Checkpointing strategies to tolerate non-memoryless failures on HPC platforms

Anne Benoit, Lucas Perotin, Yves Robert, Frédéric Vivien

Checkpointing strategies to tolerate non-memoryless failures on HPC platforms

ANNE BENOIT, Laboratoire LIP, ENS Lyon & Inria Lyon, France
LUCAS PEROTIN, Laboratoire LIP, ENS Lyon & Inria Lyon, France
YVES ROBERT*, Laboratoire LIP, ENS Lyon & Inria Lyon, France
FRÉDÉRIC VIVIEN, Laboratoire LIP, ENS Lyon & Inria Lyon, France

This paper studies checkpointing strategies for parallel applications subject to failures. The optimal strategy to minimize total execution time, or makespan, is well known when failure IATs obey an Exponential distribution, but it is unknown for non-memoryless failure distributions. We explain why the latter fact is misunderstood in recent literature. We propose a general strategy that maximizes the expected efficiency until the next failure, and we show that this strategy achieves an asymptotically optimal makespan, thereby establishing the first optimality result for arbitrary failure distributions. Through extensive simulations, we show that the new strategy is always at least as good as the Young/Daly strategy for various failure distributions. For distributions with a high infant mortality (such as LogNormal with shape parameter $k = 2.51$ or Weibull with shape parameter 0.5), the execution time is divided by a factor 1.9 on average, and up to a factor 4.2 for recently deployed platforms.

CCS Concepts: • **Software and its engineering** → *Checkpoint / restart*; • **Computer systems organization** → *Reliability*; • **Hardware** → *System-level fault tolerance*.

Additional Key Words and Phrases: checkpoint, fault-tolerance, failure, non-memoryless, Young/Daly formula

ACM Reference Format:
Anne Benoit, Lucas Perotin, Yves Robert, and Frédéric Vivien. 2023. Checkpointing strategies to tolerate non-memoryless failures on HPC platforms. *ACM Trans. Parallel Comput.* 1, 1 (September 2023), 52 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Checkpoint/restart is the standard technique to protect applications running on High Performance Computing (HPC) platforms. Such platforms experience several failures[1] per day [10, 19, 36, 38] per day. After each failure, the application executing on the faulty processor (and likely on many other processors for a large parallel application) is interrupted and must be restarted. Without checkpointing, all the work executed for the application is lost. With checkpointing, the execution can resume from the last checkpoint, after some downtime (enroll a spare to replace the faulty processor) and a recovery (read the checkpoint).

Consider a parallel application executing on a platform whose processors are subject to failures. How frequently should it be checkpointed so that expected total execution time, or makespan, is

*Also with: University of Tennessee, Knoxville, TN, USA.
[1]Failures are also called *fail-stop errors*

Authors' addresses: Anne Benoit, Laboratoire LIP, ENS Lyon & Inria Lyon, Lyon, France, anne.benoit@ens-lyon.fr; Lucas Perotin, Laboratoire LIP, ENS Lyon & Inria Lyon, Lyon, France, lucas.perotin@ens-lyon.fr; Yves Robert, Laboratoire LIP, ENS Lyon & Inria Lyon, Lyon, France, yves.robert@ens-lyon.fr; Frédéric Vivien, Laboratoire LIP, ENS Lyon & Inria Lyon, Lyon, France, frederic.vivien@inria.fr.

minimized? There is a well-known trade-off: taking too many checkpoints leads to a high overhead, especially when there are few failures, while taking too few checkpoints leads to a large re-execution time after each failure. However, the optimal strategy to minimize the expected makespan is known only when failure inter-arrival times, or IATs for short[2], obey an Exponential distribution on each processor. In that case, the optimal checkpointing period is known and can be expressed with a complicated formula (see Sections 3.1 to 3.3). In practice, the optimal checkpointing period is approximated by the Young/Daly formula as $W_{YD} = \sqrt{2\mu C}$ [13, 47], where μ is the application Mean Time Between Failures (MTBF) and C is the checkpoint duration. The Young/Daly formula is widely used across a variety of applications and platforms (see [7] for a survey) and represents a major progress compared to naive strategies where each application would checkpoint, say, every hour, independently of the values of its MTBF μ and checkpoint duration C.

This paper revisits checkpointing strategies for parallel applications on platforms subject to failures that obey arbitrary probability distributions. This is a very important topic because the most accurate probability distributions to model processor failures are LogNormal [24] and Weibull [35, 36, 41, 42] instead of Exponential. For instance, failure traces from Los Alamos National Laboratory are best fit by Weibull distributions of different shapes [18]. However, dealing with non-memoryless distributions induces dramatic difficulties when aiming at optimality. Indeed, if each processor experiences failures distributed according to some non-memoryless distribution, then the platform as a whole will NOT experience failures distributed according to the same (scaled) distribution. This is because after each failure, only the processor struck by that failure is replaced by a fresh (spare) processor. The other thousands of processors in the platform are not replaced and continue execution; even if we wanted to replace them all by fresh processors, we would not have enough spares for such a massive replacement. Hence, after a few failures have struck the platform, processors have a different history, meaning that the time since the last failure is different from one processor to another. As a consequence, the platform experiences failures which do no longer obey the same probability distribution from one failure to the next. Worse, computing the probability distribution of the time at which the next platform failure will strike would require a massive convolution over all processors. In practice, such a computation is out-of-reach as soon as the number of processors exceeds a few dozens.

The striking difference between the Exponential distribution and any other non-memoryless distribution is further detailed in Sections 3.5 and 3.6. In a nutshell, when each processor experiences failures obeying an Exponential distribution, so does each parallel application (and the whole platform). Take a parallel application with p processors; if processor IATs are $\textsc{Exp}(\lambda)$, then application IATs are $\textsc{Exp}(p\lambda)$; the MTBF for each individual processor is the expectation of $\textsc{Exp}(\lambda)$, namely $\mu_{ind} = \frac{1}{\lambda}$, while the application MTBF is $\mu = \frac{\mu_{ind}}{p}$. The knowledge of the distribution of application IATs is key to derive an analytic expression for the optimal checkpointing period, and to show that the Young/Daly formula is an accurate first-order approximation. But when each processor experiences failures obeying any other distribution, then platform failures are no longer identically distributed, and not much is known about the optimal checkpoint strategy. This does not prevent to use the Young/Daly formula, because it relies on the application MTBF only, not on any probability distribution. The application MTBF can still be computed as $\mu = \frac{\mu_{ind}}{p}$, as shown in Section 3.6. What is completely unknown though is the accuracy of the Young/Daly formula for a non-memoryless distribution.

To the best of our knowledge, this paper is the first to provide a provenly correct strategy for arbitrary distributions. The main contributions are the following:

- A synthetic overview of known results for Exponential distributions, some of which being

[2]IATs are the times elapsed between two consecutive failure events (or until the first failure at the start of the application).

frequently rediscovered;
• A detailed explanation of why non-memoryless distributions require a fully different approach;
• The design of a new checkpointing strategy, NextStep, which is asymptotically optimal for arbitrary distributions;
• A practical and fast implementation of NextStep through time discretization and numerical approximation;
• A detailed experimental comparison with the standard Young/Daly approach.

This work focuses on the classical coordinated checkpointing protocol, which has well-known limitations because of the potential bottleneck incurred when all processors transfer data to stable storage simultaneously [9, 10]. Section 2.3 surveys more complicated (multi-level) approaches designed for large-scale platforms. Future work will aim at extending the NextStep strategy to such approaches.

The paper is organized as follows. We first survey related work in Section 2. Then, we provide background on checkpointing parallel applications with Exponential or non-memoryless distributions in Section 3. We detail the design of the checkpointing strategy NextStep in Section 4, and show that it is asymptotically optimal for arbitrary distributions. The experimental evaluation in Section 5 presents extensive simulation results comparing NextStep and the usual approach à la Young/Daly. Finally, we conclude in Section 6.

2 RELATED WORK

We survey related work related to checkpointing preemptible applications in Section 2.1 and task-based applications in Section 2.2. Section 2.3 is devoted to the presentation of several extensions of the standard approach.

2.1 Checkpointing preemptible parallel applications

Checkpoint-restart is one of the most widely used strategy to deal with failures. Several variants of this policy have been studied; see [25] for an overview. The natural strategy is to checkpoint periodically, and one must decide how often to checkpoint, i.e., derive the optimal checkpointing period. An optimal strategy is defined as a strategy that minimizes the expectation of the execution time of the application. For a preemptible application, where one can checkpoint at any time, the classical formula due to Young [47] and Daly [13] states that the optimal checkpointing period is $W_{YD} = \sqrt{2\mu C}$, where μ is the application MTBF and C the checkpoint cost. This formula is a first-order approximation. For memoryless failures, Daly provides a second-order, more accurate, approximation in [13], while our previous work [8] provides the optimal value; both [13] and [8] use the Lambert function, whose Taylor expansion is key to assess the accuracy of the Young/Daly formula. The derivation in [8] is based on Equation (1) (see Section 3.2), a formula rediscovered ten years later, with a quite different proof based on a Markov model, in [38].

As explained in Section 3.6, non-memoryless failures are more difficult to deal with for parallel applications. Several papers study non-periodic checkpointing strategies, either with a single processor or with total rejuvenation of all processors after a failure [27, 32, 34]. A recent paper [31] also uses full rejuvenation while [41] wrongly assumes independent and identically distributed (IID) failures for a range of classic distributions, including Weibull and LogNormal, which are not memoryless (see Section 3.6). An unorthodox approach is used in [20], where it is assumed that the failures striking the whole platform obey a Weibull distribution; this is misleading for two reasons: (i) it is not clear what is the failure distribution on each individual processor; and (ii) after one processor is struck by a failure and rejuvenated, the platform failure distribution does not remain Weibull (see a more detailed discussion in Section 3.6).

In order to deal with non-memoryless failures, the NextFailure problem is studied in [8], where the goal is to maximize the expected amount of work completed before the next failure. This problem is solved using a dynamic programming algorithm, and it is used as a solution to the initial problem of makespan minimization. Simulations are done with Exponential and Weibull distributions, showing that the proposed algorithm outperforms existing solutions with Weibull distributions. In this paper, we propose to maximize the expected efficiency rather than the expected work, with our new NextStep heuristic. This requires a much more subtle approach, but is key to proving asymptotic optimality.

2.2 Checkpointing task-based applications

Going beyond preemptible applications, some works have studied task-based applications, using a model where checkpointing is only possible right after the completion of a task. The problem is then to determine which tasks should be checkpointed. This problem has been solved for linear workflows (where the task graph is a simple linear chain) by Toueg and Babaoglu [44], using a dynamic programming algorithm. This algorithm was later extended in [6] to cope with both fail-stop and silent errors simultaneously. Another special case is that of a workflow whose dependence graph is arbitrary but whose tasks are parallel tasks that each executes on the whole platform. In other words, the tasks have to be serialized. The problem of ordering the tasks and placing checkpoints is proven NP-complete for simple join graphs in [1], which also introduces several heuristics. For general workflows, deciding which tasks to checkpoint has been shown #P-complete [22], but several heuristics are proposed in [23].

2.3 Extensions: multi-criteria, hierarchical checkpointing, independence

For completeness, in this section we briefly reference several works that go beyond makespan optimization and independent failures. First, other optimization criteria have been considered in the literature. Indeed, I/O is a scarce resource on modern platforms, and several works aim at minimizing I/O volume while enforcing an efficient checkpoint for makespan [26, 28]. Similarly, energy-makespan bi-criteria optimization has been addressed in [18, 21].

Next, to reduce I/O overhead, various two-level checkpointing protocols have been studied [16, 39]. Some authors have also generalized two-level checkpointing to account for an arbitrary number of levels [4, 5, 14, 33].

As for failure independence, the standard model assumes IID failure IATs, on each processor, with a common distribution $\mathcal{D}$. While it is reasonable to assume that IATs are identically distributed on a given processor, because the faulty processor is rejuvenated (replaced by a spare) after each failure, it is very questionable to assume that IATs are independent across the platform. As for *temporal* dependence, it has been observed many times that when a failure occurs, it may trigger other failures that will strike different system components [3, 24, 43]. As an example, a failing cooling system may cause a series of successive crashes of different processors. Also, an outstanding error in the file system will likely be followed by several others [30, 37]. As for *spatial* dependence, it is clear that the overheating of some processor in a cabinet is quite likely to be followed by the overheating of neighbor processors (which comes atop of a temporal dependence as well!) Bautista-Gomez et al. [3] have studied nine systems, and they report periods of high failure density in all of them. They call these periods *cascade failures*. This observation has led them to revisit the temporal failure independence assumption, and to design bi-periodic checkpointing algorithms that use different periods in normal (failure-free) and degraded (with failure cascades) modes. [43] introduces a dynamic strategy called *lazy checkpointing* to adjust to changes in the failure rate. Another approach has been proposed in [2], using quantiles of consecutive IAT pairs.

2.4 Summary

While many checkpointing algorithms have been introduced, most of them have been assessed only experimentally. Optimality results, or results that provides bounds on the accuracy of the various approaches, are dramatically lacking. The single exception is when processor failures obey a memoryless distribution: in this case, the optimal periodic solution is known, and the Young/Daly approach has been proven to be a first-order approximation. However, in real-life, failure distributions on each processor obey Weibull or LogNormal distributions, whose parameters can be estimated from failure log traces. We do assume that the distribution law of failures on each processor is known, but it can be arbitrary. We show that the NextStep strategy outperforms the Young/Daly approach for a wide spectrum of such distribution laws. This work provides the first theoretical result for non-memoryless failure distributions on parallel platforms.

3 BACKGROUND

This section overviews known results for checkpoint strategies. We cover uni-processor and multi-processor applications, either with Exponential failure distributions or with arbitrary failure distributions. Beforehand, we detail the platform and application model.

3.1 Model

3.1.1 Platform and applications. We consider a large parallel platform with m identical processors. We point out that this work is agnostic to the granularity of the computing resources, which can vary from individual cores up to complete multicore nodes equipped with several GPUs. Each of the m processors is subject to failures. A failure, or fail-stop error, interrupts the execution of the processor and provokes the loss of its whole memory. There are many causes of failures, including power outages or network errors, and they cause the node to stall or crash [17, 36]. After each failure, the application executing on the faulty processor (and likely on many other processors for a large parallel application) is interrupted and must be restarted. Without checkpointing, all the work executed for the application is lost. With checkpointing, the execution can resume from the last checkpoint, as detailed below.

We consider parallel applications that can be checkpointed at any time. In scheduling terminology, the applications are preemptible. Consider a parallel application running on several processors: when one of these processors is struck by a failure, the state of the application is lost, and execution must restart from scratch, unless a fault-tolerance mechanism has been deployed. The classical technique to deal with failures makes use of a checkpoint-restart mechanism: the state of the application is periodically checkpointed, i.e., all participating processors take a checkpoint simultaneously. This is the standard coordinated checkpointing protocol, which is routinely used on large-scale platforms [11], where each processor writes its share of application data to stable storage (checkpoint of duration C). When a failure occurs, the platform is unavailable during a downtime D. which is the time to enroll a spare processor that will replace the faulty processor. Typically, a few spares are provisioned by the platform manager, in order to avoid re-scheduling the application [25]. If no spare is available at the time of the failure, the application is put back in the job waiting queue by the batch scheduler, which then re-schedules it with a high priority; this latter scenario is likely to increase the value of D together with its variability [45]. Then all application processors (including the spare, or upon re-scheduling) recover from the last valid checkpoint in a coordinated manner, reading the checkpoint file from stable storage (recovery of duration R). Finally, the execution is resumed from that point on, rather than starting again from scratch.

Note that failures can strike during checkpoint and recovery, but not during downtime (otherwise we can include the downtime in the recovery time). We make no hypothesis on the temporal

occurrence of failures, which may well strike in bursts and simultaneously on several processors. Also, when a failure hits a processor and a spare is available, that processor is replaced by the spare. This amounts to start anew with a fresh processor. In the terminology of stochastic processes, the faulty processor is rejuvenated. However, all the other processors are not rejuvenated: this would be infeasible due to the multitudinous spares needed!

3.1.2 Failures. We assume that each processor experiences failures whose IATs follow IID random variables obeying an arbitrary probability distribution $\mathcal{D}$. We only assume that $\mathcal{D}$ is continuous and of finite expectation and variance, a condition satisfied by all standard distributions. We assume that the distribution $\mathcal{D}$ is known, e.g., from the study of failure traces [18, 24, 35, 36, 42]. As detailed in Section 3.5 and 3.6, the standard approach de facto assumes that $\mathcal{D}$ is Exponential, while non-memoryless behaviors have been observed from the traces.

We let μ_{ind} denote the expectation of $\mathcal{D}$, also known as the individual processor MTBF. Even if each processor has an MTBF of several years, large-scale parallel platforms are composed of so many processors that they will experience several failures per day [10, 19]. Hence, a parallel application using a significant fraction of the platform will typically experience a failure every few hours.

3.1.3 Checkpointing strategy. Given a parallel application with base time T_{base} without checkpoints nor failures, the optimization problem is to decide when and how often to checkpoint in order to minimize the expected execution time of the application. The application base time is divided into N segments of length W_i, $1 \leq i \leq N$, each followed by a checkpoint of length C. Of course $\sum_{i=1}^{N} W_i = T_{base}$. Throughout the paper, we add a final checkpoint at the end of the last segment, e.g., to write final outputs to stable storage. Symmetrically, we add an initial recovery when re-executing the first segment of an application (e.g., to read inputs from stable storage) if it has been struck by a failure before completing the checkpoint. Adding a last checkpoint and an initial recovery brings symmetry and simplifies formulas, but it is not at all mandatory: see Section 3.4 for an extension relaxing either or both assumptions.

3.2 Uni-processor application and Exponential failure distribution

This is the simplest case. Consider an application $\mathcal{A}$ executing on a single processor experiencing failures whose IATs follow an Exponential distribution $\mathcal{D} = \textsc{Exp}(\lambda)$ of parameter $\lambda > 0$, whose probability density function (PDF) is $f(x) = \lambda e^{-\lambda x}$ for $x \geq 0$. The processor MTBF is $\mu_{ind} = \frac{1}{\lambda}$. The optimal checkpointing strategy, i.e., the strategy minimizing the expected execution time, can be derived as shown below.

Lemma 1. *The expected time $\mathbb{E}(W, C, R)$ to execute a segment of W seconds of work followed by a checkpoint of C seconds and with recovery cost R seconds is*

$$\mathbb{E}(W, C, R) = \left(\frac{1}{\lambda} + D\right) e^{\lambda R} \left(e^{\lambda(W+C)} - 1\right). \tag{1}$$

Lemma 1 comes from [8, Theorem 1]. It also applies when the segment is not followed by a checkpoint (take $C = 0$). The *slowdown function* is defined as $f(W, C, R) = \frac{\mathbb{E}(W,C,R)}{W}$. We have the following properties:

Lemma 2. *The slowdown function $W \mapsto f(W, C, R)$ has a unique minimum W_{opt} that does not depend on R, is decreasing in the interval $[0, W_{opt}]$ and is increasing in the interval $[W_{opt}, \infty)$.*

Proof. Again, see [8, Theorem 1]. The exact value of W_{opt} is obtained using the Lambert W function, but a first-order approximation is the Young/Daly formula $W_{YD} = \sqrt{\frac{2C}{\lambda}}$. □

Lemma 2 shows that infinite applications should be partitioned into segments of size W_{opt} followed by a checkpoint. What about finite applications? Back to our application $\mathcal{A}$ of duration T_{base}, we partition it into N segments of length W_i, $1 \leq i \leq N$, each followed by a checkpoint C. By linearity of the expectation, the expected time to execute application $\mathcal{A}$ is

$$\mathbb{E}(J) = \sum_{i=1}^{N} \mathbb{E}(W_i, C, R) = \left(\frac{1}{\lambda} + D\right) e^{\lambda R} \sum_{i=1}^{N} \left(e^{\lambda(W_i+C)} - 1\right),$$

where $\sum_{i=1}^{N} W_i = T_{base}$. By convexity of the Exponential function, or by using Lagrange multipliers, we see that $\mathbb{E}(J)$ is minimized when the W_i's take a constant value, i.e., all segments have same length. Thus, we obtain $W_i = \frac{T_{base}}{N}$ for all i, and we aim at finding N that minimizes

$$\mathbb{E}(J) = N\,\mathbb{E}\left(\frac{T_{base}}{N}, C, R\right) = f\left(\frac{T_{base}}{N}, C, R\right) \times T_{base}\,, \tag{2}$$

where f is the slowdown function. We let $N_{opt} = \frac{T_{base}}{W_{opt}}$, where W_{opt} achieves the minimum of the slowdown function. N_{opt} would be the optimal number of segments if we could have a non-integer number of segments. Lemma 2 shows that the optimal value N_{ME} of N is either $N_{ME} = \max(1, \lfloor N_{opt} \rfloor)$ or $N_{ME} = \lceil N_{opt} \rceil$, whichever leads to the smallest value of $\mathbb{E}(J)$. In the experiments, we use the simplified Young/Daly expression $N_{ME} = \left\lfloor \frac{T_{base}}{W_{YD}} \right\rceil$.

3.3 Parallel application and Exponential failure distribution

Because of the memoryless property of the Exponential distribution, the multi-processor case can be reduced to the uni-processor case. Everything holds by replacing the parameter λ by $p\lambda$, where p is the number of processors of application $\mathcal{A}$. To see this formally, say the application $\mathcal{A}$ is executed on p processors $\{P_q\}_{1 \leq q \leq p}$. Let $X_i^q \sim \textsc{Exp}(\lambda)$, $i \geq 1$, $1 \leq q \leq p$, denote the IID failure IATs on processor P_q. In other words, X_i^q is the random variable for the time between failure number $i-1$ (or the application start if $i = 1$) and failure number i on processor P_q. Let Y_i, $i \geq 1$, denote the failure IATs for (the p processors executing) application $\mathcal{A}$. In other words, Y_i is the random variable for the time between failure number $i-1$ (or the application start if $i = 1$) and failure number i on the whole application.

The assumption $X_i^q \sim \textsc{Exp}(\lambda)$ formally means that when processor P_q is rejuvenated (or when it is used for the first time), the next failure will strike according to a distribution $\textsc{Exp}(\lambda)$. If the application starts at time t, and the last failure struck at time $t_1 < t$ on processor P_q, what is the distribution of the probability of the next failure, knowing that P_q has been alive for $t - t_1$ seconds? The memoryless property of Exponential distributions is the key: it is still the same Exponential distribution. To keep notation simple, we let $X_i^q \sim \textsc{Exp}(\lambda)$, $i \geq 0$, denote the failure IATs on P_q once the application has started (and similarly for Y_i, $i \geq 0$).

First, we have $Y_1 = \min_q(X_1^q)$. Hence $Y_1 \sim \textsc{Exp}(p\lambda)$ (minimum of p $\textsc{Exp}(\lambda)$ distributions). Assume that the first failure for application $\mathcal{A}$ stroke at time t_2 (hence $Y_1 = t_2 - t$) on some processor, say processor P_{q_0}, which is then rejuvenated. Because of the memoryless property, knowing this failure does not change the distribution of the next failure on any other processor, and $Y_2 \sim \textsc{Exp}(p\lambda)$ for the very same reason that $Y_1 \sim \textsc{Exp}(p\lambda)$.

