HAL
open science

Using group replication for resilience on exascale systems

Marin Bougeret, Henri Casanova, Yves Robert, Frédéric Vivien, Dounia Zaidouni

To cite this version:

Marin Bougeret, Henri Casanova, Yves Robert, Frédéric Vivien, Dounia Zaidouni. Using group replication for resilience on exascale systems. [Research Report] RR-7876, INRIA. 2012. hal-00668016v2

HAL Id: hal-00668016

https://inria.hal.science/hal-00668016v2

Submitted on 28 Jun 2013

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

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

Using group replication for resilience on exascale systems

Marin Bougeret, Henri Casanova, Yves Robert, Frédéric Vivien, Dounia Zaidouni

RESEARCH REPORT
N° 7876
February 2012
Project-Team ROMA

ISSN 0249-6399 ISRN INRIA/RR--7876--FR+ENG

Using group replication for resilience on exascale systems

Marin Bougeret*, Henri Casanova†, Yves Robert‡§¶, Frédéric Vivien‖‡, Dounia Zaidouni‖‡

Project-Team ROMA

Research Report n° 7876 — version 2 — initial version February 2012 — revised version June 2013 — 55 pages

* LIRMM Montpellier, France
† University of Hawai'i at Mānoa, Honolulu, USA
‡ LIP, Ecole Normale Supérieure de Lyon, France
§ University of Tennessee Knoxville, USA
¶ Institut Universitaire de France.
‖ INRIA, Lyon, France

RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

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

Abstract: High performance computing applications must be resilient to faults, which are common occurrences especially in post-petascale settings. The traditional fault-tolerance solution is checkpoint-recovery, by which the application saves its state to secondary storage throughout execution and recovers from the latest saved state in case of a failure. An oft studied research question is that of the optimal checkpointing strategy: when should state be saved? Unfortunately, even using an optimal checkpointing strategy, the checkpointing frequency must increase as platform scale increases, leading to higher checkpointing overhead. This overhead precludes high parallel efficiency for large-scale platforms, thus mandating other more scalable fault-tolerance mechanisms. One such mechanism is replication, which can be used in addition to checkpoint-recovery. Using replication, multiple processors perform the same computation so that a processor failure does not necessarily imply application failure. While at first glance replication may seem wasteful, it may be significantly more efficient than using solely checkpoint-recovery at large scale. In this work we investigate a simple approach where entire application instances are replicated. We provide a theoretical study of checkpoint-recovery with replication in terms of expected application execution time, under an exponential distribution of failures. We design dynamic-programming based algorithms to define checkpointing dates that work under any failure distribution. We also conduct simulation experiments assuming that failures follow Exponential or Weibull distributions, the latter being more representative of real-world systems, and using failure logs from production clusters. Our results show that replication is useful in a variety of realistic application and checkpointing cost scenarios for future exascale platforms.

Key-words: Fault-tolerance, replication, checkpointing, parallel job, Weibull, exascale

La réplication pour l'amélioration de la résilience des applications sur systèmes exascales

Résumé : Les applications de calcul à haute-performance doivent être résiliantes aux pannes, car les pannes ne seront pas des évènements rares sur les plates-formes post-petascales. La tolérance aux pannes est traditionnellement réalisée par un mécanisme d'enregistrement et redémarrage, au moyen duquel l'application sauve son état sur un système de stockage secondaire et, en cas de panne, redémarre à partir du dernier état sauvegardé. Une question souvent étudiée est celle de la stratégie de sauvegarde optimale: quand l'état doit-il être sauvé ? Malheureusement, même quand on utilise une stratégie de sauvegarde optimale, la fréquence de sauvegarde doit augmenter avec la taille de la plate-forme, augmentant mécaniquement le coût des sauvegardes. Ce coût interdit d'obtenir une très bonne efficacité sur des plates-formes à très large échelle, et requiert d'utiliser d'autres mécanismes de tolérance aux pannes, qui passent mieux à l'échelle. Un mécanisme potentiel est la réplication, qui peut être utilisée conjointement avec une solution de sauvegarde et redémarrage. Avec la réplication, plusieurs processeurs exécutent le même calcul de sorte que la panne de l'un d'entre eux n'implique pas nécessairement une panne pour l'application. Alors qu'à première vue une telle approche gaspille des ressources, la réplication peut être significativement plus efficace que la seule mise en œuvre de techniques de sauvegarde et redémarrage sur des plates-formes à très grande échelle. Dans la présente étude nous considérons une approche simple où une application toute entière est répliquée. Nous fournissons une étude théorique d'un schéma d'exécution avec réplication lorsque la distribution des pannes suit une loi exponentielle. Nous proposons des algorithmes de détermination des dates de sauvegarde quand la distribution des pannes suit une loi quelconque. Nous menons aussi une étude expérimentale, au moyen de simulations, basée sur une distribution de pannes suivant une loi exponentielle, de Weibull (ce qui est plus représentatif des systèmes réels), ou tirée de logs de clusters utilisés en production. Nos résultats montrent que la réplication est bénéfique pour un ensemble de modèles d'applications et de coût de sauvegardes réalistes, dans le cadre des futures plates-formes exascales.

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

1 Introduction

As plans are made for deploying post-petascale high performance computing (HPC) systems [10, 23], solutions need to be developed to ensure resilience to failures that occur because not all faults can be automatically detected and corrected in hardware. For instance, the 224,162-core Jaguar platform is reported to experience on the order of 1 failure per day [21, 2], and its scale is modest compared to platforms in the plans for the next decade. For applications that enroll large numbers of, or perhaps all, processors a failure is the common case rather than the exception. One can recover from a failure by resuming execution from a previously saved fault-free execution state, or *checkpoint*. Checkpoints are saved to resilient storage throughout execution (usually periodically). More frequent checkpoints lead to less loss when a failure occurs but to higher overhead during fault-free execution. A *checkpointing strategy* specifies when checkpoints should be taken.

A large literature is devoted to developing efficient checkpointing strategies, i.e., ones that minimize expected job execution time, including both theoretical and practical efforts. The former typically rely on assumptions regarding the probability distributions of times to failure of the processors (e.g., Exponential, Weibull), while the latter rely on simulations driven by failure datasets obtained on real-world platforms. In a previous paper [4], we have made several contributions in this context, including optimal solutions for Exponential failures and dynamic programming solutions in the general case.

A major issue with checkpoint-recovery is scalability: the necessary checkpoint frequency for tolerating failures in large-scale platforms is so large that processors spend more time saving state than computing. It is thus expected that future platforms will lead to unacceptably low parallel efficiency if only checkpoint-recovery is used, no matter how good the checkpointing strategy. Consequently, additional mechanisms must be used. In this work we focus on *replication*: several processors perform the same computation synchronously, so that a fault on one of these processors does not lead to an application failure. Replication is an age-old fault-tolerant technique, but it has gained traction in the HPC context only relatively recently. While replication wastes compute resources in fault-free executions, it can alleviate the poor scalability of checkpoint-recovery.

Consider a parallel application that is *moldable*, meaning that it can be executed on an arbitrary number of processors, which each processor running one application process. In our *group replication* approach, multiple application instances are executed. One could, for instance, execute 2 distinct n-process application instances on a $2n$-processor platform. Each instance runs at a smaller scale, meaning that it has better parallel efficiency than a single $2n$-process instance due to a lower checkpointing frequency. Furthermore, once an instance saves a checkpoint, the other instance can use this checkpoint immediately. Given the above, our contributions in this work are:

- A theoretical analysis of the optimal number of processors to use for a checkpoint-recovery execution of a parallel application, for various parallel workload models for Exponential failure distributions;
- An effective approach for group replication, with a theoretical analysis bounding expected execution time for Exponential failure distribution, and several dynamic programming solutions working for general failure

distributions;
- Extensive simulations showing that group replication can indeed lower application running times, and that some of our proposed strategies deliver good performance.

The rest of this paper is organized as follows. Section 2 discusses related work. Section 3 defines the theoretical framework and states key assumptions. Section 4 discusses the optimal number of processors for a checkpoint-recovery execution of a parallel application under an Exponential distribution of failures. Section 5 provides several approaches for group replication, and the theoretical analysis of one of them. Section 6, resp. Section 7, describes our experimental methodology and results. Each section corresponds to a different set of results, for historical reasons. Finally, Section 8 concludes with a summary of our findings and with future perspectives.

2 Related work

Checkpointing policies have been widely studied in the literature. In [9], Daly studies periodic checkpointing policies for Exponentially distributed failures, generalizing the well-known bound obtained by Young [29]. Daly extended his work in [16] to study the impact of sub-optimal checkpointing periods. In [26], the authors develop an "optimal" checkpointing policy, based on the popular assumption that optimal checkpointing must be periodic. In [6], Bouguerra et al. *prove* that the optimal checkpointing policy is periodic when checkpointing and recovery overheads are constant, for either Exponential or Weibull failures. But their results rely on the unstated assumption that all processors are rejuvenated after each failure and after each checkpoint. In our recent work [4], we have shown that this assumption is unreasonable for Weibull failures. We have developed optimal solutions for Exponential failures and dynamic programming solutions for any failure distribution, demonstrating performance improvements over checkpointing approaches proposed in the literature in the case of Weibull and log-based failures. The Weibull distribution is recognized as a reasonable approximation of failures in real-world systems [14, 25]. The work in this paper relates to checkpointing policies in the sense that we study a replication mechanism that is used as an addition to checkpointing. Part of our results build on the algorithms and results in [4].

In spite of all the above advances, several studies have questioned the feasibility of pure checkpoint-recovery for large-scale systems (see [12] for a discussion of this issue and for references to such studies). In this work, we study the use of replication as a mechanism complementary to checkpoint-recovery. Replication has long been used as a fault-tolerance mechanism in distributed systems [13] and more recently in the context of volunteer computing [18]. The idea to use replication together with checkpoint-recovery has been studied in the context of grid computing [28].One concern about replication in HPC is the induced resource waste. However, given the scalability limitations of pure checkpoint-recovery, replication has recently received more attention in the HPC literature [24, 31, 11].

In this work we study "group replication," by which multiple application instances are executed on different groups of processors.An orthogonal approach, "process replication," was recently studied by Ferreira et al. [12] in the context

of MPI applications. To achieve fault-tolerance, each MPI process is replicated in a way that is transparent to the application developer. While this approach can lead to good fault-tolerance, one of its drawbacks is that it increases the number and the volume of communications. Let V_{tot} be the total volume of inter-processor communications for a traditional execution. With process replication using g replicas per replica-groups, each original communication now involves g sources and g destinations, hence the total communication volume becomes $V_{tot} \times g^2$. Instead, with group replication using g groups, each original communication takes place g times, hence the total communication volume increases only to $V_{tot} \times g$. Another drawback of process replication is that it requires the use of a customized MPI library (such as the prototype developed by the authors in [12]). By contrast, group replication is completely agnostic to the parallel runtime system and thus does not even require MPI. Nevertheless, even for MPI applications, group replication provides an out of the box fault-tolerance solution that can be used until process replication possibly becomes a mainstream feature in MPI implementations.

3 Framework

We consider the execution of a tightly-coupled parallel application, or *job*, on a platform composed of p processors. We use the term processor to indicate any individually scheduled compute resource (a core, a multi-core processor, a cluster node) so that our work applies regardless of the granularity of the platform. We assume that system-level checkpoint-recovery is enabled.

The job must complete $\mathcal{W}$ units of (divisible) work, which can be split arbitrarily into separate *chunks*. The job can execute on any number $q \leq p$ processors. Letting $\mathcal{W}(q)$ be the time required for a failure-free execution on q processors, we use three models:

- Perfectly parallel jobs: $\mathcal{W}(q) = \mathcal{W}/q$.
- Generic parallel jobs: $\mathcal{W}(q) = (1-\gamma)\mathcal{W}/q + \gamma\mathcal{W}$. As in Amdahl's law [1], $\gamma < 1$ is the fraction of the work that is inherently sequential.
- Numerical kernels: $\mathcal{W}(q) = \mathcal{W}/q + \gamma\mathcal{W}^{2/3}/\sqrt{q}$. 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], $q = r^2$ and each processor receives $2r$ blocks of size N^2/r^2 during the execution. Here γ is the communication-to-computation ratio of the platform.

Each participating processor is subject to *failures*. A failure causes a *downtime* period of the failing processor, of duration D. When a processor fails, the whole execution is stopped, and all processors must recover from the previous checkpoint. We let $C(q)$ denote the time needed to perform a checkpoint, and $R(q)$ the time to perform a recovery. The downtime accounts for software rejuvenation (i.e., rebooting [17, 8]) or for the replacement of the failed processor by a spare. Regardless, we assume that after a downtime the processor is fault-free and begins a new lifetime at the beginning of the recovery period. This recovery period corresponds to the time needed to restore the last checkpoint. Assuming that the application's memory footprint is V bytes, with each processor holding V/q bytes, we consider two scenarios:

- Proportional overhead: $C(q) = R(q) = \alpha V/q = C/q$ with α some constant, for cases where the bandwidth of the network card/link at each

processor is the I/O bottleneck.

- Constant overhead: $C(q) = R(q) = \alpha V = C$ with α some constant, for cases where the bandwidth to/from the resilient storage system is the I/O bottleneck.

We assume coordinated checkpointing [27], meaning that no message logging/replay is needed when recovering from failures. We assume that failures can happen during recovery or checkpointing, but not during a downtime (otherwise, the downtime period could be considered part of the recovery period). We assume that the parallel job is tightly coupled, meaning that all q processors operate synchronously throughout the job execution. These processors execute the same amount of work $\mathcal{W}(q)$ in parallel, chunk by chunk. The total time (on one processor) to execute a chunk of size ω, and then checkpointing it, is $\omega + C(q)$. Finally, we assume that failure arrivals at all processors are independent and identically distributed (*iid*).

4 Optimal number of processors for execution

Let $\mathbb{E}(q)$ be the expectation of the execution time, or *makespan*, when using q processors, and q_{opt} the value of q that minimizes $\mathbb{E}(q)$. Is it true that the optimal solution is to use all processors, i.e., $q_{opt} = p$? If not, what can we say about the value of q_{opt}? This question was partially and empirically addressed in [26], via experiments for 4 MPI applications for up to 35 processors. Our approach here is radically different since we target large-scale platforms and seek theoretical results in the form of optimal solutions. The main objective of this section is to show analytically that, for Exponential failures, $\mathbb{E}(q)$ may reach its minimum for some finite value of q (implying that q_{opt} is not necessarily equal to p).

Assume that failure inter-arrival times follow an Exponential distribution with parameter λ. In our recent work [4], we have shown that the optimal strategy to minimize the expected makespan $\mathbb{E}(q)$ is to split $\mathcal{W}$ into $K^* = \max(1, \lfloor K_0(q) \rfloor)$ or $K^* = \lceil K_0(q) \rceil$ same-size chunks, whichever leads to the smaller value, where $K_0(q) = \frac{q\lambda\mathcal{W}(q)}{1+\mathbb{L}(-e^{-q\lambda C(q)-1})}$ is the optimal (non integer) number of chunks. $\mathbb{L}$ denotes the Lambert function, defined as $\mathbb{L}(z)e^{\mathbb{L}(z)} = z$. This result shows that the optimal strategy is periodic and that the optimal expectation of the makespan is:

$$\mathbb{E}^*(q) = K^*(q)\left(\frac{1}{q\lambda} + \mathbb{E}(X_D(q))\right) e^{q\lambda R(q)} \left(e^{\frac{q\lambda\mathcal{W}(q)}{K^*(q)} + q\lambda C(q)} - 1\right) \tag{1}$$

where $\mathbb{E}(X_D(q))$ denotes the expectation of the downtime. It turns out that, although we can compute the optimal number of chunks (and thus the chunk size), we cannot compute $\mathbb{E}^*(q)$ analytically because $\mathbb{E}(X_D(q))$ is difficult to compute. This is because a processor can fail while another one is down, thus prolonging the downtime. With a single processor ($q = 1$), $X_D(q)$ has constant value D, but with several processors there could be cascading downtimes. It turns out that we can compute the following lower and upper bounds for $\mathbb{E}(X_D(q))$:

Proposition 1. *Let $X_D(q)$ denote the downtime of a group of q processors. Then*

$$D \leq \mathbb{E}(X_D(q)) \leq \frac{e^{(q-1)\lambda D} - 1}{(q-1)\lambda} \tag{2}$$

Proof. In [4], we have shown that th optimal expectation of the makespan is computed as:

$$\mathbb{E}^*(q) = K^*(q)\left(\frac{1}{q\lambda} + \mathbb{E}(T_{rec}(q))\right)\left(e^{\frac{q\lambda\mathcal{W}(q)}{K^*(q)}+q\lambda C(q)} - 1\right) \tag{3}$$

where $\mathbb{E}(T_{rec}(q))$ denotes the expectation of the recovery time, i.e., the time spent recovering from failure during the computation of a chunk. All chunks have the same recovery time because they all have the same size and because of the memoryless property of the Exponential distribution. It turns out that although we can compute the optimal number of chunks (and thus the chunk size), we cannot compute $\mathbb{E}^*(q)$ analytically because $\mathbb{E}(T_{rec}(q))$ is difficult to compute. We write the following recursion:

$$T_{rec}(q) = \begin{cases} X_D(q) + R(q) & \text{if no processor fails} \\ & \text{during } R(q) \text{ units of time,} \\ X_D(q) + T_{lost}(R(q)) + T_{rec}(q) & \text{otherwise.} \end{cases} \tag{4}$$

$X_D(q)$ is the downtime of a group of q processors, that is the time between the first failure of one of the processors and the first time at which all of them are available (accounting for the fact a processor can fail while another one is down, thus prolonging the downtime). $T_{lost}(R(q))$ is the amount of time spent computing by these processors before a first failure, knowing that the next failure occurs within the next $R(q)$ units of time. In other terms, it is the compute time that is wasted because checkpoint recovery was not completed. The time until the next failure of a group of q processors is the minimum of q *iid* Exponentially distributed variables, and is thus Exponential with parameter $q\lambda$. We can compute $\mathbb{E}(T_{lost}(R(q))) = \frac{1}{q\lambda} - \frac{R(q)}{e^{q\lambda R(q)}-1}$ (see [4] for details). Plugging this value into Equation 4 leads to:

$$\begin{aligned}\mathbb{E}(T_{rec}(q)) &= e^{-q\lambda R(q)}(\mathbb{E}(X_D(q)) + R(q)) \\ &\quad + (1 - e^{-q\lambda R(q)})\left(\mathbb{E}(X_D(q)) + \frac{1}{q\lambda} - \frac{R(q)}{e^{q\lambda R(q)} - 1} + \mathbb{E}(T_{rec}(q))\right)\end{aligned} \tag{5}$$

Equation 5 reads as follows: after the downtime $X_D(q)$, either the recovery succeeds for everybody, or there is a failure during the recovery and another attempt must be made. Both events are weighted by their respective probabilities. Simplifying the above expression we get:

$$\mathbb{E}(T_{rec}(q)) = \mathbb{E}(X_D(q))e^{q\lambda R(q)} + \frac{1}{q\lambda}(e^{q\lambda R(q)} - 1) \tag{6}$$

Plugging back this expression in Equation 3, we obtain the value given in Equation 1.

Now we establish the desired bounds on $\mathbb{E}(X_D(q))$ We always have $X_D(q) \geq X_D(1) \geq D$, hence the lower bound. For the upper bound, consider a date at which one of the q processors, say processor i_0, just had a failure and initiates its downtime period for D time units. Some other processors might be in the middle of their downtime period: for each processor i, $1 \leq i \leq q$, let t_i denote the remaining duration of the downtime of processor i. We have $0 \leq t_i \leq D$ for

$1 \leq i \leq q$, $t_{i_0} = D$, and $t_i = 0$ means that processor i is up and running. Let $X_D^{t_1,..,t_q}(q)$ be the *remaining* downtime of a group of q processors, knowing that processor i, $1 \leq i \leq q$, will still be down for a duration of t_i, and that a failure just happened (i.e., there exists i_0 such that $t_{i_0} = D$). Given the values of the t_i's, we have the following equation for the random variable $X_D^{t_1,..,t_q}(q)$:

$$X_D^{t_1,..,t_q}(q) = \begin{cases} D \\ \quad \text{if none of the processors of the group} \\ \quad \text{fails during the next } D \text{ units of time} \\ T_{lost}^{t_1,..,t_q}(D) + X_D^{t'_1,..,t'_q}(q) \\ \quad \text{otherwise.} \end{cases}$$

In the second case of the equation, consider the next D time-units. Processor i can only fail in the *last* $D - t_i$ of these time-units. Here the values of the t'_i's depend on the t_i's and on $T_{lost}^{t_1,..,t_q}(D)$. Indeed, except for the last processor to fail, say i_1, for which $t'_{i_1} = D$, we have $t'_i = \max\{t'_i - T_{lost}^{t_1,..,t_q}(D), 0\}$. More importantly we always have $T_{lost}^{t_1,..,t_q}(D) \leq T_{lost}^{D,0,...,0}(D)$ and $X_D^{t_1,..,t_q}(q) \leq X_D^{D,0,..,0}(q)$ because the probability for a processor to fail during D time units is always larger than that to fail during $D - t_i$ time-units. Thus, $\mathbb{E}(X_D^{t_1,..,t_q}(q)) \leq \mathbb{E}(X_D^{D,0,..,0}(q))$. Following the same line of reasoning, we derive an upper-bound for $X_D^{D,0,..,0}(q)$:

$$X_D^{D,0,..,0}(q) \leq \begin{cases} D \\ \quad \text{if none of the } q-1 \text{ running processors of the group} \\ \quad \text{fails during the downtime } D \\ T_{lost}^{D,0,..,0}(D) + X_D^{D,0,..,0}(q) \\ \quad \text{otherwise.} \end{cases}$$

Weighting both cases by their probability and taking expectations, we obtain

$$\begin{aligned} &\mathbb{E}\left(X_D^{D,0,..,0}(q)\right) \\ &\quad \leq e^{-(q-1)\lambda D} D + (1 - e^{-(q-1)\lambda D}) \left(E\left(T_{lost}^{D,0,..,0}(D)\right) + E\left(X_D^{D,0,..,0}(q)\right)\right) \end{aligned}$$

hence $\mathbb{E}\left(X_D^{D,0,..,0}(q)\right) \leq D + (e^{(q-1)\lambda D} - 1) E\left(T_{lost}^{D,0,..,0}(D)\right)$, with

$$E\left(T_{lost}^{D,0,..,0}(D)\right) = \frac{1}{(q-1)\lambda} - \frac{D}{e^{(q-1)\lambda D} - 1}.$$

We derive

$$\mathbb{E}\left(X_D^{t_1,..,t_q}(q)\right) \leq \mathbb{E}\left(X_D^{D,..,0}(q)\right) \leq \frac{e^{(q-1)\lambda D} - 1}{(q-1)\lambda}.$$

which concludes the proof. As a sanity check, we observe that the upper bound is at least D, using the identity $e^x \geq 1 + x$ for $x \geq 0$. □

While in a failure-free environment $\mathbb{E}^*(q)$ would always decrease as q increases, using the above lower bound on $\mathbb{E}(X_D(q))$ we obtain the following results:

Theorem 1. *When the failure distribution follows an Exponential law, $\mathbb{E}^*(q)$ reaches its minimum for some finite value of q in the following scenarios: all job types (perfectly parallel, generic and numerical) with constant overhead, and generic or numerical jobs with proportional overhead.*

Note that the only open scenario is with perfectly parallel jobs and proportional overhead. In this case the lower bound on $\mathbb{E}^*(q)$ decreases to some constant value while the upper bound goes to $+\infty$ as q increases.

Proof. We show that $\lim_{q\to+\infty} \mathbb{E}^*(q) = +\infty$ for the relevant scenarios. We first plug the lower-bound of Equation 2 into Equation 6 and obtain:

$$\mathbb{E}(T_{rec}(q)) \geq De^{q\lambda R(q)} + \frac{1}{q\lambda}\left(e^{q\lambda R(q)} - 1\right).$$

From Equation 1 we then derive the lower-bound:

$$\mathbb{E}^*(q) \geq K_0(q)\left(\frac{1}{q\lambda} + D\right) e^{q\lambda R(q)} \left(e^{\frac{q\lambda \mathcal{W}(q)}{K_0(q)} + q\lambda C(q)} - 1\right)$$

using the fact that, by definition, the expression in the right hand-side of Equation 1 is minimized by K_0, where $K_0(q) = \frac{q\lambda\mathcal{W}(q)}{1+\mathbb{L}(-e^{-q\lambda C(q)-1})}$.

