# Replication Is More Efficient Than You Think

Anne Benoit*
Anne.Benoit@ens-lyon.fr
Ecole Normale Superieure de Lyon
Lyon, France

Thomas Herault†
herault@icl.utk.edu
The University of Tennessee
Knoxville, Tennessee

Valentin Le Fèvre*
valentin.le-fevre@ens-lyon.fr
Ecole Normale Superieure de Lyon
Lyon, France

Yves Robert*†
Yves.Robert@ens-lyon.fr
Ecole Normale Superieure de Lyon
Lyon, France
The University of Tennessee
Knoxville, Tennessee

## ABSTRACT

This paper revisits replication coupled with checkpointing for fail-stop errors. Replication enables the application to survive many fail-stop errors, thereby allowing for longer checkpointing periods. Previously published works use replication with the *no-restart* strategy, which works as follows: (i) compute the application Mean Time To Interruption (MTTI) $M$ as a function of the number of processor pairs and the individual processor Mean Time Between Failures (MTBF); (ii) use checkpointing period $T^{\text{no}}_{MTTI} = \sqrt{2MC}$ à la Young/Daly, where $C$ is the checkpoint duration; and (iii) never restart failed processors until the application crashes. We introduce the *restart* strategy where failed processors are restarted after each checkpoint. We compute the optimal checkpointing period $T^{\text{rs}}_{opt}$ for this strategy, which is much larger than $T^{\text{no}}_{MTTI}$, thereby decreasing I/O pressure. We show through simulations that using $T^{\text{rs}}_{opt}$ and the *restart* strategy, instead of $T^{\text{no}}_{MTTI}$ and the usual *no-restart* strategy, significantly decreases the overhead induced by replication.

**ACM Reference Format:**
Anne Benoit, Thomas Herault, Valentin Le Fèvre, and Yves Robert. 2019. Replication Is More Efficient Than You Think. In *Proceedings of ACM Conference (SC'19).* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

## 1 INTRODUCTION

Current computing platforms have millions of cores: the Summit system at the Oak Ridge National Laboratory (ORNL) is listed at number one in the TOP500 ranking [38], and it has more than two million cores. The Chinese Sunway TaihuLight (ranked as number 3) has even more than 10 million cores. These large-scale computing systems are frequently confronted with failures, also called fail-stop errors. Indeed, even if individual cores are reliable, for instance if the *Mean Time Between Failures* (MTBF) for a core is $\mu = 10$ years, then the MTBF for a platform with a million cores ($N = 10^6$) becomes $\mu_N = \frac{\mu}{N} \approx 5.2$ minutes, meaning that a failure strikes the platform every five minutes, as shown in [24].

The classical technique to deal with failures consists of using a checkpoint-restart mechanism: the state of the application is periodically checkpointed, and when a failure occurs, we recover from the last valid checkpoint and resume the execution from that point on, rather than starting the execution from scratch. The key for an efficient checkpointing policy is to decide how often to checkpoint. Young [42] and Daly [14] derived the well-known Young/Daly formula $T_{YD} = \sqrt{2\mu_N C}$ for the optimal checkpointing period, where $\mu_N$ is the platform MTBF, and $C$ is the checkpointing duration.

Another technique that has been advocated for dealing with failures is process replication, where each process in a parallel MPI (Message Passing Interface) application is duplicated to increase the *Mean Time To Interruption* (MTTI). The MTTI is the mean time between two application failures. If a process is struck by a failure, the execution can continue until the replica of this process is also struck by a failure. More precisely, processors are arranged by pairs, i.e., each processor has a replica, and the application fails whenever both processors in a same pair have been struck by a failure. With replication, one considers the MTTI rather than the MTBF, because the application can survive many failures before crashing. Given the high rate of failures on large-scale systems, process replication is combined with periodic checkpoint-restart, as proposed for instance in [18, 35, 45] for high-performance computing (HPC) platforms, and in [28, 41] for grid computing. Then, when the application fails, one can recover from the last valid checkpoint, just as was the case without replication. Intuitively, since many failures are needed to interrupt the application, the checkpointing period should be much larger than without replication. Previous works [12, 20, 25] all use $T^{\text{no}}_{MTTI} = \sqrt{2M_N C}$ for the checkpointing period, where $M_N$ is the MTTI with $N$ processors (instead of the MTBF $\mu_N$).