3.4 Extension without final checkpoint nor initial recovery, and $\mathcal{D}$ Exponential

Consider a parallel application $\mathcal{A}$ with p processors, which is partitioned into segments. This section deals with the case where no checkpoint is enforced at the end of the last segment. By symmetry, we also deal with the case where no recovery is paid when re-executing the first segment

after a failure. Let $p\lambda$ denote the failure rate for application $\mathcal{A}$, assuming that application failures obey an Exponential distribution $\mathcal{D} = \text{Exp}(p\lambda)$.

The application is partitioned into N segments of length W_i, with checkpoint cost C_i and recovery cost R_i. Let $C_{tot} = \sum_{i=1}^{N} C_i$ and $R_{tot} = \sum_{i=1}^{N} R_i$. In the model of Sections 3.2 and 3.3, we had $C_i = C$, $R_i = R$ for $1 \leq i \leq N$, $C_{tot} = NC$, and $R_{tot} = NR$. If no checkpoint is taken after the last segment, $C_N = 0$ and $C_{tot} = (N-1)C$. If no recovery is paid when re-executing the first segment, $R_1 = 0$ and $R_{tot} = (N-1)R$.

What is the optimal strategy (number N of segments and length of each segment) to minimize the expected execution time $\mathbb{E}_N$ of the application? From Lemma 1, we have

$$\mathbb{E}_N = \sum_{i=1}^{N} \mathbb{E}(W_i, C_i, R_i) = \left(\frac{1}{p\lambda} + D\right) \sum_{i=1}^{N} e^{p\lambda R_i} \left(e^{p\lambda(W_i+C_i)} - 1\right),$$

where $\sum_{i=1}^{N} W_i = T_{base}$. Given N, $\mathbb{E}_N$ is minimized when the sum $\sum_{i=1}^{N} e^{p\lambda(W_i+C_i+R_i)}$ is minimized. By convexity of the Exponential function, or by using Lagrange multipliers, we see that $\mathbb{E}_N$ is minimized when the $W_i + C_i + R_i$'s take a constant value W_{seg}. This value is given by

$$NW_{seg} = T_{base} + C_{tot} + R_{tot}, \tag{3}$$

and the length W_i of each segment is then computed as $W_i = W_{seg} - C_i - R_i$. If $C_N = 0$, the last segment involves an additional amount C of work duration. Similarly, if $R_1 = 0$, the first segment involves an additional amount R of work duration. Then, we can derive the optimal value of N and W_{seg} as follows: Equation (3) gives $N = \frac{T_{base}-R-C+R_1+C_N}{W_{seg}-R-C}$. Plugging this value into

$$\mathbb{E}_N = \left(\frac{1}{p\lambda} + D\right) \left[(N-1)e^{p\lambda R} + e^{p\lambda R_1} + Ne^{p\lambda W_{seg}}\right]$$

enables to solve for W_{seg}, using the Lambert function in a similar way as in [8].

While the derivation is painful, the conclusion is comforting: in the optimal solution, all segments have same length of work, up to an additional recovery for the first segment and an additional checkpoint for the last one. The Young/Daly approximation still holds, as well as all the results of this paper (whose presentation is much simpler with all segments followed by a checkpoint).

3.5 Uni-processor application and arbitrary failure distribution

When failure IATs obey an arbitrary distribution $\mathcal{D}$, they are still IID, because the processor is rejuvenated (replaced by a spare) after each failure. To the best of our knowledge, even the optimal value W_{opt} for the slowdown function is not known analytically. For some distributions, W_{opt} can be computed numerically, using partial moments for the expectation of the time lost due to failures. But note that W_{opt} does not give the optimal checkpointing period for infinite applications, in contrast to the memoryless case. In fact, the optimal checkpointing strategy is not known for infinite applications, let alone for finite applications.

For instance, consider a Weibull distribution $\mathcal{D} \sim \text{Weibull}(k, \lambda)$ of shape k and scale λ; its PDF is $\mathbb{P}(X = t) = \frac{k}{\lambda}(\frac{t}{\lambda})^{k-1} e^{-(\frac{t}{\lambda})^k}$ for $t > 0$. If $k < 1$, the instantaneous failure rate of $\mathcal{D}$ is decreasing with time (infant mortality), checkpoints should be spaced more and more as time progresses since the last failure. On the contrary, if $k > 1$, the instantaneous failure rate of $\mathcal{D}$ is increasing with time (ageing) and, hence, checkpoints should be spaced less and less. This explains that the optimal checkpointing strategy is aperiodic. See [27, 32, 34] for more details.

3.6 Parallel application and arbitrary failure distribution

When $\mathcal{D}$ is arbitrary, even though the failure IATs X_i^q are IID on each processor, they are not for (the p processors executing) application $\mathcal{A}$. In other words, the Y_i are not IID, unless $\mathcal{D}$ is Exponential. However, owing to the theory on the superposition of renewal processes, whenever $\mathcal{D}$ is continuous and of finite expectation μ_{ind}, we know that the following limit exists, where n is the number of failures:

$$\lim_{n\to\infty} \mathbb{E}\left(\frac{\sum_{i=1}^{n} Y_i}{n}\right) = \frac{\mu_{ind}}{p}. \tag{4}$$

This result is given as Formula 1.4 in [29]. See also [25] for an elementary proof using Wald's equation. Equation (4) is good news because we can define the application MTBF as $\frac{\mu_{ind}}{p}$: in average, a failure will strike the application every $\frac{\mu_{ind}}{p}$ seconds. Note that the MTBF is given a new definition here: the failures striking the parallel application $\mathcal{A}$ are not IID, so there is no longer a mean time before the next failure of the application. Instead, there is a limit on the average time between failures.

At any time, the distribution of the next failure is complicated because it must account for the history of the $p-1$ processors that have not been rejuvenated when the last failure stroke. Indeed, assume that the execution of application $\mathcal{A}$ was started on p fresh processors $\{P_q\}_{1\leq q\leq p}$, and that the last failure stroke on processor P_q at time t_q (where $t_q = 0$ if P_q has never been struck). Let $i(q)$ be the index of the last failure on P_q (where $i(q) = 0$ if P_q has never been struck). To simplify notation, say that the last failure struck processor P_1, meaning that $t_q < t_1$ for $q \geq 2$. Now for $q \geq 2$, the probability that the next failure on P_q strikes at time t (it will be failure number $i(q)+1$ for P_q) is

$$\mathbb{P}(X^q_{i(q)+1} = t | X^q_{i(q)+1} \geq t - t_q).$$

In other words, only $X^1_{i(1)+1}$ follows the distribution $\mathcal{D}$, while each $X^q_{i(q)+1}$, $q \geq 2$, is shifted. To compute the distribution of the next failure of application $\mathcal{A}$, we need to compute the distribution of the minimum of all the $X^q_{i(q)+1}$'s, which are not identical because of their history.

There is a theoretical approach that simplifies the problem, namely rejuvenating all the p processors of the application after each failure (and before starting the execution of the application). Of course, this is impossible in practice when p exceeds a small number, but it is nice from a theory perspective: with total rejuvenation, each failure becomes a renewal point for the whole application, and the failure IATs that strike the application become IID: their distribution is the minimum of p IID distributions $\mathcal{D}$. Even better, there are a few failure distributions $\mathcal{D}$ such that the minimum of p IID instances also obey the same distribution $\mathcal{D}$ (with scaled parameters). For instance, consider a Weibull distribution $\mathcal{D} \sim \textsc{Weibull}(k, \lambda)$ of shape k and scale λ, whose expectation is $\mu_{ind} = \lambda\Gamma(1+\frac{1}{k})$, where Γ denotes the Gamma function $\Gamma(t) = \int_0^\infty x^{t-1}e^{-x}dx$ for $t > 0$. Then the minimum Y of p IID $\textsc{Weibull}(k, \lambda)$ is also a Weibull distribution $Y \sim \textsc{Weibull}(k, \frac{\lambda}{p^{1/k}})$. We observe that the MTBF does not scale linearly with p, unless $k = 1$: the expectation of Y is $\mu = \frac{\mu_{ind}}{p^{1/k}}$. This discussion explains the errors in [20]: (i) the platform cannot obey a Weibull distribution, unless total rejuvenation is used; and (ii) assuming total rejuvenation, the MTBF of an application is not inversely proportional to its number of processors.

A realistic approach to cope with the not-IID problem is to discretize time into small quanta, and to use dynamic programming to compute the best checkpoint strategy for application $\mathcal{A}$ up to the next failure [8]. Obviously, the smaller the quanta, the more accurate the results, but the more costly the dynamic programming algorithm. The approach in [8] greedily uses this strategy from one failure to another, up to the completion of the application. However, optimizing checkpoints up to the next failure, while optimal from one failure to the next (up to the precision of the quanta),

may well be sub-optimal for the whole application. A main contribution of this paper is to introduce a new greedy strategy and to prove an approximation bound for its performance. To the best of our knowledge, this is the first theoretical result for parallel applications with non-memoryless failures.

4 THE NEXTSTEP HEURISTIC

In this section, we present the NextStep heuristic to checkpoint parallel applications under any failure probability distribution. The main idea of NextStep is the following: after each failure, NextStep is able to find the checkpointing strategy that maximizes the expected efficiency (see below) before the next failure or the end of the application. Intuitively, optimizing the expected efficiency on a *failure-by-failure* basis should lead to a good approximation on the optimal solution, at least for large application sizes. One major contribution of this work is to show that NextStep is asymptotically optimal for arbitrary failure distributions.

We first introduce notation in Section 4.1, before formally describing NextStep in Section 4.2. Finally, we prove the asymptotic optimality in Section 4.3.

4.1 Preliminaries

Consider a parallel application $\mathcal{A}$ of length T_{base} executing on p processors, with checkpoint time C. We define one unit of work of the application $\mathcal{A}$ as the parallel work executed within one second, so that we can express application progress either in seconds or in work units, whichever is more natural for the reader. Assume that the application just experienced a failure, and it is ready to resume execution of the remaining W seconds of work (or the application is just starting, and then $W = T_{base}$). For any processor P_j, $1 \leq j \leq p$, let τ_j be the time elapsed since its last failure. In particular, if P_j has been hit by the last failure, then $\tau_j = 0$; note also that we do not assume fresh processors when starting the application. We call $\vec{\tau} = (\tau_1, \tau_2, \ldots, \tau_p)$ the *history* vector.

Given a checkpointing strategy, an application with W remaining work units and a history vector $\vec{\tau}$, the function $first(W|\vec{\tau})$ returns the size W_{first} of the segment preceding the first checkpoint.

Work. Let $\mathcal{W}(W|\vec{\tau})$ be the random variable that quantifies the number of work units successfully executed before the next failure. We have the following recursion:

$$\begin{aligned}\mathcal{W}(0|\vec{\tau}) &= 0\\ \mathcal{W}(W|\vec{\tau}) &= \begin{cases} W_{first} + \mathcal{W}(W - W_{first}|\vec{\tau} + \overrightarrow{W_{first} + C}) \\ \quad \text{if the } p \text{ processors do not fail during} \\ \quad \text{the next } W_1 + C \text{ units of time,} \\ 0 \quad \text{otherwise.} \end{cases}\end{aligned} \tag{5}$$

In Equation (5), given a scalar quantity x, $\overrightarrow{x} = (x, x, \ldots, x)$ denotes the vector with p identical components equal to x. Weighting the two cases in Equation (5) by their probabilities of occurrence, we obtain the expected number of work units successfully computed before the next failure:

$$\mathbb{E}(\mathcal{W}(W|\vec{\tau})) = \mathbb{P}_{suc}(W_{first} + C|\vec{\tau})(W_1 + \mathbb{E}(\mathcal{W}(W - W_{first}|\vec{\tau} + \overrightarrow{W_{first} + C}))), \tag{6}$$

where the probability of success $\mathbb{P}_{suc}$ is computed as

$$\mathbb{P}_{suc}(x|\vec{\tau}) = \prod_{i=1}^{p} \mathbb{P}(X \geq x + \tau_i | X \geq \tau_i). \tag{7}$$

X is a generic random variable following the probability distribution $\mathcal{D}$, the failure inter-arrival time on each processor. Given any such distribution $\mathcal{D}$, $\mathbb{P}_{suc}(x|\vec{\tau})$ can be computed in time $O(p)$.

Efficiency. Rather than focusing solely on the work done, we aim at maximizing the expected efficiency, which also depends on the number of checkpoints that have been taken. This is particularly crucial at the end of the application, where maximizing the number of work units until the next failure may not be the best strategy if the application is about to complete. Indeed, the efficiency also depends on the time spent computing; if no failures occur, it depends on the number of checkpoints that are taken. Hence, we define $\mathbb{E}_W(W, \vec{\tau}, N)$ as the maximum expected number of work units until the next failure (or the completion of the application if no failure occurs) using N checkpoints; similarly, $\mathbb{E}_{T_{next}}(W, \vec{\tau}, N)$ is the expected time until the next failure, or before the completion of the application if no failure occurs. Note that the number N of checkpoints only matters in the latter case where the application has completed.

Finally, if an application still needs to be processed for W units of time, with a history $\vec{\tau}$, we define the maximum possible efficiency among all possible numbers of checkpoints N:

$$\mathbb{E}_e(W, \vec{\tau}) = \max_N \frac{\mathbb{E}_W(W, \vec{\tau}, N)}{\mathbb{E}_{T_{next}}(W, \vec{\tau}, N)}. \tag{8}$$

Time discretization. We introduce a time quantum u, and discretize time into quanta. This means that all quantities (segment sizes, checkpoint and recovery times) are integer multiples of u, and that failures strike at the end of a quantum. More precisely, the probability that a failure happens at the end of quantum i is $\int_{(i-1)u}^{iu} f(x|\vec{\tau})dx$, where $f(x|\vec{\tau})$ is the probability density function of the platform failure distribution $\mathcal{D}$ in the continuous case conditioned by the history. This discretization restricts the search for an optimal execution to a finite set of possible executions. The trade-off is that a smaller value of u leads to a more accurate solution, but also to a higher number of states in the algorithm, hence, to a higher computing time.

In what follows, if a variable y is defined in work units, $y^* = y/u$ is the corresponding number of quanta, which we always suppose integer. For instance, the application size becomes $W^* = W/u$ and the checkpoint size $C^* = C/u$. Similarly, we let $\mathbb{P}^*_{suc}(x^*|\vec{\tau^*}) \triangleq \mathbb{P}_{suc}(ux|u\vec{\tau})$ be the probability that the next x^* quanta succeed given the history $\vec{\tau^*}$, expressed in number of quanta.

4.2 NextStep

We define NextStep formally as: find a function returning the size of the first segment to be checkpointed, such that $\mathbb{E}_e(W, \vec{\tau_0})$ is maximized. Here, $\vec{\tau_0}$ corresponds to the initial history of the platform when the execution starts. Solving NextStep analytically seems out of reach, but the recursive definition of $\mathbb{E}(\mathcal{W}(W|\vec{\tau}))$ (see Equation (6)), together with time discretization, allows us to compute the maximum efficiency. Indeed, there is no need to keep the time elapsed since the last failure of each processor as a parameter of the recursive calls. This is because the τ variables of all processors evolve identically: recursive calls only correspond to cases where no failure has occurred, hence the same quantity is added to the history of each processor. The algorithm is called again each time a failure occurs, to decide where checkpoints should be taken.

Thanks to the discretization, all the $\mathbb{E}_W(W, \vec{\tau_0}, N)$ values can be computed with a time quantum u. We let x^* be the number of quanta that remain to proceed (where initially, $x^* = W^*$). We need to find and store the best solutions for any possible values of x^* and N in the recursive call. Hence, we further consider N_p, the number of checkpoints already taken, and N_f, the number of checkpoints that can still be taken (where $N_p + N_f = N$). This corresponds to Algorithm 1: the *compE* procedure fills a table *solve* that contains, for any triple (x^*, N_p, N_f), the maximum expected work duration until the next failure for these parameters, and the best segment size W^*_{first} that achieves this. For $N_f = 1$, the only possibility is to compute x^* in its entirety and then checkpoint. Otherwise, we try all possible places for the first checkpoint, and recursively call *compE*. If a value with a given (x^*, N_p, N_f) had been computed before, we retrieve the corresponding result line 7.

Algorithm 1: *compE*(x^*, N_p, N_f)

if $x^* = 0$ **then return** 0;
if $N_f = 1$ **then**
  $\vec{\tau^*} \leftarrow \vec{\tau_0^*} + \overrightarrow{W^* - x^* + N_p C^*}$;
  $best \leftarrow x^* \mathbb{P}^*_{suc}(x^* + C^* | \vec{\tau^*})$;
  $solve[x^*][N_p][N_f] \leftarrow (best, x^*)$;
  return *best*;
if $solve[x^*][N_p][N_f] = (best, W^*_{first})$ **then return** *best*;
else
  $best \leftarrow -\infty$;
  $\vec{\tau^*} \leftarrow \vec{\tau_0^*} + \overrightarrow{W^* - x^* + N_p C^*}$;
  for $i = 1$ *to* x^* **do**
    $work \leftarrow compE(x^* - i, N_p + 1, N_f - 1)$;
    $cur \leftarrow \mathbb{P}^*_{suc}(i + C^* | \vec{\tau^*}) \times (i + work)$;
    if $cur > best$ **then**
      $best \leftarrow cur$;
      $W^*_{first} \leftarrow i$;
  $solve[x^*][N_p][N_f] \leftarrow (best, W^*_{first})$;
  return *best*;

Algorithm 2: NextStep (W^*)

/* Compute $\mathbb{E}_{T_{next}}(W^*, \vec{\tau_0^*}, n_c)$ for $n_c \in [1, W^*]$ */
$S \leftarrow 0$;
for $i = 0$ *to* $W^* - 1$ **do** $S \leftarrow S + \mathbb{P}^*_{suc}(i | \vec{\tau_0^*})$;
for $n_c = 1$ *to* W^* **do**
  for $i = 1$ *to* C^* **do**
    $S \leftarrow S + \mathbb{P}^*_{suc}(W^* + (n_c - 1)C^* + i | \vec{\tau_0^*})$;
  $\mathbb{E}_{T_{next}}(W^*, \vec{\tau_0^*}, n_c) \leftarrow S$;

/* Compute $\mathbb{E}_W(W^*, \vec{\tau_0^*}, n_c)$ (array *solve*) */
for $n_c = 1$ *to* W^* **do** $compE(W^*, 0, n_c)$;

/* Solution of NextStep */
$best \leftarrow -\infty$; $N_c \leftarrow 0$; $W^*_{first} \leftarrow 0$;
for $n_c = 1$ *to* W^* **do**
  $cur \leftarrow arg_{first}(solve[W^*][0][n_c]) / \mathbb{E}_{T_{next}}(W^*, \vec{\tau_0^*}, n_c)$;
  $cursegment \leftarrow arg_{second}(solve[W^*][0][n_c])$;
  if $cur > best$ **then**
    $best \leftarrow cur$; $N_c \leftarrow n_c$; $W^*_{first} \leftarrow cursegment$;
return (N_c, W^*_{first});

There remains to compute $\mathbb{E}_{T_{next}}(W^*, \vec{\tau^*}, N)$, i.e., the expected time until next failure or application completion. The following lemma helps us compute these values efficiently with discrete segments:

Lemma 3. *Using discrete quanta of size u, the expectation of the time before the next failure or the completion of the application, expressed in number of quanta, is the following:*

$$\mathbb{E}_{T_{next}}(W^*, \vec{\tau_0^*}, N) = \sum_{i=0}^{W^*+NC^*-1} \mathbb{P}^*_{suc}(i|\vec{\tau_0^*}).$$

Proof. Let X denote the random variable of the number of quanta executed before the next failure (or the completion of the application) given the history $\vec{\tau_0^*}$, the total number of quanta of the application W^* and the number of checkpoints N. Clearly, X is taking integer values in $[1, W^* + NC^*]$, thus

$$\begin{aligned}\mathbb{E}(X) &= \sum_{i=1}^{W^*+NC^*} i\mathbb{P}\{X = i\} = \sum_{i=1}^{W^*+NC^*} \mathbb{P}\{X \geq i\} \\ &= \sum_{i=1}^{W^*+NC^*} \mathbb{P}^*_{suc}(i-1|\vec{\tau_0^*}) = \sum_{i=0}^{W^*+NC^*-1} \mathbb{P}^*_{suc}(i|\vec{\tau_0^*}).\end{aligned}$$

□

From Lemma 3, we derive that $\mathbb{E}_{T_{next}}(W^*, \vec{\tau_0^*}, n_c{+}1) = \mathbb{E}_{T_{next}}(W^*, \vec{\tau_0^*}, n_c) + \sum_{i=W^*+n_cC^*}^{W^*+(n_c+1)C^*-1} \mathbb{P}^*_{suc}(i|\vec{\tau_0^*})$. This is used in Algorithm 2 to compute all the $\mathbb{E}_{T_{next}}$ values more efficiently on lines 1–6. Algorithm 1 is called to fill the *solve* table with all values of $\mathbb{E}_W$ in line 7. We obtain the efficiency $\mathbb{E}_e(W^*, \vec{\tau_0^*})$ for all possible number of checkpoints and keep the maximum, see lines 8–13. Finally, the algorithm returns the values for N and W^*_{first} corresponding to the maximum efficiency, which allows us to rebuild completely the corresponding solution using the table *solve*.

Proposition 1. *Using a time quantum u, and for any failure inter-arrival time distribution, Algorithm 2 computes the solution to NextStep (maximizing efficiency) in time* $O(p(W^*)^4)$.

Proof. The NextStep algorithm consists of three steps. In the first step, it computes all values of $\mathbb{E}_{T_{next}}(W^*, \vec{\tau_0^*}, n_c)$ for $n_c \in [1, W^*]$. To do so, two loops are executed. The first one has W^* steps, where each step computes a single addition. The value of $\mathbb{P}^*_{suc}(W^* + (n_c - 1)C^* + i|\vec{\tau_0^*})$ is the product of the individual probability of failure of the p processors (as in Equation (7)). We assume that the individual probabilities of failure can be computed in $O(1)$, thus the loop takes a time $O(pW^*)$. The second loop is similar, with two nested loops, and its complexity is $O(pW^*C^*)$. We can safely assume that $C^* \leq W^*$, otherwise doing any checkpoint is straightforwardly bad (if we succeed the checkpoint, we would have succeeded the entire application), and the complexity is at most $O(p(W^*)^2)$.

In the second step, the algorithm fills the table *solve* and calls $compE(W^*, 0, n_c)$ for $n_c \in [1, W^*]$. The function $compE(x, y, z)$ fills the table entry corresponding to its parameters if necessary, with eventual recursive calls to *compE* where $y + z$ is constant and x decreases. Given the initial calls with $x = W^*$ and $y + z \in [1, W^*]$ the number of entries written in the table is at most $\frac{W^{*3}}{2}$. To upper bound the overall complexity of this step, we first note that an entry may only be written in the table if the *compE* function is called with the same parameters. In the sub-case $N_f = 1$, this takes few operations and involves a call to $\mathbb{P}^*_{suc}$, thus a time $O(p)$. Otherwise, this means *compE* has been called with parameters corresponding to the last sub-case. If so, a loop is executed $x^* \leq W^*$ times, and each iteration requires a call to *compE*, which either takes time $O(1)$ or fills another entry of the table (therefore the complexity is taken into account for this other entry), and the other operations are in $O(1)$, except the computation of $\mathbb{P}^*_{suc}$ in $O(p)$. Overall, the individual cost of each entry of the table is at most in $O(pW^*)$. Finally the calls to *compE* that do not fill the table may

only be made recursively and were taken into account in the analysis. Given the size of the table in $O((W^*)^3)$, this second step is in $O(p(W^*)^4)$.