With constant overhead. Let us consider the case of perfectly parallel jobs ($\mathcal{W}(q) = \mathcal{W}/q$) with constant checkpointing overhead ($C(q) = R(q) = C$). We get the lower bound:

$$\mathbb{E}^*(q) \geq K_0(q)\left(\frac{1}{q\lambda} + D\right) e^{q\lambda C} \left(e^{\frac{\lambda\mathcal{W}}{K_0(q)} + q\lambda C} - 1\right)$$

where $K_0(q) = \frac{\lambda\mathcal{W}}{1+\mathbb{L}(-e^{-q\lambda C-1})}$. When q tends to $+\infty$, $K_0(q)$ goes to $\lambda\mathcal{W}$, while $(\frac{1}{q\lambda} + D)e^{q\lambda C}\left(e^{\frac{\lambda\mathcal{W}}{K_0(q)} + q\lambda C} - 1\right)$ goes to $+\infty$. Consequently, $\mathbb{E}^*(q)$ is bounded below by a quantity that goes to $+\infty$, which concludes the proof. This result also implies that $\mathbb{E}^*(q)$ reaches a minimum for a finite q value for other job types (generic, numerical) with constant overhead, just because the execution time is larger in those cases than with perfectly parallel jobs.

Generic parallel job with proportional overhead. Here we assume that $\mathcal{W}(q) = \mathcal{W}/q + \gamma\mathcal{W}$, and use proportional overhead: $C(q) = R(q) = \frac{C}{q}$. We get the lower bound:

$$\mathbb{E}^*(q) \geq K_0(q)\left(\frac{1}{q\lambda} + D\right) e^{\lambda C} \left(e^{\frac{\lambda\mathcal{W} + q\lambda\gamma\mathcal{W}}{K_0(q)} + \lambda C} - 1\right)$$

where $K_0(q) = \frac{\lambda\mathcal{W}+q\lambda\gamma\mathcal{W}}{1+\mathbb{L}(-e^{-\lambda C-1})}$. As before, we show that $\lim_{q\to+\infty} \mathbb{E}^*_{min}(q) = +\infty$ to get the result. When q tends to $+\infty$, $K_0(q)$ tends to $+\infty$, while $(\frac{1}{q\lambda} + D)e^{\lambda C}\left(e^{\frac{\lambda\mathcal{W}+q\lambda\gamma\mathcal{W}}{K_0(q)} + \lambda C} - 1\right)$ tends to some positive constant. This concludes the proof. Note that this proof also serves for generic parallel jobs with constant overhead, simply because the execution time is larger in that case than with proportional overhead.

Numerical kernels with proportional overhead. Here we assume that $\mathcal{W}(q) = \mathcal{W}/q + \gamma\mathcal{W}^{2/3}/\sqrt{q}$, and use proportional overhead: $C(q) = R(q) = \frac{C}{q}$. We get the lower bound:

$$\mathbb{E}^*(q) \geq K_0(q)\left(\frac{1}{q\lambda} + D\right)e^{\lambda C}\left(e^{\frac{\lambda\mathcal{W}+\lambda\gamma\mathcal{W}^{2/3}\sqrt{q}}{K_0(q)}+\lambda C} - 1\right)$$

where $K_0(q) = \frac{\lambda\mathcal{W}+\lambda\gamma\mathcal{W}^{2/3}\sqrt{q}}{1+\mathbb{L}(-e^{-\lambda C-1})}$. As before, we show that $\lim_{q\to+\infty}\mathbb{E}^*_{min}(q) = +\infty$ to get the result. When q tends to $+\infty$, $K_0(q)$ tends to $+\infty$, while $(\frac{1}{q\lambda} + D)e^{\lambda C}\left(e^{\frac{\lambda\mathcal{W}+\lambda\gamma\mathcal{W}^{2/3}\sqrt{q}}{K_0(q)}+\lambda C} - 1\right)$ tends to some positive constant. This concludes the proof. □

5 Group replication

Since using all processors to run a single application instance may not make sense in light of Theorem 1, the *group replication* approach consists in executing multiple application instances on different processor groups, where the number of processors in a group is closer to q_{opt}. All groups compute the same chunk simultaneously, and do so until one of them succeeds, potentially after several failed trials. Then all other groups stop executing that chunk and recover from the checkpoint stored by the successful group. All groups then attempt to compute the next chunk. Group replication can be implemented easily with no modification to the application, provided that the recovery implementation allows a group to recover immediately from a checkpoint produced by another group. In this section we formalize group replication as an execution protocol called ASAP (As Soon As Possible), and analyze its performance for Exponential failures. We then introduce dynamic programming solutions that work with general failure distributions.

5.1 The ASAP execution protocol

We consider g groups, where each group has q processors, with $g \times q \leq p$. A group is available for execution if and only if all its q processors are available. In case of a failure, the downtime of a group is a random variable $X_D(q) \geq D$, whose expectation is bounded in Proposition 1. If a group encounters a first processor failure at time t, the group is *down* between t and $t + X_D(q)$.

The ASAP algorithm proceeds in k macro-steps. During macro-step j, $1 \leq j \leq k$, each group independently attempts to execute the j-th chunk of size ω_j and to checkpoint, restarting as soon as possible in case of a failure. As soon as one of the groups succeeds, say at time t_j^{end}, all the other groups are immediately stopped, macro-step j is over, and macro-step $(j+1)$ starts (if $j < k$). Note that the value of k, the total number of chunks, as well as the chunk sizes, the ω_j's, are inputs to the algorithm (we always have $\sum_{j=1}^{k}\omega_j = \mathcal{W}(q)$). We provide an analytical evaluation of ASAP for Exponential failure laws, and discuss how to choose these values, in Section 5.2.

Two important aspects must be mentioned. First, before being able to start macro-step $(j+1)$, a group that has been stopped must execute a recovery, in order to restart from the checkpoint of a successful group. Second, this recovery

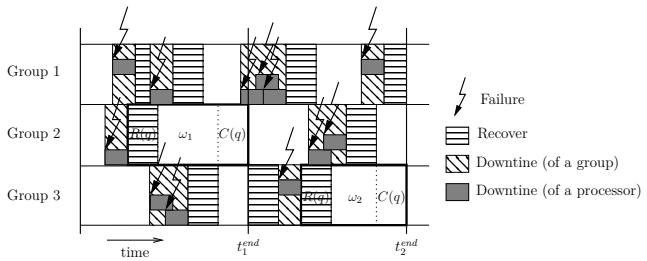


Figure 1: Execution of chunks ω_1 and ω_2 (macro-steps 1 and 2) using the ASAP protocol. At time t_1^{end}, Group 1 is not ready, and Group 2 is the only one that does not need to recover.

may start later than time t_j^{end}, in the case where the group is down at time t_j^{end}. An example is shown in Figure 1, in which group 1 cannot start the recovery at time t_j^{end}. The only groups that do not need to recover at the beginning of the next step are the groups that were successful for the previous step, except during the first step at which all groups can start computing right away.

We now provide an analytical evaluation of ASAP for Exponential failure laws, and show how to compute the number of macro-steps k and the values of the chunk sizes ω_j.

5.2 Exponential failures

Let use assume that the failure rate of each processor obeys an Exponential law of parameter λ. For the sake of the theoretical analysis, we introduce a slightly modified version of the ASAP protocol in which all groups, including the successful ones, execute a recovery at the beginning of all macro-steps, including the first one. This new version of ASAP is described in Algorithm 1. It is completely symmetric, which renders its analysis easier: for macro-step j to be successful, one of the groups must be up and running for a duration of $R(q) + \omega_j + C(q)$.

Algorithm 1: ASAP ($\omega_1, \ldots, \omega_k$)

```
for j = 1 to k do
    for each group do in parallel
        repeat
            Finish current downtime (if any)
            Try to perform a recovery, then a chunk of size ω_j, and finally to
            checkpoint
            if execution successful then
                Signal other groups to immediately stop their attempts
        until one of the groups has a successful attempt
```

Consider the j-th macro step, number the attempts of all groups by their start time, and let N_j be the index of the earliest started attempt that successfully computes chunk ω_j. In Figure 2, we have $j = 2$, the successful chunk of size $R+\omega_2+C$ is the fourth attempt, so $N_2 = 4$. To represent each attempt, we sample random variables X_i^j and Y_i^j, $1 \leq i \leq N_j$, that correspond respectively

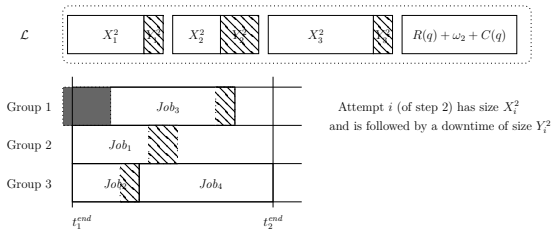


Figure 2: Zoom on macro-step 2 of the execution depicted in Figure 1, using the (X, Y) notation of Algorithm 2. Recall that Job_i has size $X_i^2 + Y_i^2$ for $1 \leq i \leq 3$, and Job_4 has size $R(q) + \omega_2 + C(q)$.

to the ith tentative execution of the chunk and to the ith downtime that follows it (if $i \neq N_j$). Note that $X_i^j < R + \omega_j + C$ for $i < N_j$, and $X_{N_j}^j \geq R + \omega_j + C$. All the X_i^j's follow the same distribution D_X, namely an Exponential law of parameter $q\lambda$. And all the Y_i^j's follow the same distribution $D_{X_D}(q)$, that of the the random variable $X_D(q)$ corresponding to the downtime of a group of q processors.

The main idea here is to view the N_j execution attempts as jobs, where the size of job i is $X_i^j + Y_i^j$, and to distribute them across the g groups using the classical online *list scheduling* algorithm for independent jobs [22, section 5.6]. This formulation (see Proposition 2) allows us to provide an upper bound for the starting time of job N_j, and hence for the length of macro-step j, using a well-known scheduling argument (see Proposition 3). We then derive an upper bound for the expected execution time of ASAP (see Theorem 2).

Algorithm 2: Step j of ASAP $(\omega_1, \ldots, \omega_k)$

1 $i \leftarrow 1$ `/* i represents the number of attempts for the job */`
2 $\mathcal{L} \leftarrow \emptyset$ `/* ℒ represents the list of attempts for the job */`
3 Sample X_i^j and Y_i^j using D_X and $D_{X_D(q)}$ respectively
4 **while** $X_i^j < R(q) + \omega_j + C(q)$ **do**
5 Add Job_i, with processing time $X_i^j + Y_i^j$, to $\mathcal{L}$
6 $i \leftarrow i + 1$
7 Sample X_i^j and Y_i^j using D_X and $D_{X_D(q)}$ respectively
8 $N_j \leftarrow i$
9 Add Job_{N_j}, with processing time $R(q) + \omega_j + C(q)$, to $\mathcal{L}$
`/* the first successful job has size` $R(q) + \omega_j + C(q)$`, not` $X_{N_j}^j + Y_{N_j}^j$ `*/`
10 From time t_{j-1}^{end} on, execute a List Scheduling algorithm to distribute jobs of $\mathcal{L}$ to the different groups (recall that some groups may not be ready at time t_{j-1}^{end})

Proposition 2. *The j-th macro-step of the ASAP protocol can be simulated using Algorithm 2: the last job scheduled by Algorithm 2 ends exactly at time t_j^{end}.*

Proof. The List Scheduling algorithm distributes the next job to the first available group. Because of the memoryless property of Exponential laws, it is

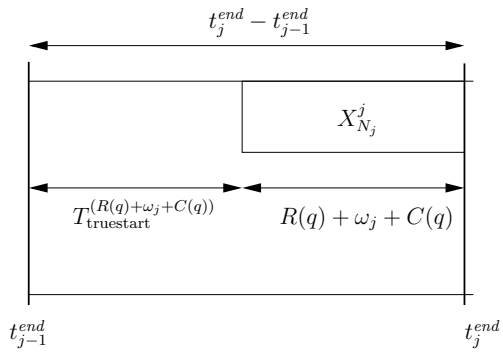


Figure 3: Notations used in Proposition 3.

equivalent (i) to generate the attempts *a priori* and greedily schedule them, or (ii) to generate them independently within each group. □

Proposition 3. *Let $T_{truestart}^{(R(q)+\omega_j+C(q))}$ be the time elapsed between t_{j-1}^{end} and the beginning of Job_{N_j} (see Figure 3). We have $\mathbb{E}\left(T_{truestart}^{(R(q)+\omega_j+C(q))}\right) \leq \mathbb{E}(Y) + \frac{\mathbb{E}(N_j)\mathbb{E}(X)-\mathbb{E}(X_j^{N_j})+(\mathbb{E}(N_j)-1)\mathbb{E}(Y)}{g}$ where X and Y are random variables corresponding to an attempt (sampled using D_X and $D_{X_D(q)}$ respectively). Moreover, we have $\mathbb{E}(N_j) = e^{\lambda q(R(q)+\omega_j+C(q))}$ and $\mathbb{E}(X_j^{N_j}) = \frac{1}{q\lambda} + R(q) + \omega_j + C(q)$.*

Proof. For group x, $1 \leq x \leq g$, let $\tilde{Y_x}$ denote the time elapsed before it is ready for macro-step j. For example in Figure 2, we have $\tilde{Y_1} > 0$ (group 1 is down at time t_{j-1}^{end}), while $\tilde{Y_2} = \tilde{Y_3} = 0$ (groups 2 and 3 are ready to compute at time t_{j-1}^{end}). Proposition 2 has shown that executing macro-step j can be simulated by executing a List Schedule on a job list $\mathcal{L}$ (see Algorithm 2). We now consider g "jobs" $\tilde{Job}_x$, $x = 1, \ldots, g$, so that $\tilde{Job}_x$ has duration $\tilde{Y_x}$. We now consider the augmented job list $\mathcal{L}' = \mathcal{L} \cup \bigcup_{x=1}^{g} \tilde{Job}_x$. Note that $\mathcal{L}'$ may contain more jobs than macro-step j: the jobs that start after the successful job Job_{N_j} are discarded from the list $\mathcal{L}'$. However, both schedules have the same makespan, and jobs common to both systems have the same start and completion dates. Thus, we have $T_{\text{truestart}}^{(R(q)+\omega_j+C(q))} \leq \frac{\sum_{x=1}^{g}(\tilde{Y_x})+\sum_{i=1}^{N_j-1}(X_i^j+Y_i^j)}{g}$: this key inequality is due to the property of list scheduling: the group which is assigned the last job is the least loaded when this assignment is decided, hence its load does not exceed the average load (which is the total load divided by the number of groups). Given that $\mathbb{E}(\tilde{Y_x}) \leq \mathbb{E}(Y)$, we derive

$$\mathbb{E}\left(T_{\text{truestart}}^{(R(q)+\omega_j+C(q))}\right) \leq \mathbb{E}(Y) + \frac{\mathbb{E}\left(\sum_{i=1}^{N_j-1} X_i^j\right) + \mathbb{E}\left(\sum_{i=1}^{N_j-1}(Y_i^j)\right)}{g}$$

But N_j is the stopping criterion of the (X_i^j) sequence, hence using Wald's theorem we have $\mathbb{E}(\sum_{i=1}^{N_j} X_i^j) = \mathbb{E}(N_j)\mathbb{E}(X)$ which leads to $\mathbb{E}(\sum_{i=1}^{N_j-1} X_i^j) = \mathbb{E}(N_j)\mathbb{E}(X) - \mathbb{E}(X_j^{N_j})$. Moreover, as N_j and Y_i^j are independent variables,

we have $\mathbb{E}(\sum_{i=1}^{N_j-1} Y_i^j) = (\mathbb{E}(N_j)-1)\mathbb{E}(Y)$, and we get the desired bound for $\mathbb{E}(T_{\text{truestart}}^{(R(q)+\omega_j+C(q))})$.

Finally, as the expected number of attempts when repeating independently until success an event of probability α is $\frac{1}{\alpha}$ (geometric law), we get $\mathbb{E}(N_j) = e^{\lambda q(R(q)+\omega_j+C(q))}$. The value of $\mathbb{E}(X_j^{N_j})$ can be directly computed from the definition, recalling that $X_j^{N_j} \geq R(q)+\omega_j+C(q)$ and each X_j^i follows an Exponential distribution of parameter $q\lambda$. □

Theorem 2. *The expected execution time of ASAP has the following upper bound:*

$$\frac{g-1}{g}\mathcal{W}(q) + \frac{1}{g}\left(\frac{1}{q\lambda}+\mathbb{E}(Y)\right)e^{\lambda q(R(q)+C(q))}k^* e^{\lambda q\frac{\mathcal{W}(q)}{k^*}} + k^*\left(\frac{g-1}{g}\left(\mathbb{E}(Y)+R(q)+C(q)\right) - \frac{1}{g}\frac{1}{q\lambda}\right)$$

which is obtained when using $k^* = \max(1, \lfloor k_0 \rfloor)$ *or* $k^* = \lceil k_0 \rceil$ *same-size chunks, whichever leads to the smaller value, where:*

$$k_0 = \frac{\lambda q \mathcal{W}(q)}{1+\mathbb{L}\left(\left(g-1+\frac{(g-1)q\lambda(R(q)+C(q))-g}{1+q\lambda\mathbb{E}(Y)}\right)e^{-(1+\lambda q(R(q)+C(q)))}\right)}.$$

Proof. From Proposition 3, the expected execution time of ASAP has upper bound $T_{ASAP} = \sum_{j=1}^{k}\alpha_j$, where

$$\alpha_j = \mathbb{E}(Y) + \frac{\mathbb{E}(N_j)\mathbb{E}(X) - \mathbb{E}(X_j^{N_j}) + (\mathbb{E}(N_j)-1)\mathbb{E}(Y)}{g} + (R(q)+\omega_j+C(q)).$$

Our objective now is to find the inputs to the ASAP algorithm, namely the number k of macro-steps together with the chunk sizes $(\omega_1,\ldots,\omega_k)$, that minimize this T_{ASAP} bound.

We first have to prove that any optimal (in expectation) policy uses only a finite number of chunks. Let α be the expectation of the ASAP makespan using a unique chunk of size $\mathcal{W}(q)$. According to Proposition 3,

$$\alpha = \mathbb{E}(T_{\text{truestart}}^{(R(q)+\mathcal{W}(q)+C(q))}) + C(q) + \mathcal{W}(q) + R(q),$$

and is finite. Thus, if an optimal policy uses k^* chunks, we must have $k^*C(q) \leq \alpha$, and thus k^* is bounded.

In the proof of Theorem 1 in [4], we have shown that any deterministic strategy uses the same sequence of chunk sizes, whatever the failure scenario, thanks to the memoryless property of the Exponential distribution. We cannot prove such a result in the current context. For instance, the number of groups performing a downtime at time t_1^{end} depends on the scenario. There is thus no reason a priori for the size of the second chunk to be independent of the scenario. To overcome this difficulty, we restrict our analysis to strategies that use the same sequence of chunk sizes whatever the failure scenario. We optimize T_{ASAP} in that context, at the possible cost of finding a larger upper bound.

We thus suppose that we have a fixed number of chunks, k, and a sequence of chunk sizes $(\omega_1,\ldots,\omega_k)$, and we look for the values of $(\omega_1,\ldots,\omega_k)$ that minimize

$T_{ASAP} = \sum_{j=1}^{k} \alpha_j$. Let us first compute one of the α_j term. Replacing $\mathbb{E}(N_j)$ and $\mathbb{E}(X_j^{N_j})$ by the values given in Proposition 3, and $\mathbb{E}(X)$ by $\frac{1}{q\lambda}$, we get

$$\alpha_j = \frac{g-1}{g}\omega_j + \frac{1}{g}e^{\lambda q(R(q)+\omega_j+C(q))}\left(\frac{1}{q\lambda} + \mathbb{E}(Y)\right) + \frac{g-1}{g}\left(\mathbb{E}(Y) + R(q) + C(q)\right) - \frac{1}{g}\frac{1}{q\lambda}$$

$$T_{ASAP} = \frac{g-1}{g}\mathcal{W} + \frac{1}{g}\left(\frac{1}{q\lambda} + \mathbb{E}(Y)\right)e^{\lambda q(R(q)+C(q))}\sum_{j=1}^{k} e^{\lambda q\omega_j} + k\left(\frac{g-1}{g}\left(\mathbb{E}(Y) + R(q) + C(q)\right) - \frac{1}{g}\frac{1}{q\lambda}\right)$$

By convexity, the expression $\sum_{j=1}^{k} e^{\lambda q\omega_j}$ is minimal when all ω_j's are equal (to $\mathcal{W}(q)/k$). Hence all the chunks should be equal for T_{ASAP} to be minimal. We obtain:

$$T_{ASAP} = \frac{g-1}{g}\mathcal{W} + \frac{1}{g}\left(\frac{1}{q\lambda} + \mathbb{E}(Y)\right)e^{\lambda q(R(q)+C(q))}ke^{\lambda q\frac{\mathcal{W}(q)}{k}} + k\left(\frac{g-1}{g}\left(\mathbb{E}(Y) + R(q) + C(q)\right) - \frac{1}{g}\frac{1}{q\lambda}\right).$$

Let $f(x) = \tau_1 x e^{\lambda q\frac{\mathcal{W}(q)}{x}} + \tau_2 x$, where

$$\tau_1 = \frac{1}{g}\left(\frac{1}{q\lambda} + \mathbb{E}(Y)\right)e^{\lambda q(R(q)+C(q))} \qquad \text{and}$$

$$\tau_2 = \left(\frac{g-1}{g}\left(\mathbb{E}(Y) + R(q) + C(q)\right) - \frac{1}{g}\frac{1}{q\lambda}\right).$$

A simple analysis using differentiation shows that f has a unique minimum, and solving $f'(x) = 0$ leads to $\tau_1 e^{\lambda q\frac{\mathcal{W}(q)}{k}}\left(1 - \frac{\lambda q\mathcal{W}(q)}{k}\right) + \tau_2 = 0$, and thus to $k = \frac{\lambda q\mathcal{W}(q)}{1+\mathbb{L}\left(\frac{\tau_2}{\tau_1\cdot e}\right)} = k^*$, which concludes the proof. □

Using the upper bound of $\mathbb{E}(Y) = \mathbb{E}(X_D(q))$ in Proposition 1, we can compute numerically the number of chunks and the expectation of the upper bound given by Theorem 2.

5.3 Group replication heuristics for general failure distributions

The results of the previous section are limited to Exponential failures. We now address turn to the general case. In Section 5.3.1 we recall the dynamic program designed in [4] to define checkpointing dates in a context *without* replication. We then discuss how to use this dynamic program in the context of ASAP (Section 5.3.2). Finally, in Section 5.3.3 we propose heuristics that correspond to more general execution protocols than ASAP.

Algorithm 3: DPNextFailure (W,τ_1, ..., τ_p)

```
1 if W = 0 then return 0
2 best ← 0;        chunksize ← 0
3 for ω = quantum to W step quantum do
4     (expected_work, 1st_chunk) ← DP-
      NextFailure(W − ω, τ_1 + ω + C(p), ..., τ_p + ω + C(p))
5     cur_exp_work ←
      P_suc(τ_1 + ω + C(p), ..., τ_p + ω + C(p) | τ_1, ..., τ_p) × (ω + expected_work)
6     if cur_exp_work > best then best ← cur_exp_work;    chunksize ← ω
7 return (best, chunksize)
```

5.3.1 Solution without replication

According to [4], the most efficient algorithm to define checkpointing dates, for *general* failure distributions and when no replication is used, is the dynamic programming approach called DPNextFailure, shown in Algorithm 3. This algorithm works on a failure by failure basis, maximizing the expectation of the amount of work completed (and checkpointed) before the next failure occurs. This algorithm only provides an approximation of the optimal solution as it relies on a time discretization. (For the sake of simplicity, we present all dynamic programs as recursive algorithms.)

