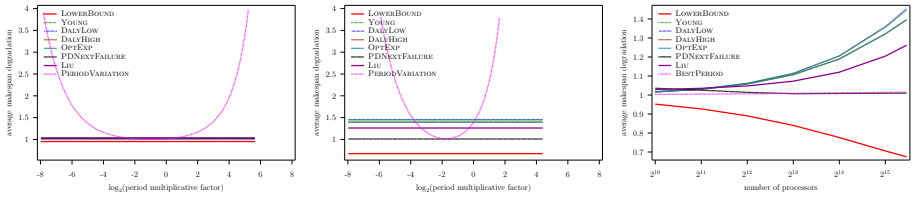
a) 1024 processors b) 45208 processors c)

Figure 23: Evaluation of the different heuristics on a Petascale platform with Weibull failures, using Amdahl law with $\gamma = 10^{-6}$, and constant overhead model



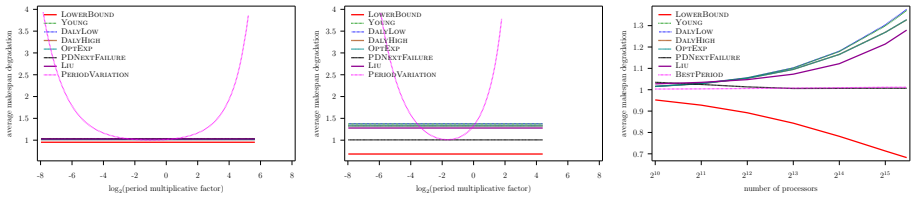
a) 1024 processors b) 45208 processors c)

Figure 24: Evaluation of the different heuristics on a Petascale platform with Weibull failures, using Numerical Kernel law with $\gamma = 0.1$, and constant overhead model



a) 1024 processors b) 45208 processors c)

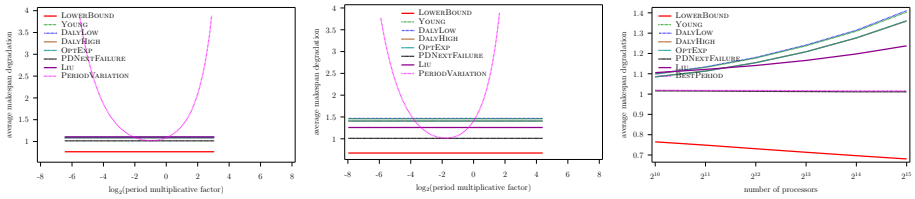
Figure 25: Evaluation of the different heuristics on a Petascale platform with Weibull failures, using Numerical Kernel law with $\gamma = 1$, and constant overhead model



a) 1024 processors b) 45208 processors c)

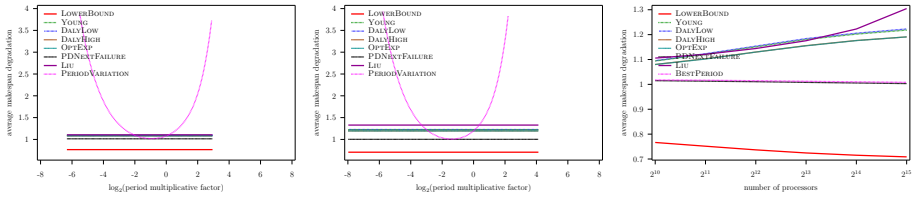
Figure 26: Evaluation of the different heuristics on a Petascale platform with Weibull failures, using Numerical Kernel law with $\gamma = 10$, and constant overhead model

A.2.2 Variable checkpoint and recovery cost



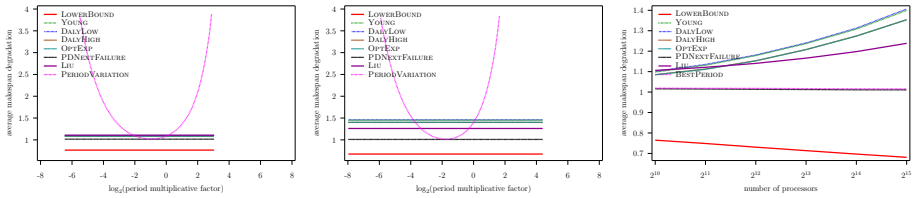
a) 1024 processors b) 45208 processors c)

Figure 27: Evaluation of the different heuristics on a Petascale platform with Weibull failures, using embarrassingly parallel job, and variable overhead model ($C(p) = 600\frac{45208}{p}$)