To illustrate the impact of replication on reliability at scale, Figure 1 compares the probability distribution of the time to application failure for: (a) a single processor, two parallel processors and a pair of replicated processors; and (b) a platform of $N = 100,000$ parallel processors, $N = 200,000$ parallel processors without replication, and $b = 100,000$ processor pairs with replication. In all cases, the

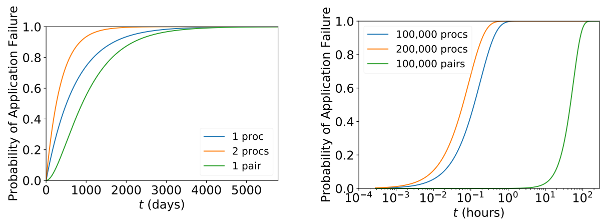
Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

*SC'19, November 2019, Denver, CO, USA*
© 2019 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

individual MTBF of a single processor is $\mu = 5$ years. The time to reach 90% chances of having a fatal failure is: (a) 1688 days for one processor, 844 days for two processors and 2178 days for a processor pair; and (b) 24 minutes for 100,000 processors, 12 minutes for 200,000 processors and 5081 minutes (almost 85 hours) for 100,000 processor pairs. We see that replication is key to safe application progress at scale! Again, the cost is that half of the resources are doing redundant work, hence time-to-solution is increased. We compare time-to-solution with and without replication in Section 7.6. We also see that in heavily failure-prone environments (small MTBF values), checkpoint/restart alone cannot ensure full reliability, and must be complemented by replication.

One major contribution of this paper is to introduce a new approach that minimizes the overhead incurred by the checkpoint-restart mechanism when coupled with replication. Previous works [12, 20, 25] use the *no-restart* strategy: if a processor was struck by a failure (but not its replica), then the processor remains failed (no recovery) until the whole application fails. Hence, there is a recovery only every $M_N$ seconds on average, whenever the application fails. Many periodic checkpoints are taken in between two application crashes, with more and more processors failing on the fly. To the best of our knowledge, analytically computing the optimal period for *no-restart* is an open problem (see Section 4.2 for more details, where we also show that non-periodic strategies are more efficient for *no-restart*), but simulations can help assess this approach.

The study of the *no-restart* strategy raises an important question: should failed processors be restarted earlier on in the execution? Instead of waiting for an application crash to rejuvenate the whole platform, a simple approach would be to restart processors immediately after each failure. Let *restart-on-failure* denote this strategy. It ensures that all processor pairs involve two live processors throughout execution, and would even suppress the notion of checkpointing periods. Instead, after each failure striking a processor, its replica would checkpoint immediately, and the spare processor replacing the failed processor would read that checkpoint to resume execution. There is a small risk of fatal crash if a second failure should strike the replica when writing its checkpoint, but (i) the risk is very small because the probability of such a cascade of two narrowly spaced failures is quite low; and (ii) if the checkpoint protocol is scalable, every other processor can checkpoint in parallel with the replica, and there is no additional time overhead. With tightly coupled applications, the other processors would likely have to wait until the spare is able to restart, and they can checkpoint instead of idling during that wait. While intuitively appealing, the *restart-on-failure* strategy may lead to too many checkpoints and restarts, especially in scenarios when failures strike frequently. However, frequent failures were exactly the reason to deploy replication in the first place, precisely to avoid having to restart after each failure.