5.3.2 Implementing the ASAP execution protocol

A key question when implementing ASAP is that of the chunk size. A naive approach would be to use DPNextFailure. Each group would call DPNextFailure to compute what would be the optimal chunk size for itself, as if there were no other groups. Then we have to *merge* these individual chunk sizes to obtain a common chunk size. This heuristic, DPNextFailureAsap, is shown in Algorithm 4 (the Alive function returns, for a list of q processors, the amount of time each has been up and running since its last downtime). For the Merge operator, one could be pessimistic and take the minimum of the chunk sizes, or be optimistic and take the maximum, or attempt a trade-off by taking the average. Two important limitations of this heuristic are: 1) the Merge operator that has no theoretical justification; and 2) the use of chunk sizes defined using an obsolete failure history. The latter limitation shows after a group is victim of a failure: the failure history has changed significantly but the chunk size is not recomputed (this has no consequences with an Exponential distribution as the Exponential is memoryless).

5.3.3 Other execution protocols

In order to circumvent both limitations of DPNextFailureAsap, we relax the constraint that all groups work with the same chunk size. Each group now works with its own chunk size that is recomputed each time the group fails, and each time one of the groups successfully complete its own chunk. This leads to the heuristic DPNextFailureSynchro in Algorithm 5. Each time a group successfully works for the duration of its chunk size and checkpoints its work, it signals its success to all groups which then interrupt their own work. This

Algorithm 4: DPNextFailureAsap(W).

```
while W ≠ 0 do
    for each group x = 1..g do in parallel
        (τ_{(x−1)q+1}, ..., τ_{(x−1)q+q}) ← Alive((x − 1)q + 1, ..., (x − 1)q + g)
        (exp_work_x, ω_x) ← DPNextFailure(W, τ_{(x−1)q+1}, ..., τ_{(x−1)q+q})
    ω ← Merge(ω_1, ..., ω_g)
    for each group do in parallel
        repeat
            Try to execute a chunk of size ω and then checkpoint
            if successful then
                Signal other groups to immediately stop their attempts
            else if failure then Complete downtime and perform recovery
        until One of the groups signals its success
    W ← W − ω
    for each group do in parallel
        if not successful on last chunk then  Perform recovery from last
        successfully completed checkpoint
```

behavior is clearly sub-optimal if the successful completion is for a very small chunk and an interrupted group that was computing a large chunk was close to completion. The problems are that: 1) chunk sizes are defined as if each group was alone; and 2) the definition of the chunk sizes does not account for the fact that the first group to complete its checkpoint defines a mandatory checkpointing date for all groups.

Algorithm 5: DPNextFailureSynchro(W).

```
for each group x = 1..g do in parallel
    while W ≠ 0 do
        (τ_{(x−1)q+1}, ..., τ_{(x−1)q+q}) ← Alive((x − 1)q + 1, ..., (x − 1)q + g)
        (exp_work_x, ω_x) ← DPNextFailure(W, τ_{(x−1)q+1}, ..., τ_{(x−1)q+q})
        Try to execute a chunk of size ω_x and then checkpoint
        if successful then
            Signal other groups to immediately stop their attempts
            W ← W − ω_x
        if failure then Complete downtime
        if failure or signal then
            Perform recovery from last successfully completed checkpoint
```

To address these problems, rather than doing another attempt at reusing DPNextFailure, we design a brand new dynamic program. In [4], without replication, we found that a dynamic program that minimizes the expectation of the makespan would require an intractable exponential number of states to record which processors fail and when. Hence we aimed instead at maximizing the expectation of the work completed before the next failure. We now extend this approach to the context of replication. The first failure will only interrupt a single group. Therefore, the objective should be to maximize the expectation of the work completed before all groups have failed. This approach ignores that once a group has failed, it will eventually restart and resume computing. However, keeping track of such restarts would require recording which processors

have failed and when, thus once again leading to an exponential number of states.

To avoid having the first completed checkpoint force a checkpoint for all other groups we design a new dynamic program, DPNextCheckpoint (Algorithm 6). DPNextCheckpoint does not define chunk sizes, i.e., amount of work to be processed before a checkpoint is taken, but instead it defines checkpoint dates. The rationale is that one checkpointing date can correspond to different amounts of work for each group, depending on when the group has started to process its chunk, after either its last failure and recovery, or its last checkpoint, or its last recovery based on another group's checkpoint. The function WorkAlreadyDone (Line 3) returns, for each group, the time since it started processing its current chunk.

DPNextCheckpoint proceeds as follows. At the checkpointing date, the amount of work completed is the maximum of the amount of work done by the different groups that successfully complete the checkpoint. Therefore, we consider all the different cases (Line 8), that is, which group x, among the successful groups, has done the most work. We compute the probability of each case (Line 11). All groups that started to work earlier than group x have failed (i.e., at least one processor in each of them has failed) but not group x (i.e., none of its processors have failed). We compute the expectation of the amount of work completed in each case (Lines 12 and 13). We then sum the contributions of all the cases (Line 14) and record the checkpointing date leading to the largest expectation (Line 15). Note that the probability computed at Line 11 explicitly states which groups have successfully completed the checkpoint and which groups have not. We choose not to take this information into account when computing the expectation (recursive call at Line 13). This is to avoid keeping track of which group had failed, thereby lowering the complexity of the dynamic program. This explains why the conditions do not evolve in the conditional probability at Line 11.

Finally, Algorithm 7 shows the algorithm, called DPNextFailure, that uses DPNextCheckpoint. Each time a group is affected by an event (a failure, a successful checkpoint by itself or by another group), it computes the next checkpoint date and signals the result of its computation to the other groups (e.g., by broadcasting it to the g group leaders). Hence, a group may have computed the next checkpoint date to be t, and that date can be either unmodified, or postponed, or advanced by events occurring on other groups and by their re-computation of the best next checkpoint date. In practice, as these algorithms rely on a time discretization, at each time quantum a group can check whether the current time is a checkpoint date or not.

6 First set of experimental evaluations

6.1 Simulation framework

In this section we detail our simulation methodology. We use both synthetic and real-world failure distributions. The source code and all simulation scenarios are publicly available at: `http://perso.ens-lyon.fr/frederic.vivien/Data/Resilience/HIPC2012/`.

Algorithm 6: DPNextCheckpoint(W, T, T_0, τ_1, ..., τ_{gq})

```
1  if W = 0 then return 0
2  best_work ← 0;      next_chkpt ← date
3  (W_1, ..., W_g) ← WorkAlreadyDone(T)    /* Time since last recovery or
   checkpoint */
4  Reorder groups in non-increasing availabilities (W_1 is maximum)
5  for t = T to T + W − W_g step quantum   /* Loop on checkpointing date */
6  do
7      cur_work ← 0
8      for x = 1 to g      /* Loop on the first group to successfully work
       until t + C(q) */
9      do
10         δ ← (t + C(q)) − T_0    /* Total time elapsed until the checkpoint
           completion */
11         proba ← ∏_{y=1}^{x−1} P_fail(τ_{(y−1)q+1} + δ, ..., τ_{(y−1)q+p} + δ | τ_{(y−1)q+1}, ..., τ_{(y−1)q+p})
                   × P_suc(τ_{(x−1)q+1} + δ, ..., τ_{(x−1)q+p} + δ | τ_{(x−1)q+1}, ..., τ_{(x−1)q+p})
12         ω ← min{W − W_x, t − T}   /* Work done between T and t by group
           x */
13         (rec_ω, rec_t) ← DP-
           NextCheckpoint(W − W_x − ω, T + ω + C(q) + R(q), T_0, τ_1, ..., τ_p)
14         cur_work ← cur_work + proba × (W_x + ω + rec_ω)
15     if cur_work > best_work then
       best_work ← cur_work;   next_chkpt ← t
16 return (best_work, next_chkpt)
```

Algorithm 7: DPNextFailure(W).

```
1  for each group x = 1..g do in parallel
2      while W ≠ 0 do
3          (τ_1, ..., τ_gq) ← Alive(1, ..., gq)
4          T_0 ← Time()                                   /* Current time */
5          date ← DPNextCheckpoint(W, T_0, T_0, τ_1, ..., τ_gq)
6          Signal all processors that the next checkpoint date is now date
7          Try to work until date and then checkpoint
8          if successful work until date and checkpoint then
9              Let y be the longest running group without failure among the
               successful groups
10             Let ω be the work performed by y since its last recovery or
               checkpoint
11             W ← W − ω
12             if group x last recovery or checkpoint was strictly later than that of
               y then
13                 Perform a recovery
14         if failure then Complete downtime
15         if failure or signal then  Perform recovery from last successfully
           completed checkpoint
```

6.1.1 Heuristics

Our simulator implements the following eight checkpointing policies:

- Two versions of the ASAP protocol: OptExp, that uses the optimal and periodic policy established in [4] for Exponential failure distributions and no replication; OptExpGroup, that uses the periodic policy defined by Theorem 2 for Exponential distributions.
- The six dynamic programming approaches in Section 5.3: DPNextFailure, the 3 variants of DPNextFailureAsap, DPNextFailureSynchro, and DPNextFailure.

Our simulator also implements BestPeriod, which is a numerical search for the optimal period for ASAP. We evaluate each candidate period on 50 randomly generated scenarios. To build the candidate periods, the period computed for OptExp is multiplied and divided by $1 + 0.05 \times i$ with $i \in \{1, ..., 180\}$, and by 1.1^j with $j \in \{1, ..., 60\}$. BestPeriod corresponds to the periodic policy that uses the best period found by the search. The evaluation of BestPeriod on a configuration requires running 24,000 simulations (which would be prohibitive in practice), but we include it for reference. Based on the results in [4], we do not consider any additional checkpointing policy, such as those defined by Young [29] or Daly [9] for instance. We point out that OptExp and OptExpGroup compute the checkpointing period based solely on the MTBF, implicitly assuming 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 are not exponentially distributed. To use OptExp with g groups we use the period from [4] computed with $\lfloor p/g \rfloor$ processors.

6.1.2 Platforms

We target two types of platforms, depending on the type of the failure distribution. For synthetic distributions, we consider platforms containing from 32,768 to 1,048,576 processors. For platforms with failures based on failure logs from production clusters, because of the limited scale of those clusters, we restrict the size of the platforms to a maximum of 131,072 processors, starting with 4,096 processors. For both platform types, we determine the job size $\mathcal{W}$ so that a job using the whole platform would use it for a significant amount of time in the absence of failures, namely ≈ 3.5 days on the largest platforms for synthetic failures ($\mathcal{W} = 10,000$ years), and ≈ 2.8 days on those for log-based failures ($\mathcal{W} = 1,000$ years). Otherwise, we use the same parameters as in [4]: $C = R = 600\ s$, $D = 60\ s$, $\gamma = 10^{-6}$ for generic parallel jobs, and $\gamma = 0.1$ for numerical kernels. Note that the checkpointing overheads come from the scenarios in [7] and are smaller than those used in [12].

6.1.3 Failure scenarios

Synthetic failure distributions – To choose failure distribution parameters that are representative of realistic systems, we use failure statistics from the Jaguar platform. Jaguar contains $45,208$ processors and is said to experience on the order of 1 failure per day [21, 2]. Assuming a 1-day platform MTBF gives us a processor MTBF equal to $\frac{45,208}{365} \approx 125$ years. For the Exponential

distribution, we then have λ as $\lambda = \frac{1}{MTBF}$ and for Weibull, which requires two parameters k and λ, we have $\lambda = MTBF/\Gamma(1 + 1/k)$ and we fix k to 0.7 based on the results of [25]. (We have shown in [4] that the general trends were not influenced by the exact values used for k nor for the MTBF.)

Log-based failure distributions – We also consider failure distributions based on failure logs from production clusters. We used logs from the largest clusters among the preprocessed logs in the *Failure trace archive* [19], i.e., from clusters at the Los Alamos National Laboratory [25]. In these logs, each failure is tagged by the node —and not just the processor— on which the failure occurred. Among the 26 possible clusters, we opted for the only two clusters with more than 1,000 nodes, as we needed a sample history sufficiently large to simulate platforms with more than 10,000 nodes. The two chosen logs are for clusters 18 and 19 in the archive (referred to as 7 and 8 in [25]). For each log, we record the set $\mathcal{S}$ of availability intervals. A discrete failure distributionfor the simulation is then generated as follows: the conditional probability $\mathbb{P}(X \geq t \mid X \geq \tau)$ that a node stays up for a duration t, knowing that it has been up for a duration τ, is set to the ratio of the number of availability durations in $\mathcal{S}$ greater than or equal to t, over the number of availability durations in $\mathcal{S}$ greater than or equal to τ.

Scenario generation – Given a p-processor job, a failure trace is a set of failure dates for each processor over a fixed time horizon h (set to 2 years). The job start time is assumed to be 1 year for synthetic distribution platforms, and 0.25 year for log-based distribution platforms. We use a non-null start time to avoid side-effects related to the synchronous initialization of all processors. Given the distribution of inter-arrival times at a processor, for each processor we generate a trace via independent sampling until the target time horizon is reached.For simulations where the only varying parameter is the number of processors $a \leq p \leq b$, we first generate traces for b processors. For experiments with p processors we then simply select the first p traces. This ensures that simulation results are coherent when varying p. Finally, the two clusters used for computing our log-based failure distributions consist of 4-processor nodes. Hence, to simulate a 131,072-processor platform we generate 32,768 failure traces, one for each four-processor node.

6.2 Simulation results

In this section we discuss simulation results, but only show graphs for perfectly parallel applications under the constant overhead scenario. The reason is that all trends and conclusions are similar regardless of the application and overhead models. The full results are provided in Appendix A. All results are averages over at least 50 instances.

6.2.1 Exponential failures

Figure 4 plots average makespan vs. number of processors in the case of exponential failures for our various algorithms. A first observation is that OptExp is close to BestPeriod (the two curves are indistinguishable), and better than all replication algorithms up to 2^{20} processors. In other terms, assuming exponential failures, group replication is not worthwhile until the platform becomes

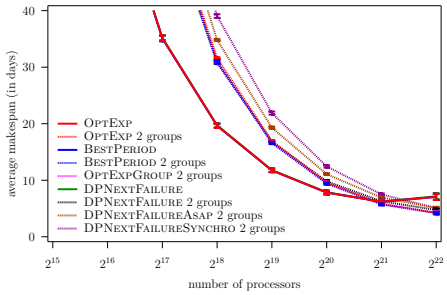


Figure 4: Exponential failures with MTBF=125 years.

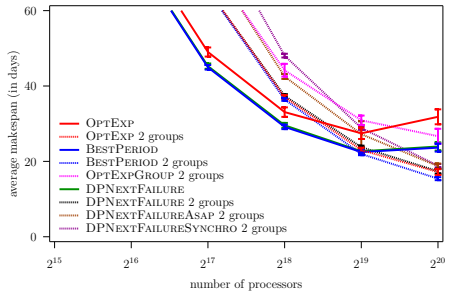


Figure 5: Weibull failures with MTBF=125 years and $k = 0.70$.

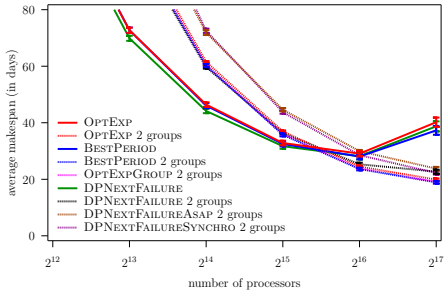


Figure 6: Failures based on the failure log of LANL cluster 18.

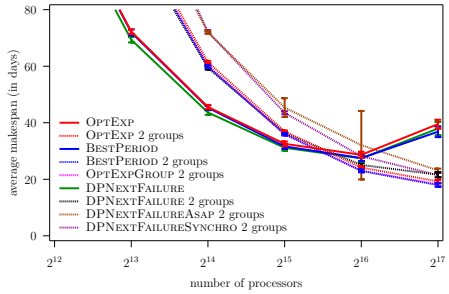


Figure 7: Failures based on the failure log of LANL cluster 19.

very large (i.e., two millions of processors or more). Considering those algorithms that use replication, we find that OptExpGroup with $g = 2$ delivers performance very close to that of the numerical lower bound on any periodic ASAP policy (BestPeriod with $g = 2$) and slightly better than the performance of OptExp with $g = 2$. The implication is that determining chunk sizes according to Theorem 2 is more efficient than using two application instances that each use the chunk size determined by OptExp. In other terms, it is better to compute the chunk sizes by explicitly considering that multiple groups are used. Turning to the dynamic programming solutions, we find that DPNextFailureSynchro leads to the worse results, outperformed by DPNextFailureAsap. The three variants of this algorithm (with Merge being the minimum, the average, or the maximum) are indistinguishable which is why Figure 4 shows only one curve. By contrast, DPNextFailure is the best among the dynamic programming algorithms, leading to equivalent or better results than its competitors. Furthermore, its performance is close to that of BestPeriod with $g = 2$, meaning that it is close to the optimal periodic policy. Perhaps the most striking result is that DPNextFailure is (slightly) outperformed by the periodic policy OptExp with $g = 2$. This result is somewhat expected because this policy is specifically designed for exponential failures. However, it computes the checkpointing period ignoring that replication is used! By contrast, DPNextFailure is based on strong theoretical foundations and was designed specifically to handle multiple groups efficiently. Yet, with

exponential failures, it does not outperform OptExp with $g = 2$. We conclude that our dynamic programming approach is not able to do better than a naïve approach in the presence of exponentially distributed failures. Although the results in this section are interesting from a theoretical point of view, recall that failures in real-world systems are known to not be exponentially distributed.

6.2.2 Log-based failures

For log-based failures with the constant overhead scenario, using all the available processors to run a single application instance leads to significantly larger makespans. This is seen in Figures 7, the no-replication strategies, shoot upward when p reaches a large enough value. For instance, with traces based on the logs of LANL cluster 18, the increase in makespan is more than 37% when going from $p = 2^{16}$ to $p = 2^{17}$.

6.2.3 Weibull failures

Figure 5 is similar to Figure 4 but shows results for Weibull failures with $k = 0.7$. This figure does not include results for DPNextFailureSynchro and DPNextFailureAsap, as these algorithms, as in the case of exponential failures, lead to results strictly worse than that of DPNextFailure. We see that using replication pays off at lower scale (2^{19} processors). Another difference, is that OptExpGroup with $g = 2$ is no longer close to the numerical lower bound BestPeriod with $g = 2$. This is expected because failures are no longer exponentially distributed. Other observations are more or less unchanged, including the fact that DPNextFailure with $g = 2$ still does not outperform OptExp with $g = 2$. While this result was perhaps not a concern in the previous section since failures were exponentially distributed, it is here a very surprising result. One might expect that OptExp, which assumes exponential failures, would lead to poor results when confronted with Weibull failures, or at least poorer results than an approach designed specifically to handle replication with general failures.

It turns out that the Weibull distribution with $k = 0.7$ is too close to an exponential distribution to be detrimental to the performance of OptExp. Smaller values of k are reasonable, as seen for instance in [20] ($k \approx 0.5$) and in [25] ($0.33 \leq k \leq 0.49$). Figure 8 shows average makespan results for k values between 0.4 and 0.95, for OptExp, BestPeriod, and DPNextFailure all with $g = 2$. We observe that for lower $k \leq 0.5$, OptExp is more than a factor 2 worse than BestPeriod, while DPNextFailure is within less than 10% of it. Confirming the observation made on Figure 5, once k reaches 0.7 then OptExp is closer to BestPeriod than DPNextFailure. To further illustrate the impact of the value of k, Figure 9 shows results like those in Figure 5 but for $k = 0.5$. Results for $g = 3$ groups are also included in this figure since, unlike for exponential failures and for Weibull failures with $k = 0.7$, using more than 2 groups can prove beneficial with low k. The main observation is that, regardless of the number of groups used, OptExp is significantly outperformed by DPNextFailure, which remains close to BestPeriod. We conclude that overall, and unlike OptExp, our dynamic programming algorithm provides performance close to the numerical lower bound for the whole range of k, the parameter defining the possible shape of Weibull distributions.

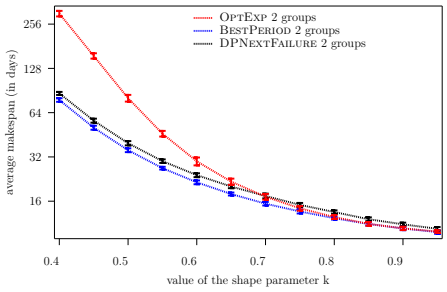


Figure 8: Weibull failures, MTBF=125y, 2^{20} procs.

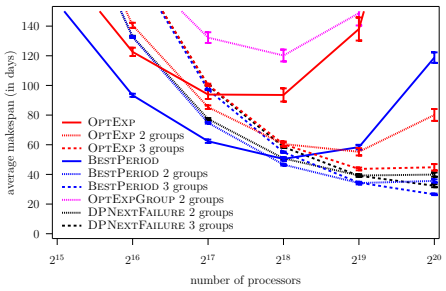


Figure 9: Weibull failures, MTBF=125y, $k = 0.50$.

7 Second set of experimental evaluations

7.1 Simulation methodology

In this section we detail our simulation methodology. Source codes and simulation scenarios are publicly available at `http://perso.ens-lyon.fr/frederic.vivien/Data/Resilience/Replication`.

7.1.1 Evaluated algorithms

Our simulator implements two versions of the ASAP protocol in the case of exponentially distributed failures. The first version, OptExp, simply uses for each group the optimal and periodic policy established in [4] for Exponential failure distributions and no replication. To use OptExp with g groups we use the period from [4] computed with $\lfloor p/g \rfloor$ processors. The second, OptExpGroup, uses the periodic policy defined by Theorem 2. Both OptExp and OptExpGroup compute the checkpointing period based solely on the MTBF, assuming that failures are exponentially distributed. We nevertheless include them in all our experiments, simply using the MTBF value even when failures are not exponentially distributed. The simulator also implements BestPeriod (Section 6.1.1) and DPNextFailure (Section 5.3.3). Note that the execution times reported when using DPNextFailure include the time needed to run Algorithms 6 and 7. Based on the results in [4], we do not consider any additional checkpointing policy, such as those defined by Young [29] or Daly [9] for instance.

7.1.2 Platform and job parameters

We consider platforms containing from 32,768 to 4,194,304 processors. We determine the job size $\mathcal{W}$ so that a job using the whole platform would use it for a significant amount of time in the absence of failures, namely ≈ 21 hours on the largest platforms ($\mathcal{W} = 10,000$ years). In all experiments we use $D = 60$ s, and $C = R = 60$ s, 600 s, and 6000 s, thus spanning the spectrum from relatively fast to relatively slow checkpointing/recovery. We also ran experiments with a very short $C = R = 6$ s, but the results are virtually identical to those obtained with $C = R = 60$ s and we do not present them. Finally, we use $\gamma = 10^{-6}$ for generic parallel jobs, and $\gamma = 0.1$ for numerical kernels (see Section 3). Here,

we only present and discuss the constant overhead scenario ($C(q) = R(q) = C$). Results from the proportional overhead scenario are consistent with those for the constant overhead scenario and can be found in the companion research report [5].

7.1.3 Failure distributions

To choose failure distribution parameters that are representative of realistic systems, we use failure statistics from the Jaguar platform. Jaguar contained $45,208$ processors and is said to have experienced on the order of 1 failure per day [30]. Assuming a 1-day platform MTBF leads to a processor MTBF equal to $\frac{45,208}{365} \approx 125$ years. We generate both Exponential and Weibull failures, the former serving as a best case yet unrealistic scenario and the latter being representative of failure behavior in production systems [14, 25, 20, 15]. For the Exponential distribution of failure inter-arrival times, we simply set $\lambda = \frac{1}{MTBF}$. For the Weibull distribution, which requires two parameters, a shape parameter k and a scale parameter λ, and has density $\frac{k}{\lambda}(\frac{x}{\lambda})^{k-1}e^{-(x/\lambda)^k}$ for $x \geq 0$, we have $\lambda = MTBF/\Gamma(1 + 1/k)$. Based on the results in [14, 25, 20, 15] we use for the value of k either 0.5 or 0.7. For small values of the shape parameter k, the Weibull distribution is far from an Exponential distribution, meaning that it is far from being memoryless. We resort to generating synthetic failure traces because it is unclear how to extrapolate production failure logs for current platforms, e.g., as available in [19], to post-petascale platforms in a reasonable manner. One option is to use available smaller failure logs and use oversampling to simulate failures on larger platforms. Unfortunately, such oversampling introduces biases, and the validity of the obtained results would be questionable.