Finally, the last step returning the solution consists of a loop with W^* iterations, where each iteration is done in $O(1)$, which gives a complexity in $O(W^*)$. The complexity is dominated by the second step; hence, the result. □

4.3 Asymptotic analysis

We proceed in two steps. First, we show that for an infinite application and for any integer n, the expected work completed by NextStep before the n-th failure (which happens at a random time) is larger than or equal to the one of any other strategy. Then, we show that the expected work processed within T units of time is asymptotically optimal with T (Theorem 1).

4.3.1 Expected work completed before the n-th failure. We compare NextStep to any other strategy for an infinite application that has an infinite number of failures. We assume that the application starts at time $t_0 = 0$. For $k \geq 1$, let t_k be the random variable representing the date of the k-th failure, and for $k \geq 0$, let $\vec{\tau}_k$ be the random variable representing the history of the application at time t_k. Note that neither t_k nor $\vec{\tau}_k$ depend on the checkpointing strategy. For any $n \geq 1$, let WF_n be the random variable corresponding to the expected work executed from the start up to failure n by NextStep, and let OPT_n denote the same variable for an optimal strategy, both depending on the initial history $\vec{\tau}_0$. We show that $\mathbb{E}(WF_n) \geq \mathbb{E}(OPT_n)$.

We define wf_k (resp. opt_k) the random variable representing the work executed by NextStep (resp. an optimal strategy) between times t_{k-1} and t_k. At any decision point, i.e., after each failure, the expected time before the next failure or the completion is in fact the expected time before the next failure, because the application is infinite; hence, this time does not depend upon the number of checkpoints. From Equation (8), we deduce that, for any history, NextStep maximizes the expected work accomplished before the next failure. Hence, we have for any $k \geq 1$ and any possible history $\vec{\tau}$ at the $(k-1)$-th failure (or $\vec{\tau} = \vec{\tau}_0$ if $k = 1$):

$$\mathbb{E}(wf_k | \vec{\tau}_{k-1} = \vec{\tau}) \geq \mathbb{E}(opt_k | \vec{\tau}_{k-1} = \vec{\tau}).$$

This inequality holds for any possible history $\vec{\tau}$, i.e., for any possible event $\{\vec{\tau}_{k-1} = \vec{\tau}\}$, and $\vec{\tau}_{k-1}$ does not depend on the strategy. Therefore, this inequality can be directly extended to $\mathbb{E}(wf_k | \vec{\tau}_{k-1})$ and $\mathbb{E}(opt_k | \vec{\tau}_{k-1})$. Note that these expectations are conditioned by a random variable instead of an event, thus are random variables themselves (constant for $k = 1$ if we consider $\vec{\tau}_0$ as a constant random variable) which always verify $\mathbb{E}(wf_k | \vec{\tau}_{k-1}) \geq \mathbb{E}(opt_k | \vec{\tau}_{k-1})$. In particular:

$$\mathbb{E}(\mathbb{E}(wf_k | \vec{\tau}_{k-1})) \geq \mathbb{E}(\mathbb{E}(opt_k | \vec{\tau}_{k-1})).$$

This result can be combined with the property $\mathbb{E}(X) = \mathbb{E}(\mathbb{E}(X|Y))$ (Law of Total Expectation) whenever both sides exist [40, p. 179] to obtain, for all $k \geq 1$,

$$\mathbb{E}(wf_k) = \mathbb{E}(\mathbb{E}(wf_k | \vec{\tau}_{k-1})) \geq \mathbb{E}(\mathbb{E}(opt_k | \vec{\tau}_{k-1})) = \mathbb{E}(opt_k).$$

Finally, we obtain:

$$\begin{aligned} \mathbb{E}(WF_n) &= \mathbb{E}\left(\sum_{k=1}^{n} wf_k\right) = \sum_{k=1}^{n} \mathbb{E}(wf_k) \\ &\geq \sum_{k=1}^{n} \mathbb{E}(opt_k) = \mathbb{E}\left(\sum_{k=1}^{n} opt_k\right) = \mathbb{E}(OPT_n). \end{aligned}$$

This shows that the expected work completed by NextStep before the n-th failure is larger than or equal to the one of any other strategy.

4.3.2 NextStep is asymptotically optimal. For any T, we show how to define $n(T)$, the index of a failure striking at a time close enough to T, so that the relative work difference performed between T and $t_{n(T)}$ (whichever comes first) is negligible:

Lemma 4. *Let* $n(T) = p\lfloor \frac{T}{\mathbb{E}(X)} \rfloor$, *then* $\lim_{T\to\infty} \frac{\mathbb{E}(|T - t_{n(T)}|)}{T} = 0$.

Proof. Consider an infinitely long application executing on p processors P_i, $1 \le i \le p$. Let X denote the random variable for failure IATs on each processor if there is no history. For $T > 0$, we fix $K(T) = \left\lfloor \frac{T}{\mathbb{E}(X)} \right\rfloor$, thus $n(T) = pK(T)$.

Let $t_{i,j}$ be the random variable representing the time when processor P_i fails for the j-th time. Clearly, for all i and $k > 0$, $t_{i,k+1} - t_{i,k}$ follows the distribution X, because P_i is rejuvenated after each failure. Therefore, $\mathbb{E}(t_{i,j}) = \mathbb{E}(X_{i,0}) + (j-1)\mathbb{E}(X)$, where $\mathbb{E}(X_{i,0})$ depends on the initial state of processor P_i. We then use a variant of the strong law of large numbers [12, Ex. 8 p. 137]: If $(X_1, X_2, \ldots, X_j)$ are identically distributed with finite expectations and $S_j = \sum_{k=1}^{j} X_k$, then $\frac{S_j}{j} \to \mathbb{E}(X_1)$ in L^1, i.e., $\lim_{j\to\infty} \mathbb{E}\left(\left|\frac{S_j}{j} - \mathbb{E}(X_1)\right|\right) = 0$. Applying this result with $X_k = t_{i,k+1} - t_{i,k}$, we obtain $S_{j-1} = t_{i,j} - t_{i,1}$ and

$$\forall i, \lim_{j\to\infty} \mathbb{E}\left(\left|\frac{t_{i,j} - t_{i,1}}{j-1} - \mathbb{E}(X)\right|\right) = 0.$$

For any given j, since the $t_{i,j} - t_{i,1}$'s are identically distributed for all i, we can define a function $\epsilon(j)$ verifying $\lim_{j\to\infty} \epsilon(j) = 0$ and such that, using triangular inequalities:

$$\mathbb{E}(|t_{i,j} - j\mathbb{E}(X)|) \le j\epsilon(j) + \mathbb{E}(X) + t_{i,1}$$

for all i and j. Finally, $\min_{1\le i\le p} t_{i,K(T)} \le t_{n(T)} \le \max_{1\le i\le p} t_{i,K(T)}$, because the total number of failures is the sum of the number of failures of each processor and $n(T) = pK(T)$. Hence,

$$\begin{aligned}
\mathbb{E}(|t_{n(T)} - K(T)\mathbb{E}(X)|) &\le \mathbb{E}(\max_{1\le i\le p}(|t_{i,K(T)} - K(T)\mathbb{E}(X)|)) \\
&\le \mathbb{E}(\sum_{i=1}^{p}(|t_{i,K(T)} - K(T)\mathbb{E}(X)|)) \\
&\le pK(T)\epsilon(K(T)) + p\mathbb{E}(X) + \sum_{i=1}^{p} t_{i,1}.
\end{aligned}$$

By definition, $K(T)\mathbb{E}(X) \le T \le (K(T)+1)\mathbb{E}(X)$, and, because of the triangular inequality, we have:

$$\begin{aligned}
\frac{\mathbb{E}(|T - t_{n(T)}|)}{T} &= \frac{\mathbb{E}(|(T - K(T)\mathbb{E}(X)) + (K(T)\mathbb{E}(X) - t_{n(T)})|)}{T} \\
&\le \frac{\mathbb{E}(X)}{K(T)\mathbb{E}(X)} + \frac{K(T)p\epsilon K(T)}{K(T)\mathbb{E}(X)} + \frac{p\mathbb{E}(X)}{K(T)\mathbb{E}(X)} + \frac{\sum_{i=1}^{p} t_{i,1}}{K(T)\mathbb{E}(X)} \\
&= \frac{1}{K(T)} + \frac{p}{\mathbb{E}(X)}\epsilon(K(T)) + \frac{p}{K(T)} + \frac{\sum_{i=1}^{p} t_{i,1}}{K(T)\mathbb{E}(X)}.
\end{aligned}$$

Here, $p, \mathbb{E}(X)$, and $\sum_{i=1}^{p} t_{i,1}$ are fixed, while $\lim_{T\to\infty} K(T) = \infty$. Hence the result, using $\lim_{K(T)\to\infty} \epsilon(K(T)) = 0$. □

Theorem 1. *For any* T, *with* $n(T) = p\lfloor \frac{T}{\mathbb{E}(X)} \rfloor$, *let* $w^*_{n(T)}$ *be the optimal expected work done up to time* $t_{n(T)}$ *(from the start to the* $n(T)$*-th failure). Then, for any checkpointing strategy, we have* $\frac{\mathbb{E}_W([0,T])}{T} \le \frac{w^*_{n(T)}}{T} + o(1)$. *Furthermore, with NextStep, we have* $\frac{\mathbb{E}_W([0,T])}{T} \ge \frac{w^*_{n(T)}}{T} - o(1)$. *Hence, NextStep is asymptotically optimal.*

Proof. Assuming $\mathbb{E}_W([a,b]) = 0$ if $a > b$, thanks to Lemma 4, we obtain that, for any strategy,

$$\frac{\mathbb{E}_W(T)}{T} \leq \frac{\mathbb{E}_W([0,t_{n(T)}])}{T} + \frac{\mathbb{E}_W([t_{n(T)},T])}{T}$$
$$\leq \frac{w^*_{n(T)}}{T} + \frac{\mathbb{E}(|T - t_{n(T)}|)}{T} = \frac{w^*_{n(T)}}{T} + o(1).$$

Furthermore, for NextStep, we have:

$$\frac{\mathbb{E}_W(T)}{T} \geq \frac{\mathbb{E}_W([0,t_{n(T)}])}{T} - \frac{\mathbb{E}_W([T,t_{n(T)}])}{T}$$
$$\geq \frac{w^*_{n(T)}}{T} - \frac{\mathbb{E}(|T - t_{n(T)}|)}{T} = \frac{w^*_{n(T)}}{T} - o(1),$$

which concludes the proof. □

4.3.3 Counter-example to optimality. The NextStep heuristic is asymptotically optimal but not always optimal. This is because, for short applications, maximizing the efficiency until the next failure is not exactly equivalent to minimizing the makespan. Appendix A, gives an example where NextStep is not optimal, for an Exponential distribution. The example is designed as a worst-case scenario and shows that the number of checkpoints may differ between NextStep and the optimal.

4.3.4 A note on the optimal solution for an Exponential distribution. Sections 3.2 and 3.3 have shown how to statically compute the optimal strategy to minimize the expected makespan of an application when the failures obey an Exponential distribution. This optimal strategy is *static*, meaning that we compute the number and length of the application segments once and for all, before starting the execution. On the contrary, the NextStep strategy is dynamic, since it is called after each failure. One may envision a dynamic version of the optimal static strategy, where one would recompute the number and length of the application segments after each failure (and maybe after each checkpoint too), as a function of the remaining size of the application. Section B of the appendix shows that this dynamic approach is identical to the static one. This new result demonstrates the fairness of comparing NextStep with a static approach.

5 PERFORMANCE EVALUATION

In this section, we evaluate and compare the performance of NextStep with the Young/Daly periodic checkpointing heuristic, using simulations on synthetic applications with various parameters, and subject to failures that are sampled from a wide range of probability distributions. Section 5.1 details application parameters and failure distributions. Section 5.2 presents all simulation results.

To the best of our knowledge, the Young/Daly periodic checkpointing heuristic is the only competitor to our new approach. Indeed, we deal with parallel applications, arbitrary failure probability distributions, and after a failure we only rejuvenate the faulty processor. While some strategies have been derived for Weibull probability distributions [27, 32, 34], they either assume a single processor or total rejuvenation, so that application IATs always obey a scaled Weibull distribution. As already mentioned, total rejuvenation is not an option for HPC platforms.

5.1 Simulation Setup

Algorithms. We compare the performance of NextStep with YoungDaly, the Young/Daly periodic checkpointing strategy (see Sections 3.2 and 3.3). For YoungDaly, an application $\mathcal{A}$ of length T_{base} and using p processors is divided into $N_{ME} = \left\lceil \frac{T_{base}}{W_{YD}} \right\rceil$ equal-size segments, each followed by a checkpoint. Here, $W_{YD} = \sqrt{\frac{2C\mu_{ind}}{p}}$. By default, we use $\mu_{ind} = 10$ years in the simulations.

Because YoungDaly is a *periodic* strategy, the size of its checkpointed segments are defined once and for all. On the contrary, NextStep adapts its checkpointing strategy to the failure history. Hence, after each failure, NextStep must recompute the size of its checkpointed segments. We take this recomputation time into account in the simulation, and conservatively add it to the recovery time[3]. To keep the recomputation time as low as possible, we introduce two optimizations to Algorithm 2. The goal is to dramatically shorten its execution time while maintaining the quality of the produced solution.

The first optimization is about the loop at Line 9. In Algorithm 2, the loop is over all possible numbers of checkpoints, ranging from a single checkpoint to one checkpoint per time quantum (i.e., W^* checkpoints). This latter solution would lead to a huge number of checkpoints. A natural conjecture is that the expected performance of NextStep would be a bell-shaped function of the number of checkpoints that are taken, first increasing and then decreasing after a threshold number has been reached. Therefore, in our implementation, we have replaced the `for` loop at Line 9 by a `while` loop that continues to look for a solution involving one additional checkpoint only if at least one of the five prior attempts leads to the best solution overall.

The second optimization is about the computation of the probability of success in Algorithm 1, Lines 4 and 13; and Algorithm 2, Line 2 and 5. This probability is the product of the individual probabilities of the processors. Hence, the execution time of this step is linear in the number of processors, while we want to consider platforms with tens of thousands of processors. Furthermore, this probability is computed many times, for many different values of i and n_c. We replace the exact computation by the following approximation. We first sort the values in τ_0^* and we retain the smallest ten and largest ten values; then we approximate the remaining values using 100 quantiles, according to the distribution. When i and n_c vary, they add an additive term to the history, which does not change the ranking of the values. We can thus replace the exact computation by one that uses the 10 smallest and 10 largest values of the history, and the 100 quantiles with their frequency of occurrences (if there are k values for a quantile, we compute a single probability and its exponentiation rather than k probabilities that we multiply). Hence, for a pre-processing cost of $O(p \log p)$ we approximate in constant time the probability of success, since the processors that define the 120 history values remain the same up to the next failure.

Probability Distributions. We experiment with a wide range of probability distributions:

• **The Exponential distribution**, with probability density function $f(t) = \frac{e^{-t/\mu_{ind}}}{\mu_{ind}}$;

• **The Weibull distribution**, with probability density function $f(t;k,\lambda) = \frac{k}{\lambda}\left(\frac{t}{\lambda}\right)^{k-1} e^{-(t/\lambda)^k}$, where k is the shape parameter and λ is the scale parameter. To obtain an MTBF of μ_{ind}, we chose $\lambda = \frac{\mu_{ind}}{\Gamma(1+\frac{1}{k})}$, and therefore the probability density function becomes $f(t,k) = \frac{k\Gamma(1+\frac{1}{k})}{\mu_{ind}}\left(\frac{t\Gamma(1+\frac{1}{k})}{\mu_{ind}}\right)^{k-1} e^{-\left(\frac{t\Gamma(1+\frac{1}{k})}{\mu_{ind}}\right)^k}$. In the experiments, k is varied in $\{0.5, 0.7, 1.5\}$. The first two shape values are realistic values taken from [18, 35, 36]; for $k < 1$, processors are more likely to fail if the processor is recent (infant mortality). The last shape value $k = 1.5$ provides an example of a distribution whose instantaneous failure rate increases with time. Note that $k = 1$ corresponds to the Exponential distribution;

• **The Gamma distribution**, with probability density function $f(t,k,\Theta) = \frac{t^{k-1}e^{-\frac{t}{\Theta}}}{\Gamma(k)\Theta^k}$, where k is the shape parameter and Θ is the scale parameter. To obtain an MTBF of μ_{ind}, we scale it using $\Theta = \frac{\mu_{ind}}{k}$ and obtain $f(t,k) = \frac{k^k t^{k-1} e^{-\frac{kt}{\mu_{ind}}}}{\Gamma(k)\mu_{ind}^k}$, where k is the shape parameter and Γ is the Gamma function. In

[3]An alternative would be to perform this recomputation on dedicated resources, and in parallel to the recovery. We study the most costly scenario for NextStep.

the experiments, k is varied in $\{0.5, 0.7\}$. Note that $k = 1$ corresponds again to the Exponential distribution;

• **The LogNormal distribution**, with probability density function $f(t, \mu, \sigma) = \frac{1}{t\sigma\sqrt{2\pi}} e^{\left(-\frac{(\ln t - \mu)^2}{2\sigma^2}\right)}$ and with expectation $e^{\mu+\sigma^2/2}$, where μ and σ are respectively the expectation and the standard deviation of the variable's natural logarithm. We tested with two sets of (μ, σ), used in [15] and [46]: $(\mu_1 = 6.6025, \sigma_1 = 1.6206)$ and $(\mu_2 = 10.89, \sigma_2 = 1.08)$. In order to harmonize with the other probability distributions, we aim at having $\mu_{ind} = e^{\mu+\sigma^2/2} = 10$ years. To achieve this without altering the shape of the probability distribution, we fix a parameter $k = \mu/\sigma^2$, in order to express the probability density function with (μ_{ind}, k). After scaling, we obtain two sets $(\mu_{ind} = 10, k = 2.51)$ and $(\mu_{ind} = 10, k = 9.34)$ that we consider in the experiments. We retrieve the probability density function with:

$$\mu = \frac{\ln(\mu_{ind})}{1 + \frac{1}{2k}}; \qquad \sigma = \sqrt{\frac{\mu}{k}} = \sqrt{\frac{\ln(\mu_{ind})}{k + \frac{k}{2}}}.$$

Traces. We generate a failure trace for each failure distribution and for each processor. In that trace, failure IATs obey the distribution, and the last failure happens after time h, where h is the horizon of the failure trace. The horizon is set to two years ($h = 730$ days) for all the traces. The different heuristics are then evaluated using the trace, thereby making sure that all heuristics are evaluated using the very same failure scenario. If during a simulation, a checkpointing strategy reaches time h before the completion of the application, the simulation is said to fail.

Simulation Parameters. In the experiments, we compare both checkpointing strategies under the following parameter settings:

• The number of processors p, logarithmically varied in the range 10^3 to 10^5. These values represent mid-size to large parallel platforms.

• The checkpoint/recovery/downtime $C = R = 10D$, in seconds, varied in $\{60, 600\}$. In practice, the small value of C is optimistic while the later is pessimistic; and the low value of $D = \frac{C}{10}$ assumes that spares are immediately available.

• The duration of the application T_{base}, in hours, varied in $\{1, 3, 10, 48\}$. T_{base} corresponds to the total length of the application, excluding checkpoints, if no failure occurs, when it is run on p processors (weak scaling). This corresponds to the duration range of typical HPC applications, lasting from one hour up to two days.

• The age of the platform T_{plat}, in days, varied in $\{0, 10, 30, 100, 365\}$. This is the time from which we start using the failure traces: either from their very beginning if $T_{plat} = 0$, or from a later time if $T_{plat} > 0$. The age of the platform plays an important role for non-memoryless failure distributions. At the creation of the platform, all the processors are new and without any failure history. After a failure, the processor that failed is replaced/rejuvenated, but the other processors are not and keep their history. For instance, if the processors experience infant mortality, we expect the number of failures to be much higher with $T_{plat} = 0$, when all processors are new, than after a year of service ($T_{plat} = 365$).

• The duration of the quantum is set to the platform MTBF divided by 300. This holds unless the time needed to complete the remaining work and checkpoint it, becomes shorter than the MTBF; in that latter case, the quantum is set to the remaining completion time without failure divided by 300.

Evaluation Methodology. For each possible choice of parameters, we generate 50 different failure scenarios. For each failure scenario, the simulated makespan (duration of the whole execution) of both heuristics is computed. We include the time spent to compute the segment sizes of NextStep.

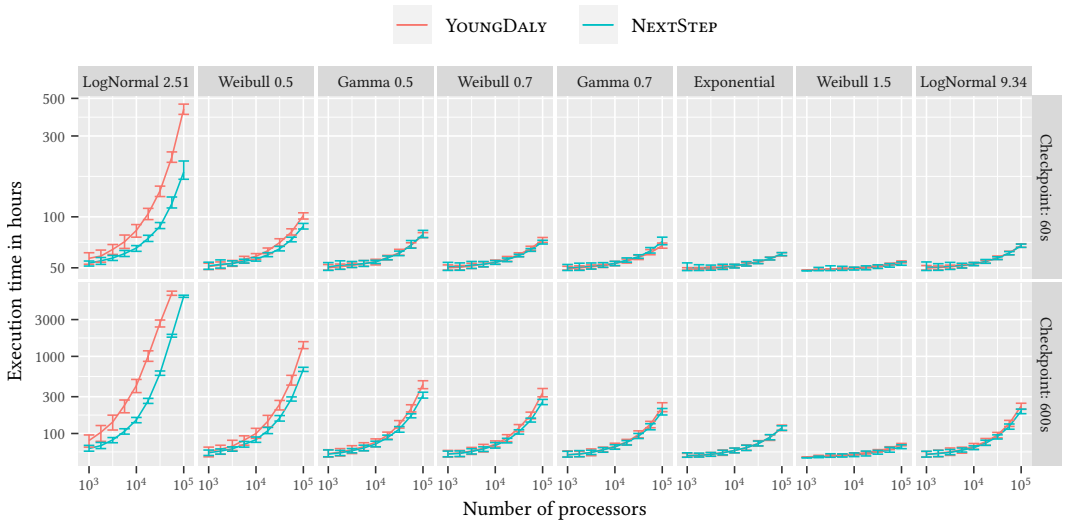

Fig. 1. Expected performance of both heuristics under all failure distributions for a 100 days old platform with a workflow of T_{base} = 48 hours.

Table 1. Ratio of the execution time achieved by YoungDaly to that of NextStep for the 8 failure distributions, when T_{base} = 48 and T_{plat} = 100, and when aggregating all results.

	LogNormal 2.51	Weibull 0.5	Gamma 0.5	Weibull 0.7	Gamma 0.7	Exponential	Weibull 1.5	LogNormal 9.34
$T_{base}=48$, $T_{plat}=100$	1.89 (2.02)	1.15 (1.34)	1.04 (1.17)	1.04 (1.14)	1 (1.1)	1.01 (1.06)	1.03 (1.06)	1.02 (1.11)
Aggregated	2.48 (2.26)	1.44 (1.6)	1.24 (1.43)	1.13 (1.28)	1.07 (1.21)	1.01 (1.07)	1.04 (1.07)	1.03 (1.09)

On the plots, we report the average makespan over these 50 instances, together with the tenth and the ninetieth percentiles, as a function of the number of processors. The YoungDaly heuristic is shown in red, and NextStep in blue. In all figures, the y-axis is the makespan in hours, and the x-axis corresponds in most cases to the number of processors; both axes are in log-scale.