**(a) CDFs of the probability distribution of time to app. failure for one processor, two parallel processors and one proc. pair (replication).**

**(b) CDFs of the proba. distrib. of time to app. failure for 100,000 parallel proc., 200,0000 parallel proc. and 100,000 proc. pairs (replication).**

**Figure 1: Comparison of CDFs with and without replication.**

In this work, we introduce the *restart* strategy, which requires any failed processor to recover each time a checkpoint is taken. This ensures that after any checkpoint at the end of a successful period, all processors are alive. This is a middle ground between the *no-restart* and *restart-on-failure* strategies, because failed processors are restarted at the end of each period with *restart.* On the one hand, a given period may well include many failures, hence *restart* restarts processors less frequently than *restart-on-failure.* On the other hand, there will be several periods in between two application crashes, hence *restart* restarts processors more frequently than *no-restart.*

Periodic checkpointing is optimal with the *restart* strategy: the next period should have same length as the previous one, because we have the same initial conditions at the beginning of each period. Restarting failed processors when checkpointing can introduce additional overhead, but we show that it is very small, and even non-existent when in-memory (a.k.a. buddy) checkpointing is used as the first-level of a hierarchical multi-level checkpointing protocol (such state-of-the-art protocols are routinely deployed on large-scale platforms [3, 11, 29]). A key contribution of this paper is a mathematical analysis of the *restart* strategy, with a closed-form formula for its optimal checkpointing period. We show that the optimal checkpointing period for the *restart* strategy has the order $\Theta(\mu^{\frac{2}{3}})$, instead of the $\Theta(\mu^{\frac{1}{2}})$ used in previous works for *no-restart* as an extension of the Young/Daly formula [12, 20, 25]. Hence, as the error rate increases, the optimal period becomes much longer than the value that has been used in all previous works (with *no-restart*). Consequently, checkpoints are much less frequent, thereby dramatically decreasing the pressure on the I/O system.

The main contributions of this paper are the following:

- We provide the first closed-form expression of the application MTTI $M_N$ with replication;
- We introduce the *restart* strategy for replication, where we recover failed processors during each checkpoint;
- We formally analyze the *restart* strategy, and provide the optimal checkpointing period with this strategy;
- We apply these results to applications following Amdahl's law, i.e., applications that are not fully parallel but have an inherent sequential part, and compare the time-to-solution achieved with and without replication;
- We validate the model through comprehensive simulations, by showing that analytical results, using first-order approximations and making some additional assumptions (no failures during checkpoint and recovery), are quite close to simulation results; for these simulations, we use both randomly generated failures and log traces.
- We compare through simulations the overhead obtained with the optimal strategy introduced in this work (*restart* strategy, optimal checkpointing period) to those used in all previous works (*no-restart* strategy, extension of the Young/Daly checkpointing period), as

well as with strategies that use partial replication or that restart only at some of the checkpoints, and demonstrate that we can significantly decrease both total execution time and utilization of the I/O file system.

The paper is organized as follows. We first describe the model in Section 2. We recall how to compute the optimal checkpointing period when no replication is used in Section 3. The core contribution is presented in Section 4, where we explain how to compute the MTTI with $b$ $(= \frac{N}{2})$ processor pairs, detail the *restart* strategy, and show how to derive the optimal checkpointing period with this *restart* strategy. Results are applied to applications following Amdahl's law in Section 5. An asymptotic analysis of *no-restart* and *restart* is provided in Section 6. The experimental evaluation in Section 7 presents extensive simulation results, demonstrating that replication is indeed *more efficient than you think*, when enforcing the *restart* strategy instead of the *no-restart* strategy. Finally, we discuss related work in Section 8, and conclude in Section 9.

## 2 MODEL

This section describes the model, with an emphasis on the cost of a combined checkpoint-restart operation.

*Fail-stop errors.* Throughout the text, we consider a platform with $N$ identical processors. The platform is subject to fail-stop errors, or failures, that interrupt the application. Similarly to previous work [17, 20, 25], for the mathematical analysis, we assume that errors are independent and identically distributed (IID), and that they strike each processor according to an exponential probability distribution $\exp(\lambda)$ with support $[0, \infty)$, probability density function (PDF) $f(t) = \lambda e^{-\lambda t}$ and cumulative distribution function (CDF) $F(T) = \mathbb{P}(X \leq T) = 1 - e^{-\lambda T}$. We also introduce the reliability function $G(T) = 1 - F(T) = e^{-\lambda T}$. The expected value $\mu = \frac{1}{\lambda}$ of the $\exp(\lambda)$ distribution is the MTBF on one processor. We lift the IID assumption in the performance evaluation section by using trace logs from real platforms.