7.1.4 Failure scenario generation

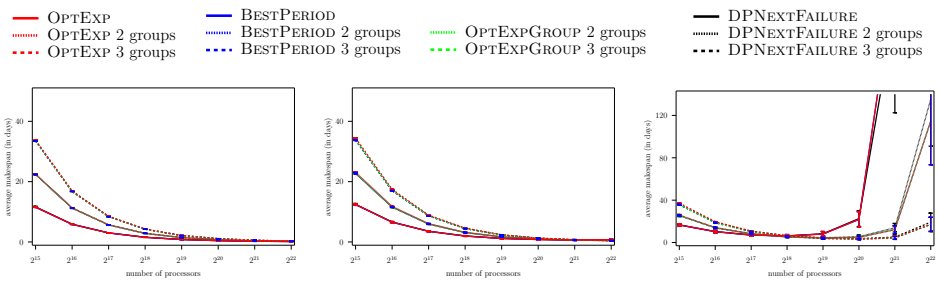
Given a p-processor job, a failure trace is a set of failure dates for each processor over a fixed time horizon, which we set to 2 years in our simulations. The job start time is assumed to be at 1 year. We use a non-zero start time to avoid side-effects related to the synchronous initialization of all processors. Given the distribution of inter-arrival times at a processor, for each processor we generate a trace via independent sampling until the target time horizon is reached.

7.2 Simulation results

In this section, we only present simulation results for perfectly parallel applications under the constant overhead model. All trends and conclusions are similar regardless of the application and overhead models. For completeness, we provide the full results in Appendix B. All results are averages over at least 50 instances, and all graphs show one-standard-deviation error bars.

7.2.1 Exponential failures

Figure 10 shows average makespan vs. the number of processors for our algorithms, each used assuming $g = 1$, 2, or 3 groups, assuming Exponential failures. A first observation is that many curves overlap each other: for a given g all algorithms lead to similar average makespan. For instance, for $C = R = 600$ s and



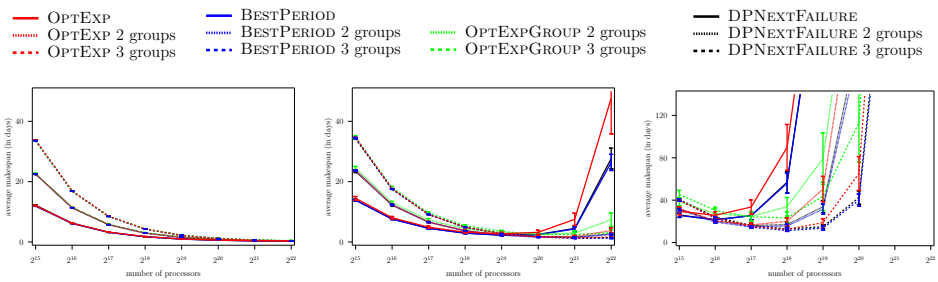
(a) $C = R = 60$ s (b) $C = R = 600$ s (c) $C = R = 6000$ s

Figure 10: Average makespan vs. number of processors, Exponential failures, $MTBF = 125$ years.

$g = 2$, and taking OptExp as a reference, the relative difference between the average makespan of OptExp and that of the other three algorithms is at most 6.81% (and only 2.31% when averaged over all considered numbers of processors). In spite of such small differences, several trends emerge. OptExp almost always leads to higher average makespan than OptExpGroup (note that for $g = 1$ the two algorithms are equivalent). Over the 8 numbers of processors considered, the 3 values for $R = C$, and the 3 values for g, i.e., 72 scenarios, OptExp leads to average makespans shorter than that of OptExpGroup only 4 times (for $R = C = 6000$ s, for 2^{18} to 2^{21} processors, and by at most 3.27%). BestPeriod never leads to an average makespan higher than that of OptExp or OptExpGroup, and outperforms them by up to several percents across all the $R = C$ and g values. DPNextFailure leads to mixed results, with equal or shorter average makespan than OptExpGroup, resp. BestPeriod, for 31, resp. 24, of the 72 different scenarios.

A second observation is that the use of $g > 1$ (i.e., multiple groups) often does not help and can even lead to larger average makespans. For $R = C = 60$ s, increasing g from 1 to 2, or from 2 to 3, never leads to a lower average makespan for any of our algorithms. For $R = C = 600$ s, the only improvements are seen when going from 1 to 2 groups, for the OptExp, OptExpGroup, and BestPeriod algorithms, and only with more than 2^{21} processors. The relative improvements are at most 7.75% for 2^{21} processors, and between 25.40% and 41.09% for 2^{22} processors. No improvements are achieved when going from 2 to 3 groups. More improvements are seen for $C = R = 6000$ s. When going from 1 to 2 groups, improvements are achieved starting at 2^{18} processors, with improvements up to between 93.64% and 95.17% at large scale, for all four algorithms. When going from 2 to 3 groups, relative improvements are seen starting at 2^{19} processors, reaching up to between 85.09% and 85.78% for all four algorithms.

For low and moderate checkpointing overheads, $C = R = 60$ s or 600 s, the average makespan decreases as the number of processors increases. Instead, for high checkpointing overheads, $C = R = 6000$ s, the average makespan initially decreases but starts increasing at large scale. This is particularly noticeable when using $g = 1$ group. For instance, the average makespan using OptExp goes from 21.83 s with 2^{20} processors to 249.39 s with 2^{21} processors, or an increase by a factor 11.42. The increase is similar with BestPeriod and marginally lower with DPNextFailure (a factor 9.72). The reason for



(a) $C = R = 60$ s (b) $C = R = 600$ s (c) $C = R = 6000$ s

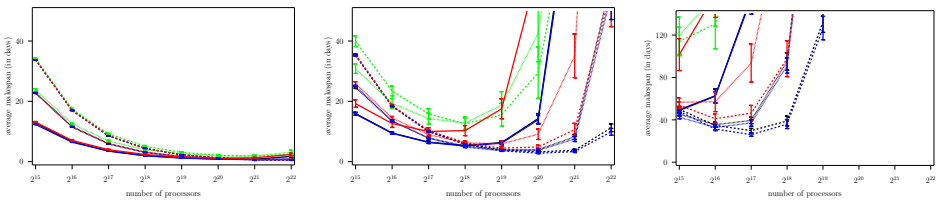
Figure 11: Average makespan vs. number of processors, Weibull failures, $k = 0.7$, $MTBF = 125$ years.

this makespan increase is simply that with a high checkpointing overhead, the parallel efficiency is low as processors spend more time in checkpointing activities than in actual computation. This observation is precisely the motivation for using $g > 1$ (see Section 1). With $g = 2$, we still see increases in average makespans, but only by a factor between 2.46 and 2.53 when going from 2^{20} processors to 2^{21} processors for all algorithms. With $g = 3$, this factor is between 1.34 and 1.39 for all algorithms. Therefore, the use of group replication improves parallel efficiency and can lead to scalability improvements. For instance, with $g = 1$ or $g = 2$, regardless of the algorithm in use, it is not advisable to use 2^{20} processors as the makespan is lower when using 2^{19} processors. With $g = 3$, instead, there is a reduction in average makespan when going from 2^{19} processors to 2^{20} processors for all our algorithms (the relative percentage reductions are between 14.58% and 18.81%).

Based on the above, we conclude that for Exponential failures group replication can be useful when the checkpointing overhead is relatively large and/or when the scale of the execution is large. While large checkpointing overheads decrease parallel efficiency, the use of group replication makes it possible to limit this decrease or even to increase parallel efficiency at some scales. All our algorithms lead to comparable performance, with BestPeriod leading to good results even though marginally outperformed by DPNextFailure in some instances. While these results are interesting, and although Exponential failures have been studied in all previously published works, their relevance to practice is not clear given that real-world failures follow non-memoryless distributions. In the next section we present results for Weibull failures, which are more representative of real-world failure scenarios.

7.2.2 Weibull failures

Figures 11 and 12 show results for Weibul failures with $k = 0.7$ and $k = 0.5$, respectively. For low $R = C = 60$ s and for $k = 0.7$ (Figure 11(a)), results are similar to those seen in the previous section for Exponential failures: the use of multiple groups does not help, and all algorithms lead to sensibly the same performance. The gaps between the algorithms become larger for $k = 0.5$, i.e., when the failure distribution is farther from the Exponential distribution, with the advantage to BestPeriod (Figure 12(a)). For instance, for $k = 0.5$, 2^{20} processors, and using $g = 2$ groups, BestPeriod leads to an average



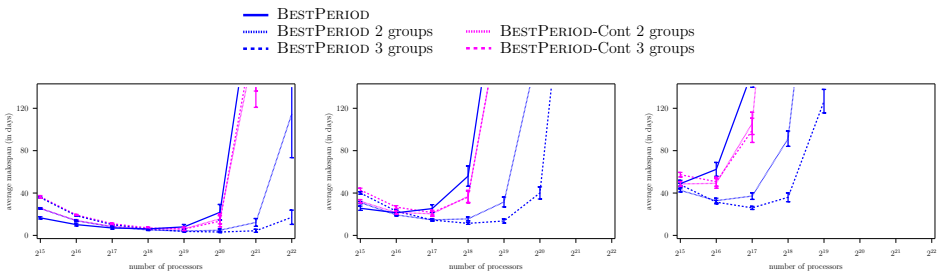
(a) $C = R = 60$ s (b) $C = R = 600$ s (c) $C = R = 6000$ s

Figure 12: Average makespan vs. number of processors, Weibull failures, $k = 0.5$, $MTBF = 125$ years.

makespan lower than that of OptExp, OptExpGroup, and DPNextFailure by 10.46%, 51.04%, and 2.08%, respectively. A general observation in all the results for replication ($g > 1$) with Weibull failures, regardless of the value of $C = R$, is that OptExpGroup leads to much poorer results than all the other algorithms. This is because the analytical development of Theorem 2 relies heavily on the Exponential failure assumption. As a result, OptExpGroup is even outperformed by OptExp, even though this algorithm also assumes Exponential failures. In all that follows we no longer discuss the results for OptExpGroup.

For $C = R = 600$ s and $k = 0.7$, and unlike the results for Exponential failures, at large scale the average makespan of the $g = 1$ executions increases sharply while the average makespans for $g > 1$ executions remain more stable (Figure 11(b)). In other words, even when checkpointing overheads are moderate, group replication is useful for increasing parallel efficiency once the scale is large enough. This result is amplified when failures are further from being Exponential, i.e., for $k = 0.5$ (Figure 12(b)). For $k = 0.5$, going from $g = 1$ to $g = 2$ groups is beneficial for OptExp starting at 2^{17} processors and for BestPeriod and DPNextFailure starting at 2^{18} processors. Going from $g = 2$ to $g = 3$ groups is beneficial for OptExp andBestPeriod starting at 2^{19} processors, and for DPNextFailure starting at 2^{20} processors. In terms of comparing the algorithms with each other, in Figure 12(b) all algorithms experience a makespan increase after the initial decrease. Only BestPeriod and DPNextFailure, when using $g = 3$ groups, have a decreasing makespan up to 2^{20} processors. When going to 2^{21} processors, these algorithms lead to relative increases in makespan of 18.50% and 14.99%, and larger increases when going from 2^{21} to 2^{22} processors. Across the board, BestPeriod with $g = 3$ groups leads to the lowest average makespan, with DPNextFailure with $g = 3$ groups a close second. The average makespan of DPNextFailure is at most 15.66% larger than that of BestPeriod, and in fact is shorter at low scales (for 2^{15} and 2^{16} processors).

Results for $C = R = 6000$ s show similar but accentuated trends. For $k = 0.7$ (Figure 11(c)) the main results are similar to those obtained for $k = 0.5$ with $C = R = 600$ s. The best two algorithms are BestPeriod and DPNextFailure using $g = 3$ groups, but both algorithms show an increase in makespan starting at 2^{19} processors. For $k = 0.5$ (Figure 12(c)) this increase occurs at 2^{18} processors and is sharper for DPNextFailure than BestPeriod. Even though group replication helps, with such large checkpointing overheads parallel efficiency cannot be maintained beyond 2^{17} processors.



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

Figure 13: Average makespan vs. number of processors, $C = R = 6000$ s, $MTBF = 125$ years.

We conclude that although with Exponential failures all our algorithms are more or less equivalent (see Section 7.2.1), with more realistic Weibul failures BestPeriod emerges as the best algorithm. The only algorithm that leads to makespans comparable to those of BestPeriod is DPNextFailure, but it never leads to a lower average makespan than BestPeriod at large scale. Even though DPNextFailure relies on a sophisticated DP approach, the brute-force but pragmatic approach used by BestPeriod turns out to be more effective. Even when using BestPeriod, our results show that application scalability is hindered by higher checkpoint overheads, which is expected, but also by lower k values, i.e., by less exponentially distributed failures.

7.2.3 Checkpointing contention

The results presented so far are obtained assuming that the checkpointing overhead ($R = C$) does not depend on the number of groups. There are cases in which this assumption could give an unfair advantage to group replication. Consider an application with a given memory footprint V, in bytes, running on a platform with a total of q processors. With no replication ($g = 1$) the total volume of data involved in a checkpoint is V. Assuming that V is no larger than the aggregate RAM capacity of q/g processors, then group replication can be used with $g > 1$ groups. In this case, since each group executes the application, the total volume of data involved in a checkpoint at each group is also V. Since groups may checkpoint/recover at the same time, the amount of data involved can be up to $g \times V$, or a factor g larger than in the no-replication case.

To evaluate the impact of group replication on checkpointing overhead, we introduce a checkpointing contention model in our simulation. Whenever multiple checkpointing/recovery operations are concurrent, they receive a fair share of the checkpointing/recovery bandwidth. For instance, if n checkpointing operations begin at the same time, and no other checkpointing or recovery occurs over the next $n \times C$ time units, then all n checkpointing operations finish after $n \times C$ time units. More generally, considering that a checkpointing/recovery operations requires C units of activity, over a time interval Δt during which there are n ongoing such operations each operation performs $\frac{1}{n}/\Delta t$ units of activity (if one of these operations requires fewer units of work to complete, consider a shorter Δt interval).

Our objective in this section is to determine whether group replication can

still be beneficial when considering checkpointing contention. We repeated all the experiments presented in Sections 7.2.1 and 7.2.2. For $C = R = 60$ s, checkpointing contention has negligible impact on the results, and the impact for $C = R = 600$ s is lower than that for $C = R = 6000$ s. This is expected since the larger the checkpointing/recovery overhead, the more likely that more than one group is engaged in checkpointing or recovery at the same time. Thus, among all our results, those for $C = R = 6000$ s should be the most disadvantageous for group replication. These are the results presented in Figure 13, which shows average makespan vs. number of processors for BestPeriod without and with contention (denoted by BestPeriod-Cont), for $g = 1, 2$, and 3, for $C = R = 6000$ s, for Exponential failures and for Weibull failures with $k = 0.7$ and $k = 0.5$.

As expected the average makespan of BestPeriod is increased due to checkpointing contention when multiple groups are used. However, even with contention, group replication outperforms the no-replication case at large scale. For Exponential failures, using $g = 2$ groups outperforms using $g = 1$ group as soon as the number of processors reaches 2^{18}, both with and without contention. Using $g = 3$ groups outperforms using $g = 2$ groups when there are either 2^{19} or 2^{20} processors with contention. The lowest average makespans with contention are achieved using either 2^{18} processors split in $g = 2$ groups, or 2^{19} processors split in $g = 3$ groups. For Weibull failures with $k = 0.7$, using $g = 2$ groups outperforms using $g = 1$ group starting at 2^{16} processors, with or without checkpointing contention. With contention, using $g = 3$ groups never outperforms using $g = 2$ groups, and ties its performance starting at 2^{18} processors. For Weibull failures with $k = 0.5$, using $g = 2$ groups outperforms using $g = 1$ group starting at 2^{15} processors with or without contention. With contention, using $g = 3$ groups is beneficial over using $g = 2$ groups when there are 2^{17} processors but the lowest makespan overall is achieved with $g = 2$ groups and 2^{15} processors.

We conclude that although checkpointing contention increases the makespan of group replication executions, the makespans of these executions are still shorter than that of no-replication execution at the same or slightly higher scales than when no contention takes place. One difference due to contention is that in our experiments using $g = 3$ groups is never worthwhile.

8 Conclusion

In this paper, we have studied group replication as a fault-tolerance mechanism for parallel applications on large-scale platforms. While others have studied process replication [12], group replication is a more general, simpler, and less intrusive approach. We have defined an execution protocol for group replication, ASAP. We have derived a bound on the expected application makespan using this protocol when failures are exponentially distributed. We have also proposed several dynamic programming algorithms to minimize application makespan that apply regardless of the failure distribution. We have evaluated all these approaches, along with no-replication approaches proposed in previous work, in simulation. Our main findings are that (i) replication can significantly lower the execution time of applications on very large scale platforms, for failure and checkpointing characteristics corresponding to today's platforms; and (ii) our DPNextFailure dynamic programming approach is close to the optimal peri-

odic solution (determined via a numerical search for the best period that would be prohibitively expensive in practice). Some of the approaches that we have evaluated are more effective than DPNextFailure when the failure distribution is close to the exponential distribution. However, studies have shown that failures in production platforms today are far from being exponentially distributed [14, 25, 20, 15].

An interesting direction for future work is to compare group replication with process replication [12], both theoretically and experimentally, thereby determining in which regimes one replication method is better than the other, if at all. A more ambitious longer-term objective is to generalize this work beyond the case of coordinated checkpointing, for instance in the case of hierarchical checkpointing schemes based on message logging.

Acknowledgments

This work was supported in part by the ANR RESCUE project, and by the INRIA-Illinois Joint Laboratory for Petascale Computing.

References

[1] G. Amdahl. The validity of the single processor approach to achieving large scale computing capabilities. In *Proc. of AFIPS*, volume 30, pages 483–485, 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] M. Bougeret, H. Casanova, M. Rabie, Y. Robert, and F. Vivien. Checkpointing strategies for parallel jobs. In *Proc. Int. Conf. High Performance Computing, Networking, Storage and Analysis SC'11*. ACM Press, 2011.

[5] M. Bougeret, H. Casanova, Y. Robert, F. Vivien, and D. Zaidouni. Using group replication for resilience on exascale systems. Research Report RR-7876, INRIA, ENS Lyon, France, 2012. Available at `http://hal.inria.fr/hal-00668016`.

[6] M.-S. Bouguerra, T. Gautier, D. Trystram, and J.-M. Vincent. A flexible checkpoint/restart model in distributed systems. In *Proc. of PPAM*, volume 6067 of *LNCS*, pages 206–215, 2010.

[7] F. Cappello, H. Casanova, and Y. Robert. Checkpointing vs. migration for post-petascale supercomputers. In *Proc. of the International Conference on Parallel Processing*. 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 Journal of Research and Development*, 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] J. Dongarra, P. Beckman, P. Aerts, F. Cappello, T. Lippert, S. Matsuoka, P. Messina, T. Moore, R. Stevens, A. Trefethen, and M. Valero. The international exascale software project: a call to cooperative action by the global high-performance community. *International Journal of High Performance Computing Applications*, 23(4):309–322, 2009.

[11] C. Engelmann, H. H. Ong, and S. L. Scorr. The case for modular redundancy in large-scale high performance computing systems. In *Proc. of the 8th IASTED Infernational Conference on Parallel and Distributed Computing and Networks)*, pages 189–194, 2009.

[12] K. Ferreira, J. Stearley, J. H. III Laros, R. Oldfield, K. Pedretti, R. Brightwell, R. Riesen, P. G. Bridges, and D. Arnold. Evaluating the Viability of Process Replication Reliability for Exascale Systems. In *Proc. of the ACM/IEEE Conference on Supercomputing*, 2011.

[13] F. Gärtner. Fundamentals of fault-tolerant distributed computing in asynchronous environments. *ACM Computing Surveys*, 31(1), 1999.

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

[15] R. Heien, D. Kondo, A. Gainaru, D. LaPine, B. Kramer, and F. Cappello. Modeling and Tolerating Heterogeneous Failures on Large Parallel System. In *Proc. of the IEEE/ACM Supercomputing Conference (SC)*, 2011.

[16] W.M. Jones, J.T. Daly, and N. DeBardeleben. Impact of sub-optimal checkpoint intervals on application efficiency in computational clusters. In *Proc. of the International ACM Symposium on High-Performance Parallel and Distributed Computing*, pages 276–279, 2010.

[17] N. Kolettis and N. D. Fulton. Software Rejuvenation: Analysis, Module and Applications. In *Proc. of International Symposium on Fault-Tolerant Computing*, page 381, 1995.

[18] D. Kondo, A. Chien, and H. Casanova. Scheduling Task Parallel Applications for Rapid Application Turnaround on Enterprise Desktop Grids. *Journal of Grid Computing*, 5(4):379–405, 2007.

[19] D. Kondo, B. Javadi, A. Iosup, and D. Epema. The Failure Trace Archive: Enabling Comparative Analysis of Failures in Diverse Distributed Systems. In *Proc. of the IEEE International Symposium on Cluster Computing and the Grid*, pages 398–407, Los Alamitos, CA, USA, 2010. IEEE Computer Society.

[20] 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 *Proc. of International Parallel and Distributed Processing Symposium*, pages 1–9, 2008.

[21] 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`.

[22] M. Pinedo. *Scheduling: theory, algorithms, and systems (3rd edition)*. Springer, 2008.

[23] 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`.

[24] B. Schroeder and G. Gibson. Understanding failures in petascale computers. *Journal of Physics: Conference Series*, 78(1), 2007.

[25] B. Schroeder and G. A. Gibson. A Large-Scale Study of Failures in High-Performance Computing Systems. In *Proc. of International Conference on Dependable Systems and Networks*, pages 249–258, 2006.

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

[27] L. Wang, P. Karthik, Z. Kalbarczyk, R.K. Iyer, L. Votta, C. Vick, and A. Wood. Modeling Coordinated Checkpointing for Large-Scale Supercomputers. In *Proc. of the International Conference on Dependable Systems and Networks*, pages 812–821, June 2005.

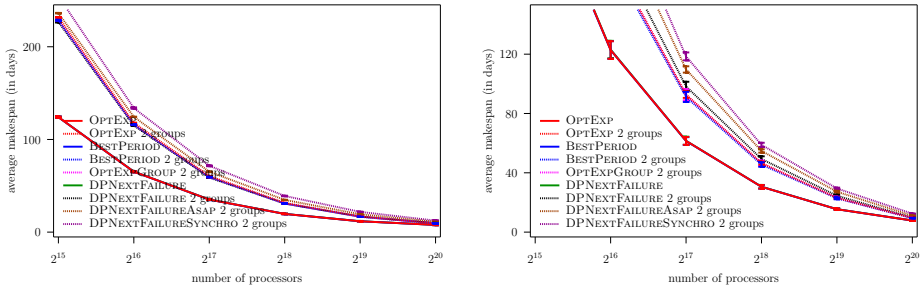
[28] S. Yi, D. Kondo, B. Kim, G. Park, and Y. Cho. Using Replication and Checkpointing for Reliable Task Management in Computational Grids. In *Proc. of the International Conference on High Performance Computing & Simulation*, 2010.

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

[30] Gengbin Zheng, Xiang Ni, and L.V. Kale. A scalable double in-memory checkpoint and restart scheme towards exascale. In *Dependable Syst. and Networks Workshops*, 2012.

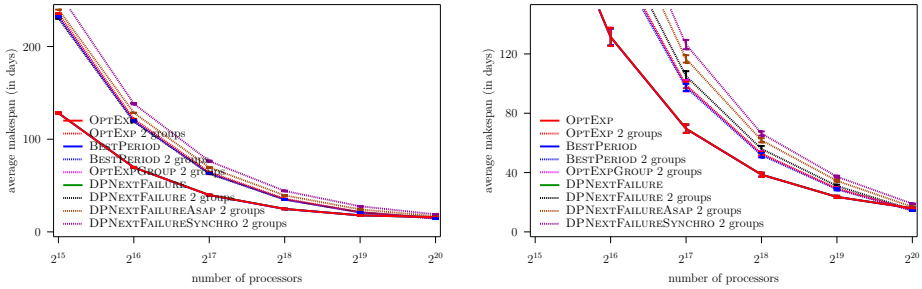
[31] Z. Zheng and Z. Lan. Reliability-aware scalability models for high performance computing. In *Proc. of the IEEE Conference on Cluster Computing*, 2009.