In the tables, we report the relative performance of YoungDaly and NextStep. More precisely, for each failure scenario, we compute the ratio of the makespan achieved by YoungDaly divided by that of NextStep. Hence, NextStep achieves a better makespan when the ratio is greater than 1; the larger the ratio, the higher the benefit of using NextStep. To produce meaningful statistics on these ratios, we compute and report their geometric mean and geometric standard deviation (in parentheses). For a few configurations, YoungDaly does not succeed to complete the application before the trace horizon. For these cases, in order to be able to compute statistics, we take for the execution time of YoungDaly a lower bound, namely the time at which the execution was stopped: $h - T_{plat}$. We checked that using this lower-bound or computing the statistics while just discarding these configurations leads to almost identical results (differences below 1%). The simulation code for all experiments is publicly available at http://perso.ens-lyon.fr/frederic.vivien/resilience/non-memoryless-checkpoint.

5.2 Results

We first compare the behavior of both checkpointing heuristics with the different probability distributions on a particular set of parameters, before studying the impact of the different parameters.

Table 2. Ratio of the execution time achieved by YoungDaly to that of NextStep for the 8 failure distributions for the different number of processors and when averaging over all the other parameters.

Platform size	LogNormal 2.51		Weibull 0.5		Gamma 0.5		Weibull 0.7		Gamma 0.7		Exponential		Weibull 1.5		LogNormal 9.34	
1000	1.34	(1.6)	1.14	(1.33)	1.08	(1.22)	1.03	(1.11)	1.01	(1.08)	1	(1.04)	1.01	(1.04)	1.01	(1.04)
1778	1.53	(1.82)	1.18	(1.41)	1.11	(1.3)	1.03	(1.12)	1.02	(1.1)	1	(1.04)	1.01	(1.04)	1.01	(1.04)
3162	1.88	(2.07)	1.26	(1.49)	1.16	(1.37)	1.05	(1.16)	1.02	(1.13)	1.01	(1.04)	1.02	(1.04)	1.01	(1.05)
5623	2.22	(2.28)	1.33	(1.54)	1.19	(1.38)	1.07	(1.19)	1.04	(1.14)	1.01	(1.06)	1.03	(1.04)	1.02	(1.06)
10000	2.72	(2.33)	1.38	(1.58)	1.23	(1.42)	1.09	(1.24)	1.06	(1.18)	1.01	(1.05)	1.03	(1.05)	1.02	(1.06)
17783	3.2	(2.33)	1.51	(1.66)	1.26	(1.44)	1.14	(1.29)	1.08	(1.22)	1.01	(1.07)	1.04	(1.06)	1.03	(1.07)
31623	3.74	(2.18)	1.72	(1.72)	1.36	(1.5)	1.22	(1.36)	1.12	(1.27)	1.02	(1.08)	1.07	(1.08)	1.05	(1.12)
56234	3.65	(2.02)	1.76	(1.63)	1.37	(1.5)	1.24	(1.35)	1.14	(1.3)	1.01	(1.1)	1.08	(1.09)	1.05	(1.13)
100000	3.5	(1.92)	1.85	(1.55)	1.41	(1.48)	1.33	(1.41)	1.15	(1.32)	1.01	(1.09)	1.08	(1.1)	1.07	(1.16)

Table 3. Ratio of the execution time achieved by YoungDaly to that of NextStep for the 8 failure distributions as a function of platform age, with $p = 56234$ and $T_{base} = 48$ and when averaging over the two checkpoint sizes. The last column provides an average over all distributions.

Platform age	LogNormal 2.51		Weibull 0.5		Gamma 0.5		Weibull 0.7		Gamma 0.7		Exponential		Weibull 1.5		LogNormal 9.34		**Average**	
0	4.17	(2.06)	2.33	(1.48)	1.85	(1.44)	1.42	(1.38)	1.28	(1.32)	1.03	(1.08)	1.08	(1.07)	1.08	(1.07)	**1.58**	(1.78)
10	3.27	(2.26)	1.61	(1.66)	1.29	(1.46)	1.13	(1.29)	1.06	(1.2)	1.01	(1.06)	1.04	(1.06)	1.02	(1.06)	**1.31**	(1.71)
30	2.57	(2.17)	1.36	(1.55)	1.15	(1.32)	1.08	(1.21)	1.03	(1.16)	1	(1.06)	1.03	(1.06)	1	(1.06)	**1.21**	(1.58)
100	1.89	(2.02)	1.15	(1.34)	1.04	(1.17)	1.04	(1.14)	1	(1.1)	1.01	(1.06)	1.03	(1.06)	1.02	(1.11)	**1.12**	(1.42)
365	1.42	(1.72)	1.05	(1.17)	1.01	(1.1)	1.02	(1.11)	1	(1.08)	1	(1.06)	1.01	(1.07)	1.03	(1.13)	**1.06**	(1.27)

Only a selection of results is presented here, but exhaustive results for all combinations of parameters can be found in Section C of the appendix.

5.2.1 Comparison of Probability Distributions. Figure 1 compares the two heuristics for the different failure distributions, with a checkpoint length of one or ten minutes, where the application base time is 48 hours and the platform is 100 days old. In this case, although the platform is not new, we see that the NextStep heuristic is performing either better than or similarly to YoungDaly. Moreover, the difference tends to be more important when the checkpoint length is higher (bottom graphs). Recall that lower is better, since we plot execution times.

Although the MTBF of any individual processor is the same for all failure distributions ($\mu_{ind} = 10$ years), the shape of these distributions significantly impacts the number of failures that occur during the processing of the application, as well as the distribution of the failures. For instance, if the processors tend to have infant mortality (which corresponds to distributions on the left of the figure), and if the platform is not very old, then applications may actually experience more failures than expected. This is the case for the LogNormal distribution with $k = 2.51$ or Weibull with $k = 0.5$ in Figure 1. This explains the higher execution times of YoungDaly for both heuristics.

Furthermore, YoungDaly does not checkpoint often enough, as it considers the global long term MTBF of the platform instead of its actual instantaneous failure rate. This is because YoungDaly does not take the failure history into account. On the contrary, NextStep does take that history into account. Therefore, it correctly estimates the instantaneous failure rate. This results in a makespan that can be up to two times lower.

There are some distributions for which processors tend to be more robust at the beginning because of their young age (distributions on the right of the figure). In this case, when the platform is rather young, the number of failures is lower than what would be expected regarding the MTBF of the platform. A good example is the Weibull distribution with $k = 1.5$ in Figure 1. In that case, the actual instantaneous failure rate of the platform is lower than expected, YoungDaly tends to over-checkpoint because it does not take into account this actual failure rate, whereas NextStep

Table 4. Ratio of the execution time achieved by YoungDaly to that of NextStep for the 8 failure distributions for the two checkpoint durations (in seconds) when averaging over all the other parameters.

Checkpoint duration	LogNormal 2.51		Weibull 0.5		Gamma 0.5		Weibull 0.7		Gamma 0.7		Exponential		Weibull 1.5		LogNormal 9.34	
60	2.18	(2.27)	1.33	(1.52)	1.17	(1.37)	1.08	(1.2)	1.03	(1.14)	1	(1.03)	1.02	(1.04)	1.01	(1.04)
600	2.83	(2.2)	1.56	(1.66)	1.3	(1.47)	1.18	(1.34)	1.11	(1.26)	1.02	(1.09)	1.06	(1.08)	1.05	(1.12)

adapts its checkpointing strategy according to this history, showing once again its versatility. Yet this time, the difference between heuristics is low, because the overall checkpointing cost remains small in both cases.

The number of failures per simulation varies a lot. For instance, focusing on the case of a 100 days old platform including 10^5 processors with a workflow of $T_{base} = 48$ hours, the number of failures ranges between 7 and 7911 failures. The mean number of failures for a given distribution ranges from 13.3 for Weibull 1.5 up to 4883 for LogNormal 2.51. In contexts where very few failures occur (including cases with no failures), one single additional failure, or one single lacking failure, may have a very significant impact on the execution time. Hence, the high standard deviations that are observed.

Finally, if the platform actual instantaneous failure rate is in accordance with the expected MTBF, as is the case for an Exponential distribution, YoungDaly is optimal. We check that the performance of NextStep and YoungDaly are similar in this setting.

Altogether, these results show that NextStep always adapts to the actual instantaneous failure rate, because it accounts for the failure history of processors. Its versatility makes it a better strategy in all the cases: Table 1 summarizes the results, reporting the ratio of the execution time achieved by YoungDaly to that of NextStep (geometric average, geometric standard deviation). We point out that the difference is more significant for the realistic probability distributions that have been advocated in the literature: namely Weibull with a shape parameter smaller than one [18, 35, 36], and LogNormal [15, 46].

The previous study was for long applications lasting 48 hours and $T_{plat} = 100$. We also present the aggregated results over all application base times and platform ages in Table 1. We observe that NextStep achieves even larger gains. For instance, for LogNormal 2.51, the average ratio becomes 2.48, instead of 1.89 for the scenario with $T_{plat} = 100$.

5.2.2 Impact of the Different Parameters. **Impact of the number of processors.** It can be observed on Figure 1: the more processors, the more failures, and the larger the makespan for both heuristics, as one could have foretold. In most settings, the performance of YoungDaly worsens relatively to that of NextStep when the number of processors increases (recall that the y-axis is in log-scale). Again, this can be explained as follows: the difference between the estimated failure rate and the instantaneous failure rate increases with the number of processors; hence, worse results for YoungDaly. On the contrary, NextStep adapts to the instantaneous failure rate. Table 2 provides a comprehensive summary of results for each number of processors, averaging over all other parameters. The table confirms this observation.

Impact of the age of the platform. The age of the platform has a great impact on the performance of both heuristics, because the instantaneous failure rate of the platform highly depends on it. When processors have a high infant mortality, a younger platform leads to more errors and thus to a higher makespan for both heuristics. This can be observed in Figure 2, especially on the leftmost graphs. On this figure, the x-axis is now the age of the platform (in a linear scale). The number of processors is fixed to $p = 56234$ and the application execution time is $T_{base} = 48$ hours.

For all distributions to the left of the Exponential, the newer the platform, the higher the difference between the heuristics. Indeed, for younger platforms, processors are more likely to fail due to

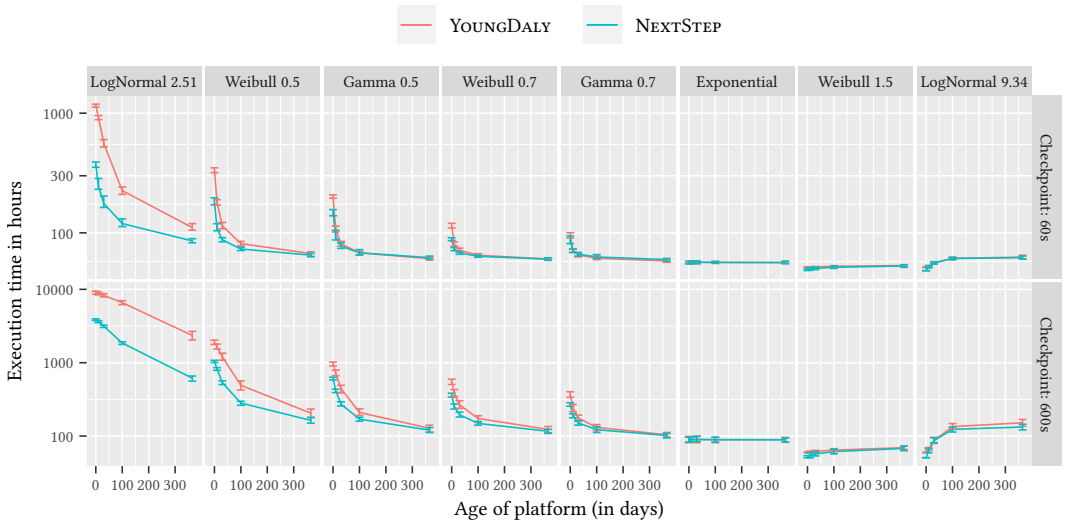

Fig. 2. Expected performance of both heuristics under all failure distributions, with $p = 56234$ and $T_{base} = 48$ hours.

infant mortality; and the older the platform, the more the instantaneous failure rate resembles an Exponential. The same observation can be made for Weibull 1.5, although in this case, this is due to low infant mortality. Indeed, with this distribution, less failures occur for younger platforms.

LogNormal 9.34 behaves slightly differently. For this distribution, once again, YoungDaly does not adapt to the instantaneous failure rate: either it underestimates the instantaneous failure rate of a new platform and does not checkpoint enough, or it overestimates the instantaneous failure rate of an old platform and checkpoints too much. On the contrary, NextStep adjusts the checkpointing strategy for both cases. For intermediate platform ages, both heuristics have close performance because this is where the instantaneous failure rate is closest to what is expected ($\frac{\mu_{ind}}{p}$) by YoungDaly. Nevertheless, the variance is different from that of the Exponential distribution, and NextStep achieves slightly better performance.

Table 3 summarizes these results. NextStep always achieves a performance at least similar to YoungDaly, and much better in many cases. As discussed above, most gains are obtained for young platforms, because Young/Daly either underestimates or overestimates the instantaneous failure rate of the processors, while NextStep nicely adjusts to the rigth checkpoint frequency. When the platform ages a few years, its processors will have diverse histories depending upon when they have failed; then NextStep and Young/Daly become closer to each other, since the potential fact that young processors could be more likely to fail (for example) is balanced by the presence of middle-aged and older processors, resulting in a global distribution closer to Exponential platform-wise.

Impact of the checkpoint time. As expected, the larger the checkpoint cost, the larger the execution time for both heuristics, as shown in Figure 1. Having a larger checkpoint cost exacerbates the differences between both heuristics. Indeed, when checkpoints cost more, both heuristics execute fewer checkpoints and thus lose more time at each failure. In the end, this increases the performance

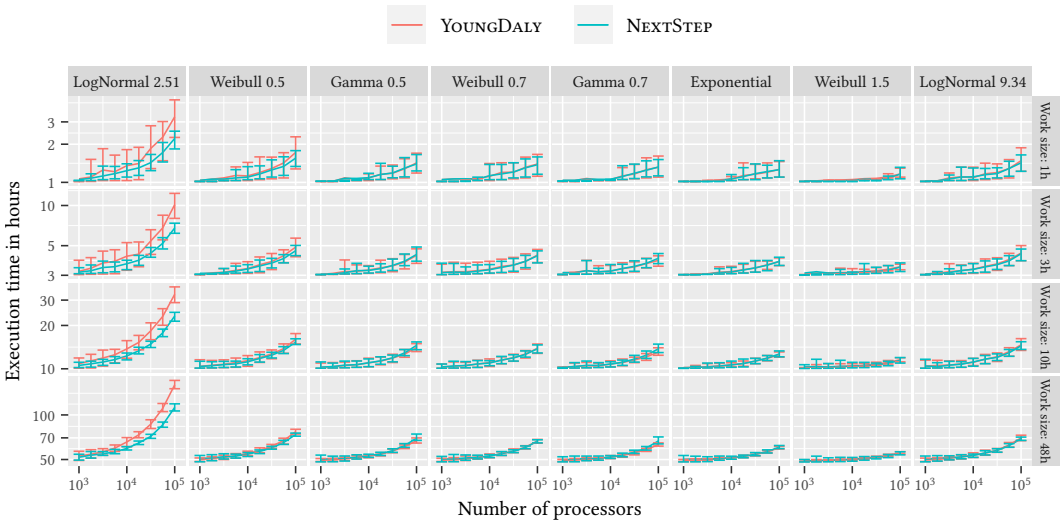

Fig. 3. Expected performance of both heuristics under all failure distributions on a 365 day old platform, with $C = 60$ seconds.

Table 5. Ratio of the execution time achieved by YoungDaly to that of NextStep for the 8 failure distributions for the different application base times (in hours) and when averaging over all the other parameters.

application length	LogNormal 2.51	Weibull 0.5	Gamma 0.5	Weibull 0.7	Gamma 0.7	Exponential	Weibull 1.5	LogNormal 9.34
1	2.36 (2.84)	1.42 (1.81)	1.25 (1.58)	1.12 (1.39)	1.07 (1.29)	1.02 (1.09)	1.05 (1.08)	1.04 (1.12)
3	2.83 (2.47)	1.52 (1.71)	1.29 (1.49)	1.15 (1.32)	1.09 (1.25)	1.01 (1.07)	1.04 (1.08)	1.03 (1.1)
10	2.63 (2)	1.47 (1.51)	1.25 (1.35)	1.14 (1.23)	1.07 (1.17)	1 (1.05)	1.04 (1.06)	1.03 (1.08)
48	2.16 (1.6)	1.35 (1.32)	1.16 (1.21)	1.11 (1.15)	1.05 (1.11)	0.997 (1.03)	1.03 (1.05)	1.02 (1.06)

loss due to a bad checkpointing strategy. Table 4 summarizes the results for the two checkpoint costs (one minute or ten minutes).

Impact of the application base time. Again, the larger the application base time, the larger the execution time for both heuristics, as shown in Figure 3. Moreover, the error bars are much wider for a small workload, because having larger applications will smooth the impact of each individual failure. Table 5 summarizes results for the four application base times by aggregating all results. Overall, more gain can be achieved with smaller application lengths. Indeed, relative to the lengths of applications, checkpoints are more expensive for small applications. This conclusion is similar to that on the impact of the cost of checkpoints. This phenomenon can be observed by comparing Tables 4 and 5, where the impact of increasing the checkpoint cost is similar to the impact of decreasing the application base time.

5.2.3 Summary. Here are four key findings from the whole simulation campaign:

- The impact of NextStep gets more significant for larger and longer applications, because the MTBF decreases when the number of processors increases, and the number of failures increases when the base time increases;
- The impact of NextStep is very high for young platforms, because it nicely adapts its checkpoint frequency to the actual instantaneous failure rate of the processors;

- The parameters of the distribution laws tend to have a stronger influence on the results than the distribution laws themselves, because these parameters have a strong impact on the shape of the distributions (e.g., Weibull 0.5 has a strong infant mortality contrarily to Weibull 1.5 which has a rapidly decreasing instantaneous failure rate);
- NextStep is always at least as good as Young/Daly, and much better for several scenarios. This holds true over the whole range of distribution types and application/platform parameters.

6 CONCLUSION

We have investigated checkpointing strategies to protect parallel applications from non-memoryless failures. We have designed a general strategy, NextStep, which maximizes the expected efficiency until the next failure. While it may not be optimal because of side-effects towards the end of the application, we proved that this strategy is asymptotically optimal for very long applications. Instead of maximizing the expected efficiency until the next failure, traditional solutions consist in checkpointing periodically according to the platform MTBF (YoungDaly strategy). Our extensive simulation results show that this periodic strategy works well for Exponential distributions, but not for the other distributions, because it either underestimates or overestimates the actual instantaneous failure rate. On the contrary, NextStep is always at least as good as YoungDaly for any failure distribution, and significantly outperforms it in many cases. Overall, our study demonstrates the interest of always using NextStep instead of YoungDaly. In particular, the difference between NextStep and YoungDaly is very important for distributions whose infant mortality of the distribution is high, e.g., LogNormal 2.51 or Weibull 0.5. The latter distributions have been advocated to model failures on real-life platforms [18, 24, 35, 36, 41, 42], which further evidences the impact and significance of NextStep.

Future work will focus on checkpointing strategies for workflows composed of parallel tasks with dependencies, instead of single parallel applications as in this study. The criticality of some tasks in the workflow may lead to checkpoint them more often than prescribed by the NextStep strategy tuned for a given non-memoryless failure distribution. Another interesting perspective is to assess the robustness of the NextStep strategy with only partial knowledge of the probability distribution law of failures on each processor.

Acknowledgements. We would like to thank the reviewers for their comments and suggestions, which greatly helped improve the final version of the paper.

REFERENCES

[1] G. Aupy, A. Benoit, H. Casanova, and Y. Robert. Scheduling computational workflows on failure-prone platforms. *Int. J. of Networking and Computing*, 6(1):2–26, 2016.
[2] G. Aupy, Y. Robert, and F. Vivien. Assuming failure independence: are we right to be wrong? In *FTS'2017*, 2017.
[3] L. Bautista-Gomez, A. Gainaru, S. Perarnau, D. Tiwari, S. Gupta, C. Engelmann, F. Cappello, and M. Snir. Reducing waste in extreme scale systems through introspective analysis. In *IPDPS*, pages 212–221. IEEE, 2016.
[4] L. Bautista-Gomez, S. Tsuboi, D. Komatitsch, F. Cappello, N. Maruyama, and S. Matsuoka. FTI: High performance fault tolerance interface for hybrid systems. In *Proc. SC'11*, 2011.
[5] A. Benoit, A. Cavelan, V. Le Fèvre, Y. Robert, and H. Sun. Towards optimal multi-level checkpointing. *IEEE Trans. Computers*, 66(7):1212–1226, 2017.
[6] A. Benoit, A. Cavelan, Y. Robert, and H. Sun. Assessing general-purpose algorithms to cope with fail-stop and silent errors. *ACM Trans. Parallel Computing*, 3(2), 2016.
[7] A. Benoit, Y. Du, T. Herault, L. Marchal, G. Pallez, L. Perotin, Y. Robert, H. Sun, and F. Vivien. Checkpointing à la Young/Daly: an overview. In *IC3, the 14th Int. Conf. on Contemporary Computing*. ACM Press, 2022.
[8] M. Bougeret, H. Casanova, M. Rabie, Y. Robert, and F. Vivien. Checkpointing strategies for parallel jobs. In *Proc. of SC'11*, 2011.
[9] J. Cao, K. Arya, R. Garg, S. Matott, D. K. Panda, H. Subramoni, J. Vienne, and G. Cooperman. System-level scalable checkpoint-restart for petascale computing. In *22nd Int. Conf.Parallel and Distributed Systems (ICPADS)*. IEEE, 2016.