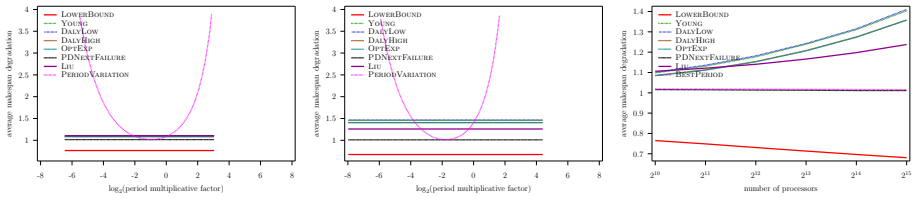
a) 1024 processors b) 45208 processors c)

Figure 28: Evaluation of the different heuristics on a Petascale platform with Weibull failures, using Amdahl law with $\gamma = 10^{-4}$, and variable overhead model $(C(p) = 600\frac{45208}{p})$



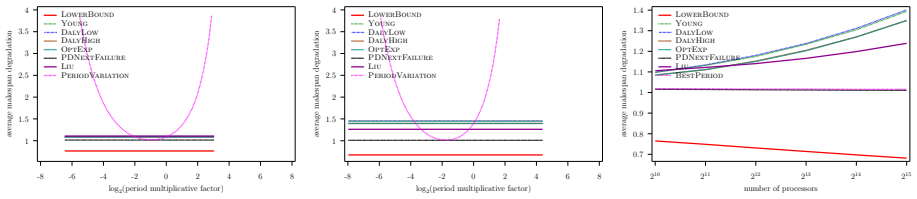
a) 1024 processors b) 45208 processors c)

Figure 29: Evaluation of the different heuristics on a Petascale platform with Weibull failures, using Amdahl law with $\gamma = 10^{-6}$, and variable overhead model $(C(p) = 600\frac{45208}{p})$



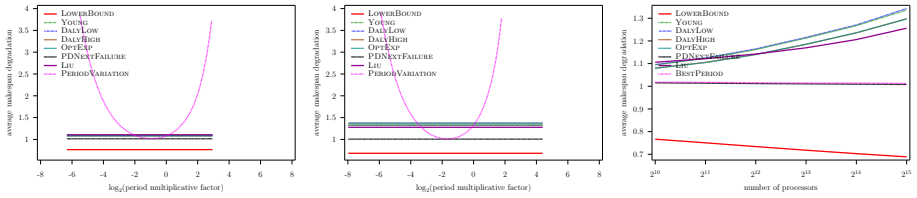
a) 1024 processors b) 45208 processors c)

Figure 30: Evaluation of the different heuristics on a Petascale platform with Weibull failures, using Numerical Kernel law with $\gamma = 0.1$, and variable overhead model $(C(p) = 600\frac{45208}{p})$



a) 1024 processors b) 45208 processors c)

Figure 31: Evaluation of the different heuristics on a Petascale platform with Weibull failures, using Numerical Kernel law with $\gamma = 1$, and variable overhead model $(C(p) = 600\frac{45208}{p})$



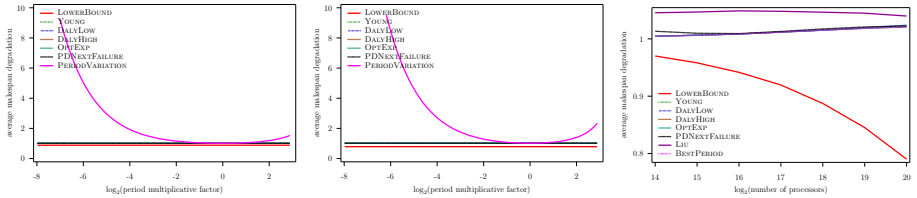
a) 1024 processors b) 45208 processors c)

Figure 32: Evaluation of the different heuristics on a Petascale platform with Weibull failures, using Numerical Kernel law with $\gamma = 10$, and variable overhead model ($C(p) = 600\frac{45208}{p}$)