A Full results for the first set of experimental evaluations



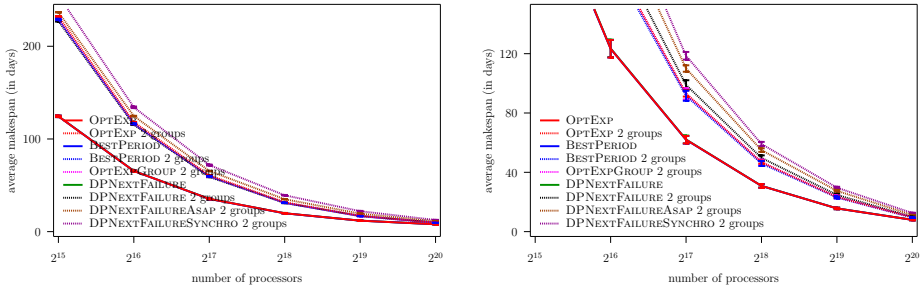
a) constant overhead model b) proportional overhead model

(1) Perfectly parallel jobs.



a) constant overhead model b) proportional overhead model

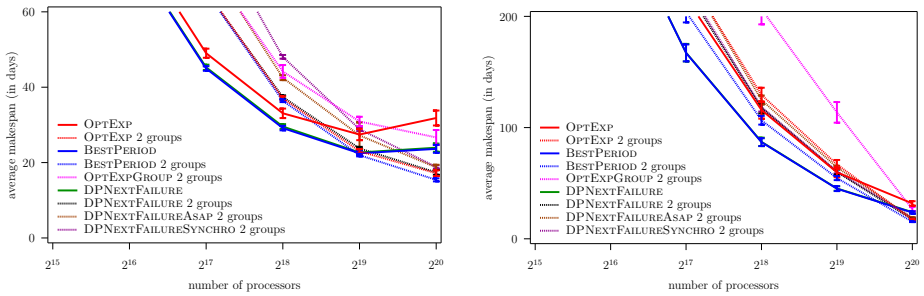
(2) Generic parallel jobs.



a) constant overhead model b) proportional overhead model

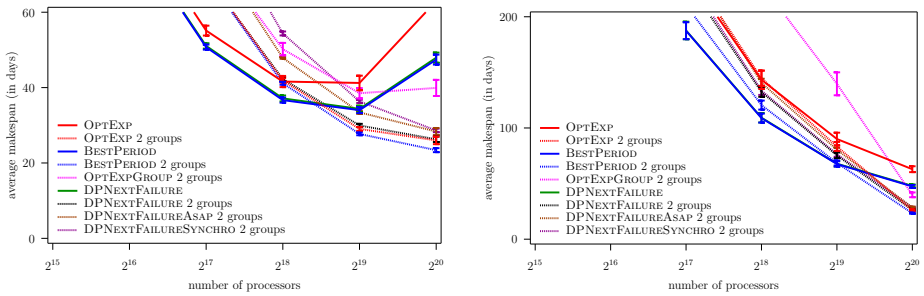
(3) Numerical kernels.

Figure 14: Evaluation of the different heuristics on a platform with **Exponential failures** ($MTBF = 125$ years).



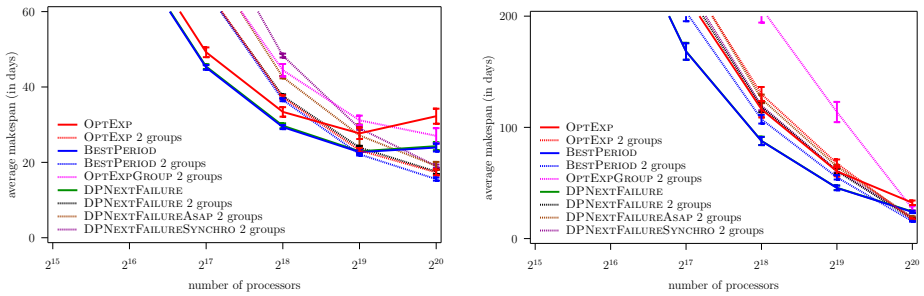
a) constant overhead model b) proportional overhead model

(1) Perfectly parallel jobs.



a) constant overhead model b) proportional overhead model

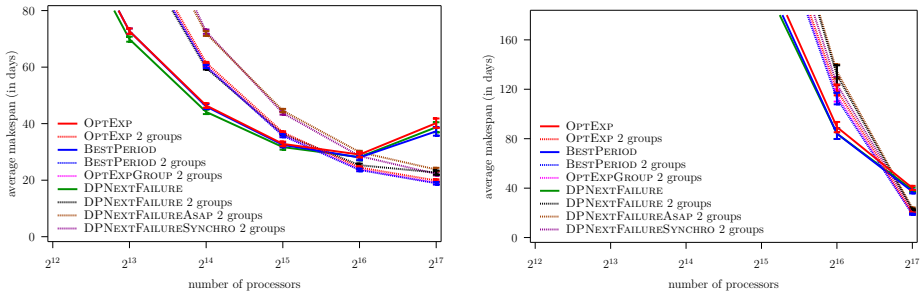
(2) Generic parallel jobs.



a) constant overhead model b) proportional overhead model

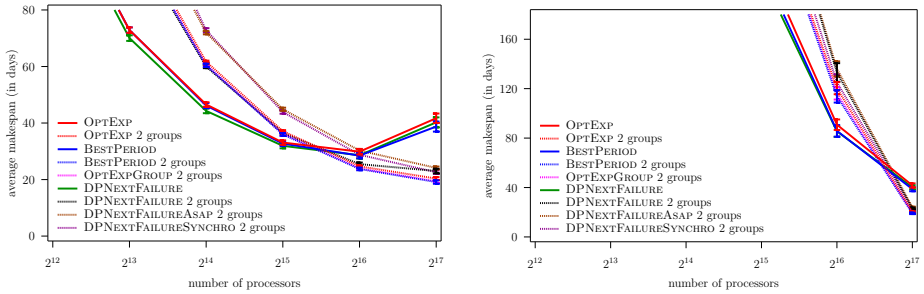
(3) Numerical kernels.

Figure 15: Evaluation of the different heuristics on a platform with **Weibull failures** ($MTBF = 125$ years, and $k = 0.70$).



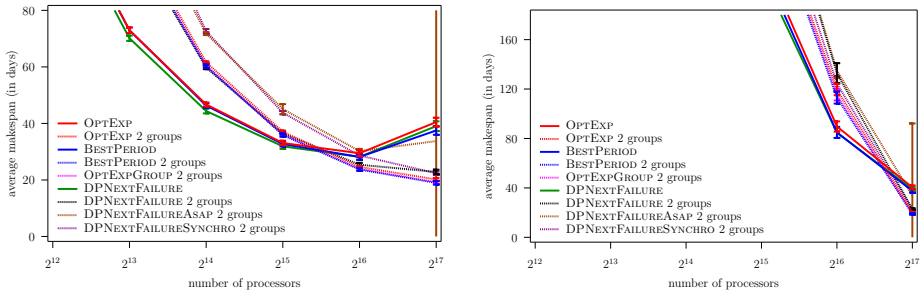
a) constant overhead model b) proportional overhead model

(1) Perfectly parallel jobs.



a) constant overhead model b) proportional overhead model

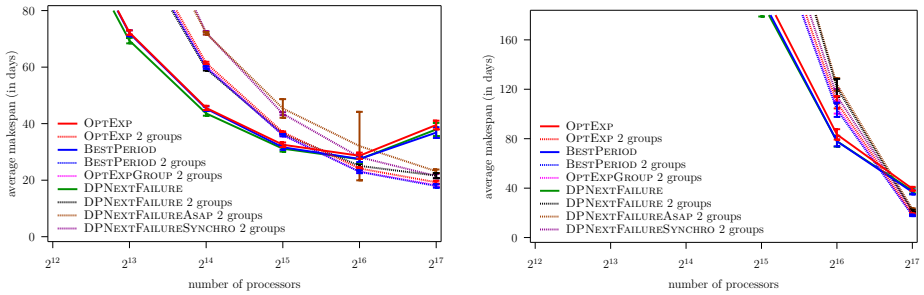
(2) Generic parallel jobs.



a) constant overhead model b) proportional overhead model

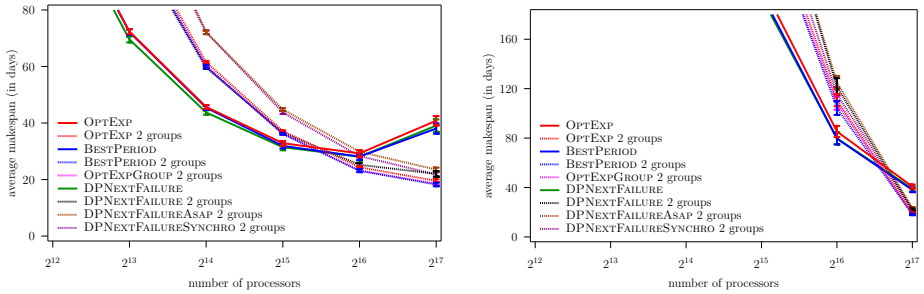
(3) Numerical kernels.

Figure 16: Evaluation of the different heuristics on a platform with failures based on the failure log of **LANL cluster 18**.



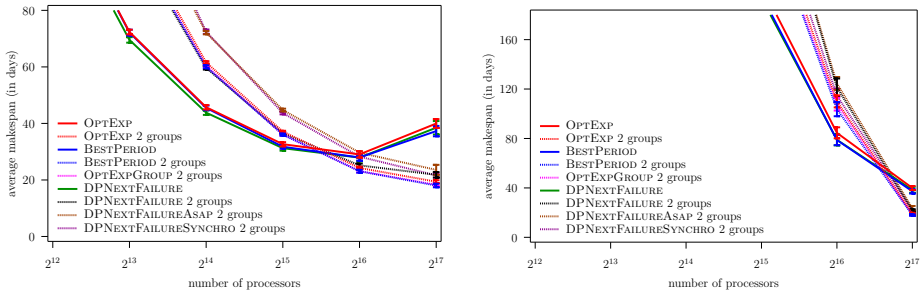
a) constant overhead model b) proportional overhead model

(1) Perfectly parallel jobs.



a) constant overhead model b) proportional overhead model

(2) Generic parallel jobs.



a) constant overhead model b) proportional overhead model

(3) Numerical kernels.

Figure 17: Evaluation of the different heuristics on a platform with failures based on the failure log of **LANL cluster 19**.

	$g = 1$			$g = 2$							
p	OptExp	BestPeriod	DPNextFailure	OptExp	OptExpGroup	BestPeriod	DPNextFailureAsap			DPNextFailureSynchro	DPNextFailure
							MIN	AVG	MAX		
32768	124.04 ± 0.80	124.04 ± 0.80	124.04 ± 0.77	231.69 ± 0.23	228.70 ± 0.55	228.52 ± 0.61	236.23 ± 0.33	225.76 ± 48.69	225.76 ± 48.69	254.43 ± 1.51	226.77 ± 0.67
65536	65.19 ± 0.58	65.19 ± 0.58	65.33 ± 0.53	117.95 ± 0.18	116.08 ± 0.50	116.06 ± 0.55	124.43 ± 0.18	121.67 ± 18.34	118.89 ± 25.64	133.88 ± 0.79	115.08 ± 0.40
131072	35.15 ± 0.48	35.08 ± 0.52	35.22 ± 0.45	60.62 ± 0.14	59.57 ± 0.42	59.52 ± 0.33	65.23 ± 0.13	62.40 ± 13.30	63.81 ± 9.51	71.65 ± 0.52	58.88 ± 0.35
262144	19.67 ± 0.38	19.62 ± 0.37	19.73 ± 0.34	31.61 ± 0.15	31.02 ± 0.31	30.95 ± 0.30	34.83 ± 0.16	34.85 ± 0.17	34.81 ± 0.14	39.04 ± 0.32	30.77 ± 0.22
524288	11.75 ± 0.31	11.73 ± 0.31	11.79 ± 0.35	16.97 ± 0.15	16.70 ± 0.26	16.60 ± 0.20	19.30 ± 0.19	18.53 ± 4.07	18.83 ± 2.87	21.89 ± 0.28	16.83 ± 0.21
1048576	7.82 ± 0.32	7.82 ± 0.32	7.87 ± 0.32	9.58 ± 0.14	9.51 ± 0.26	9.42 ± 0.18	11.08 ± 0.16	11.15 ± 0.57	11.08 ± 0.13	12.45 ± 0.22	9.81 ± 0.23
2097152	6.24 ± 0.34	6.24 ± 0.34	4.77 ± 2.66	5.87 ± 0.15	5.86 ± 0.22	5.77 ± 0.18	5.66 ± 2.69	6.06 ± 2.37	5.31 ± 2.96	6.10 ± 2.90	6.31 ± 0.21
4194304	7.07 ± 0.52	7.07 ± 0.52	6.40 ± 2.27	4.23 ± 0.17	4.24 ± 0.27	4.19 ± 0.22	4.44 ± 1.67	4.59 ± 1.52	4.52 ± 1.72	4.56 ± 1.51	4.70 ± 0.26

(a) Perfectly parallel jobs.

	$g = 1$			$g = 2$							
p	OptExp	BestPeriod	DPNextFailure	OptExp	OptExpGroup	BestPeriod	DPNextFailureAsap			DPNextFailureSynchro	DPNextFailure
							MIN	AVG	MAX		
32768	128.21 ± 0.78	128.16 ± 0.77	128.22 ± 0.76	235.49 ± 0.23	232.45 ± 0.55	232.29 ± 0.62	240.14 ± 0.31	225.11 ± 58.12	225.10 ± 58.12	258.61 ± 1.55	230.46 ± 0.60
65536	69.53 ± 0.59	69.53 ± 0.59	69.67 ± 0.53	121.82 ± 0.17	119.97 ± 0.53	119.92 ± 0.57	128.50 ± 0.17	125.43 ± 19.59	122.37 ± 27.36	138.33 ± 0.78	118.83 ± 0.38
131072	39.76 ± 0.50	39.69 ± 0.59	39.88 ± 0.46	64.58 ± 0.16	63.44 ± 0.46	63.35 ± 0.39	69.54 ± 0.22	69.51 ± 0.17	69.51 ± 0.19	76.38 ± 0.58	62.72 ± 0.35
262144	24.79 ± 0.41	24.75 ± 0.44	24.86 ± 0.40	35.77 ± 0.18	35.11 ± 0.34	35.04 ± 0.34	39.39 ± 0.16	39.41 ± 0.18	39.43 ± 0.22	44.20 ± 0.41	34.81 ± 0.26
524288	17.85 ± 0.37	17.83 ± 0.35	17.90 ± 0.37	21.41 ± 0.16	21.07 ± 0.33	20.94 ± 0.23	24.32 ± 0.20	23.75 ± 3.58	24.28 ± 0.15	27.65 ± 0.34	21.23 ± 0.24
1048576	15.92 ± 0.49	15.89 ± 0.49	16.00 ± 0.49	14.58 ± 0.17	14.46 ± 0.32	14.32 ± 0.20	16.89 ± 0.33	16.47 ± 2.49	16.83 ± 0.20	18.96 ± 0.27	14.90 ± 0.28
2097152	19.29 ± 0.63	19.28 ± 0.64	17.80 ± 5.29	12.07 ± 0.22	12.08 ± 0.35	11.88 ± 0.23	13.56 ± 3.06	11.12 ± 5.81	12.81 ± 4.16	13.83 ± 4.50	12.84 ± 0.28

(b) Generic parallel jobs.

	$g = 1$			$g = 2$							
p	OptExp	BestPeriod	DPNextFailure	OptExp	OptExpGroup	BestPeriod	DPNextFailureAsap			DPNextFailureSynchro	DPNextFailure
							MIN	AVG	MAX		
32768	124.43 ± 0.82	124.43 ± 0.82	124.45 ± 0.78	232.12 ± 0.23	229.15 ± 0.56	228.94 ± 0.70	236.71 ± 0.30	231.57 ± 34.52	231.53 ± 34.52	254.88 ± 1.53	227.18 ± 0.65
65536	65.42 ± 0.59	65.42 ± 0.59	65.57 ± 0.53	118.26 ± 0.16	116.48 ± 0.49	116.44 ± 0.50	124.77 ± 0.31	119.08 ± 25.99	124.75 ± 0.15	134.20 ± 0.86	115.36 ± 0.37
131072	35.35 ± 0.49	35.31 ± 0.60	35.46 ± 0.45	60.84 ± 0.16	59.76 ± 0.41	59.72 ± 0.43	65.47 ± 0.16	62.63 ± 13.35	62.62 ± 13.35	71.91 ± 0.54	59.10 ± 0.33
262144	19.86 ± 0.38	19.77 ± 0.35	19.88 ± 0.33	31.78 ± 0.15	31.17 ± 0.31	31.14 ± 0.31	35.05 ± 0.22	34.20 ± 5.28	35.03 ± 0.16	39.26 ± 0.37	30.94 ± 0.22
524288	11.88 ± 0.34	11.87 ± 0.33	11.94 ± 0.34	17.10 ± 0.15	16.84 ± 0.25	16.73 ± 0.19	19.44 ± 0.19	19.06 ± 2.76	17.45 ± 5.89	22.09 ± 0.33	16.97 ± 0.20
1048576	7.95 ± 0.34	7.95 ± 0.34	8.01 ± 0.33	9.69 ± 0.13	9.61 ± 0.24	9.52 ± 0.18	11.18 ± 0.14	10.95 ± 1.60	11.19 ± 0.15	12.56 ± 0.22	9.93 ± 0.21
2097152	6.39 ± 0.39	6.39 ± 0.39	6.15 ± 1.31	5.97 ± 0.16	5.97 ± 0.28	5.88 ± 0.18	6.61 ± 1.71	6.46 ± 1.95	6.93 ± 1.03	7.44 ± 1.11	6.41 ± 0.25

(c) Numerical kernels.

Table 1: Evaluation of the different heuristics on a platform with **Exponential failures** ($MTBF = 125$ years) under the constant overhead model.

	$g=1$			$g=2$							
p	OptExp	BestPeriod	DPNextFailure	OptExp	OptExpGroup	BestPeriod	DPNextFailureAsap			DPNextFailureSynchro	DPNextFailure
							MIN	AVG	MAX		
65536	122.82 ± 5.95	122.82 ± 5.95	123.13 ± 5.90	185.77 ± 4.98	185.87 ± 7.70	182.95 ± 5.33	219.23 ± 4.47	214.15 ± 32.56	209.34 ± 45.36	236.69 ± 7.06	197.28 ± 6.68
131072	61.55 ± 2.63	61.47 ± 2.61	61.60 ± 2.75	92.78 ± 2.22	92.63 ± 3.73	91.17 ± 3.42	109.71 ± 2.14	104.94 ± 22.47	107.29 ± 16.14	118.51 ± 2.68	98.29 ± 3.12
262144	30.59 ± 1.20	30.59 ± 1.20	30.68 ± 1.17	46.59 ± 1.05	46.46 ± 1.77	45.56 ± 1.51	54.76 ± 1.14	54.76 ± 1.15	54.76 ± 1.14	58.87 ± 1.42	49.33 ± 1.81
524288	15.48 ± 0.56	15.47 ± 0.50	15.51 ± 0.54	23.09 ± 0.50	23.12 ± 0.81	22.82 ± 0.68	27.29 ± 0.58	26.04 ± 5.71	26.67 ± 4.11	29.44 ± 0.59	24.56 ± 0.69
1048576	7.82 ± 0.32	7.82 ± 0.32	7.87 ± 0.32	9.58 ± 0.14	9.51 ± 0.26	9.42 ± 0.18	11.08 ± 0.16	11.15 ± 0.57	11.08 ± 0.13	12.45 ± 0.22	9.81 ± 0.23

(a) Perfectly parallel jobs.

	$g=1$			$g=2$							
p	OptExp	BestPeriod	DPNextFailure	OptExp	OptExpGroup	BestPeriod	DPNextFailureAsap			DPNextFailureSynchro	DPNextFailure
							MIN	AVG	MAX		
65536	131.49 ± 6.17	131.37 ± 5.66	131.47 ± 5.77	191.89 ± 5.08	191.66 ± 8.18	188.71 ± 6.00	226.05 ± 5.06	220.46 ± 34.77	215.16 ± 48.35	243.64 ± 6.92	203.93 ± 7.01
131072	69.59 ± 2.85	69.47 ± 2.67	69.72 ± 2.88	99.11 ± 2.12	98.65 ± 3.79	97.46 ± 2.39	116.68 ± 2.46	116.68 ± 2.46	116.68 ± 2.46	126.32 ± 3.14	105.08 ± 3.30
262144	38.63 ± 1.39	38.57 ± 1.27	38.63 ± 1.32	52.79 ± 1.12	52.54 ± 1.69	51.72 ± 1.53	61.80 ± 1.37	61.80 ± 1.37	61.80 ± 1.36	66.36 ± 1.40	55.99 ± 1.97
524288	23.60 ± 0.75	23.58 ± 0.73	23.65 ± 0.72	29.21 ± 0.57	29.15 ± 0.82	28.77 ± 0.74	34.46 ± 0.64	33.68 ± 5.12	34.46 ± 0.64	37.22 ± 0.66	31.02 ± 0.84
1048576	15.92 ± 0.49	15.89 ± 0.49	16.00 ± 0.49	14.58 ± 0.17	14.46 ± 0.32	14.32 ± 0.20	16.89 ± 0.33	16.47 ± 2.49	16.83 ± 0.20	18.96 ± 0.27	14.90 ± 0.28

(b) Generic parallel jobs.

	$g=1$			$g=2$							
p	OptExp	BestPeriod	DPNextFailure	OptExp	OptExpGroup	BestPeriod	DPNextFailureAsap			DPNextFailureSynchro	DPNextFailure
							MIN	AVG	MAX		
65536	123.32 ± 5.91	123.32 ± 5.91	123.60 ± 5.88	186.27 ± 4.91	186.38 ± 7.72	183.16 ± 5.69	219.41 ± 4.54	209.67 ± 45.96	219.41 ± 4.54	236.77 ± 6.85	197.70 ± 6.67
131072	62.01 ± 2.52	61.92 ± 2.57	62.15 ± 2.66	93.19 ± 2.09	92.90 ± 3.74	91.64 ± 3.38	110.03 ± 2.16	105.17 ± 22.53	105.17 ± 22.53	118.61 ± 2.59	98.96 ± 3.14
262144	30.85 ± 1.22	30.85 ± 1.22	30.90 ± 1.14	46.86 ± 1.00	46.86 ± 1.70	45.94 ± 1.68	54.99 ± 1.20	53.75 ± 8.38	55.01 ± 1.22	59.11 ± 1.44	49.70 ± 1.75
524288	15.65 ± 0.56	15.65 ± 0.56	15.70 ± 0.54	23.35 ± 0.56	23.43 ± 0.92	23.06 ± 0.69	27.53 ± 0.62	26.99 ± 3.94	24.72 ± 8.36	29.66 ± 0.61	24.78 ± 0.70
1048576	7.95 ± 0.34	7.95 ± 0.34	8.01 ± 0.33	9.69 ± 0.13	9.61 ± 0.24	9.52 ± 0.18	11.18 ± 0.14	10.95 ± 1.60	11.19 ± 0.15	12.56 ± 0.22	9.93 ± 0.21

(c) Numerical kernels.

Table 2: Evaluation of the different heuristics on a platform with **Exponential failures** ($MTBF = 125$ years) under the proportional overhead model.