[10] F. Cappello, A. Geist, W. Gropp, S. Kale, B. Kramer, and M. Snir. Toward exascale resilience: 2014 update. *Supercomputing frontiers and innovations*, 1(1), 2014.
[11] K. M. Chandy and L. Lamport. Distributed snapshots: Determining global states of distributed systems. *ACM Transactions on Computer Systems*, 3(1):63–75, 1985.
[12] K. L. Chung. *A Course in Probability Theory*. Stanford University, 3 edition, 2000.
[13] J. T. Daly. A higher order estimate of the optimum checkpoint interval for restart dumps. *FGCS*, 22(3):303–312, 2006.
[14] S. Di, M. S. Bouguerra, L. Bautista-Gomez, and F. Cappello. Optimization of multi-level checkpoint model for large scale HPC applications. In *IPDPS*. IEEE, 2014.
[15] S. Di, H. Guo, R. Gupta, E. R. Pershey, M. Snir, and F. Cappello. Exploring properties and correlations of fatal events in a large-scale HPC system. *Trans. on Parallel and Distributed Systems*, 2018.
[16] S. Di, Y. Robert, F. Vivien, and F. Cappello. Toward an optimal online checkpoint solution under a two-level HPC checkpoint model. *IEEE Trans. Parallel & Distributed Systems*, 2016.
[17] N. El-Sayed and B. Schroeder. Reading between the lines of failure logs: Understanding how hpc systems fail. In *43rd Int. Conf. Dependable Systems and Networks (DSN)*, pages 1–12. IEEE, 2013.
[18] N. El-Sayed and B. Schroeder. To checkpoint or not to checkpoint: Understanding energy-performance-i/o tradeoffs in hpc checkpointing. In *CLUSTER*, pages 93–102, 2014.
[19] K. Ferreira, J. Stearley, J. H. I. Laros, R. Oldfield, K. Pedretti, R. Brightwell, R. Riesen, P. G. Bridges, and D. Arnold. Evaluating the Viability of Process Replication Reliability for Exascale Systems. In *SC'11*. ACM, 2011.
[20] A. Frank, M. Baumgartner, R. Salkhordeh, and A. Brinkmann. Improving checkpointing intervals by considering individual job failure probabilities. In *IPDPS*, pages 299–309, 2021.
[21] E. Gelenbe, P. Boryszko, M. Siavvas, and J. Domanska. Optimum checkpoints for time and energy. In *28th MASCOTS*, pages 1–8. IEEE, 2020.
[22] L. Han, L.-C. Canon, H. Casanova, Y. Robert, and F. Vivien. Checkpointing workflows for fail-stop errors. *IEEE Trans. Computers*, 67(8):1105–1120, 2018.
[23] L. Han, V. Le Fèvre, L.-C. Canon, Y. Robert, and F. Vivien. A generic approach to scheduling and checkpointing workflows. In *ICPP'2018, the 47th Int. Conf. on Parallel Processing*, 2018.
[24] E. Heien, D. Kondo, A. Gainaru, D. LaPine, B. Kramer, and F. Cappello. Modeling and tolerating heterogeneous failures in large parallel systems. In *Proc. SC'11*, 2011.
[25] T. Herault and Y. Robert, editors. *Fault-Tolerance Techniques for High-Performance Computing*, Computer Communications and Networks. Springer Verlag, 2015.
[26] T. Herault, Y. Robert, A. Bouteiller, D. Arnold, K. B. Ferreira, G. Bosilca, and J. Dongarra. Checkpointing strategies for shared high-performance computing platforms. *International Journal of Networking and Computing*, 9(1):28–52, 2019.
[27] S. Hiroyama, T. Dohi, and H. Okamura. Aperiodic checkpoint placement algorithms—survey and comparison. *Journal of Software Engineering and Applications*, 6(4A):41–53, 2013.
[28] W. Jones, J. Daly, and N. DeBardeleben. Impact of sub-optimal checkpoint intervals on application efficiency in computational clusters. In *HPDC'10*, pages 276–279. ACM, 2010.
[29] O. Kella and W. Stadje. Superposition of renewal processes and an application to multi-server queues. *Statistics & probability letters*, 76(17):1914–1924, 2006.
[30] S. Y. Ko, I. Hoque, B. Cho, and I. Gupta. Making cloud intermediate data fault-tolerant. In *Proc. 1st ACM Symposium on Cloud Computing*, SoCC '10. ACM, 2010.
[31] S. Levy and K. B. Ferreira. An examination of the impact of failure distribution on coordinated checkpoint/restart. In *FTXS Workshop*, pages 35–42. ACM, 2016.
[32] Y. Ling, J. Mi, and X. Lin. A variational calculus approach to optimal checkpoint placement. *IEEE Trans. Computers*, pages 699–708, 2001.
[33] A. Moody, G. Bronevetsky, K. Mohror, and B. R. d. Supinski. Design, Modeling, and Evaluation of a Scalable Multi-level Checkpointing System. In *Proc. SC'10*, 2010.
[34] H. Okamura and T. Dohi. Comprehensive evaluation of aperiodic checkpointing and rejuvenation schemes in operational software system. *Journal of Systems and Software*, 83(9):1591–1604, 2010.
[35] B. Schroeder and G. A. Gibson. A large-scale study of failures in high-performance computing systems. In *Proc. of DSN*, pages 249–258, 2006.
[36] B. Schroeder and G. A. Gibson. Understanding Failures in Petascale Computers. *Journal of Physics: Conf. Series*, 78(1), 2007.
[37] K. Schroiff, P. Gemsjaeger, and C. Bolik. Cascading failover of a data management application for shared disk file systems in loosely coupled node clusters, 2006. US Patent 6,990,606.
[38] P. Sigdel, X. Yuan, and N. Tzeng. Realizing best checkpointing control in computing systems. *IEEE TPDS*, 32(2):315–329, 2021.

[39] L. Silva and J. Silva. Using two-level stable storage for efficient checkpointing. *IEE Proceedings - Software*, 145(6):198–202, 1998.
[40] D. Stirzaker. *Elementary Probability*. Cambridge University Press, 2 edition, 2003.
[41] O. Subasi, G. Kestor, and S. Krishnamoorthy. Toward a general theory of optimal checkpoint placement. In *CLUSTER*, pages 464–474. IEEE, 2017.
[42] O. Subasi, T. Martsinkevich, F. Zyulkyarov, O. Unsal, J. Labarta, and F. Cappello. Unified fault-tolerance framework for hybrid task-parallel message-passing applications. *IJHPCA*, 32(5):641–657, 2018.
[43] D. Tiwari, S. Gupta, and S. S. Vazhkudai. Lazy checkpointing: Exploiting temporal locality in failures to mitigate checkpointing overheads on extreme-scale systems. In *44th Int. Conf. on Dependable Systems and Networks*, pages 25–36. IEEE, 2014.
[44] S. Toueg and O. Babaoğlu. On the optimum checkpoint selection problem. *SIAM J. Comput.*, 13(3), 1984.
[45] K. Yamamoto, A. Uno, H. Murai, T. Tsukamoto, F. Shoji, S. Matsui, R. Sekizawa, F. Sueyasu, H. Uchiyama, M. Okamoto, N. Ohgushi, K. Takashina, D. Wakabayashi, Y. Taguchi, and M. Yokokawa. The K computer Operations: Experiences and Statistics. *Procedia Computer Science (ICCS)*, 29:576–585, 2014.
[46] N. Yigitbasi, M. Gallet, D. Kondo, A. Iosup, and D. Epema. Analysis and modeling of time-correlated failures in large-scale distributed systems. *Parallel and Distrib. Syst. Report Series*, 2010.
[47] J. W. Young. A first order approximation to the optimum checkpoint interval. *Comm. of the ACM*, 17(9):530–531, 1974.

A NEXTSTEP IS NOT OPTIMAL FOR EXPONENTIAL DISTRIBUTIONS

A.1 Introduction to the example

Consider a single-processor application of length $T_{base} = T = 0.062249$, which we need to checkpoint at the end. The duration of a checkpoint is $C = 0.001$ and we have $R = D = 0$. The failure probability density function is $f(x) = e^{-x}$, an Exponential distribution with parameter $\lambda = 1$. We show that NextStep is not optimal for this example.

We start with the computation of a fundamental function: if a failure strikes between instants a and b, what is the expected time $\mathbb{E}_{lost}(a, b)$ between a and the time of the failure? Here, a will always be either equal to 0 ($a = 0$) or the time the last checkpoint was completed. Then, b can be any instant before the end of the next checkpoint. Thus, $\mathbb{E}_{lost}(a, b)$ corresponds to the expected work lost after the last checkpoint.

Next, we compute the expected makespan with a single checkpoint $\mathbb{E}^*_{stat}(T, N_c = 1)$. We also show that this is the best static strategy, by proving that the optimal makespan of a static strategy with two checkpoints, $\mathbb{E}^*_{stat}(T, N_c = 2)$, is strictly greater, which gives the result according to Equation (2) in Section 3.2. Indeed, with the notations of Section 3.2, if two checkpoints are worse than one checkpoint, that means $N_{opt} \leq 2$, hence $N_{ME} \leq 2$.

Finally, we lower-bound the expected makespan of NextStep. To do so, we first find the optimal expected efficiency $\mathbb{E}^*_e(T, N_c)$ achievable by NextStep for $N_c = 1$ and $N_c = 2$, and show that the efficiency is better for $N_c = 2$, implying that the initial strategy includes at least two checkpoints. Then, for any $N_c \geq 2$, we suppose that NextStep does N_c checkpoints in the first call and we lower-bound the expected makespan $\hat{\mathbb{E}}_{WF}(T, N_c)$ in this case. Rephrased differently, $\hat{\mathbb{E}}_{WF}(T, N_c)$ corresponds to the expected makespan of the algorithm that optimizes the efficiency up to the next failure starting with N_c checkpoints, and applies NextStep afterwards. We check that for all $N_c \geq 2$, $\hat{\mathbb{E}}_{WF}(T, N_c) > \mathbb{E}^*_{stat}(T, 1)$. In particular, if the actual number of checkpoints of NextStep planned in the first call is $N^*_c \geq 2$, its expected makespan $\mathbb{E}_{WF}(T)$ verifies $\mathbb{E}_{WF}(T) = \hat{\mathbb{E}}_{WF}(T, N^*_c) > \mathbb{E}^*_{stat}(T, 1)$, which shows that it is not optimal.

A.2 Computation of $\mathbb{E}_{lost}(a, b)$

The expected time lost if we have a failure between a and b, with a backup at time a, is computed as follows:

$$\begin{aligned}
\mathbb{E}_{lost}(a, b) &= \left(\int_0^\infty x \mathbb{P}\{X = x | a < X < b\} dx \right) - a \\
&= \frac{1}{\mathbb{P}\{a < X < b\}} \left(\int_a^b x f(x) dx \right) - a \\
&= \frac{1}{\mathbb{P}\{X < b\} - \mathbb{P}\{X < a\}} \left(\int_a^b x e^{-x} dx \right) - a \\
&= \frac{1}{e^{-a} - e^{-b}} \left(\int_a^b x e^{-x} dx \right) - a
\end{aligned}$$

$$\begin{aligned}
\int_a^b x e^{-x} dx &= [-x e^{-x}]_a^b - \int_a^b -e^{-x} dx \\
&= a e^{-a} - b e^{-b} - [e^{-x}]_a^b \\
&= (a + 1) e^{-a} - (b + 1) e^{-b}
\end{aligned}$$

$$\begin{aligned}\mathbb{E}_{lost}(a,b) &= \frac{(a+1)e^{-a}-(b+1)e^{-b}}{e^{-a}-e^{-b}} - a \\ &= 1 + \frac{ae^{-a}-be^{-b}-a(e^{-a}-e^{-b})}{e^{-a}-e^{-b}} \\ &= 1 - \frac{(b-a)e^{-b}}{e^{-a}-e^{-b}}\end{aligned}$$

$$\mathbb{E}_{lost}(a,b) = 1 - \frac{b-a}{e^{b-a}-1}$$

We check that $\mathbb{E}_{lost}(a,b)$ only depends of $b-a$, which is comforting because the distribution is memoryless. In the following, we will use the difference $b-a$ as single parameter x, i.e.,

$$\mathbb{E}_{lost}(x) = 1 - \frac{x}{e^x - 1} \tag{9}$$

A.3 Computation of $\mathbb{E}^*_{stat}(T, N_c = 1)$

With a single checkpoint, the makespan is $T+C$ if there are no failures. Otherwise we lose the time of processing before the first failure occurs, $\mathbb{E}_{lost}(T+C)$, and we need to start again. Therefore, $\mathbb{E}^*_{stat}(T, N_c = 1)$ can be expressed as follows:

$$\begin{aligned}\mathbb{E}^*_{stat}(T,1) = \mathbb{P}\{X > T+C\}(T+C) \\ + \mathbb{P}\{X < T+C\}(\mathbb{E}^*_{stat}(T,1) + \mathbb{E}_{lost}(T+C))\end{aligned}$$

$$\mathbb{E}^*_{stat}(T,1) = T + C + \frac{1-e^{-(T+C)}}{e^{-(T+C)}}\mathbb{E}_{lost}(T+C) \tag{10}$$

With $T = 0.062249$ and $C = 0.001$, using Equation (9), we obtain $\mathbb{E}^*_{stat}(T,1) \approx 0.06529206$.

A.4 Computation of $\mathbb{E}^*_{stat}(T, N_c = 2)$

In this case, recall from Section 3.2 that the best solution is to have two equal-sized segments, therefore we can use Equation (10) with $\mathbb{E}^*_{stat}\left(\frac{T}{2}, 1\right)$ and obtain:

$$\begin{aligned}\mathbb{E}^*_{stat}(T, N_c = 2) &= 2 \times \mathbb{E}^*_{stat}\left(\frac{T}{2}, 1\right) \\ &= 2\left(\frac{T}{2} + C + \frac{1-e^{-\left(\frac{T}{2}+C\right)}}{e^{-\left(\frac{T}{2}+C\right)}}\mathbb{E}_{lost}\left(\frac{T}{2}+C\right)\right)\end{aligned}$$

Using Equations (9) and (10), we get $\mathbb{E}^*_{stat}(T,2) \approx 0.06529212 > \mathbb{E}^*_{stat}(T,1)$ with $T = 0.062249$ and $C = 0.001$. As stated in Section A.1, this result shows that the optimal checkpointing heuristic uses a single checkpoint at the end.

A.5 Computation of $\mathbb{E}^*_e(T, N_c = 1)$ and $\mathbb{E}^*_e(T, N_c = 2)$

We now show that NextStep does at least two checkpoints in the first call, by computing the best expected efficiency $\mathbb{E}^*_e(T, N_c)$ using one or two checkpoints, and checking that indeed $\mathbb{E}^*_e(T,2) > \mathbb{E}^*_e(T,1)$.

With a single checkpoint at the end, we can either process 0 units of work if we have a failure, or process the whole application. Therefore,

$$\begin{aligned}\mathbb{E}^*_W(T,1) &= 0 \times \mathbb{P}\{X < T+C\} + T\mathbb{P}\{X > T+C\} \\ &= Te^{-(T+C)}\end{aligned}$$

Furthermore, the expected time before the next failure or the end of the application can be computed as follows:

$$\begin{aligned}\mathbb{E}_{T_{next}}(T,1) &= \mathbb{P}\{X > T+C\}(T+C)\\ &\quad + \mathbb{P}\{X < T+C\}\mathbb{E}_{lost}(T+C)\\ &= e^{-(T+C)}(T+C) + (1-e^{-(T+C)})\left(1 - \frac{T+C}{e^{T+C}-1}\right)\end{aligned}$$

Hence, with no additional checkpoint,

$$\mathbb{E}^*_e(T,1) = \frac{Te^{-(T+C)}}{e^{-(T+C)}(T+C) + (1-e^{-(T+C)})\left(1 - \frac{T+C}{e^{T+C}-1}\right)} \tag{11}$$

With $T = 0.062249$ and $C = 0.001$, we obtain $\mathbb{E}^*_e(T,0,1) \approx 0.95339305$.

If we add one checkpoint, we could place it anywhere; in that case, if we checkpoint after $w \in (0,T)$ units of work, the work done before the next failure or the end of the application can either be 0 if a failure occurs in $[0, w+C]$, w if a failure occurs in $(w+C, T+2C)$ or T if no failure occurs. Thus,

$$\begin{aligned}\mathbb{E}^*_W(T, N_c = 2) &= \max_w \mathbb{E}_W(T,2,w)\\ &= \max_w (w\mathbb{P}\{w + c < X < T+2C\} + T\mathbb{P}\{X > T+2C\})\\ &= \max_w (w(e^{-(w+C)} - e^{-(T+2C)}) + Te^{-(T+2C)})\\ &= \max_w ((T-w)e^{-(T+2C)} + we^{-(w+C)})\end{aligned}$$

The expected time before the next failure or the end of the application is:

$$\begin{aligned}\mathbb{E}_{T_{next}}(T, N_c = 2) &= \mathbb{P}\{X > T+2C\}(T+2C)\\ &\quad + \mathbb{P}\{X < T+2C\}\mathbb{E}_{lost}(0, T+2C)\\ &= e^{-(T+2C)}(T+2C) + (1-e^{-(T+2C)})\left(1 - \frac{T+2C}{e^{T+2C}-1}\right)\end{aligned}$$

Altogether, these results give the formula for $\mathbb{E}^*_e(T, N_c = 2)$:

$$\mathbb{E}^*_e(T,2) = \frac{\max_w((T-w)e^{-(T+2C)} + we^{-(w+C)})}{e^{-(T+2C)}(T+2C) + (1-e^{-(T+2C)})\left(1 - \frac{T+2C}{e^{T+2C}-1}\right)} \tag{12}$$

The best choice for w is hard to express analytically, but with $T = 0.062249$ and $C = 0.001$, we numerically obtain $w^* \approx 0.0313732$ and $\mathbb{E}^*_e(T,2) > 0.95339312 > \mathbb{E}^*_e(T,1)$. This shows that NextStep will not execute a single checkpoint at the end. We will finally show that for all $N_c \geq 2$, $\hat{\mathbb{E}}_{WF}(T, N_c) > \mathbb{E}^*_{stat}(T,1)$.

A.6 Lower bound of $\hat{\mathbb{E}}_{WF}(T, N_c = 2)$

We have already shown that if NextStep does two checkpoints in the first call, the first one will be placed after w^* units of work executed, and the second one will be placed at the end. To compute $\hat{\mathbb{E}}_{WF}(T,2)$, we note that if no failures happen, the makespan is $T+2C$, whereas if the failure strikes in the second segment, we lose $\mathbb{E}_{lost}(T+C-w^*)$ and need to reprocess the remaining $T-w^*$ units of time, which takes times $\mathbb{E}_{WF}(T-w^*)$ (because the distribution is memoryless and $R = D = 0$). Finally, if a failure strikes in the first segment, we lose $\mathbb{E}_{lost}(w^*+C)$ and need to retry from the beginning. Overall, $\hat{\mathbb{E}}_{WF}(T,2)$ can be expressed as follow:

$$\begin{aligned}\hat{\mathbb{E}}_{WF}(T,2) &= \mathbb{P}\{X > T+2C\}(T+2C)\\ &\quad + (\mathbb{P}\{X > w^*+C\} - \mathbb{P}\{X > T+2C\})\\ &\quad \times (w^* + C + \mathbb{E}_{lost}(T+C-w^*) + \mathbb{E}_{WF}(T-w^*))\\ &\quad + (1 - \mathbb{P}\{X > w^*+C\})(\mathbb{E}_{lost}(w^*+C) + \hat{\mathbb{E}}_{WF}(T,2))\end{aligned}$$

We isolate $\hat{\mathbb{E}}_{WF}(T,2)$ at the left side of the equation and divide by $\mathbb{P}\{X > w^* + C\}$ to obtain:

$$\begin{aligned}\hat{\mathbb{E}}_{WF}(T,2) = &\frac{1}{e^{-(w^*+C)}}\\ &\times ((T+2C)e^{-(T+2C)} + (1-e^{-(w^*+C)})\mathbb{E}_{lost}(w^*+C))\\ &+ \frac{e^{-(w^*+C)} - e^{-(T+2C)}}{e^{-(w^*+C)}}\\ &\times (w^* + C + \mathbb{E}_{lost}(T+C-w^*) + \mathbb{E}_{WF}(T-w^*)))\end{aligned}$$

The only unknown value is $\mathbb{E}_{WF}(T-w^*)$, which we may lower-bound by the optimal strategy given a segment of length $T-w^*$. If we chose to do a single checkpoint at the end and stick with this strategy, the expected makespan of this segment would be $\mathbb{E}^*_{stat}(T-w^*,1) \approx 0.0324$, whereas if we chose to do at least one additional checkpoint, the expected makespan is larger than $w^*+2C \approx 0.0329 > \mathbb{E}^*_{stat}(T-w^*,1)$, therefore the first choice of an optimal strategy is not doing any additional checkpoint. Clearly, if a failure occurs, changing the strategy can only result in an increase in expected makespan, so we have $\mathbb{E}_{WF}(T-w^*) \geq \mathbb{E}^*_{stat}(T-w^*,1)$. Finally,

$$\begin{aligned}\hat{\mathbb{E}}_{WF}(T,2) \geq &\frac{1}{e^{-(w^*+C)}}\\ &\times ((T+2C)e^{-(T+2C)} + (1-e^{-(w^*+C)})\mathbb{E}_{lost}(w^*+C))\\ &+ \frac{e^{-(w^*+C)} - e^{-(T+2C)}}{e^{-(w^*+C)}}\\ &\times (w^* + C + \mathbb{E}_{lost}(T+C-w^*) + \mathbb{E}^*_{stat}(T-w^*,1))\end{aligned}$$

Using Equations (9) and (10), with $T = 0.062249$ and $C = 0.001$, we get $\hat{\mathbb{E}}_{WF}(T,2) > 0.06529218 > \mathbb{E}^*_{stat}(T,1)$.

A.7 Lower bound of $\hat{\mathbb{E}}_{WF}(T, N_c = 3)$

We now assume NextStep initially splits the application into three segments and considers the longest segment, of length $L \geq \frac{T}{3}$, which starts after w units of work and has s segments before. In particular, $w = 0$ if $s = 0$, $w \in (0, T-L)$ if $s = 1$, and $w = T - L$ if $s = 2$.

To lower-bound the expected makespan in that scenario, we consider several cases:

- If no failure occurs, the makespan is $T + 3C$. This happens with probability $e^{-(T+3C)}$.
- If a failure occurs in the longest segment, the conditional expected makespan is at least $T + C + \mathbb{E}_{lost}\left(\frac{T}{3} + C\right)$, because in total we have to process at least $T + C$ for the application in total, and we lose an additional time $\mathbb{E}_{lost}(L + C) \geq \mathbb{E}_{lost}\left(\frac{T}{3} + C\right)$ because a failure occurred in the segment. This happens with probability $f(w, s, L)$.
- Finally, if a failure occurs during the first processing in another segment, the conditional expected makespan is at least $T + C$, the strongest lower-bound for the processing. This scenario happens with a probability $1 - e^{-(T+3C)} - f(w, s, L)$.

We are always in one and only one of these three scenarios, therefore we can lower-bound the expected makespan using previous bounds of the conditional expected makespan combined with their probability of occurrence:

$$\begin{aligned}\hat{\mathbb{E}}_{WF}(T, N_c = 3) &\geq (T+3C)e^{-(T+3C)} \\ &+ \left(T + C + \mathbb{E}_{lost}\left(\frac{T}{3} + C\right)\right) f(w, s, L) \\ &+ (T+C)\left(1 - e^{-(T+3C)} - f(w, s, L)\right) \\ \hat{\mathbb{E}}_{WF}(T, N_c = 3) &\geq (T+3C)e^{-(T+3C)} \\ &+ \mathbb{E}_{lost}\left(\frac{T}{3} + C\right) f(w, s, L) + (T+C)\left(1 - e^{-(T+3C)}\right)\end{aligned}$$

Clearly, our bound increases when $f(w, s, L)$ increases. Therefore, we need to find a lower bound for $f(w, s, L)$ to conclude. But $f(w, s, L)$ represents the probability that a failure strikes the longest segment during the first try of the processing. This is the conjunction of the success of the segments before and the failure of the longest segment. Therefore,

$$f(w, s, L) = e^{-(w+sC)}(1 - e^{-(L+C)}).$$

This function is clearly increasing with L and decreasing with w and s. The largest possible s is 2. As $L \geq \frac{T}{3}$ and $w \leq T - L$, the smallest possible L is $\frac{T}{3}$ whereas the largest possible w is $\frac{2T}{3}$. Clearly, $f(w, s, L) \geq f\left(\frac{2T}{3}, 2, \frac{T}{3}\right)$, and our lower bound on $\hat{\mathbb{E}}_{WF}(T, N_c = 3)$ becomes:

$$\begin{aligned}\hat{\mathbb{E}}_{WF}(T, 3) &\geq (T+3C)e^{-(T+3C)} + (T+C)\left(1 - e^{-(T+3C)}\right) \\ &+ \mathbb{E}_{lost}\left(\frac{T}{3} + C\right) e^{-\left(\frac{2T}{3}+2C\right)}\left(1 - e^{-\left(\frac{T}{3}+C\right)}\right)\end{aligned}$$

Using Equation (9), with $T = 0.062249$ and $C = 0.001$, we get $\hat{\mathbb{E}}_{WF}(T, 3) > 0.06534 > \mathbb{E}^*_{stat}(T, 1)$.