B Appendix : Results for Exascale platforms

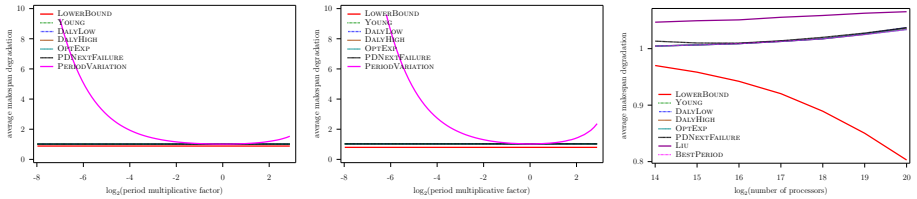
B.1 Exponential law

B.1.1 Fixed checkpoint and recovery cost



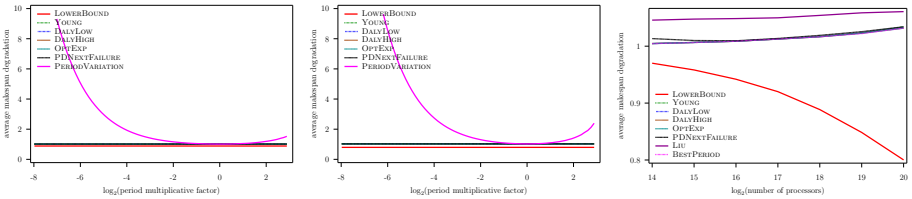
a) 262144 processors b) 1048576 processors c)

Figure 33: Evaluation of the different heuristics on a Exascale platform with Exponential failures, using Amdahl law with $\gamma = 10^{-6}$, and constant overhead model



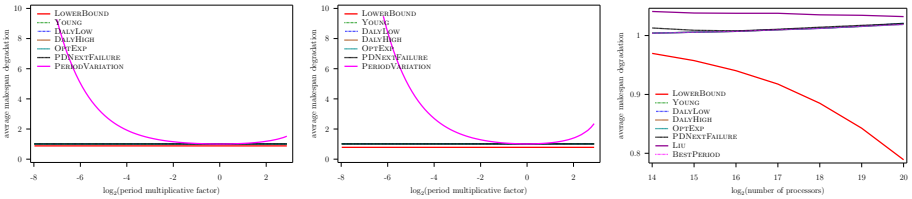
a) 262144 processors b) 1048576 processors c)

Figure 34: Evaluation of the different heuristics on a Exascale platform with Exponential failures, using Numerical Kernel law with $\gamma = 0.1$, and constant overhead model



a) 262144 processors b) 1048576 processors c)

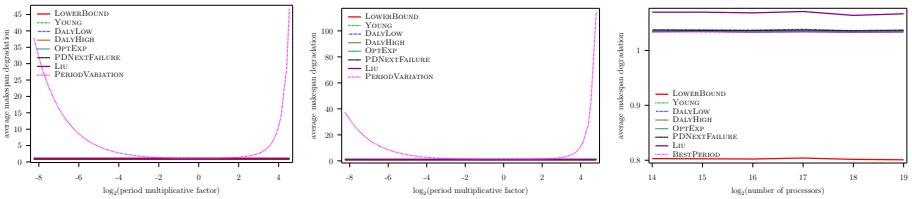
Figure 35: Evaluation of the different heuristics on a Exascale platform with Exponential failures, using Numerical Kernel law with $\gamma = 1$, and constant overhead model



a) 262144 processors b) 1048576 processors c)

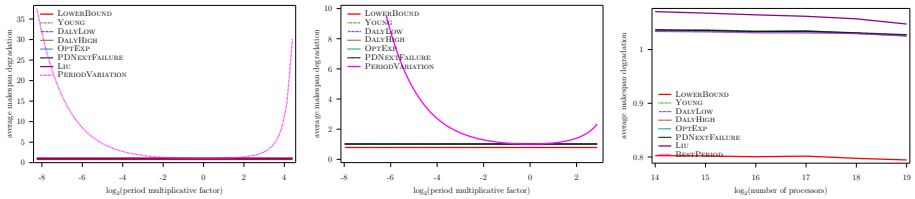
Figure 36: Evaluation of the different heuristics on a Exascale platform with Exponential failures, using Numerical Kernel law with $\gamma = 10$, and constant overhead model

B.1.2 Variable checkpoint and recovery cost



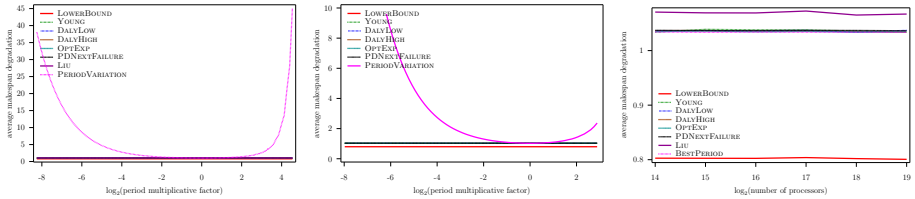
a) 262144 processors b) 1048576 processors c)