	$g = 1$			$g = 2$							
p	OptExp	BestPeriod	DPNextFailure	OptExp	OptExpGroup	BestPeriod	DPNextFailureAsap			DPNextFailureSynchro	DPNextFailure
							MIN	AVG	MAX		
32768	185.25 ± 3.61	155.85 ± 1.11	156.57 ± 1.33	252.55 ± 1.47	290.05 ± 4.79	246.77 ± 0.59	263.00 ± 0.54	253.53 ± 48.80	234.72 ± 81.31	300.82 ± 1.41	243.12 ± 0.88
65536	122.64 ± 2.82	93.39 ± 1.05	93.85 ± 1.08	140.56 ± 1.67	182.93 ± 4.02	132.63 ± 0.60	150.79 ± 0.49	150.72 ± 0.49	140.98 ± 37.03	171.12 ± 0.85	132.83 ± 0.65
131072	93.87 ± 3.00	62.39 ± 1.11	63.22 ± 1.07	85.44 ± 1.23	132.04 ± 3.79	74.82 ± 0.56	87.50 ± 0.55	80.62 ± 23.28	87.38 ± 0.48	98.41 ± 0.77	77.45 ± 0.63
262144	93.61 ± 4.46	50.45 ± 1.21	51.78 ± 1.28	60.58 ± 1.70	120.19 ± 3.88	46.64 ± 0.61	56.20 ± 0.59	56.09 ± 0.53	56.11 ± 0.55	60.35 ± 0.44	50.80 ± 0.60
524288	138.07 ± 7.80	58.40 ± 1.53	59.48 ± 1.62	55.25 ± 2.63	148.70 ± 8.00	34.24 ± 0.92	42.41 ± 2.57	40.49 ± 8.05	41.14 ± 25.87	41.51 ± 0.85	39.33 ± 0.97
1048576	316.08 ± 13.15	118.78 ± 3.53	121.30 ± 3.81	80.28 ± 3.95	252.39 ± 11.14	35.61 ± 1.08	47.45 ± 22.38	40.49 ± 7.59	44.50 ± 23.21	39.08 ± 1.05	39.95 ± 1.55

(a) Perfectly parallel jobs.

	$g = 1$			$g = 2$							
p	OptExp	BestPeriod	DPNextFailure	OptExp	OptExpGroup	BestPeriod	DPNextFailureAsap			DPNextFailureSynchro	DPNextFailure
							MIN	AVG	MAX		
32768	191.09 ± 3.64	160.93 ± 1.09	161.58 ± 1.36	256.24 ± 1.48	294.34 ± 5.02	250.75 ± 0.73	267.21 ± 0.50	267.18 ± 0.51	257.87 ± 48.74	305.94 ± 1.38	247.00 ± 0.76
65536	130.01 ± 2.90	99.44 ± 1.05	99.86 ± 1.11	145.06 ± 1.73	187.85 ± 4.38	136.87 ± 0.84	155.62 ± 0.59	155.63 ± 0.66	150.47 ± 27.48	176.47 ± 0.86	137.07 ± 0.72
131072	105.84 ± 2.95	70.49 ± 1.07	71.27 ± 1.12	90.66 ± 1.33	140.19 ± 3.64	79.78 ± 0.52	93.01 ± 0.43	90.27 ± 17.49	92.99 ± 0.56	104.70 ± 0.68	82.53 ± 0.62
262144	115.87 ± 4.79	63.41 ± 1.16	64.86 ± 1.17	68.21 ± 1.80	134.36 ± 3.52	52.58 ± 0.68	63.32 ± 0.84	67.97 ± 38.89	63.24 ± 0.61	68.08 ± 0.61	57.33 ± 0.85
524288	194.87 ± 8.45	86.94 ± 1.79	88.51 ± 1.89	68.98 ± 2.23	180.03 ± 8.58	43.22 ± 0.78	53.22 ± 2.38	46.92 ± 16.63	50.80 ± 10.01	52.32 ± 0.88	49.70 ± 0.82

(b) Generic parallel jobs.

	$g = 1$			$g = 2$							
p	OptExp	BestPeriod	DPNextFailure	OptExp	OptExpGroup	BestPeriod	DPNextFailureAsap			DPNextFailureSynchro	DPNextFailure
							MIN	AVG	MAX		
32768	185.51 ± 3.74	156.19 ± 1.18	156.94 ± 1.41	252.93 ± 1.47	290.51 ± 4.28	247.23 ± 0.68	263.44 ± 0.56	196.05 ± 114.95	251.16 ± 55.47	301.47 ± 1.56	243.64 ± 0.79
65536	123.09 ± 2.87	93.72 ± 1.01	94.20 ± 1.04	140.97 ± 1.68	183.66 ± 4.29	132.98 ± 0.72	151.27 ± 0.68	104.61 ± 72.16	140.70 ± 38.06	171.60 ± 0.93	133.25 ± 0.72
131072	94.51 ± 3.13	62.75 ± 1.06	63.63 ± 1.15	85.90 ± 1.38	133.04 ± 3.39	75.20 ± 0.58	87.73 ± 0.48	66.77 ± 37.44	87.75 ± 0.74	98.78 ± 0.80	77.83 ± 0.64
262144	94.14 ± 4.71	50.78 ± 1.26	52.08 ± 1.35	60.45 ± 1.80	120.91 ± 4.18	46.74 ± 0.56	56.36 ± 0.58	46.10 ± 21.56	54.83 ± 8.91	60.57 ± 0.52	50.91 ± 0.71
524288	140.61 ± 8.15	59.03 ± 1.68	60.43 ± 1.97	55.74 ± 2.51	150.01 ± 7.45	34.42 ± 0.76	48.99 ± 21.60	40.00 ± 11.35	42.70 ± 3.74	41.79 ± 0.82	39.66 ± 0.91
1048576	317.15 ± 13.22	120.35 ± 3.57	122.22 ± 3.46	80.85 ± 4.14	257.38 ± 11.92	35.80 ± 1.01	50.37 ± 38.55	42.11 ± 0.99	62.52 ± 52.12	39.18 ± 1.02	40.04 ± 1.27

(c) Numerical kernels.

Table 3: Evaluation of the different heuristics on a platform with **Weibull failures** ($MTBF = 125$ years and $k = 0.50$) under the constant overhead model.

	$g = 1$			$g = 2$			$g = 3$		
p	OptExp	BestPeriod	DPNextFailure	OptExp	BestPeriod	DPNextFailure	OptExp	BestPeriod	DPNextFailure
32768	185.25 ± 3.61	155.85 ± 1.11	156.57 ± 1.33	252.55 ± 1.47	246.77 ± 0.59	243.12 ± 0.88			
65536	122.64 ± 2.82	93.39 ± 1.05	93.85 ± 1.08	140.56 ± 1.67	132.63 ± 0.60	132.83 ± 0.65	182.92 ± 0.83	181.94 ± 0.52	182.63 ± 0.58
131072	93.87 ± 3.00	62.39 ± 1.11	63.22 ± 1.07	85.44 ± 1.23	74.82 ± 0.56	77.45 ± 0.63	100.48 ± 0.77	97.30 ± 0.44	99.97 ± 0.32
262144	93.61 ± 4.46	50.45 ± 1.21	51.78 ± 1.28	60.58 ± 1.70	46.64 ± 0.61	50.80 ± 0.60	60.36 ± 1.20	54.97 ± 0.51	58.52 ± 0.61
524288	138.07 ± 7.80	58.40 ± 1.53	59.48 ± 1.62	55.25 ± 2.63	34.24 ± 0.92	39.33 ± 0.97	43.81 ± 1.22	34.55 ± 0.55	39.17 ± 0.71
1048576	316.08 ± 13.15	118.78 ± 3.53	121.30 ± 3.81	80.28 ± 3.95	35.61 ± 1.08	39.95 ± 1.55	44.38 ± 2.22	26.54 ± 0.47	32.18 ± 1.02

(a) Perfectly parallel jobs.

	$g = 1$			$g = 2$			$g = 3$		
p	OptExp	BestPeriod	DPNextFailure	OptExp	BestPeriod	DPNextFailure	OptExp	BestPeriod	DPNextFailure
32768	191.09 ± 3.64	160.93 ± 1.09	161.58 ± 1.36	256.24 ± 1.48	250.75 ± 0.73	247.00 ± 0.76			
65536	130.01 ± 2.90	99.44 ± 1.05	99.86 ± 1.11	145.06 ± 1.73	136.87 ± 0.84	137.07 ± 0.72	186.94 ± 0.73	185.82 ± 0.42	187.17 ± 0.40
131072	105.84 ± 2.95	70.49 ± 1.07	71.27 ± 1.12	90.66 ± 1.33	79.78 ± 0.52	82.53 ± 0.62	104.71 ± 1.12	101.58 ± 0.56	104.04 ± 0.40
262144	115.87 ± 4.79	63.41 ± 1.16	64.86 ± 1.17	68.21 ± 1.80	52.58 ± 0.68	57.33 ± 0.85	65.50 ± 1.27	59.60 ± 0.57	63.85 ± 0.58
524288	194.87 ± 8.45	86.94 ± 1.79	88.51 ± 1.89	68.98 ± 2.23	43.22 ± 0.78	49.70 ± 0.82	51.42 ± 1.66	40.58 ± 0.58	46.85 ± 0.67

(b) Generic parallel jobs.

	$g = 1$			$g = 2$			$g = 3$		
p	OptExp	BestPeriod	DPNextFailure	OptExp	BestPeriod	DPNextFailure	OptExp	BestPeriod	DPNextFailure
32768	185.51 ± 3.74	156.19 ± 1.18	156.94 ± 1.41	252.93 ± 1.47	247.23 ± 0.68	243.64 ± 0.79			
65536	123.09 ± 2.87	93.72 ± 1.01	94.20 ± 1.04	140.97 ± 1.68	132.98 ± 0.72	133.25 ± 0.72	183.40 ± 0.77	182.21 ± 0.43	183.13 ± 0.54
131072	94.51 ± 3.13	62.75 ± 1.06	63.63 ± 1.15	85.90 ± 1.38	75.20 ± 0.58	77.83 ± 0.64	100.52 ± 0.98	97.55 ± 0.49	100.21 ± 0.42
262144	94.14 ± 4.71	50.78 ± 1.26	52.08 ± 1.35	60.45 ± 1.80	46.74 ± 0.56	50.91 ± 0.71	60.91 ± 1.08	55.28 ± 0.48	59.12 ± 0.68
524288	140.61 ± 8.15	59.03 ± 1.68	60.43 ± 1.97	55.74 ± 2.51	34.42 ± 0.76	39.66 ± 0.91	43.88 ± 1.66	34.55 ± 0.54	39.60 ± 0.63
1048576	317.15 ± 13.22	120.35 ± 3.57	122.22 ± 3.46	80.85 ± 4.14	35.80 ± 1.01	40.04 ± 1.27	45.58 ± 2.00	27.05 ± 0.56	33.15 ± 0.98

(c) Numerical kernels.

Table 4: Evaluation of the different heuristics on a platform with **Weibull failures** ($MTBF$ = 125 years and k = 0.50) under the constant overhead model for g=1, g=2 and g=3.

	$g=1$			$g=2$							
p	OptExp	BestPeriod	DPNextFailure	OptExp	OptExpGroup	BestPeriod	DPNextFailureAsap			DPNextFailureSynchro	DPNextFailure
							MIN	AVG	MAX		
32768	142.66 ± 1.91	137.19 ± 1.14	137.51 ± 1.22	236.16 ± 0.87	241.59 ± 2.61	236.06 ± 0.82	251.98 ± 1.46	252.17 ± 1.06	252.38 ± 1.52	277.78 ± 1.27	232.66 ± 0.75
65536	80.44 ± 1.45	76.17 ± 0.79	76.37 ± 0.81	122.54 ± 0.85	128.82 ± 2.50	122.49 ± 0.49	133.43 ± 0.49	133.52 ± 0.52	133.53 ± 0.55	150.89 ± 0.91	121.11 ± 0.63
131072	48.93 ± 1.25	44.92 ± 0.64	45.21 ± 0.66	65.51 ± 0.95	72.56 ± 2.60	65.29 ± 0.45	74.34 ± 0.84	74.41 ± 0.94	74.22 ± 0.57	84.39 ± 0.72	65.51 ± 0.56
262144	33.15 ± 1.25	29.16 ± 0.56	29.52 ± 0.59	37.07 ± 0.53	44.13 ± 1.63	36.41 ± 0.33	42.83 ± 0.50	42.76 ± 0.34	42.87 ± 0.88	48.52 ± 0.56	37.85 ± 0.52
524288	27.43 ± 1.45	22.49 ± 0.62	22.72 ± 0.70	23.00 ± 0.58	30.85 ± 1.23	21.98 ± 0.34	26.60 ± 0.89	26.43 ± 0.58	26.43 ± 0.56	28.73 ± 0.38	24.37 ± 0.50
1048576	31.83 ± 1.93	23.67 ± 1.01	23.96 ± 1.04	17.16 ± 0.77	26.73 ± 1.92	15.38 ± 0.43	18.53 ± 0.43	18.52 ± 0.43	18.71 ± 0.98	18.84 ± 0.44	17.98 ± 0.76

(a) Perfectly parallel jobs.

	$g=1$			$g=2$							
p	OptExp	BestPeriod	DPNextFailure	OptExp	OptExpGroup	BestPeriod	DPNextFailureAsap			DPNextFailureSynchro	DPNextFailure
							MIN	AVG	MAX		
32768	147.35 ± 1.87	141.72 ± 1.18	142.06 ± 1.22	240.03 ± 0.87	245.59 ± 2.66	239.98 ± 0.84	255.91 ± 0.80	256.44 ± 0.96	256.59 ± 0.76	282.32 ± 1.29	236.52 ± 0.69
65536	85.76 ± 1.43	81.10 ± 0.81	81.30 ± 0.78	126.54 ± 0.89	132.99 ± 2.50	126.52 ± 0.52	137.70 ± 0.37	137.74 ± 0.36	137.79 ± 0.35	155.87 ± 1.02	125.06 ± 0.62
131072	55.16 ± 1.29	50.76 ± 0.66	51.13 ± 0.74	69.78 ± 1.00	77.40 ± 2.75	69.54 ± 0.49	79.25 ± 1.06	79.06 ± 0.40	79.01 ± 0.29	89.92 ± 0.77	69.77 ± 0.59
262144	41.62 ± 1.47	36.69 ± 0.68	37.16 ± 0.77	41.84 ± 0.61	50.12 ± 1.71	41.11 ± 0.40	48.43 ± 0.57	48.38 ± 0.40	48.54 ± 0.82	54.77 ± 0.59	42.77 ± 0.53
524288	41.25 ± 1.93	34.07 ± 0.78	34.37 ± 0.86	29.02 ± 0.50	38.49 ± 1.37	27.70 ± 0.34	33.16 ± 0.32	33.22 ± 0.31	33.22 ± 0.33	36.17 ± 0.45	30.66 ± 0.49
1048576	63.14 ± 2.71	47.57 ± 1.38	48.12 ± 1.48	25.99 ± 0.98	39.94 ± 2.13	23.41 ± 0.52	28.47 ± 1.46	28.26 ± 1.02	28.18 ± 0.46	28.70 ± 0.51	27.28 ± 0.73

(b) Generic parallel jobs.

	$g=1$			$g=2$							
p	OptExp	BestPeriod	DPNextFailure	OptExp	OptExpGroup	BestPeriod	DPNextFailureAsap			DPNextFailureSynchro	DPNextFailure
							MIN	AVG	MAX		
32768	143.09 ± 1.86	137.57 ± 1.14	137.93 ± 1.27	236.61 ± 0.88	242.03 ± 2.62	236.51 ± 0.85	252.38 ± 0.71	252.74 ± 0.93	252.92 ± 1.29	278.28 ± 1.32	233.11 ± 0.74
65536	80.83 ± 1.45	76.51 ± 0.79	76.67 ± 0.78	122.85 ± 0.84	129.19 ± 2.47	122.83 ± 0.50	133.61 ± 0.34	133.80 ± 0.47	133.75 ± 0.46	151.34 ± 0.90	121.45 ± 0.63
131072	49.15 ± 1.29	45.14 ± 0.62	45.39 ± 0.64	65.77 ± 0.94	72.96 ± 2.63	65.53 ± 0.45	74.52 ± 0.68	74.57 ± 0.62	74.71 ± 1.28	84.73 ± 0.71	65.74 ± 0.54
262144	33.37 ± 1.27	29.36 ± 0.60	29.70 ± 0.63	37.27 ± 0.54	44.42 ± 1.61	36.59 ± 0.36	43.14 ± 0.76	43.04 ± 0.51	43.07 ± 0.82	48.75 ± 0.51	38.10 ± 0.48
524288	27.73 ± 1.45	22.72 ± 0.64	22.99 ± 0.75	23.18 ± 0.58	31.09 ± 1.22	22.13 ± 0.32	26.47 ± 0.29	26.59 ± 0.35	26.64 ± 0.48	28.95 ± 0.38	24.53 ± 0.53
1048576	32.34 ± 1.96	24.00 ± 1.06	24.37 ± 1.09	17.34 ± 0.78	27.07 ± 2.02	15.56 ± 0.41	18.74 ± 0.44	18.68 ± 0.41	18.75 ± 0.46	19.02 ± 0.42	18.23 ± 0.68

(c) Numerical kernels.

Table 5: Evaluation of the different heuristics on a platform with **Weibull failures** ($MTBF = 125$ years and $k = 0.70$) under the constant overhead model.

	$g=1$			$g=2$							
p	OptExp	BestPeriod	DPNextFailure	OptExp	OptExpGroup	BestPeriod	DPNextFailureAsap			DPNextFailureSynchro	DPNextFailure
							MIN	AVG	MAX		
131072	216.62 ± 12.54	167.14 ± 7.60	167.39 ± 7.89	238.79 ± 9.69	352.61 ± 7.05	203.97 ± 9.39	239.73 ± 5.73	239.68 ± 5.75	216.01 ± 72.22	227.66 ± 7.16	224.49 ± 6.80
262144	115.80 ± 7.79	86.91 ± 3.49	87.20 ± 3.79	129.42 ± 6.46	206.46 ± 13.63	106.61 ± 3.92	124.88 ± 4.01	122.11 ± 18.84	124.85 ± 4.00	118.61 ± 4.14	117.78 ± 4.98
524288	59.82 ± 4.43	45.10 ± 2.17	45.30 ± 2.01	66.96 ± 3.96	113.70 ± 9.27	54.63 ± 1.83	64.24 ± 1.48	61.33 ± 13.46	61.35 ± 13.47	60.71 ± 1.85	60.08 ± 2.19
1048576	31.83 ± 1.98	23.65 ± 1.05	23.96 ± 1.09	17.17 ± 0.77	26.68 ± 1.97	15.42 ± 0.40	18.73 ± 0.70	19.53 ± 3.92	18.61 ± 6.40	18.86 ± 0.44	17.40 ± 0.63

(a) Perfectly parallel jobs.

	$g=1$			$g=2$							
p	OptExp	BestPeriod	DPNextFailure	OptExp	OptExpGroup	BestPeriod	DPNextFailureAsap			DPNextFailureSynchro	DPNextFailure
							MIN	AVG	MAX		
131072	240.53 ± 13.44	187.48 ± 7.62	187.70 ± 7.91	247.26 ± 6.62	357.09 ± 4.09	210.27 ± 4.24	249.53 ± 1.68	249.55 ± 1.81	249.55 ± 1.80	237.94 ± 4.10	233.14 ± 5.74
262144	143.33 ± 8.19	108.90 ± 4.01	109.25 ± 4.27	144.80 ± 6.88	229.27 ± 12.39	120.63 ± 4.06	140.61 ± 3.71	140.56 ± 3.73	140.56 ± 3.75	133.63 ± 3.85	132.39 ± 4.34
524288	90.09 ± 5.70	67.64 ± 2.40	67.89 ± 2.48	83.42 ± 4.01	139.75 ± 10.33	68.56 ± 2.19	80.49 ± 1.82	76.99 ± 16.70	78.71 ± 12.01	76.31 ± 2.17	75.41 ± 2.53
1048576	62.95 ± 2.64	47.43 ± 1.33	47.94 ± 1.35	25.98 ± 0.99	39.92 ± 2.14	23.41 ± 0.53	28.36 ± 0.94	29.63 ± 6.25	24.11 ± 9.97	28.64 ± 0.47	26.28 ± 0.72

(b) Generic parallel jobs.

	$g=1$			$g=2$							
p	OptExp	BestPeriod	DPNextFailure	OptExp	OptExpGroup	BestPeriod	DPNextFailureAsap			DPNextFailureSynchro	DPNextFailure
							MIN	AVG	MAX		
131072	218.14 ± 12.52	168.22 ± 7.36	168.41 ± 7.64	237.66 ± 9.63	351.56 ± 7.18	203.78 ± 8.32	239.66 ± 5.12	239.66 ± 5.17	239.80 ± 5.09	228.46 ± 5.97	223.59 ± 6.42
262144	116.29 ± 7.33	87.73 ± 3.62	87.89 ± 3.82	129.91 ± 6.40	207.17 ± 12.93	107.36 ± 3.92	125.69 ± 3.87	123.09 ± 18.55	120.54 ± 25.68	119.45 ± 4.06	118.48 ± 4.76
524288	60.52 ± 4.55	45.55 ± 2.12	45.73 ± 2.05	67.64 ± 3.54	113.81 ± 9.09	55.07 ± 1.93	64.76 ± 1.52	64.76 ± 1.51	64.77 ± 1.48	61.25 ± 1.74	60.79 ± 2.17
1048576	32.25 ± 2.00	23.94 ± 1.04	24.30 ± 1.05	17.34 ± 0.79	27.06 ± 2.04	15.56 ± 0.41	19.07 ± 1.07	18.79 ± 0.47	22.71 ± 22.40	19.03 ± 0.42	17.56 ± 0.62

(c) Numerical kernels.

Table 6: Evaluation of the different heuristics on a platform with **Weibull failures** ($MTBF = 125$ years and $k = 0.70$) under the proportional overhead model.

	$g = 1$			$g = 2$							
p	OptExp	BestPeriod	DPNextFailure	OptExp	OptExpGroup	BestPeriod	DPNextFailureAsap			DPNextFailureSynchro	DPNextFailure
							MIN	AVG	MAX		
4096	124.15 ± 1.33	124.07 ± 1.42	120.94 ± 1.45	203.13 ± 0.58	198.94 ± 1.41	198.95 ± 0.90	221.99 ± 3.39	218.18 ± 4.22	216.48 ± 4.02	234.28 ± 1.75	197.38 ± 0.87
8192	72.71 ± 0.94	72.71 ± 0.94	69.80 ± 0.92	109.17 ± 0.74	107.04 ± 1.49	106.88 ± 0.69	122.11 ± 2.80	119.21 ± 3.65	117.79 ± 2.63	131.25 ± 1.58	106.44 ± 0.79
16384	46.38 ± 0.82	45.97 ± 0.94	44.12 ± 0.80	61.41 ± 0.87	60.27 ± 1.43	60.23 ± 0.59	71.23 ± 2.52	68.32 ± 1.75	67.34 ± 1.67	72.44 ± 0.67	60.08 ± 0.65
32768	32.88 ± 0.78	32.53 ± 0.84	31.76 ± 0.88	36.92 ± 0.39	35.91 ± 0.59	35.74 ± 0.50	44.06 ± 1.63	42.26 ± 1.54	41.55 ± 1.32	43.76 ± 0.56	36.71 ± 0.51
65536	29.12 ± 0.96	28.12 ± 1.00	28.22 ± 1.15	24.53 ± 0.36	23.82 ± 0.45	23.64 ± 0.45	30.02 ± 1.49	28.43 ± 1.23	28.06 ± 1.24	28.57 ± 0.55	25.62 ± 0.54
131072	40.30 ± 1.70	37.42 ± 1.59	38.75 ± 1.70	19.92 ± 0.45	19.09 ± 0.53	18.94 ± 0.44	23.67 ± 0.91	22.99 ± 1.08	22.74 ± 1.18	22.28 ± 0.58	22.68 ± 0.72

(a) Perfectly parallel jobs.

	$g = 1$			$g = 2$							
p	OptExp	BestPeriod	DPNextFailure	OptExp	OptExpGroup	BestPeriod	DPNextFailureAsap			DPNextFailureSynchro	DPNextFailure
							MIN	AVG	MAX		
4096	124.27 ± 1.32	124.22 ± 1.41	121.15 ± 1.63	203.23 ± 0.57	199.05 ± 1.40	199.02 ± 0.89	222.15 ± 4.37	218.33 ± 3.89	216.09 ± 3.64	234.33 ± 1.78	197.50 ± 0.87
8192	72.88 ± 0.93	72.88 ± 0.93	69.95 ± 0.92	109.27 ± 0.74	107.14 ± 1.49	107.00 ± 0.67	122.45 ± 3.50	119.00 ± 3.09	117.90 ± 3.01	131.26 ± 1.44	106.57 ± 0.72
16384	46.57 ± 0.80	46.17 ± 0.93	44.33 ± 0.82	61.54 ± 0.87	60.37 ± 1.44	60.35 ± 0.47	70.85 ± 1.97	68.71 ± 1.77	67.99 ± 1.70	72.69 ± 0.77	60.21 ± 0.70
32768	33.13 ± 0.79	32.76 ± 0.85	31.97 ± 0.88	37.07 ± 0.40	36.08 ± 0.58	35.91 ± 0.51	44.31 ± 1.74	42.74 ± 1.63	41.87 ± 1.37	43.91 ± 0.58	36.88 ± 0.52
65536	29.94 ± 1.01	28.56 ± 1.02	28.67 ± 1.18	24.71 ± 0.36	23.98 ± 0.44	23.77 ± 0.45	30.21 ± 1.74	28.72 ± 1.37	28.47 ± 1.55	28.79 ± 0.58	25.78 ± 0.54
131072	41.80 ± 1.71	38.82 ± 1.78	40.27 ± 1.85	20.40 ± 0.48	19.39 ± 0.54	19.15 ± 0.60	24.05 ± 1.07	23.29 ± 1.11	22.95 ± 1.18	22.59 ± 0.59	22.97 ± 0.74

(b) Generic parallel jobs.