A.8 Lower bound of $\hat{\mathbb{E}}_{WF}(T, N_c)$ for $N_c \geq 4$

If NextStep chooses to do N_c checkpoints at first, the resulting the makespan is $T + N_cC$ if no failures occur, and at least $T + C$ otherwise. Therefore,

$$\hat{\mathbb{E}}_{WF}(T, N_c) \geq (T + N_cC)e^{-(T+N_cC)} + (1 - e^{-(T+N_cC)})(T + C).$$

As $T + N_cC > T + C$, this bound is clearly decreasing when $e^{(-(T+N_CC))}$ decreases, therefore when N_C increases. We can safely use $N_C = 4$ to get

$$\hat{\mathbb{E}}_{WF}(T, N_c) \geq (T + 4C)e^{-(T+4C)} + (1 - e^{-(T+4C)})(T + C).$$

With $T = 0.062249$ and $C = 0.001$, we get $\hat{\mathbb{E}}_{WF}(T, N_c) \geq \hat{\mathbb{E}}_{WF}(T, 4) > 0.066 > \mathbb{E}^*_{stat}(T, 1)$.

We have shown that for all $N_c \geq 2$, $\hat{\mathbb{E}}_{WF}(T, N_c) > \mathbb{E}^*_{stat}(T, 1)$; therefore NextStep is strictly worse than the best static strategy, which shows that it is not optimal. This concludes the analysis of the counter-example.

B ON THE DYNAMIC VERSION OF THE OPTIMAL STATIC STRATEGY FOR AN EXPONENTIAL DISTRIBUTION

In this section, we show that the dynamic version of the optimal static strategy is identical to the static version when failures obey an Exponential distribution. We start with a few notations before formally stating this result.

B.1 Notations

In the following, we consider a sequential or parallel application of length T_{base}. A checkpointing strategy $\mathcal{S}$ is defined as $\mathcal{S} = \{c_1, c_2, \ldots, c_m\}$, where each $c_k \in (0, T_{base})$ denotes the amount of the work executed until checkpoint number k. Note that we assume that there is a checkpoint at the end, i.e., $c_m = T_{base}$.

When a failure occurs, let $\mathbb{E}(R)$ be the expected time before the processors are ready to work again. This includes the downtime and a recovery time, but may be longer if we encounter another failure during the recovery time[4]. For a given checkpointing strategy $\mathcal{S}$ and a work $w \in \mathcal{S} \cup \{0\}$, we denote by $\mathbb{E}([0, w], \mathcal{S})$ the expected time between the start of the application and the completion of the checkpoint corresponding to w units of work. Similarly, we denote by $\mathbb{E}([w, T_{base}], \mathcal{S})$ the expected time between the moment the checkpoint corresponding to w units of work is completed (or the start of the application if $w = 0$) and the moment the application finishes the completion, including the last checkpoint. If we do not have $w \in \mathcal{S} \cup \{0\}$, both expectations are considered infinite. With these definitions, we clearly have:

$$\forall w \in \mathcal{S}, \mathbb{E}([0, T_{base}], \mathcal{S}) = \mathbb{E}([0, w], \mathcal{S}) + \mathbb{E}([w, T_{base}], \mathcal{S}).$$

Finally, given a work $w \in [0, T_{base}]$, we let $\mathcal{S}^*_w$ be a checkpointing strategy such that for all $\mathcal{S}$, we have $\mathbb{E}([w, T_{base}], \mathcal{S}) \geq \mathbb{E}([w, T_{base}], \mathcal{S}^*_w)$. Although multiple checkpointing strategies may minimize this expectation, the value of this expectation $\mathbb{E}^*_w \triangleq \mathbb{E}([w, T_{base}], \mathcal{S}^*_w)$ is unique and well defined. Intuitively, $\mathcal{S}^*_w$ is an optimal checkpointing strategy for the end of the application after w units have been processed and checkpointed.

B.2 Main result

Theorem 2. *For an application of length T_{base}, consider the following two approaches:*

- *(A) Static Strategy: Find an optimal checkpointing heuristic $\mathcal{S}^*_0$ that minimizes the total expected makespan $\mathbb{E}^*_0$ and does not update the strategy until the application is completed.*
- *(B) Dynamic Strategy: Start with the best static strategy $\mathcal{S}^*_0$, then whenever an **event** occurs, i.e., a segment is completed or a failure happens, find an optimal static checkpointing strategy minimizing the remaining expected makespan. If the remaining expected makespan is strictly smaller with the new strategy, update the checkpointing strategy accordingly.*

The static strategy (A) and the dynamic strategy (B) are identical.

The optimal static strategy (A) is well-known and uses N_{ME} segments, where N_{ME} is given in Section 3.2. The value of N_{ME} depends upon the length of the application that remains to be processed, so strategy (B) could compute a different value when called after the first checkpoint or the first failure. The proof shows that this is never the case.

Proof. Initially, both strategies are identical by definition. We assume that the strategy (B) can diverge from the strategy (A) and obtain a contradiction. Suppose that both strategies are different. Then, there exists a failure scenario in which both strategies diverge. Consider such scenario and let W be the total work executed and backed-up when the first *event* e occurs, after which strategy (B) becomes different from strategy (A).

After this event e, the expected resulting makespan of strategy(A) is $\mathbb{E}^A_{remain} = t(e) + \mathbb{E}([W, T_{base}], \mathcal{S}^*_0)$, where $t(e) = 0$ if the event is the end of a segment, $t(e) = D$ if the event is a failure in the first segment for the model in which a recovery is not necessary for the first segment, and $t(e) = \mathbb{E}(R)$ otherwise. In any case, $t(e)$ is a cost in execution time independent to the checkpointing heuristic. We must finish the processing of the as planned with strategy $\mathcal{S}^*_0$. The latter is identical to $\mathbb{E}([W, T_{base}], \mathcal{S}^*_0)$, because the distribution is memoryless, therefore we are exactly at the same point after the recovery as we were when we first succeeded the checkpoint corresponding to W units of work.

Strategy (B) also needs to spend $t(e)$ units of time to deal with the event and is able to find a new checkpointing strategy such that the overall expected remaining makespan is reduced. As before, because the distribution is memoryless, an optimal strategy is $\mathcal{S}^*_W$. By assumption, this new strategy reduces the total expected makespan. Thus,

$$\mathbb{E}(R) + \mathbb{E}([W, T_{base}], \mathcal{S}^*_W) = \mathbb{E}^B_{remain} < \mathbb{E}^A_{remain}$$
$$\mathbb{E}(R) + \mathbb{E}([W, T_{base}], \mathcal{S}^*_W) < \mathbb{E}(R) + \mathbb{E}([W, T_{base}], \mathcal{S}^*_0)$$

[4]Similarly to Equation (1), we have $\mathbb{E}(R) = De^{\lambda R} + \frac{1}{\lambda}\left(e^{\lambda R} - 1\right)$. But we do not need to know the value of $\mathbb{E}(R)$ in this derivation.

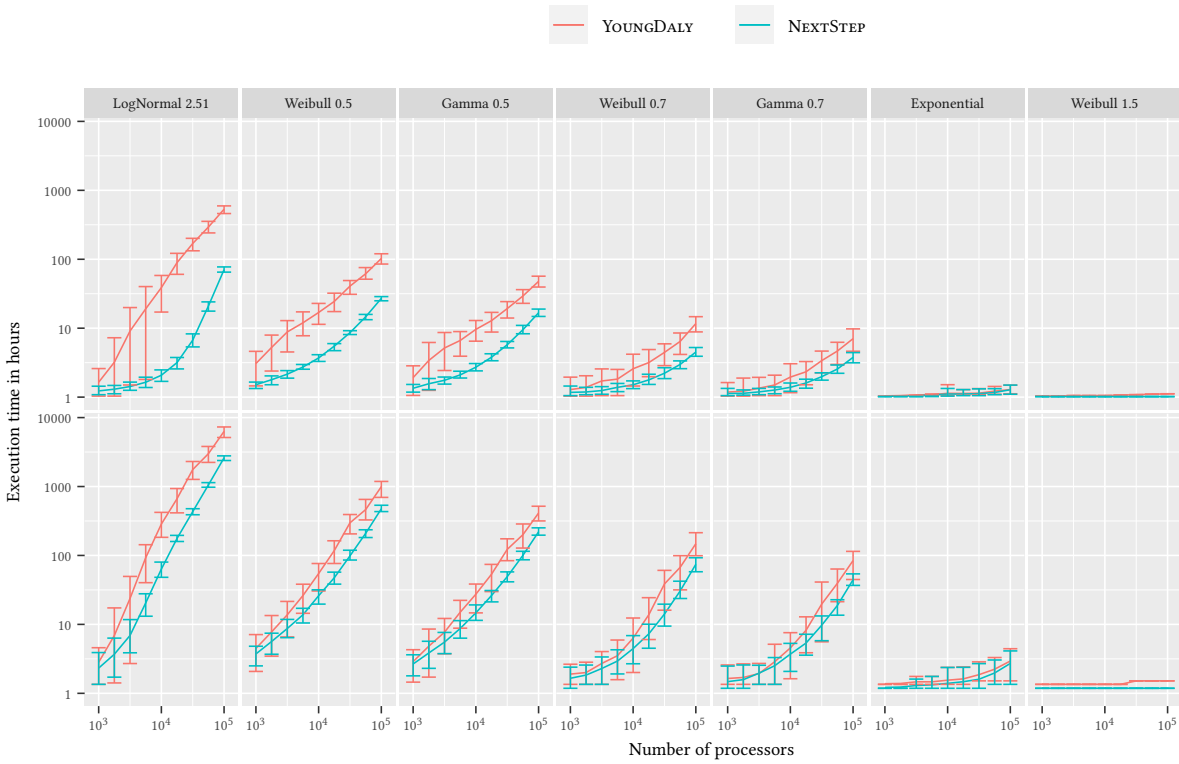

Fig. 4. Expected performance of the three heuristics under all failure distribution on a 0 day old platform and with a workflow W of 1 hour. Top: $C = 60$, bottom: $C = 600$.

Finally,

$$\mathbb{E}([W, T_{base}], \mathcal{S}_W^*) < \mathbb{E}([W, T_{base}], \mathcal{S}_0^*).$$

Now, suppose that we had applied the strategy $S_2 = \left(\mathcal{S}_0^* \setminus [W, T_{base}]\right) \cup \left(\mathcal{S}_W^* \setminus (0, W)\right)$. The total expectation would have been:

$$\begin{aligned}
\mathbb{E}([0, T_{base}], \mathcal{S}_2) &= \mathbb{E}([0, W], \mathcal{S}_2) + \mathbb{E}([W, T_{base}], \mathcal{S}_2) \\
&= \mathbb{E}([0, W], \mathcal{S}_0^*) + \mathbb{E}([W, T_{base}], \mathcal{S}_W^*) \\
\mathbb{E}([0, T_{base}], \mathcal{S}_2) &< \mathbb{E}([0, W], \mathcal{S}_0^*) + \mathbb{E}([W, T_{base}], \mathcal{S}_0^*) \\
\mathbb{E}([0, T_{base}], \mathcal{S}_2) &< \mathbb{E}([0, T_{base}], \mathcal{S}_0^*) \\
\mathbb{E}([0, T_{base}], \mathcal{S}_2) &< \mathbb{E}_0^*
\end{aligned}$$

This contradicts the definition of $\mathcal{S}_0^*$ and $\mathbb{E}_0^*$. □

C ALL SIMULATION RESULTS

The following figures are the results of the simulations with all combinations of parameters.

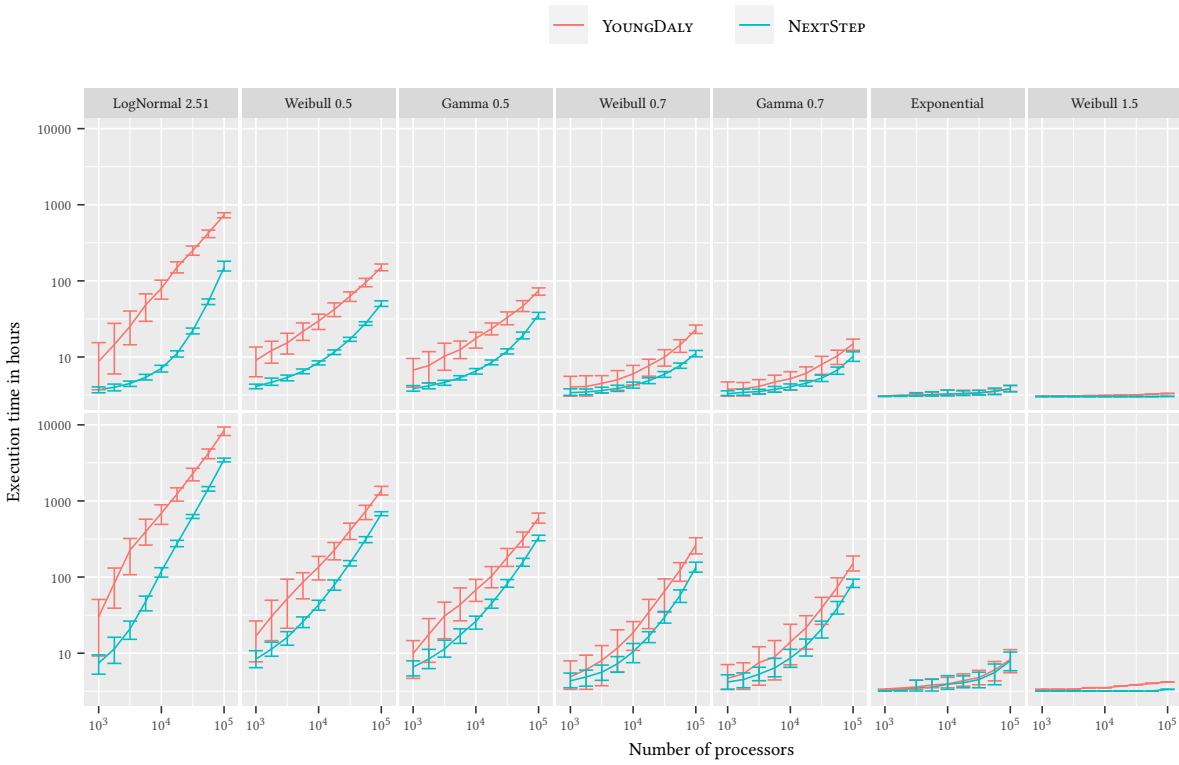

Fig. 5. Expected performance of the three heuristics under all failure distribution on a 0 day old platform and with a workflow W of 3 hours. Top: $C = 60$, bottom: $C = 600$.

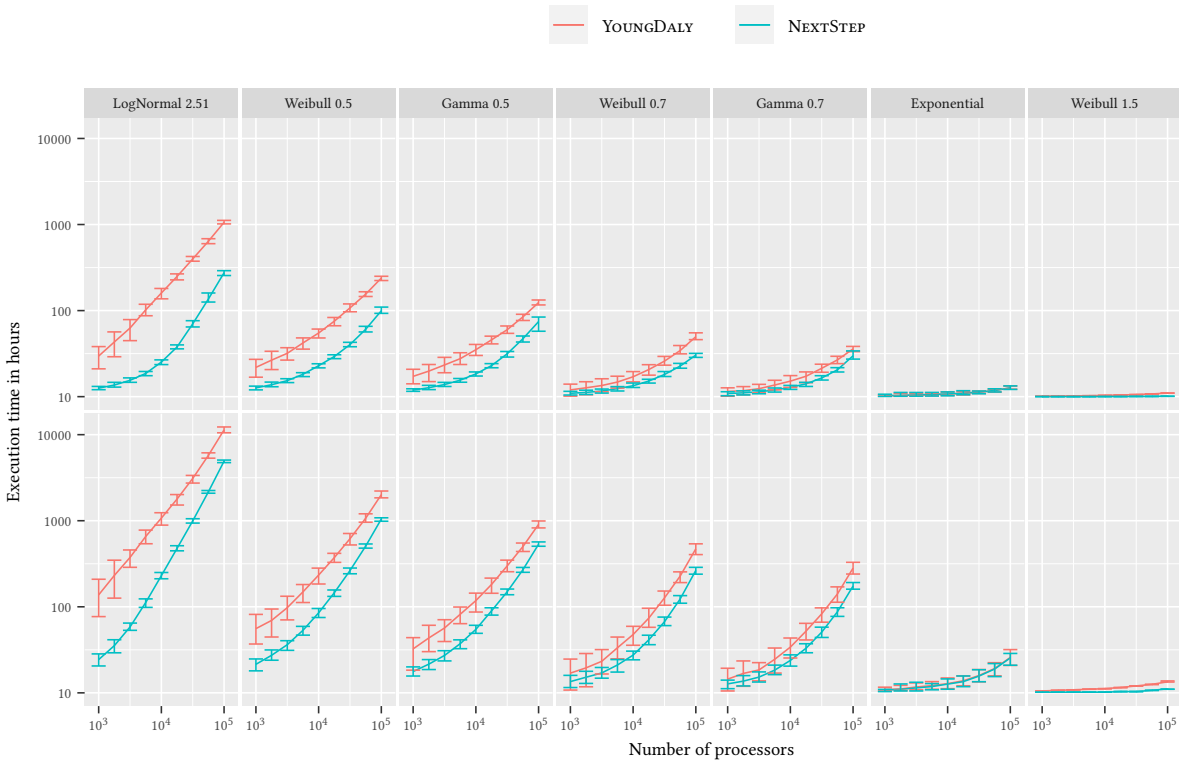

Fig. 6. Expected performance of the three heuristics under all failure distribution on a 0 day old platform and with a workflow W of 10 hours. Top: $C = 60$, bottom: $C = 600$.

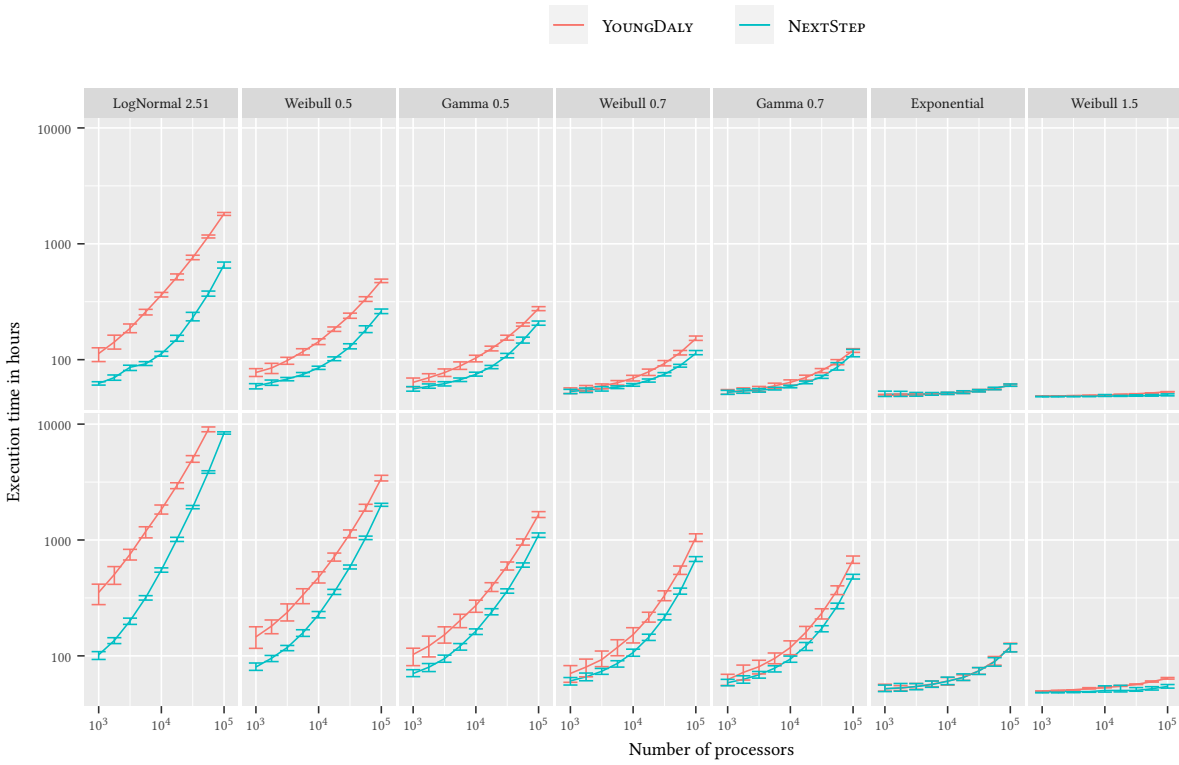

Fig. 7. Expected performance of the three heuristics under all failure distribution on a 0 day old platform and with a workflow W of 48 hours. Top: $C = 60$, bottom: $C = 600$.

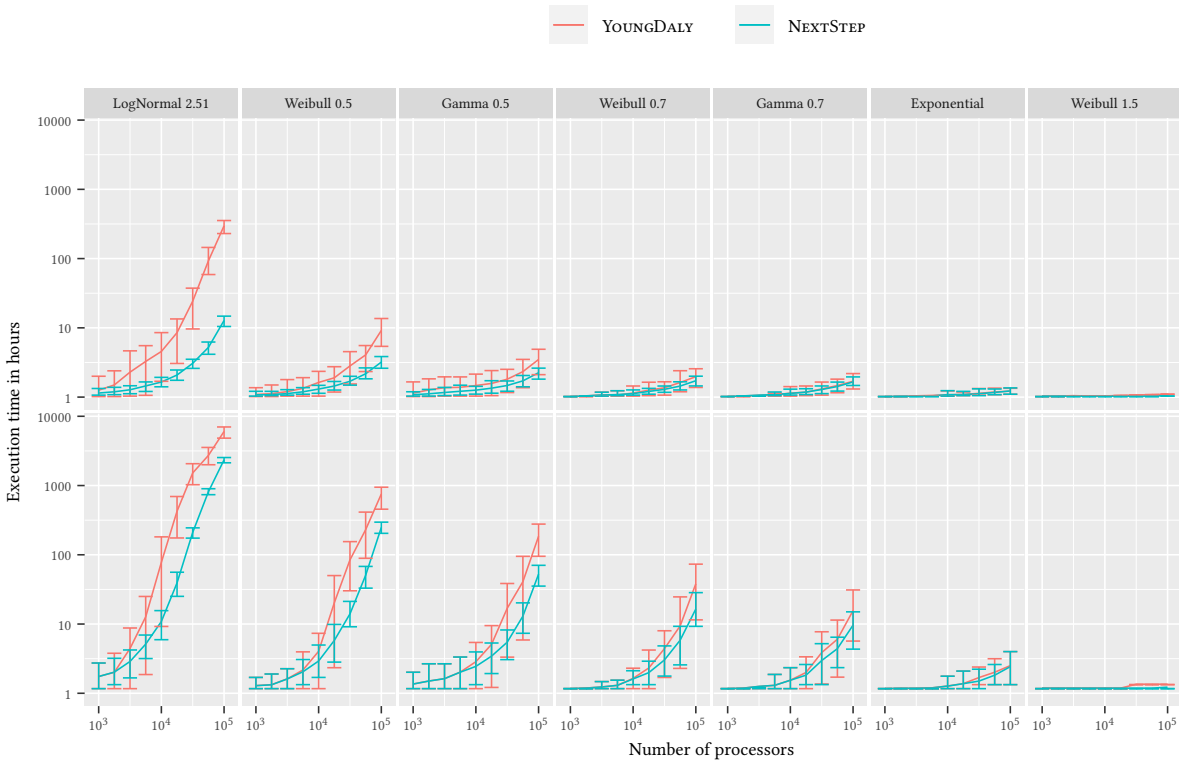

Fig. 8. Expected performance of the three heuristics under all failure distribution on a 10 day old platform and with a workflow W of 1 hour. Top: $C = 60$, bottom: $C = 600$.