Figure 37: Evaluation of the different heuristics on a Exascale platform with Exponential failures, using embarrassingly parallel job, and variable overhead model ($C(p) = 600\frac{1048576}{p}$)



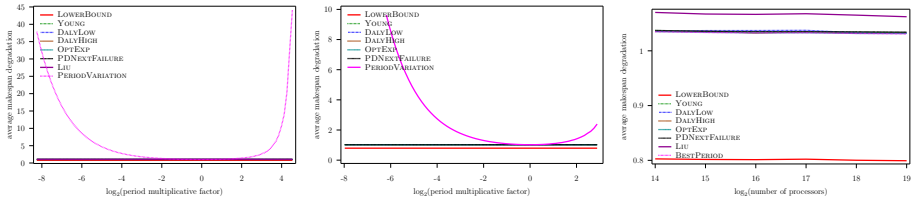
a) 262144 processors b) 1048576 processors c)

Figure 38: Evaluation of the different heuristics on a Exascale platform with Exponential failures, using Amdahl law with $\gamma = 10^{-6}$, and variable overhead model ($C(p) = 600\frac{1048576}{p}$)



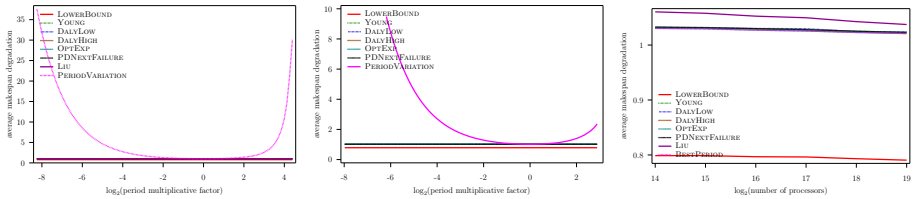
a) 262144 processors b) 1048576 processors c)

Figure 39: Evaluation of the different heuristics on a Exascale platform with Exponential failures, using Numerical Kernel law with $\gamma = 0.1$, and variable overhead model ($C(p) = 600\frac{1048576}{p}$)



a) 262144 processors b) 1048576 processors c)

Figure 40: Evaluation of the different heuristics on a Exascale platform with Exponential failures, using Numerical Kernel law with $\gamma = 1$, and variable overhead model ($C(p) = 600\frac{1048576}{p}$)

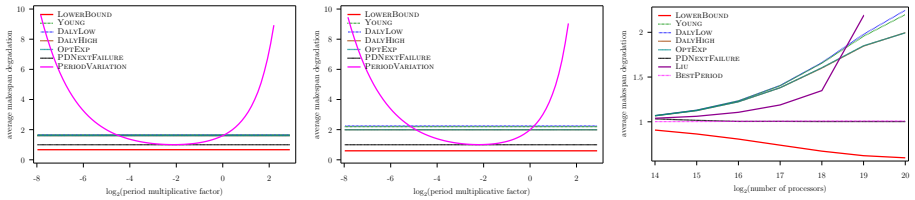


a) 262144 processors b) 1048576 processors c)

Figure 41: Evaluation of the different heuristics on a Exascale platform with Exponential failures, using Numerical Kernel law with $\gamma = 10$, and variable overhead model ($C(p) = 600\frac{1048576}{p}$)

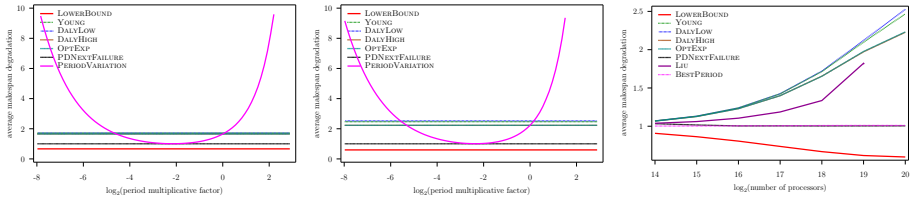
B.2 Weibull law

B.2.1 Fixed checkpoint and recovery cost



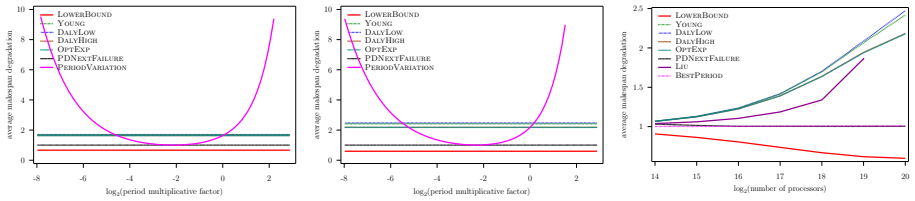
a) 262144 processors b) 1048576 processors c)