	$g = 1$			$g = 2$							
p	OptExp	BestPeriod	DPNextFailure	OptExp	OptExpGroup	BestPeriod	DPNextFailureAsap			DPNextFailureSynchro	DPNextFailure
							MIN	AVG	MAX		
4096	124.28 ± 1.32	124.23 ± 1.40	121.01 ± 1.48	203.37 ± 0.57	199.19 ± 1.40	199.15 ± 0.89	221.89 ± 3.01	218.15 ± 3.97	216.89 ± 3.71	234.46 ± 1.61	197.66 ± 0.82
8192	72.90 ± 0.92	72.90 ± 0.92	70.03 ± 0.94	109.33 ± 0.73	107.21 ± 1.49	107.01 ± 0.68	122.93 ± 2.61	119.05 ± 3.12	118.16 ± 3.06	131.47 ± 1.58	106.73 ± 0.89
16384	46.58 ± 0.80	46.19 ± 0.94	44.28 ± 0.83	61.56 ± 0.85	60.40 ± 1.42	60.36 ± 0.59	71.36 ± 1.90	68.65 ± 1.92	67.83 ± 1.58	72.60 ± 0.68	60.22 ± 0.66
32768	32.99 ± 0.80	32.64 ± 0.84	31.86 ± 0.88	37.05 ± 0.39	36.00 ± 0.60	35.85 ± 0.51	44.58 ± 1.37	42.34 ± 1.56	41.88 ± 1.49	43.90 ± 0.54	36.84 ± 0.50
65536	29.63 ± 0.98	28.24 ± 0.99	28.40 ± 1.17	24.63 ± 0.36	23.90 ± 0.45	23.72 ± 0.45	29.87 ± 1.39	28.56 ± 1.27	28.07 ± 1.26	28.68 ± 0.57	25.73 ± 0.51
131072	40.73 ± 1.69	37.86 ± 1.66	39.14 ± 1.73	20.22 ± 0.46	19.23 ± 0.53	18.99 ± 0.60	23.89 ± 0.98	23.18 ± 1.17	22.83 ± 1.02	22.41 ± 0.58	22.85 ± 0.77

(c) Numerical kernels.

Table 7: Evaluation of the different heuristics on a platform with failures based on the failure log of **LANL cluster 18** and under under the constant overhead model.

	$g = 1$			$g = 2$							
p	OptExp	BestPeriod	DPNextFailure	OptExp	OptExpGroup	BestPeriod	DPNextFailureAsap			DPNextFailureSynchro	DPNextFailure
							MIN	AVG	MAX		
32768	225.85 ± 12.20	216.94 ± 13.24	210.63 ± 12.11	292.28 ± 11.12	286.29 ± 13.57	285.62 ± 14.10	336.16 ± 14.64	331.58 ± 14.40	335.13 ± 15.56	310.40 ± 14.96	334.08 ± 18.93
65536	89.40 ± 4.16	84.08 ± 4.29	84.13 ± 4.35	119.39 ± 4.34	114.84 ± 4.76	112.46 ± 4.66	134.44 ± 5.68	131.03 ± 5.65	132.19 ± 6.03	123.99 ± 5.79	131.50 ± 7.86
131072	40.19 ± 1.66	37.29 ± 1.54	38.73 ± 1.77	19.89 ± 0.44	19.05 ± 0.52	18.91 ± 0.44	23.78 ± 0.60	22.81 ± 0.55	22.67 ± 0.60	22.22 ± 0.56	22.71 ± 0.73

(a) Perfectly parallel jobs.

	$g = 1$			$g = 2$							
p	OptExp	BestPeriod	DPNextFailure	OptExp	OptExpGroup	BestPeriod	DPNextFailureAsap			DPNextFailureSynchro	DPNextFailure
							MIN	AVG	MAX		
32768	228.11 ± 12.01	219.07 ± 13.30	212.50 ± 12.03	294.47 ± 11.87	289.90 ± 13.54	289.03 ± 15.88	340.13 ± 15.66	313.31 ± 83.69	337.35 ± 16.45	313.06 ± 16.52	338.60 ± 20.48
65536	90.84 ± 4.23	85.85 ± 4.71	85.49 ± 4.49	120.40 ± 4.90	116.24 ± 5.01	113.74 ± 4.98	136.00 ± 6.11	128.93 ± 22.66	126.94 ± 31.41	125.13 ± 6.26	133.05 ± 7.56
131072	41.70 ± 1.68	38.76 ± 1.78	40.24 ± 1.76	20.35 ± 0.52	19.37 ± 0.56	19.11 ± 0.60	24.13 ± 0.48	23.15 ± 0.57	22.98 ± 0.61	22.61 ± 0.61	23.03 ± 0.80

(b) Generic parallel jobs.

	$g = 1$			$g = 2$							
p	OptExp	BestPeriod	DPNextFailure	OptExp	OptExpGroup	BestPeriod	DPNextFailureAsap			DPNextFailureSynchro	DPNextFailure
							MIN	AVG	MAX		
32768	228.56 ± 12.22	221.25 ± 13.14	213.61 ± 12.71	292.61 ± 10.65	287.49 ± 13.03	285.89 ± 14.77	336.91 ± 15.54	312.08 ± 78.23	297.19 ± 107.86	310.60 ± 15.35	335.47 ± 20.40
65536	89.73 ± 4.11	84.94 ± 4.46	84.65 ± 4.18	119.58 ± 4.59	115.53 ± 4.71	113.00 ± 4.77	134.96 ± 6.01	131.79 ± 6.15	129.33 ± 23.86	124.48 ± 5.92	132.86 ± 8.07
131072	40.47 ± 1.54	37.52 ± 1.54	39.03 ± 1.73	20.17 ± 0.51	19.21 ± 0.56	18.96 ± 0.61	33.75 ± 58.42	22.97 ± 0.57	22.86 ± 0.67	22.40 ± 0.61	22.86 ± 0.79

(c) Numerical kernels.

Table 8: Evaluation of the different heuristics on a platform with failures based on the failure log of **LANL cluster 18** and under the proportional overhead model.

	$g=1$			$g=2$							
p	OptExp	BestPeriod	DPNextFailure	OptExp	OptExpGroup	BestPeriod	DPNextFailureAsap			DPNextFailureSynchro	DPNextFailure
							MIN	AVG	MAX		
4096	123.81 ± 1.28	123.67 ± 1.41	120.13 ± 1.46	203.31 ± 0.55	198.87 ± 1.39	198.97 ± 0.92	224.81 ± 5.58	219.66 ± 3.81	216.53 ± 4.32	233.54 ± 1.80	196.71 ± 0.83
8192	72.18 ± 0.82	71.84 ± 0.96	69.28 ± 0.95	109.42 ± 0.77	107.05 ± 1.36	106.98 ± 0.57	122.92 ± 2.72	119.69 ± 1.92	118.55 ± 2.32	130.38 ± 1.57	106.39 ± 0.82
16384	45.50 ± 0.74	45.27 ± 0.79	43.57 ± 0.78	61.36 ± 0.87	60.08 ± 1.37	59.88 ± 0.46	71.57 ± 2.56	68.21 ± 1.98	67.59 ± 2.00	72.15 ± 0.63	59.81 ± 0.59
32768	32.54 ± 0.76	31.63 ± 0.84	31.20 ± 0.96	36.97 ± 0.29	36.00 ± 0.42	35.96 ± 0.42	44.26 ± 2.14	41.86 ± 1.33	41.32 ± 1.21	43.54 ± 0.54	36.84 ± 0.53
65536	28.74 ± 1.10	27.57 ± 1.15	27.54 ± 1.25	24.18 ± 0.40	23.06 ± 0.50	23.01 ± 0.48	29.78 ± 1.08	28.40 ± 1.39	27.99 ± 1.40	28.22 ± 0.56	25.21 ± 0.57
131072	39.69 ± 1.47	36.93 ± 1.85	38.08 ± 2.12	19.26 ± 0.49	18.24 ± 0.52	17.92 ± 0.57	22.95 ± 1.24	21.68 ± 0.94	21.32 ± 0.85	21.48 ± 0.72	21.45 ± 0.75

(a) Perfectly parallel jobs.

	$g=1$			$g=2$							
p	OptExp	BestPeriod	DPNextFailure	OptExp	OptExpGroup	BestPeriod	DPNextFailureAsap			DPNextFailureSynchro	DPNextFailure
							MIN	AVG	MAX		
4096	123.93 ± 1.30	123.80 ± 1.44	120.24 ± 1.51	203.42 ± 0.55	199.00 ± 1.40	199.07 ± 0.91	224.16 ± 4.98	219.79 ± 3.74	216.99 ± 5.19	233.77 ± 1.86	196.76 ± 0.82
8192	72.33 ± 0.81	72.01 ± 0.95	69.43 ± 0.91	109.52 ± 0.76	107.20 ± 1.38	107.09 ± 0.58	123.09 ± 2.93	120.09 ± 1.92	118.02 ± 2.04	130.28 ± 1.52	106.34 ± 0.82
16384	45.73 ± 0.68	45.48 ± 0.78	43.79 ± 0.78	61.47 ± 0.88	60.21 ± 1.39	60.02 ± 0.47	71.60 ± 2.12	68.41 ± 2.00	67.49 ± 1.91	72.27 ± 0.63	59.89 ± 0.55
32768	32.82 ± 0.77	31.90 ± 0.85	31.49 ± 0.94	37.11 ± 0.29	36.07 ± 0.41	36.07 ± 0.41	44.16 ± 2.01	42.09 ± 1.65	41.51 ± 1.34	43.70 ± 0.59	36.98 ± 0.53
65536	29.27 ± 1.11	28.04 ± 1.20	27.98 ± 1.26	24.39 ± 0.40	23.26 ± 0.51	23.19 ± 0.50	30.06 ± 1.07	28.56 ± 1.43	28.16 ± 1.40	28.44 ± 0.57	25.40 ± 0.59
131072	40.93 ± 1.57	38.12 ± 1.79	39.26 ± 2.20	19.59 ± 0.48	18.54 ± 0.52	18.23 ± 0.56	23.35 ± 1.14	22.09 ± 0.96	21.56 ± 0.89	21.82 ± 0.73	21.90 ± 0.84

(b) Generic parallel jobs.

	$g=1$			$g=2$							
p	OptExp	BestPeriod	DPNextFailure	OptExp	OptExpGroup	BestPeriod	DPNextFailureAsap			DPNextFailureSynchro	DPNextFailure
							MIN	AVG	MAX		
4096	124.01 ± 1.31	123.90 ± 1.44	120.39 ± 1.51	203.53 ± 0.56	199.13 ± 1.39	199.20 ± 0.90	224.18 ± 4.70	219.70 ± 3.98	217.95 ± 4.53	234.13 ± 1.92	196.94 ± 0.74
8192	72.31 ± 0.82	72.00 ± 0.95	69.42 ± 0.96	109.59 ± 0.76	107.24 ± 1.33	107.14 ± 0.56	123.37 ± 2.12	120.10 ± 2.08	118.64 ± 2.19	130.39 ± 1.53	106.51 ± 0.74
16384	45.68 ± 0.70	45.42 ± 0.79	43.73 ± 0.78	61.48 ± 0.86	60.21 ± 1.36	60.02 ± 0.46	71.17 ± 2.71	68.95 ± 1.83	67.73 ± 2.00	72.28 ± 0.65	59.87 ± 0.55
32768	32.69 ± 0.77	31.78 ± 0.85	31.38 ± 0.96	37.08 ± 0.29	36.07 ± 0.41	36.05 ± 0.36	44.01 ± 1.59	41.93 ± 1.28	41.54 ± 1.31	43.63 ± 0.54	36.92 ± 0.50
65536	28.98 ± 1.09	27.77 ± 1.18	27.70 ± 1.23	24.30 ± 0.40	23.17 ± 0.50	23.10 ± 0.49	29.94 ± 0.99	28.46 ± 1.33	28.14 ± 1.41	28.36 ± 0.57	25.33 ± 0.59
131072	40.00 ± 1.50	37.25 ± 1.82	38.43 ± 2.22	19.40 ± 0.48	18.34 ± 0.51	18.03 ± 0.57	23.21 ± 1.11	21.80 ± 0.87	21.37 ± 0.84	21.60 ± 0.73	21.60 ± 0.84

(c) Numerical kernels.

Table 9: Evaluation of the different heuristics on a platform with failures based on the failure log of **LANL cluster 19** and under the constant overhead model.

	$g = 1$			$g = 2$							
p	OptExp	BestPeriod	DPNextFailure	OptExp	OptExpGroup	BestPeriod	DPNextFailureAsap			DPNextFailureSynchro	DPNextFailure
							MIN	AVG	MAX		
32768	202.16 ± 10.07	193.98 ± 10.97	191.11 ± 12.24	282.88 ± 15.22	270.91 ± 14.48	270.91 ± 14.48	327.30 ± 15.04	280.86 ± 107.68	297.63 ± 97.70	299.75 ± 16.03	328.45 ± 22.09
65536	83.47 ± 4.21	77.92 ± 4.11	78.00 ± 4.39	109.43 ± 4.90	105.31 ± 4.98	103.38 ± 5.71	123.86 ± 5.16	114.05 ± 28.17	110.22 ± 33.81	114.48 ± 5.65	121.19 ± 7.17
131072	39.52 ± 1.53	36.84 ± 1.90	37.96 ± 2.32	19.25 ± 0.51	18.21 ± 0.54	17.94 ± 0.61	23.15 ± 0.63	21.47 ± 3.56	21.61 ± 0.67	21.45 ± 0.76	21.61 ± 0.90

(a) Perfectly parallel jobs.

	$g = 1$			$g = 2$							
p	OptExp	BestPeriod	DPNextFailure	OptExp	OptExpGroup	BestPeriod	DPNextFailureAsap			DPNextFailureSynchro	DPNextFailure
							MIN	AVG	MAX		
32768	204.85 ± 10.14	196.93 ± 10.88	193.99 ± 11.64	283.04 ± 16.73	270.94 ± 15.46	270.94 ± 15.46	327.08 ± 17.00	290.77 ± 95.42	317.19 ± 60.76	300.30 ± 16.34	328.56 ± 22.29
65536	85.32 ± 4.55	79.53 ± 4.56	79.83 ± 4.96	110.71 ± 4.95	108.39 ± 5.33	104.37 ± 5.59	125.08 ± 5.27	113.88 ± 29.27	121.33 ± 6.30	115.50 ± 5.62	121.36 ± 7.15
131072	40.83 ± 1.64	38.06 ± 1.82	39.09 ± 2.20	19.59 ± 0.52	18.53 ± 0.55	18.25 ± 0.62	23.53 ± 0.73	22.34 ± 0.74	21.95 ± 0.73	21.80 ± 0.79	22.02 ± 0.97

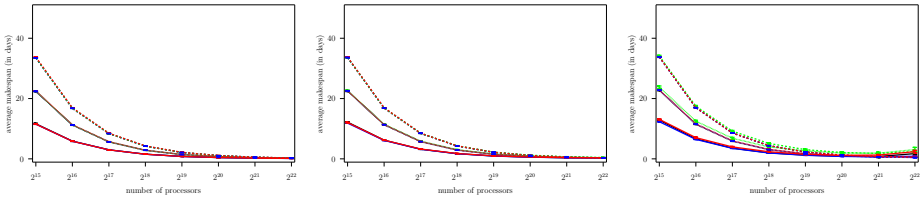
(b) Generic parallel jobs.

	$g = 1$			$g = 2$							
p	OptExp	BestPeriod	DPNextFailure	OptExp	OptExpGroup	BestPeriod	DPNextFailureAsap			DPNextFailureSynchro	DPNextFailure
							MIN	AVG	MAX		
32768	203.83 ± 9.84	195.93 ± 11.42	193.63 ± 12.23	283.43 ± 16.33	271.53 ± 15.56	271.42 ± 16.71	327.53 ± 15.42	281.06 ± 109.68	316.16 ± 60.91	298.87 ± 16.55	328.76 ± 21.71
65536	84.55 ± 4.46	78.74 ± 4.17	78.91 ± 4.48	110.03 ± 4.88	107.74 ± 5.12	103.76 ± 5.73	124.57 ± 5.24	114.00 ± 29.40	121.12 ± 8.23	114.66 ± 5.64	121.31 ± 7.20
131072	39.96 ± 1.58	37.29 ± 1.84	38.48 ± 2.37	19.38 ± 0.53	18.31 ± 0.54	18.03 ± 0.62	23.58 ± 1.78	21.58 ± 3.78	21.71 ± 0.73	21.62 ± 0.79	21.83 ± 0.98

(c) Numerical kernels.

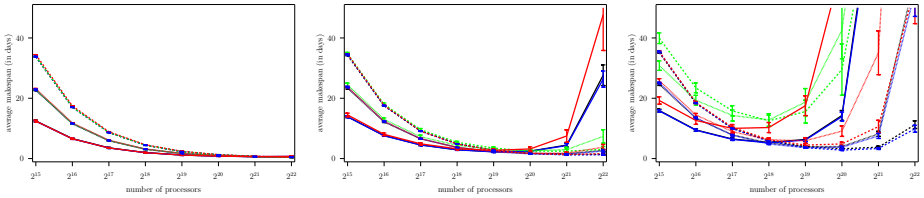
Table 10: Evaluation of the different heuristics on a platform with failures based on the failure log of **LANL cluster 19** and under the proportional overhead model.

B Full results for the second set of experimental evaluations



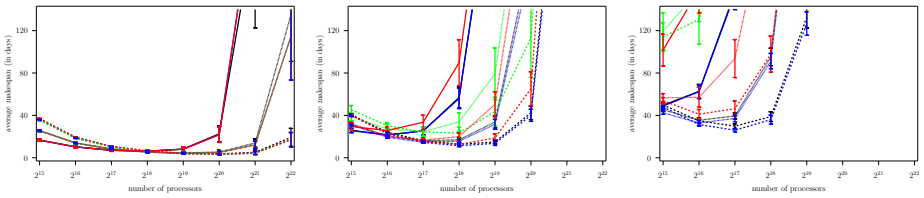
a) Exponential b) Weibull $k = 0.7$ c) Weibull $k = 0.5$

(2) $C = R = 60$, $D = 60$.



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

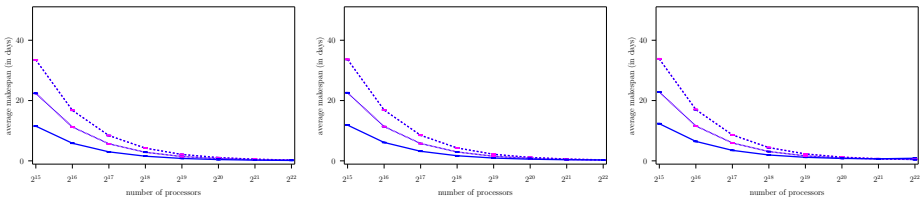
(3) $C = R = 600$, $D = 60$.



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

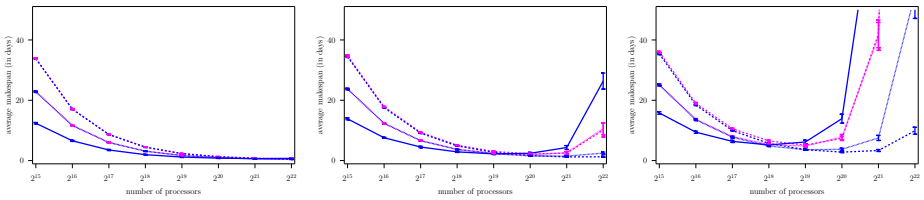
(4) $C = R = 6000$, $D = 60$.

Figure 18: Evaluation of the different heuristics on a platform where $MTBF =$ 125 years.



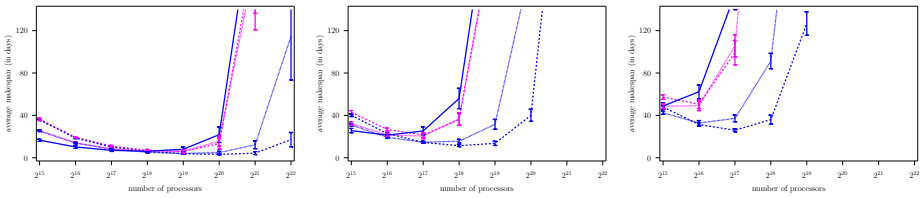
a) Exponential b) Weibull $k = 0.7$ c) Weibull $k = 0.5$

(2) $C = R = 60$, $D = 60$.



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

(3) $C = R = 600$, $D = 60$.



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

(4) $C = R = 6000$, $D = 60$.

Figure 19: Evaluation of the different heuristics on a platform where $MTBF =$ 125 years, in a system with contention.