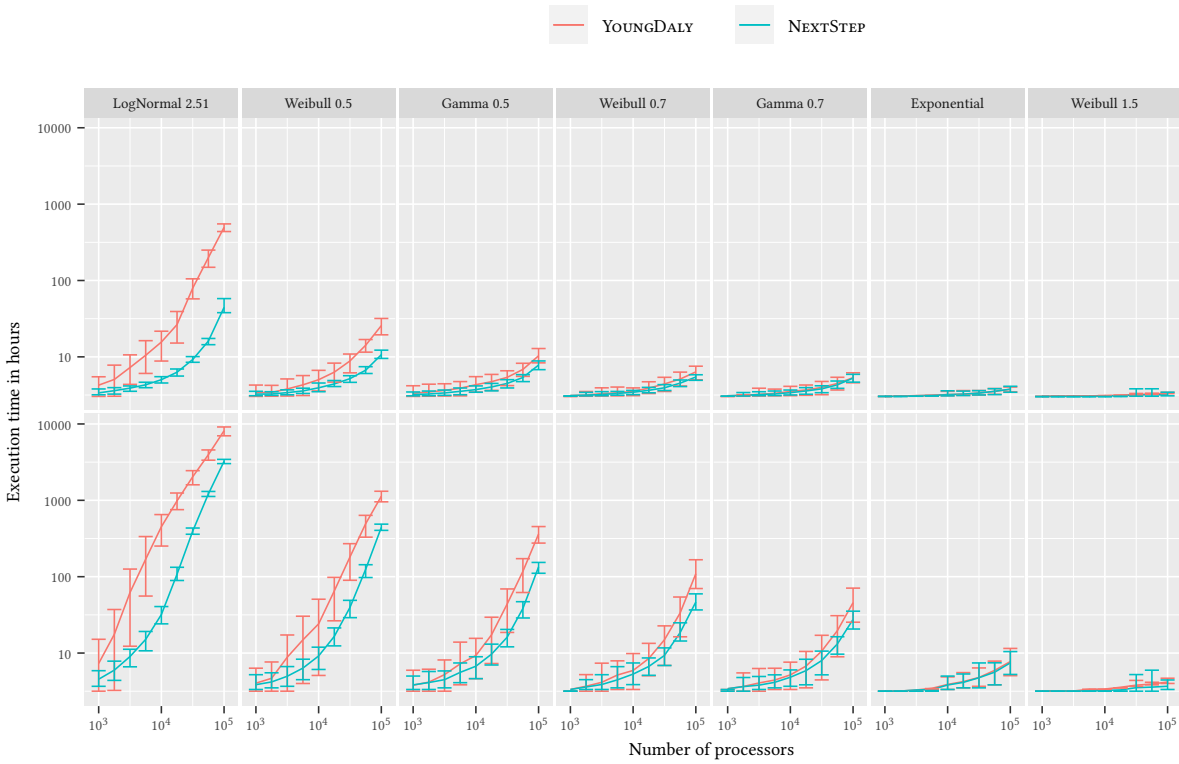

Fig. 9. Expected performance of the three heuristics under all failure distribution on a 10 day old platform and with a workflow W of 3 hours. Top: $C = 60$, bottom: $C = 600$.

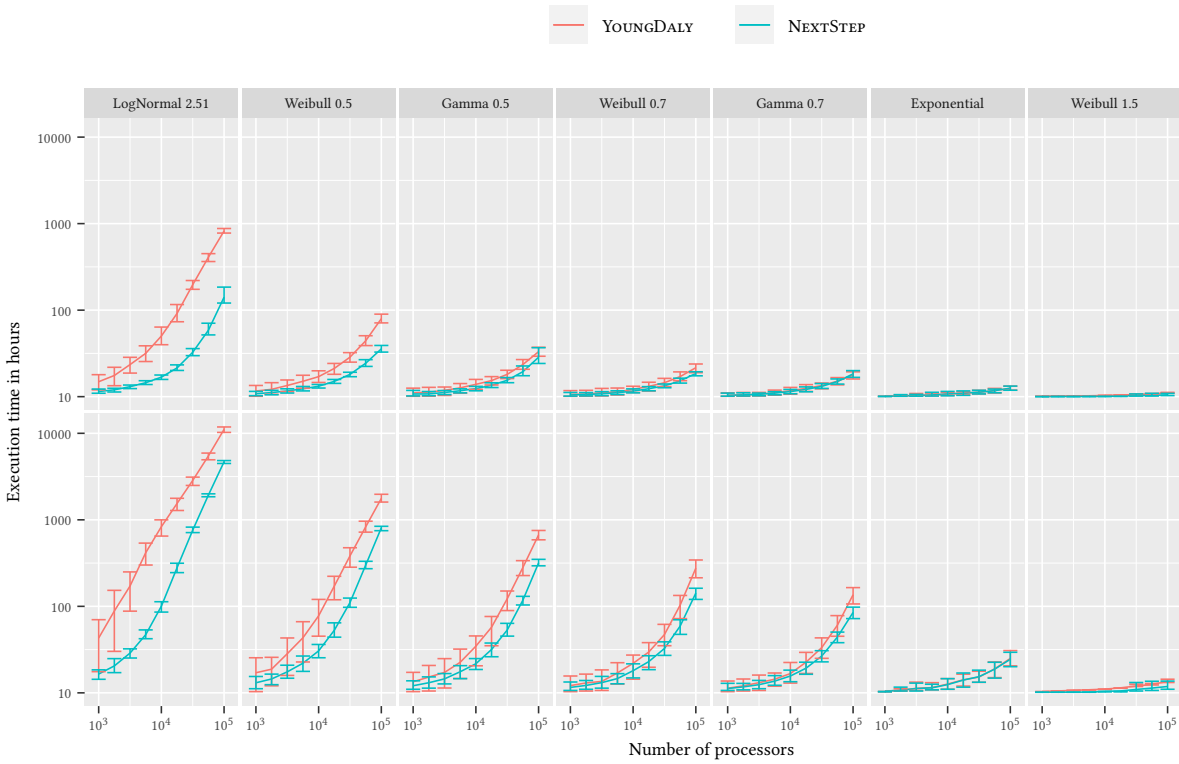

Fig. 10. Expected performance of the three heuristics under all failure distribution on a 10 day old platform and with a workflow W of 10 hours. Top: $C = 60$, bottom: $C = 600$.

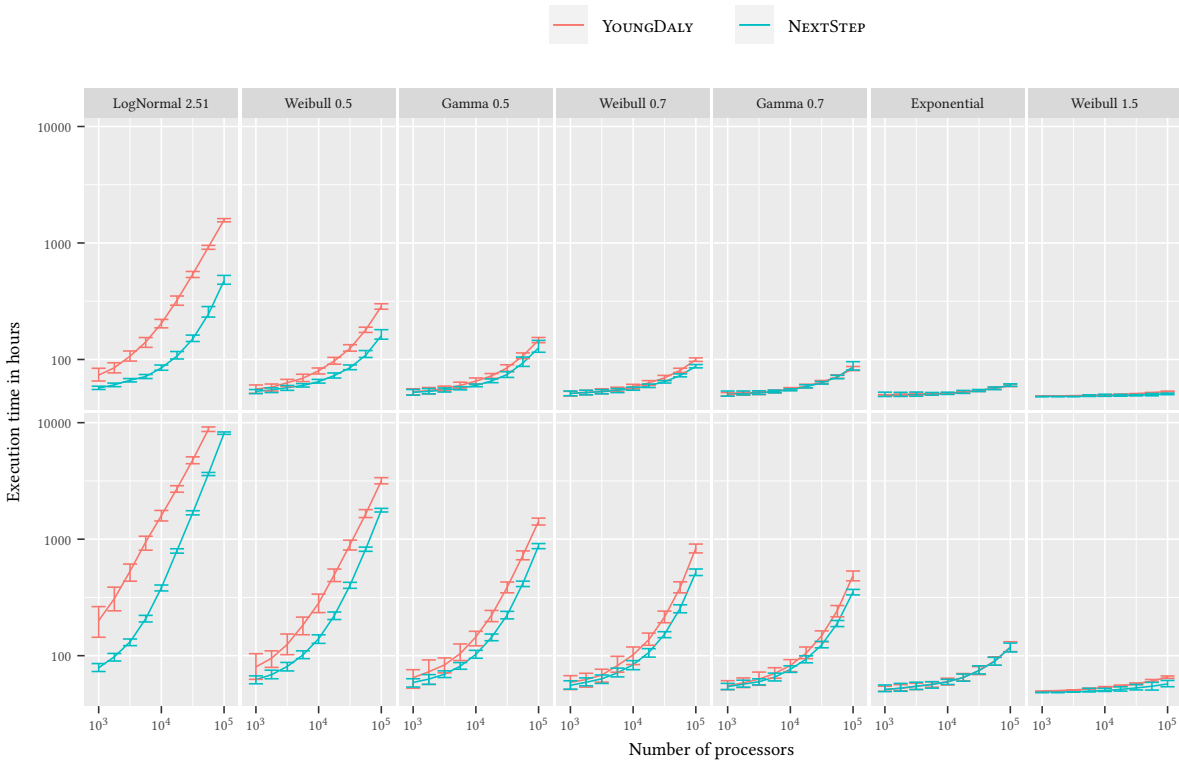

Fig. 11. Expected performance of the three heuristics under all failure distribution on a 10 day old platform and with a workflow W of 48 hours. Top: $C = 60$, bottom: $C = 600$.

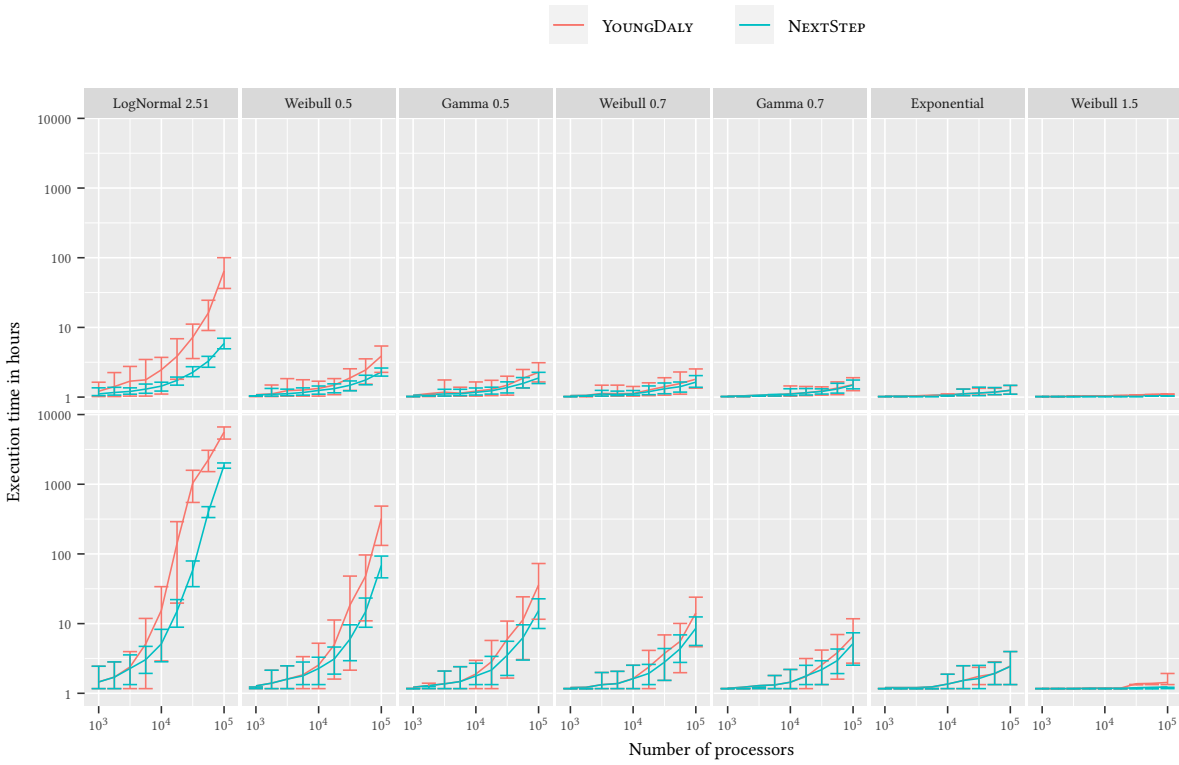

Fig. 12. Expected performance of the three heuristics under all failure distribution on a 30 day old platform and with a workflow W of 1 hour. Top: $C = 60$, bottom: $C = 600$.

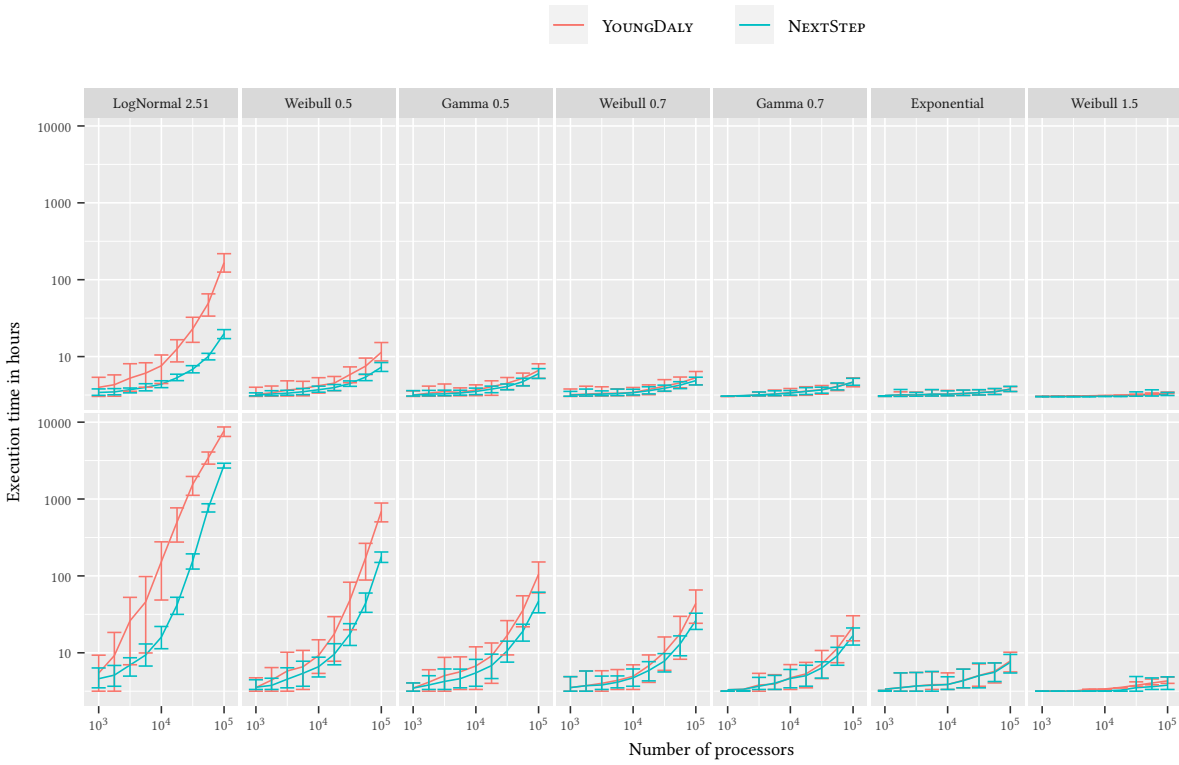

Fig. 13. Expected performance of the three heuristics under all failure distribution on a 30 day old platform and with a workflow W of 3 hours. Top: $C = 60$, bottom: $C = 600$.

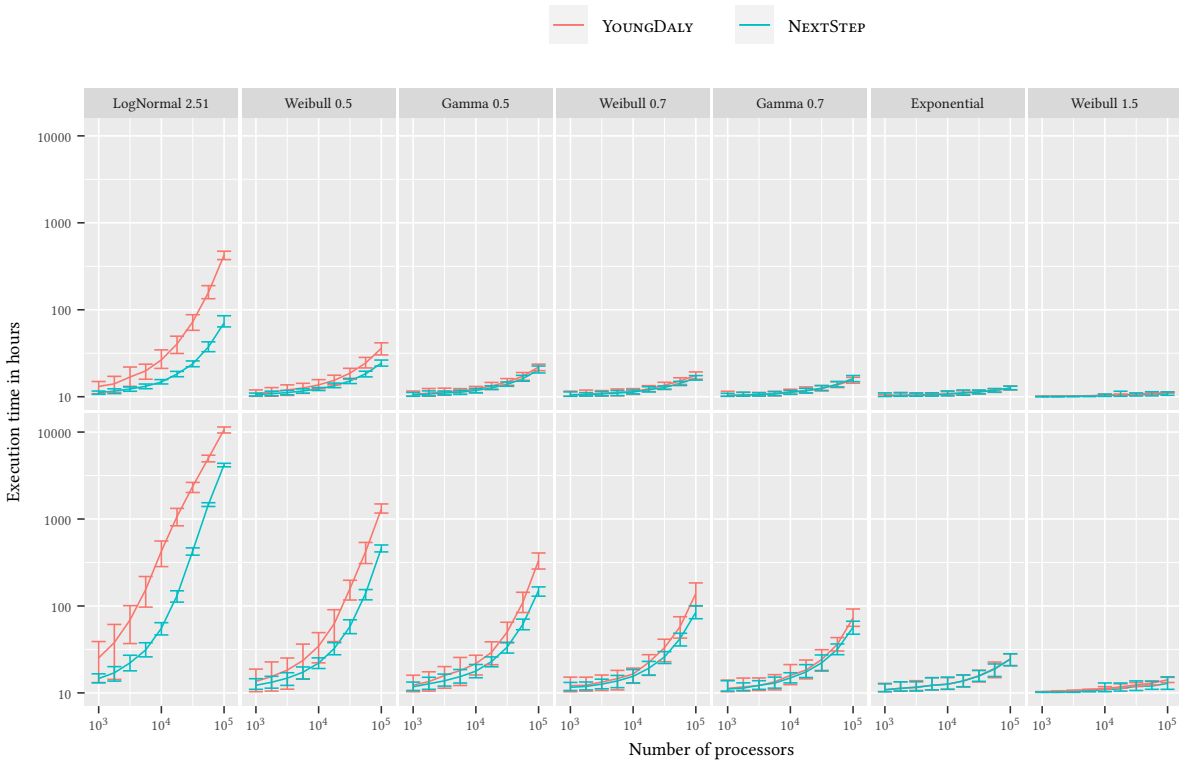

Fig. 14. Expected performance of the three heuristics under all failure distribution on a 30 day old platform and with a workflow W of 10 hours. Top: $C = 60$, bottom: $C = 600$.

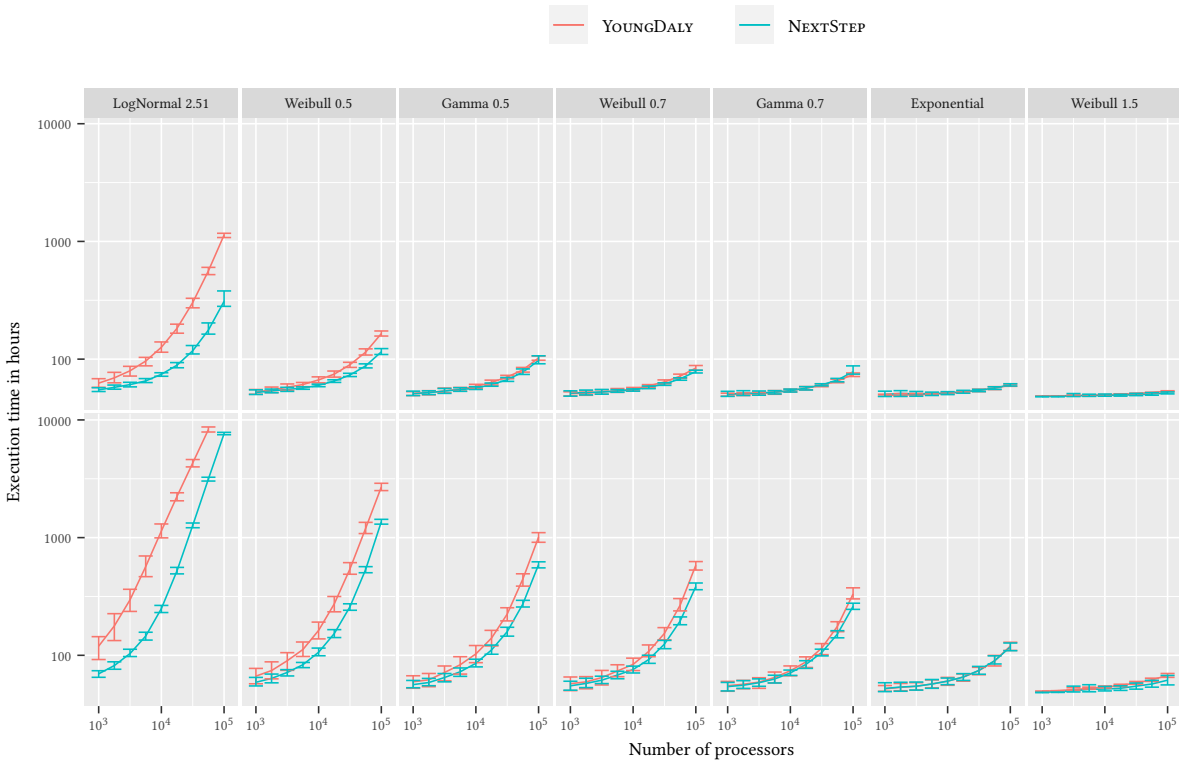

Fig. 15. Expected performance of the three heuristics under all failure distribution on a 30 day old platform and with a workflow W of 48 hours. Top: $C = 60$, bottom: $C = 600$.