Figure 42: Evaluation of the different heuristics on a Exascale platform with Weibull failures, using Amdahl law with $\gamma = 10^{-6}$, and constant overhead model



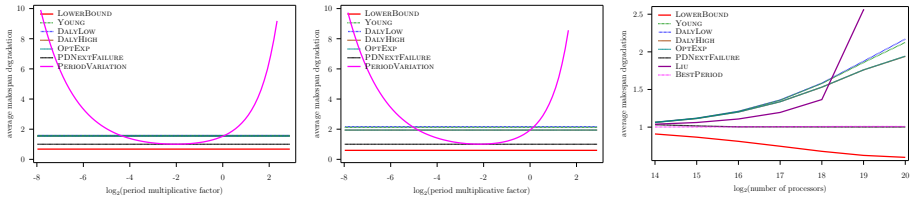
a) 262144 processors b) 1048576 processors c)

Figure 43: Evaluation of the different heuristics on a Exascale platform with Weibull failures, using Numerical Kernel law with $\gamma = 0.1$, and constant overhead model



a) 262144 processors b) 1048576 processors c)

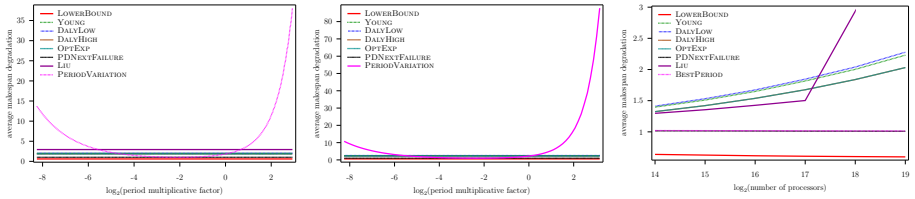
Figure 44: Evaluation of the different heuristics on a Exascale platform with Weibull failures, using Numerical Kernel law with $\gamma = 1$, and constant overhead model



a) 262144 processors b) 1048576 processors c)

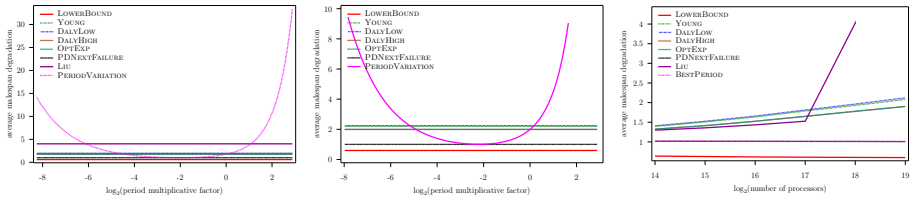
Figure 45: Evaluation of the different heuristics on a Exascale platform with Weibull failures, using Numerical Kernel law with $\gamma = 10$, and constant overhead model

B.2.2 Variable checkpoint and recovery cost



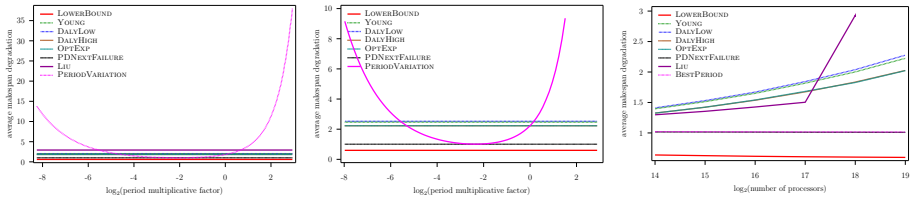
a) 262144 processors b) 1048576 processors c)

Figure 46: Evaluation of the different heuristics on a Exascale platform with Weibull failures, using embarrassingly parallel job, and variable overhead model ($C(p) = 600\frac{1048576}{p}$)