	$g=1$			$g=2$					$g=3$				
p	OptExp	BestPeriod	DPNextFailure	OptExp	OptExpGroup	BestPeriod	DPNextFailure	BestPeriod-Cont	OptExp	OptExpGroup	BestPeriod	DPNextFailure	BestPeriod-Cont
32768	11.52 ± 0.08	11.52 ± 0.08	11.65 ± 0.18	22.55 ± 0.01	22.42 ± 0.03	22.39 ± 0.07	22.33 ± 0.02	22.40 ± 0.07	33.75 ± 0.00	33.55 ± 0.00	33.48 ± 0.02	33.47 ± 0.00	33.50 ± 0.02
65536	5.85 ± 0.05	5.85 ± 0.06	5.89 ± 0.09	11.33 ± 0.01	11.25 ± 0.04	11.23 ± 0.06	11.20 ± 0.09	11.24 ± 0.06	16.94 ± 0.00	16.80 ± 0.01	16.76 ± 0.02	16.75 ± 0.02	16.77 ± 0.03
131072	2.99 ± 0.05	2.99 ± 0.05	3.01 ± 0.06	5.71 ± 0.01	5.66 ± 0.04	5.65 ± 0.04	5.62 ± 0.05	5.66 ± 0.04	8.52 ± 0.00	8.42 ± 0.01	8.40 ± 0.02	8.39 ± 0.01	8.41 ± 0.02
262144	1.54 ± 0.03	1.54 ± 0.03	1.55 ± 0.04	2.89 ± 0.01	2.85 ± 0.03	2.85 ± 0.03	2.82 ± 0.01	2.85 ± 0.03	4.29 ± 0.00	4.23 ± 0.01	4.22 ± 0.01	4.21 ± 0.01	4.23 ± 0.01
524288	0.80 ± 0.03	0.80 ± 0.03	0.81 ± 0.03	1.47 ± 0.01	1.45 ± 0.02	1.45 ± 0.01	1.43 ± 0.01	1.45 ± 0.01	2.17 ± 0.00	2.13 ± 0.01	2.12 ± 0.01	2.12 ± 0.00	2.13 ± 0.02
1048576	0.44 ± 0.02	0.43 ± 0.01	0.45 ± 0.02	0.75 ± 0.01	0.74 ± 0.01	0.74 ± 0.01	0.73 ± 0.01	0.74 ± 0.01	1.11 ± 0.00	1.08 ± 0.01	1.07 ± 0.02	1.08 ± 0.00	1.08 ± 0.02
2097152	0.24 ± 0.02	0.24 ± 0.02	0.24 ± 0.06	0.39 ± 0.01	0.39 ± 0.01	0.38 ± 0.01	0.38 ± 0.01	0.39 ± 0.01	0.57 ± 0.00	0.55 ± 0.01	0.55 ± 0.01	0.55 ± 0.00	0.55 ± 0.01
4194304	0.15 ± 0.02	0.14 ± 0.02	0.15 ± 0.02	0.21 ± 0.01	0.21 ± 0.01	0.21 ± 0.01	0.21 ± 0.01	0.21 ± 0.01	0.30 ± 0.00	0.28 ± 0.01	0.28 ± 0.00	0.29 ± 0.00	0.29 ± 0.01

(a) $C = R = 60$, $D = 60$

	$g=1$			$g=2$					$g=3$				
p	OptExp	BestPeriod	DPNextFailure	OptExp	OptExpGroup	BestPeriod	DPNextFailure	BestPeriod-Cont	OptExp	OptExpGroup	BestPeriod	DPNextFailure	BestPeriod-Cont
32768	12.42 ± 0.29	12.38 ± 0.25	12.49 ± 0.27	23.19 ± 0.08	22.86 ± 0.14	22.82 ± 0.22	22.64 ± 0.15	22.84 ± 0.21	34.46 ± 0.01	33.89 ± 0.06	33.77 ± 0.18	33.74 ± 0.03	33.90 ± 0.18
65536	6.56 ± 0.20	6.56 ± 0.20	6.61 ± 0.18	11.82 ± 0.07	11.63 ± 0.16	11.62 ± 0.12	11.53 ± 0.16	11.63 ± 0.12	17.46 ± 0.02	17.06 ± 0.05	17.00 ± 0.09	17.02 ± 0.03	17.09 ± 0.24
131072	3.53 ± 0.14	3.51 ± 0.19	3.56 ± 0.15	6.09 ± 0.06	5.99 ± 0.14	5.99 ± 0.14	5.90 ± 0.12	6.00 ± 0.17	8.90 ± 0.02	8.65 ± 0.07	8.62 ± 0.09	8.67 ± 0.07	8.71 ± 0.14
262144	1.96 ± 0.13	1.96 ± 0.13	1.99 ± 0.12	3.19 ± 0.06	3.12 ± 0.13	3.12 ± 0.13	3.09 ± 0.11	3.13 ± 0.10	4.59 ± 0.02	4.44 ± 0.07	4.43 ± 0.10	4.44 ± 0.04	4.50 ± 0.10
524288	1.16 ± 0.12	1.16 ± 0.13	1.18 ± 0.12	1.72 ± 0.05	1.68 ± 0.10	1.68 ± 0.08	1.69 ± 0.07	1.69 ± 0.07	2.41 ± 0.03	2.31 ± 0.07	2.28 ± 0.08	2.33 ± 0.04	2.35 ± 0.08
1048576	0.81 ± 0.12	0.81 ± 0.11	0.83 ± 0.11	0.98 ± 0.05	0.96 ± 0.10	0.95 ± 0.08	0.97 ± 0.07	0.98 ± 0.08	1.29 ± 0.02	1.24 ± 0.05	1.24 ± 0.05	1.27 ± 0.04	1.30 ± 0.08
2097152	0.66 ± 0.12	0.64 ± 0.13	0.63 ± 0.17	0.61 ± 0.06	0.61 ± 0.08	0.60 ± 0.07	0.64 ± 0.07	0.65 ± 0.09	0.73 ± 0.02	0.69 ± 0.05	0.69 ± 0.05	0.74 ± 0.04	0.79 ± 0.07
4194304	0.73 ± 0.20	0.73 ± 0.20	0.63 ± 0.32	0.44 ± 0.07	0.43 ± 0.09	0.43 ± 0.07	0.47 ± 0.10	0.55 ± 0.15	0.45 ± 0.03	0.43 ± 0.05	0.43 ± 0.05	0.48 ± 0.05	0.60 ± 0.12

(b) $C = R = 600$, $D = 60$

	$g=1$			$g=2$					$g=3$				
p	OptExp	BestPeriod	DPNextFailure	OptExp	OptExpGroup	BestPeriod	DPNextFailure	BestPeriod-Cont	OptExp	OptExpGroup	BestPeriod	DPNextFailure	BestPeriod-Cont
32768	16.63 ± 1.20	16.56 ± 1.04	16.67 ± 1.16	25.99 ± 0.50	25.49 ± 1.06	25.44 ± 0.73	25.43 ± 0.94	25.57 ± 0.66	37.21 ± 0.23	35.67 ± 0.59	35.67 ± 0.59	35.87 ± 0.34	36.46 ± 1.15
65536	10.27 ± 1.13	10.20 ± 1.24	10.41 ± 1.11	14.15 ± 0.49	13.92 ± 0.81	13.86 ± 0.64	13.94 ± 0.87	14.10 ± 0.85	19.60 ± 0.24	18.60 ± 0.52	18.60 ± 0.52	19.08 ± 0.42	19.38 ± 0.54
131072	7.02 ± 1.03	7.02 ± 1.03	7.13 ± 0.99	8.26 ± 0.61	8.16 ± 0.89	8.08 ± 0.59	8.44 ± 0.68	8.50 ± 0.91	10.73 ± 0.22	10.31 ± 0.48	10.16 ± 0.37	10.70 ± 0.43	11.04 ± 0.74
262144	6.28 ± 1.27	6.15 ± 1.30	6.22 ± 1.25	5.30 ± 0.53	5.39 ± 0.81	5.15 ± 0.53	5.58 ± 0.68	6.08 ± 0.82	6.29 ± 0.25	6.02 ± 0.54	5.90 ± 0.40	6.42 ± 0.48	7.06 ± 0.82
524288	8.06 ± 2.21	8.06 ± 2.21	8.24 ± 2.19	4.15 ± 0.68	4.21 ± 0.85	4.12 ± 0.78	4.51 ± 0.82	6.15 ± 1.55	4.04 ± 0.35	3.97 ± 0.53	3.79 ± 0.44	4.32 ± 0.57	5.99 ± 1.58
1048576	21.83 ± 7.38	21.83 ± 7.38	22.58 ± 7.43	4.89 ± 1.23	5.05 ± 1.56	4.89 ± 1.23	5.50 ± 1.66	15.62 ± 4.73	3.28 ± 0.48	3.23 ± 0.63	3.17 ± 0.56	3.69 ± 0.72	13.41 ± 5.28
2097152	249.39 ± 64.46	249.39 ± 64.46	219.53 ± 97.02	12.04 ± 3.49	12.41 ± 3.54	12.25 ± 3.68	13.95 ± 4.00	183.96 ± 63.07	4.50 ± 1.35	4.36 ± 1.40	4.36 ± 1.40	5.14 ± 1.98	198.67 ± 62.37
4194304	—	—	—	114.74 ± 41.30	114.74 ± 41.30	114.74 ± 41.30	135.31 ± 44.37	0.00 ± 0.00	17.10 ± 6.73	17.10 ± 6.73	17.10 ± 6.73	19.24 ± 8.49	0.00 ± 0.00

(c) $C = R = 6000$, $D = 60$

Table 11: Evaluation of the different heuristics on a platform with **Exponential failures** ($MTBF = 125$ years) under the constant overhead model.

	$g=1$			$g=2$					$g=3$				
p	OptExp	BestPeriod	DPNextFailure	OptExp	OptExpGroup	BestPeriod	DPNextFailure	BestPeriod-Cont	OptExp	OptExpGroup	BestPeriod	DPNextFailure	BestPeriod-Cont
32768	12.03 ± 0.15	11.86 ± 0.07	12.19 ± 0.13	22.61 ± 0.04	22.67 ± 0.14	22.57 ± 0.06	22.44 ± 0.13	22.58 ± 0.06	33.76 ± 0.01	33.60 ± 0.08	33.60 ± 0.08	33.57 ± 0.06	33.66 ± 0.09
65536	6.21 ± 0.08	6.10 ± 0.05	6.22 ± 0.09	11.39 ± 0.03	11.49 ± 0.10	11.38 ± 0.04	11.28 ± 0.08	11.38 ± 0.04	16.96 ± 0.01	16.87 ± 0.06	16.86 ± 0.04	16.83 ± 0.05	16.90 ± 0.04
131072	3.27 ± 0.08	3.18 ± 0.04	3.21 ± 0.07	5.77 ± 0.03	5.87 ± 0.10	5.76 ± 0.04	5.69 ± 0.06	5.76 ± 0.04	8.53 ± 0.01	8.51 ± 0.07	8.47 ± 0.03	8.45 ± 0.03	8.51 ± 0.03
262144	1.77 ± 0.08	1.70 ± 0.05	1.73 ± 0.05	2.95 ± 0.02	3.06 ± 0.09	2.94 ± 0.03	2.90 ± 0.03	2.95 ± 0.03	4.31 ± 0.01	4.34 ± 0.07	4.28 ± 0.03	4.26 ± 0.01	4.31 ± 0.03
524288	1.00 ± 0.06	0.94 ± 0.04	0.97 ± 0.04	1.53 ± 0.03	1.64 ± 0.07	1.52 ± 0.02	1.50 ± 0.02	1.53 ± 0.02	2.19 ± 0.01	2.24 ± 0.06	2.18 ± 0.02	2.17 ± 0.01	2.21 ± 0.02
1048576	0.60 ± 0.05	0.56 ± 0.03	0.60 ± 0.05	0.81 ± 0.02	0.93 ± 0.07	0.81 ± 0.02	0.81 ± 0.02	0.82 ± 0.02	1.13 ± 0.01	1.22 ± 0.07	1.12 ± 0.01	1.13 ± 0.01	1.15 ± 0.02
2097152	0.41 ± 0.05	0.37 ± 0.02	0.36 ± 0.16	0.46 ± 0.02	0.57 ± 0.07	0.45 ± 0.02	0.46 ± 0.02	0.46 ± 0.02	0.60 ± 0.01	0.68 ± 0.06	0.60 ± 0.01	0.60 ± 0.01	0.62 ± 0.02
4194304	0.35 ± 0.07	0.29 ± 0.03	0.40 ± 0.06	0.28 ± 0.02	0.42 ± 0.06	0.27 ± 0.01	0.31 ± 0.02	0.28 ± 0.01	0.33 ± 0.01	0.45 ± 0.06	0.33 ± 0.01	0.35 ± 0.01	0.35 ± 0.01

(a) $C = R = 60$, $D = 60$

	$g=1$			$g=2$					$g=3$				
p	OptExp	BestPeriod	DPNextFailure	OptExp	OptExpGroup	BestPeriod	DPNextFailure	BestPeriod-Cont	OptExp	OptExpGroup	BestPeriod	DPNextFailure	BestPeriod-Cont
32768	14.46 ± 0.60	13.84 ± 0.31	13.89 ± 0.28	23.73 ± 0.21	24.46 ± 0.60	23.73 ± 0.23	23.30 ± 0.29	23.74 ± 0.23	34.64 ± 0.11	34.73 ± 0.50	34.43 ± 0.24	34.23 ± 0.14	34.73 ± 0.37
65536	8.04 ± 0.42	7.57 ± 0.19	7.63 ± 0.19	12.38 ± 0.21	13.00 ± 0.46	12.34 ± 0.21	12.13 ± 0.18	12.37 ± 0.22	17.66 ± 0.09	18.00 ± 0.42	17.56 ± 0.14	17.50 ± 0.06	17.82 ± 0.21
131072	4.89 ± 0.38	4.49 ± 0.25	4.54 ± 0.24	6.61 ± 0.23	7.24 ± 0.54	6.58 ± 0.18	6.52 ± 0.14	6.64 ± 0.20	9.12 ± 0.10	9.57 ± 0.39	9.11 ± 0.13	9.15 ± 0.08	9.36 ± 0.16
262144	3.24 ± 0.35	2.89 ± 0.19	2.95 ± 0.19	3.72 ± 0.18	4.42 ± 0.35	3.68 ± 0.10	3.72 ± 0.11	3.77 ± 0.12	4.86 ± 0.11	5.40 ± 0.33	4.86 ± 0.09	4.91 ± 0.07	5.12 ± 0.15
524288	2.74 ± 0.47	2.22 ± 0.23	2.27 ± 0.22	2.31 ± 0.20	3.03 ± 0.47	2.23 ± 0.11	2.35 ± 0.15	2.41 ± 0.13	2.74 ± 0.13	3.48 ± 0.37	2.72 ± 0.09	2.83 ± 0.09	3.04 ± 0.18
1048576	3.20 ± 0.69	2.41 ± 0.32	2.49 ± 0.31	1.74 ± 0.25	2.60 ± 0.49	1.56 ± 0.13	1.73 ± 0.16	1.97 ± 0.19	1.71 ± 0.15	2.48 ± 0.41	1.66 ± 0.10	1.79 ± 0.10	2.20 ± 0.22
2097152	7.50 ± 2.03	4.32 ± 0.65	4.47 ± 0.72	1.80 ± 0.38	2.96 ± 0.78	1.47 ± 0.18	1.61 ± 0.20	2.58 ± 0.42	1.32 ± 0.16	2.45 ± 0.68	1.21 ± 0.13	1.39 ± 0.13	2.48 ± 0.43
4194304	47.73 ± 11.89	26.38 ± 2.67	27.65 ± 3.42	3.79 ± 0.90	7.37 ± 2.22	2.48 ± 0.34	2.74 ± 0.36	10.55 ± 2.03	1.63 ± 0.37	3.07 ± 0.94	1.26 ± 0.18	1.42 ± 0.18	10.10 ± 2.31

(b) $C = R = 600$, $D = 60$

	$g=1$			$g=2$					$g=3$				
p	OptExp	BestPeriod	DPNextFailure	OptExp	OptExpGroup	BestPeriod	DPNextFailure	BestPeriod-Cont	OptExp	OptExpGroup	BestPeriod	DPNextFailure	BestPeriod-Cont
32768	29.50 ± 3.30	25.64 ± 2.00	25.91 ± 1.97	31.91 ± 1.77	38.59 ± 3.81	31.01 ± 1.06	31.73 ± 1.12	32.24 ± 1.58	40.09 ± 1.10	45.40 ± 3.82	40.09 ± 1.20	40.62 ± 0.85	42.88 ± 1.68
65536	25.70 ± 3.36	21.35 ± 2.31	21.66 ± 2.09	20.27 ± 1.47	27.30 ± 4.53	19.35 ± 1.08	20.49 ± 1.02	21.57 ± 1.41	23.13 ± 1.16	30.30 ± 2.94	22.88 ± 1.06	24.00 ± 0.68	26.73 ± 1.31
131072	33.49 ± 6.68	25.29 ± 3.73	25.37 ± 3.87	16.62 ± 2.18	24.66 ± 4.26	14.65 ± 0.99	16.04 ± 1.61	20.43 ± 2.51	15.10 ± 1.41	24.19 ± 4.82	14.53 ± 0.92	16.11 ± 0.98	21.51 ± 2.29
262144	89.78 ± 21.68	55.97 ± 9.58	56.89 ± 10.05	19.97 ± 3.32	33.80 ± 8.73	15.62 ± 1.81	16.96 ± 2.25	36.78 ± 5.68	12.49 ± 1.87	23.37 ± 5.32	11.40 ± 0.95	12.80 ± 1.23	36.18 ± 5.58
524288	334.82 ± 27.02	283.30 ± 35.95	285.86 ± 38.05	50.46 ± 11.97	79.17 ± 24.40	31.55 ± 4.65	33.79 ± 5.71	221.35 ± 32.85	18.78 ± 3.78	43.58 ± 13.40	13.60 ± 1.99	14.86 ± 1.97	223.14 ± 36.22
1048576	—	—	—	271.08 ± 51.53	318.41 ± 44.99	182.39 ± 18.93	195.18 ± 21.52	0.00 ± 0.00	65.18 ± 16.01	113.07 ± 37.19	40.06 ± 5.72	42.42 ± 6.40	0.00 ± 0.00
2097152	—	—	—	—	—	—	—	—	523.27 ± 73.80	579.27 ± 72.56	353.83 ± 42.69	370.01 ± 39.48	0.00 ± 0.00

(c) $C = R = 6000$, $D = 60$

Table 12: Evaluation of the different heuristics on a platform with **Weibull failures of shape parameter 0.7** ($MTBF = 125$ years) under the constant overhead model.

	$g=1$			$g=2$					$g=3$				
p	OptExp	BestPeriod	DPNextFailure	OptExp	OptExpGroup	BestPeriod	DPNextFailure	BestPeriod-Cont	OptExp	OptExpGroup	BestPeriod	DPNextFailure	BestPeriod-Cont
32768	13.10 ± 0.21	12.33 ± 0.07	12.87 ± 0.21	22.87 ± 0.12	23.79 ± 0.40	22.82 ± 0.07	22.71 ± 0.12	22.83 ± 0.08	33.81 ± 0.03	34.12 ± 0.24	33.75 ± 0.06	33.81 ± 0.16	33.85 ± 0.07
65536	7.04 ± 0.25	6.46 ± 0.07	6.64 ± 0.15	11.62 ± 0.08	12.51 ± 0.30	11.57 ± 0.05	11.44 ± 0.06	11.58 ± 0.05	17.01 ± 0.03	17.44 ± 0.28	16.99 ± 0.05	16.98 ± 0.07	17.06 ± 0.05
131072	3.97 ± 0.17	3.49 ± 0.04	3.56 ± 0.07	6.00 ± 0.09	6.83 ± 0.31	5.93 ± 0.04	5.85 ± 0.06	5.94 ± 0.04	8.60 ± 0.03	9.19 ± 0.25	8.60 ± 0.03	8.57 ± 0.04	8.66 ± 0.05
262144	2.38 ± 0.12	1.95 ± 0.06	2.01 ± 0.06	3.18 ± 0.05	3.98 ± 0.21	3.09 ± 0.03	3.04 ± 0.03	3.10 ± 0.03	4.39 ± 0.02	5.13 ± 0.24	4.39 ± 0.02	4.37 ± 0.03	4.45 ± 0.04
524288	1.52 ± 0.14	1.18 ± 0.04	1.24 ± 0.06	1.77 ± 0.07	2.63 ± 0.23	1.66 ± 0.02	1.65 ± 0.03	1.67 ± 0.03	2.30 ± 0.03	3.07 ± 0.24	2.28 ± 0.02	2.27 ± 0.02	2.33 ± 0.03
1048576	1.17 ± 0.15	0.80 ± 0.05	0.89 ± 0.07	1.05 ± 0.08	1.92 ± 0.25	0.94 ± 0.03	0.96 ± 0.04	0.95 ± 0.03	1.25 ± 0.04	2.12 ± 0.25	1.21 ± 0.02	1.23 ± 0.02	1.27 ± 0.03
2097152	1.27 ± 0.19	0.69 ± 0.05	0.80 ± 0.40	0.74 ± 0.06	1.89 ± 0.33	0.59 ± 0.03	0.65 ± 0.04	0.62 ± 0.03	0.75 ± 0.04	1.82 ± 0.31	0.69 ± 0.01	0.72 ± 0.02	0.75 ± 0.03
4194304	2.39 ± 0.41	0.88 ± 0.11	1.75 ± 0.35	0.71 ± 0.11	3.10 ± 0.71	0.45 ± 0.03	0.62 ± 0.05	0.51 ± 0.03	0.53 ± 0.05	2.41 ± 0.46	0.43 ± 0.02	0.52 ± 0.03	0.52 ± 0.02

(a) $C = R = 60$, $D = 60$

	$g=1$			$g=2$					$g=3$				
p	OptExp	BestPeriod	DPNextFailure	OptExp	OptExpGroup	BestPeriod	DPNextFailure	BestPeriod-Cont	OptExp	OptExpGroup	BestPeriod	DPNextFailure	BestPeriod-Cont
32768	19.26 ± 1.21	15.79 ± 0.40	15.94 ± 0.46	25.85 ± 0.65	30.83 ± 1.57	25.05 ± 0.27	24.57 ± 0.32	25.13 ± 0.32	35.47 ± 0.29	39.92 ± 1.77	35.39 ± 0.26	35.15 ± 0.24	36.00 ± 0.26
65536	12.65 ± 1.29	9.41 ± 0.35	9.50 ± 0.38	14.46 ± 0.51	19.33 ± 1.65	13.52 ± 0.26	13.40 ± 0.27	13.67 ± 0.28	18.60 ± 0.36	23.45 ± 1.61	18.42 ± 0.18	18.36 ± 0.14	18.99 ± 0.31
131072	10.02 ± 1.28	6.36 ± 0.35	6.47 ± 0.40	8.95 ± 0.68	14.30 ± 1.84	7.74 ± 0.27	7.87 ± 0.26	7.96 ± 0.28	10.36 ± 0.38	15.82 ± 1.69	9.95 ± 0.14	10.08 ± 0.12	10.57 ± 0.23
262144	10.29 ± 1.67	5.19 ± 0.43	5.35 ± 0.45	6.24 ± 0.66	12.82 ± 2.08	4.82 ± 0.19	5.16 ± 0.25	5.29 ± 0.29	6.27 ± 0.39	12.55 ± 1.88	5.69 ± 0.14	5.93 ± 0.16	6.53 ± 0.29
524288	17.53 ± 3.28	6.17 ± 0.59	6.34 ± 0.61	5.93 ± 0.90	18.73 ± 4.50	3.61 ± 0.21	3.94 ± 0.23	4.82 ± 0.42	4.48 ± 0.38	15.58 ± 3.78	3.61 ± 0.16	3.97 ± 0.16	5.12 ± 0.33
1048576	64.08 ± 11.50	13.90 ± 1.48	14.30 ± 1.64	9.10 ± 1.74	42.61 ± 9.48	3.75 ± 0.34	4.03 ± 0.37	7.81 ± 0.84	4.86 ± 0.60	29.53 ± 8.51	2.81 ± 0.16	3.25 ± 0.20	7.36 ± 0.70
2097152	320.94 ± 39.95	95.82 ± 8.73	98.60 ± 9.13	35.09 ± 7.27	129.07 ± 26.58	7.51 ± 0.84	8.20 ± 0.84	41.20 ± 4.61	10.61 ± 2.09	93.63 ± 17.85	3.34 ± 0.31	3.77 ± 0.36	42.02 ± 4.60
4194304	—	—	—	242.63 ± 31.87	439.07 ± 53.70	55.13 ± 8.00	59.96 ± 5.17	385.97 ± 20.98	56.78 ± 12.03	235.65 ± 37.40	9.89 ± 1.16	11.24 ± 1.25	413.58 ± 24.81

(b) $C = R = 600$, $D = 60$

	$g=1$			$g=2$					$g=3$				
p	OptExp	BestPeriod	DPNextFailure	OptExp	OptExpGroup	BestPeriod	DPNextFailure	BestPeriod-Cont	OptExp	OptExpGroup	BestPeriod	DPNextFailure	BestPeriod-Cont
32768	101.62 ± 15.08	48.86 ± 3.20	49.57 ± 3.13	56.54 ± 4.02	119.49 ± 17.52	42.50 ± 1.50	45.37 ± 1.71	48.63 ± 2.17	53.89 ± 2.97	114.02 ± 13.42	47.92 ± 1.29	49.86 ± 1.25	57.32 ± 2.16
65536	158.69 ± 22.05	62.34 ± 6.56	62.67 ± 6.57	57.14 ± 9.36	157.84 ± 29.28	32.83 ± 2.30	35.40 ± 2.38	49.13 ± 4.58	41.08 ± 4.53	130.61 ± 23.50	31.40 ± 1.60	34.82 ± 1.55	50.66 ± 4.23
131072	333.89 ± 24.28	154.39 ± 14.54	156.07 ± 13.83	93.62 ± 17.95	238.64 ± 45.36	37.13 ± 3.14	39.46 ± 3.02	105.76 ± 10.49	46.33 ± 7.28	223.43 ± 37.62	25.99 ± 1.48	29.81 ± 2.11	99.21 ± 11.44
262144	—	—	—	254.49 ± 35.94	544.74 ± 72.34	91.29 ± 7.23	95.18 ± 8.26	463.68 ± 35.47	97.79 ± 17.00	327.72 ± 34.05	35.99 ± 4.20	38.86 ± 4.62	0.00 ± 0.00
524288	—	—	—	838.67 ± 27.23	1171.23 ± 104.81	383.79 ± 55.84	407.06 ± 45.95	0.00 ± 0.00	351.82 ± 38.34	817.45 ± 110.70	126.72 ± 11.09	132.65 ± 10.10	0.00 ± 0.00

(c) $C = R = 6000$, $D = 60$

Table 13: Evaluation of the different heuristics on a platform with **Weibull failures of shape parameter 0.5** ($MTBF = 125$ years) under the constant overhead model.

RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

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

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

ISSN 0249-6399