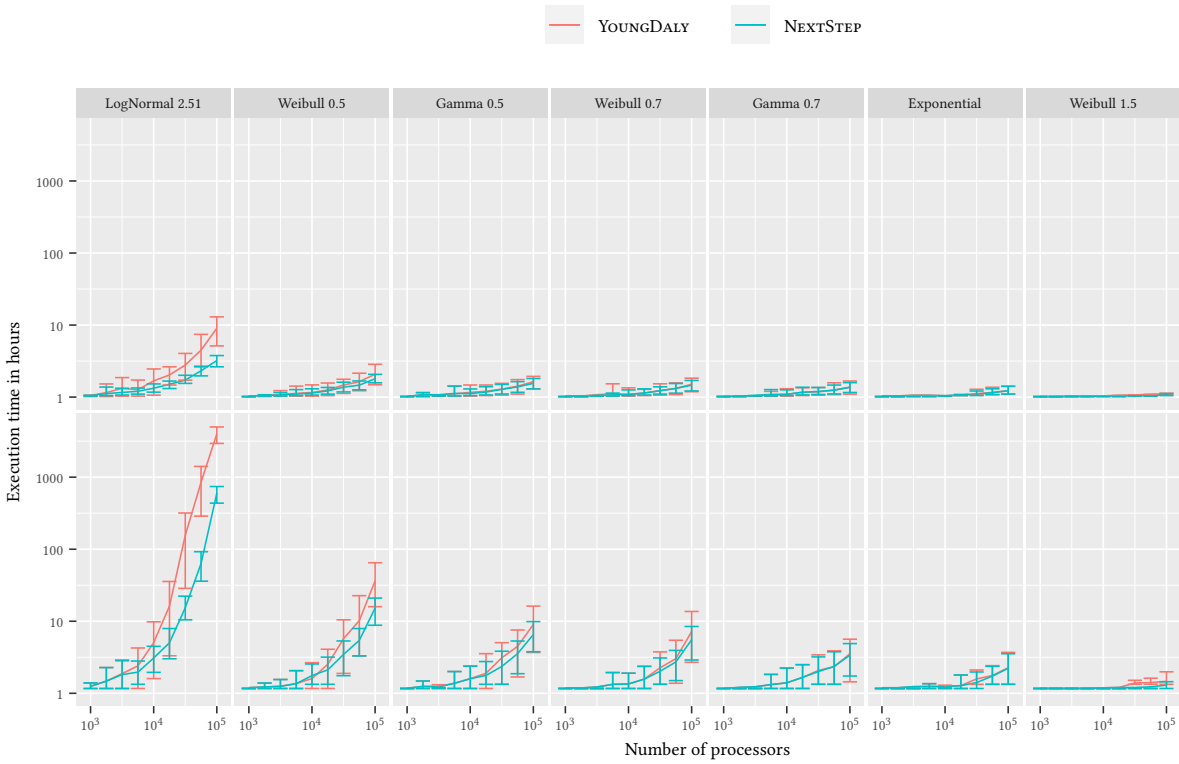

Fig. 16. Expected performance of the three heuristics under all failure distribution on a 100 day old platform and with a workflow W of 1 hour. Top: $C = 60$, bottom: $C = 600$.

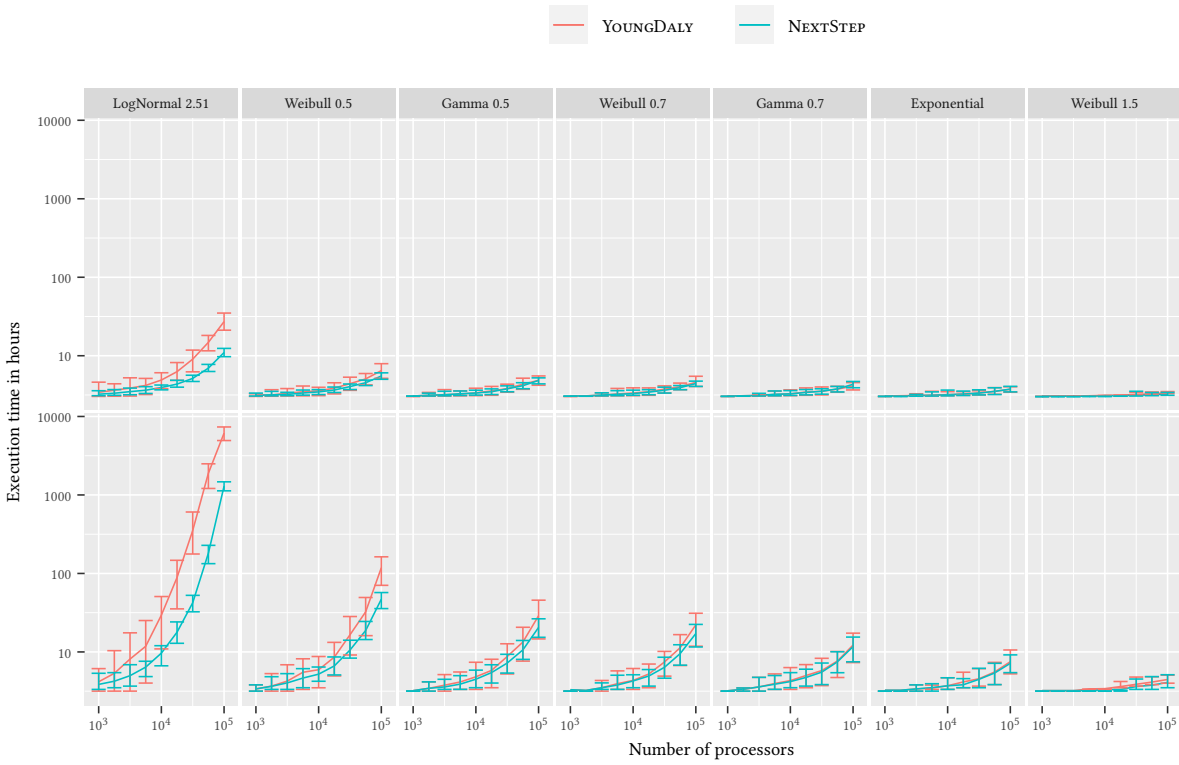

Fig. 17. Expected performance of the three heuristics under all failure distribution on a 100 day old platform and with a workflow W of 3 hours. Top: $C = 60$, bottom: $C = 600$.

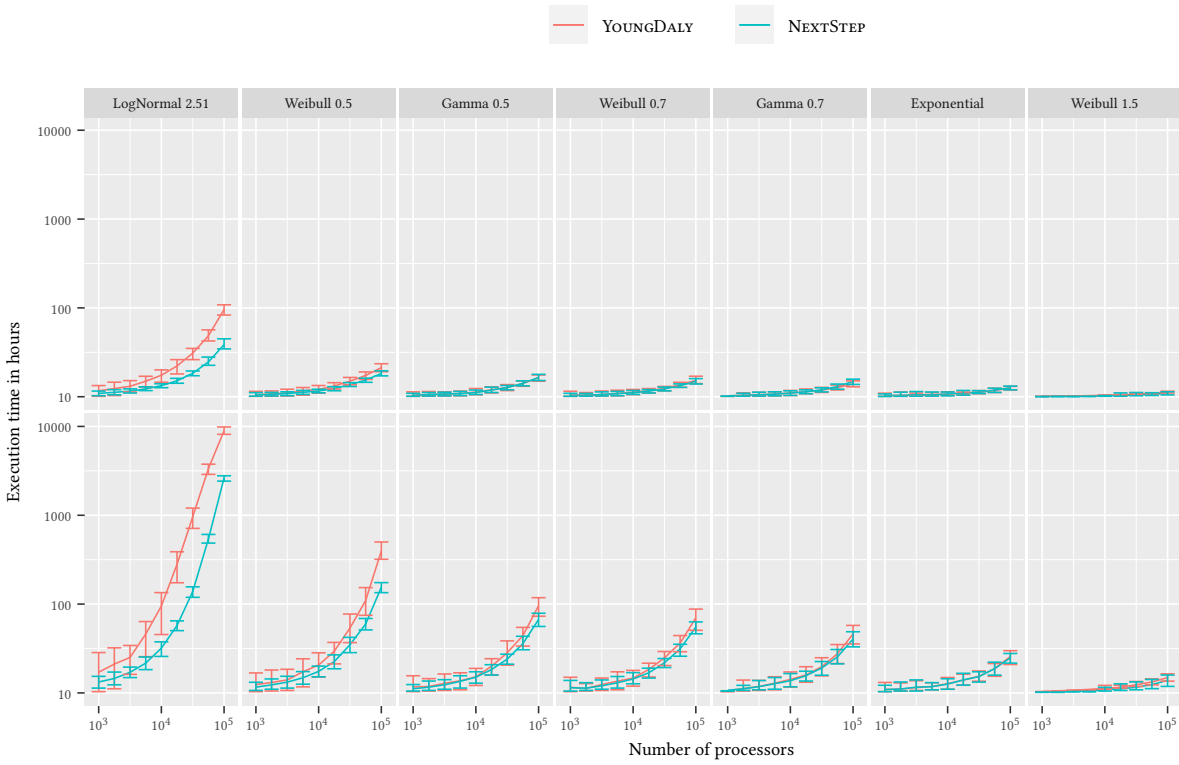

Fig. 18. Expected performance of the three heuristics under all failure distribution on a 100 day old platform and with a workflow W of 10 hours. Top: $C = 60$, bottom: $C = 600$.

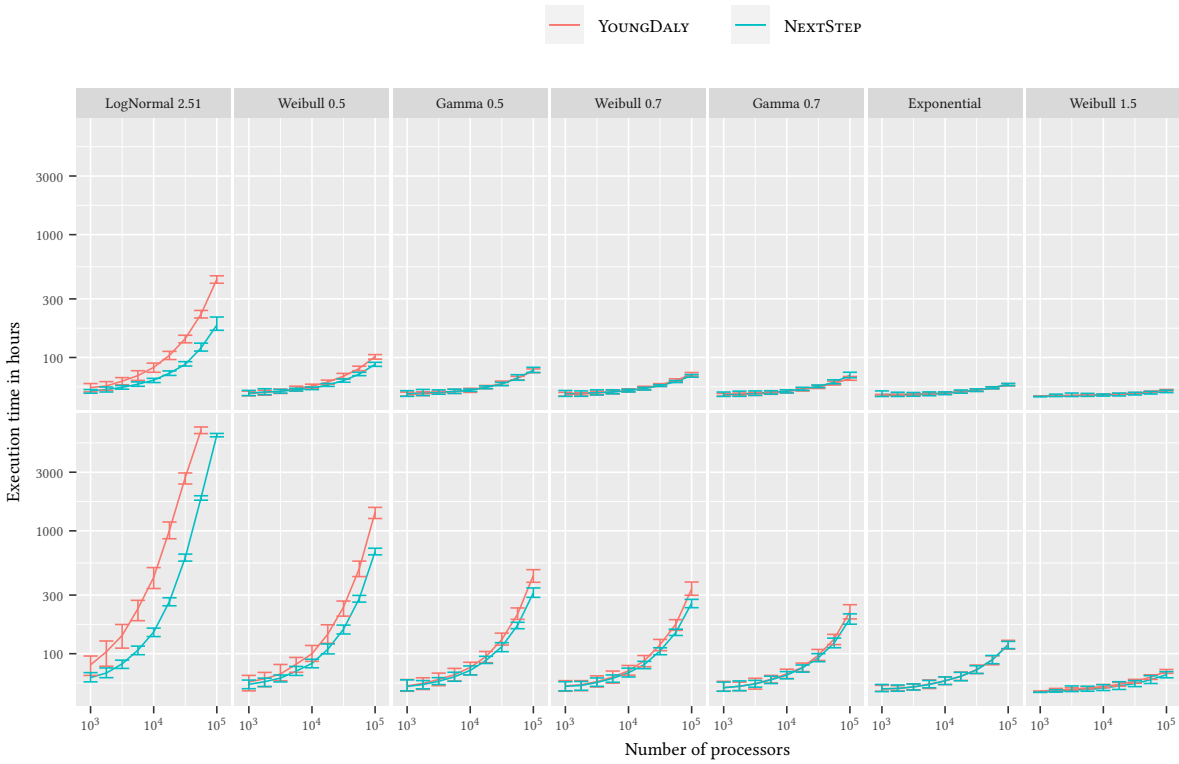

Fig. 19. Expected performance of the three heuristics under all failure distribution on a 100 day old platform and with a workflow W of 48 hours. Top: $C = 60$, bottom: $C = 600$.

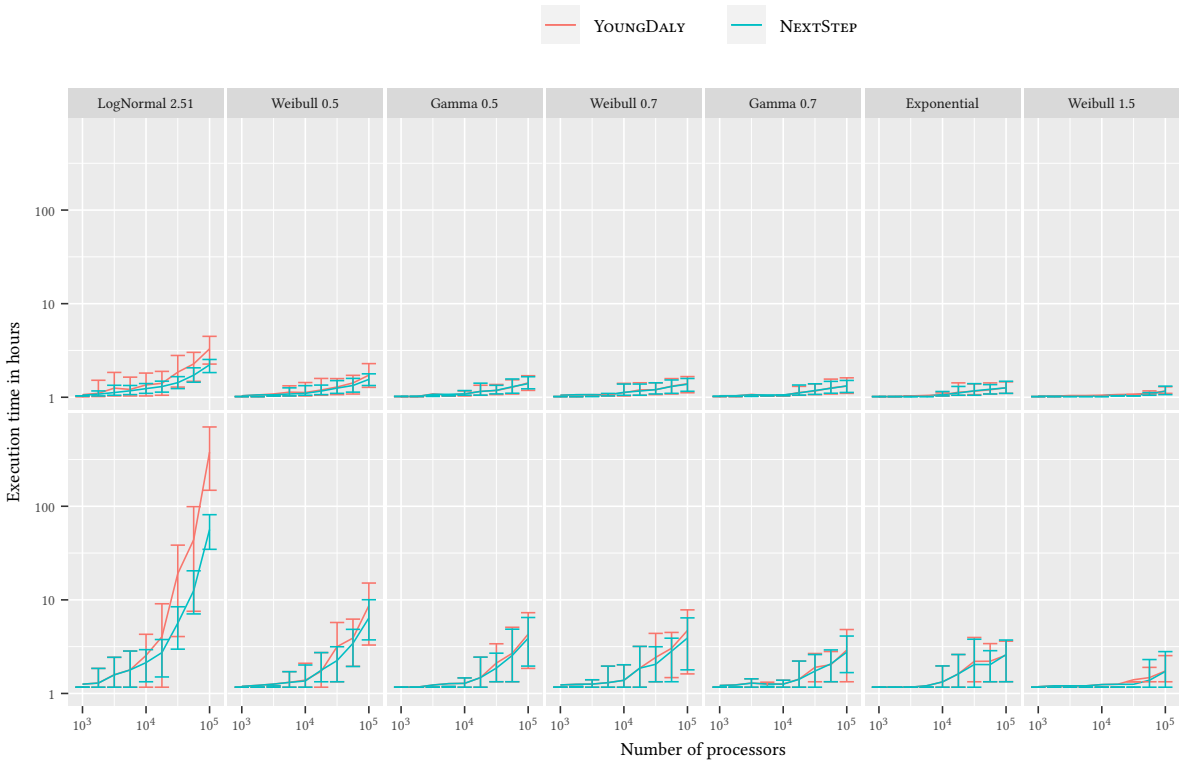

Fig. 20. Expected performance of the three heuristics under all failure distribution on a 365 day old platform and with a workflow W of 1 hour. Top: $C = 60$, bottom: $C = 600$.

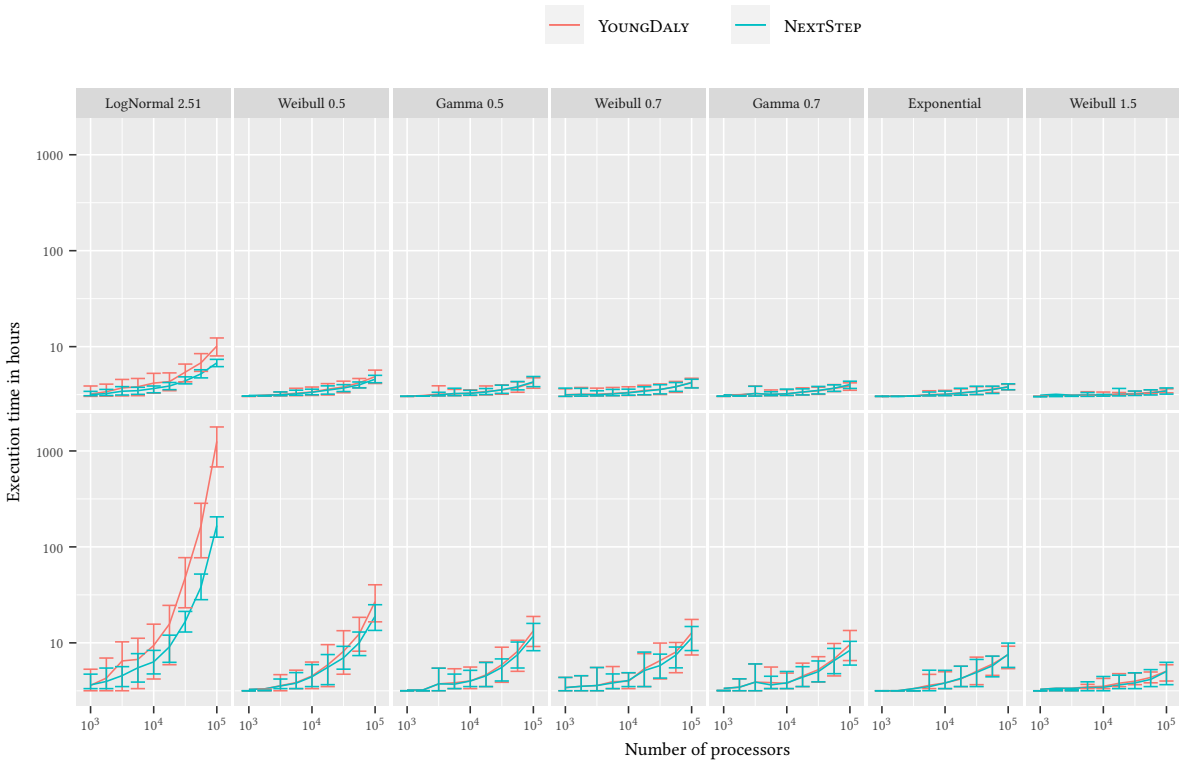

Fig. 21. Expected performance of the three heuristics under all failure distribution on a 365 day old platform and with a workflow W of 3 hours. Top: $C = 60$, bottom: $C = 600$.

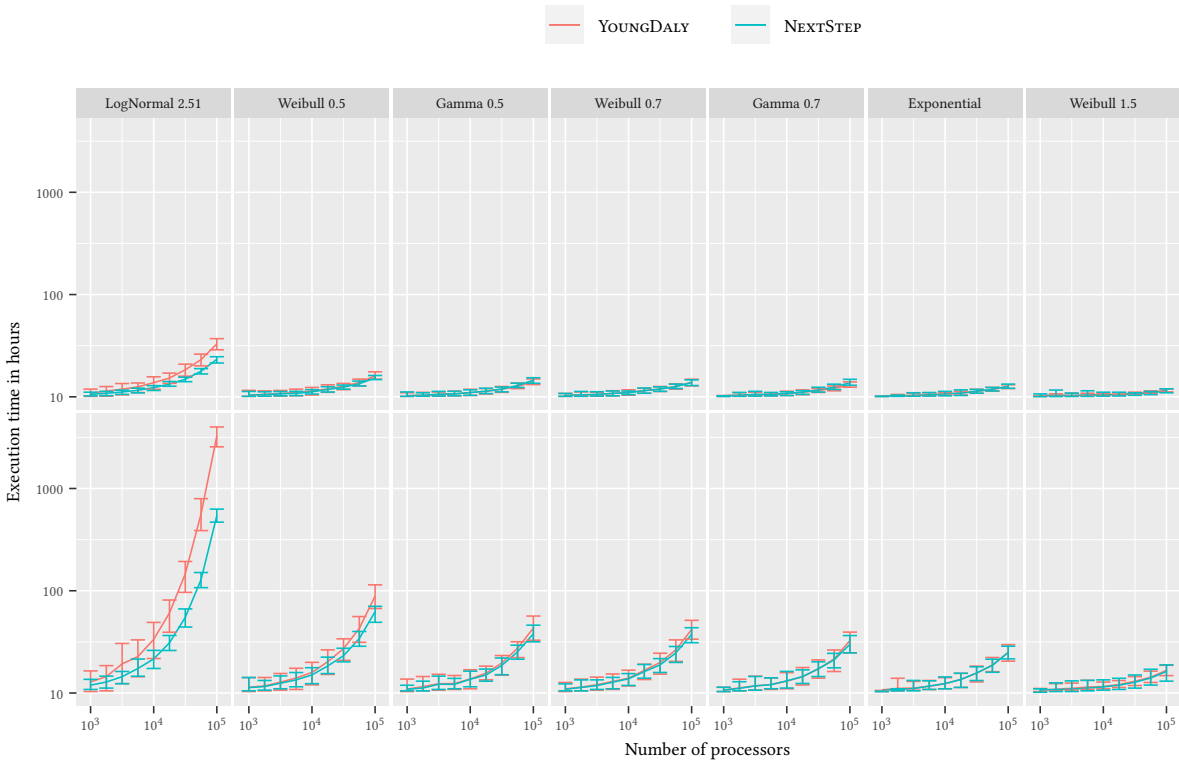

Fig. 22. Expected performance of the three heuristics under all failure distribution on a 365 day old platform and with a workflow W of 10 hours. Top: $C = 60$, bottom: $C = 600$.

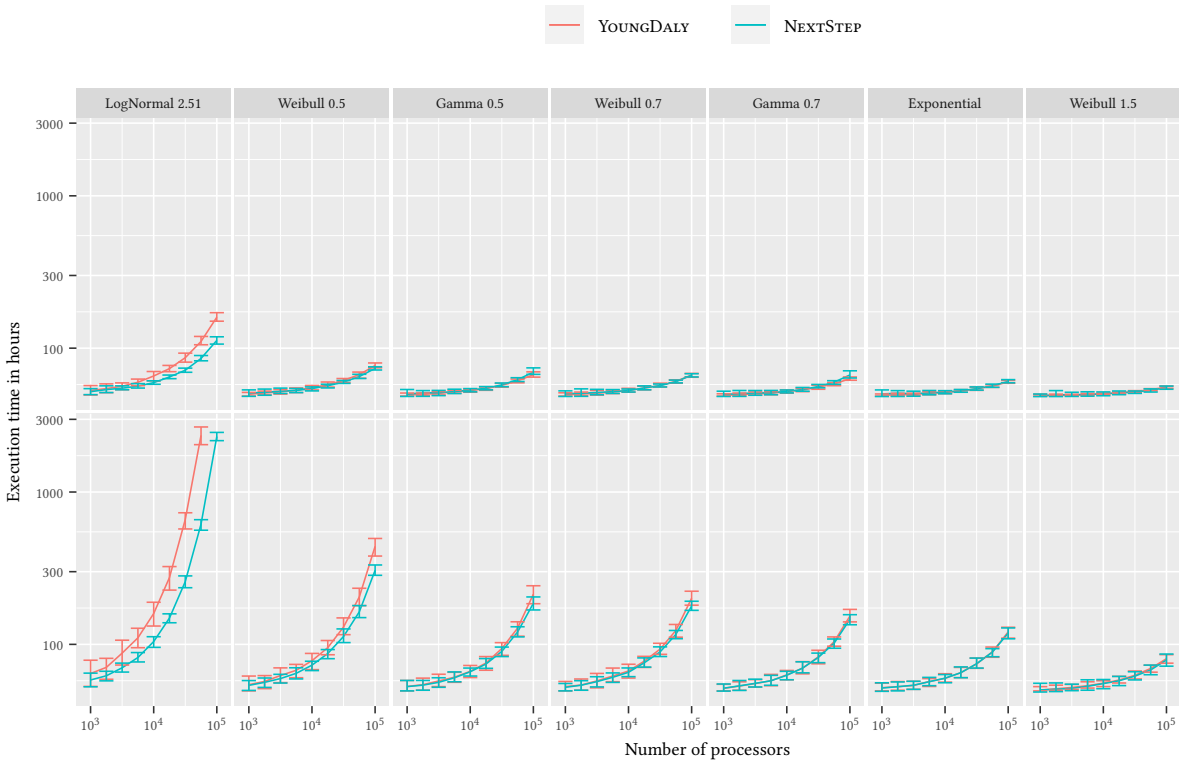

Fig. 23. Expected performance of the three heuristics under all failure distribution on a 365 day old platform and with a workflow W of 48 hours. Top: $C = 60$, bottom: $C = 600$.