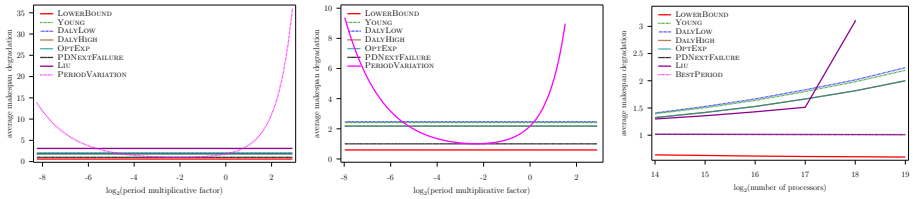
a) 262144 processors b) 1048576 processors c)

Figure 47: Evaluation of the different heuristics on a Exascale platform with Weibull failures, using Amdahl law with $\gamma = 10^{-6}$, and variable overhead model ($C(p) = 600\frac{1048576}{p}$)



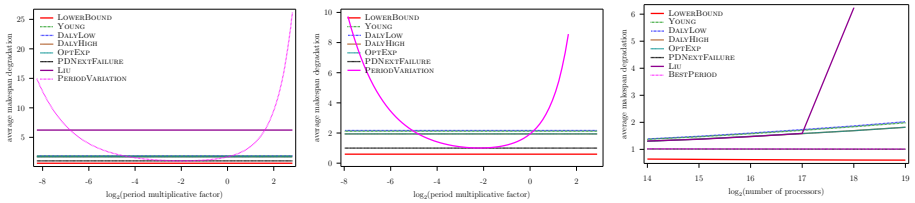
a) 262144 processors b) 1048576 processors c)

Figure 48: Evaluation of the different heuristics on a Exascale platform with Weibull failures, using Numerical Kernel law with $\gamma = 0.1$, and variable overhead model ($C(p) = 600\frac{1048576}{p}$)



a) 262144 processors b) 1048576 processors c)

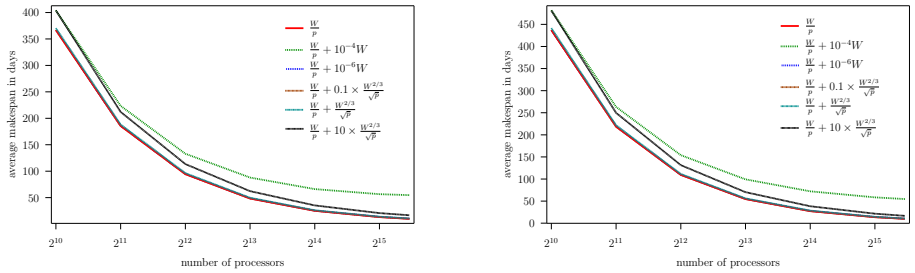
Figure 49: Evaluation of the different heuristics on a Exascale platform with Weibull failures, using Numerical Kernel law with $\gamma = 1$, and variable overhead model ($C(p) = 600\frac{1048576}{p}$)



a) 262144 processors b) 1048576 processors c)

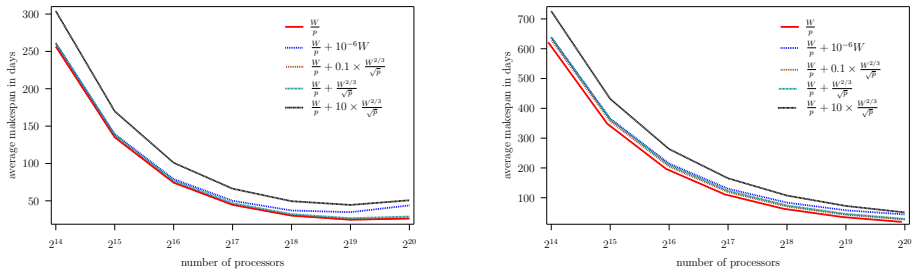
Figure 50: Evaluation of the different heuristics on a Exascale platform with Weibull failures, using Numerical Kernel law with $\gamma = 10$, and variable overhead model ($C(p) = 600\frac{1048576}{p}$)

C Appendix: results for fixed heuristic according to application model variation



a) b)

Figure 51: Evolution of the makespan for OptExp as a function of the platform size for the different application profiles and a constant (a)) or platform-dependent (b)) checkpoint cost (exponential law).



a) b)

Figure 52: Evolution of the makespan for PDNextFailure as a function of the platform size for the different application profiles and a constant (a)) or platform-dependent (b)) checkpoint cost (Weibull law).

Centre de recherche INRIA Grenoble – Rhône-Alpes
655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399