*Checkpointing.* To cope with errors, we use periodic coordinated checkpointing. We assume that the divisible application executes for a very long time (asymptotically infinite) and we partition the execution into periods. Each period $\mathcal{P}$ consists of a work segment of duration $T$ followed by a checkpoint of duration $C$. After an error, there is a downtime of duration $D$ (corresponding to the time needed to migrate to a spare processor), a recovery of size $R$, and then one needs to re-execute the period from its beginning.

*Replication.* We use another fault tolerance technique, namely replication. Each process has a replica, which follows the exact same states in its execution. To ensure this, when a process receives a message, its replica also receives the same message, and messages are delivered in the same order to the application (an approach called *active* replication; see [20, 23]). If a crash hits a process at any time, and its replica is still alive, the replica continues the execution alone until a new process can replace the dead one.

We rely on the traditional process allocation strategy that assigns processes and their replicas on remote parts of the system (typically different racks) [9]. This strategy mitigates the risk that a process and its replica would both fail within a short time interval (much shorter than the expected MTTI). As stated in [16], when failure correlations are observed, their correlation diminishes when the processes are far away from each other in the memory hierarchy, and becomes undistinguishable from the null hypothesis (no correlation) when processes belong to different racks.

*Combined checkpoint-restart.* In this paper, we propose the *restart* strategy where failed processes are restarted as soon as the next checkpoint wave happens. When that happens, and processes need to be restarted, the cost of a checkpoint and restart wave, $C^R$, is then increased: one instance of each surviving process must save their state, then processes for the missing instances of the replicas must be allocated; the new processes must load the current state, which has been checkpointed, and join the system to start acting as a replica. The first part of the restart operation, allocating processes to replace the failed ones, can be managed in parallel with the checkpoint of the surviving processes. Using spare processes, this allocation time can be very small and we will consider it negligible compared to the checkpoint saving and loading times. Similarly, integrating the newly spawned process inside the communication system when using spares is negligible when using mechanisms such as the ones described in [8].

There is a large variety of checkpointing libraries and approaches to help applications save their state. [3, 11, 29] are typically used in HPC systems for coordinated checkpointing, and use the entire memory hierarchy to speed up the checkpointing cost: the checkpoint is first saved on local memory, then uploaded onto local storage (SSD, NVRAM if available), and eventually to the shared file system. As soon as a copy of the state is available on the closest memory, the checkpoint is considered as taken. Loading that checkpoint requires that the application state from the closest memory be sent to the memory of the new hosting process.

Another efficient approach to checkpoint is to use in-memory checkpoint replication using the memory of a 'buddy' process (see [31, 44]). To manage the risk of losing the checkpoint in case of failure of two buddy processes, the checkpoint must also be saved on reliable media, as is done in the approaches above. Importantly, in-memory checkpointing is particularly fitted for the *restart* strategy, because the buddy process and the replica are the same process: in that case, the surviving processes upload their checkpoint directly onto the memory of the newly spawned replicas; as soon as this communication is done, the processes can continue working. Contrary to traditional buddy checkpointing, it is not necessary to exchange the checkpoints between a pair of surviving buddies since, per the replication technique, both checkpoints are identical.

In the worst case, if a sequential approach is used, combining checkpointing and restart takes at most twice the time to checkpoint only; in the best case, using buddy checkpointing, the overhead of adding the restart to the checkpoint is negligible. We consider the full spectrum $C \leq C^R \leq 2C$ in the simulations.

As discussed in [20, 32], checkpoint time varies significantly depending upon the target application and the hardware capabilities. We will consider a time to checkpoint within two reasonable limits: $60s \leq C \leq 600s$, following [25].

*First-order approximation.* Throughout the paper, we are interested in first-order approximations, because exact formulas are not analytically tractable. We carefully state the underlying hypotheses that are needed to enforce the validity of first-order results.

Basically, the first-order approximation will be the first, and most meaningful, term of the Taylor expansion of the overhead occurring every period when the error rate $\lambda$ tends to zero.

## 3 BACKGROUND

In this section, we briefly summarize well-known results on the optimal checkpointing period when replication is not used, starting with a single processor in Section 3.1, and then generalizing to the case with $N$ processors in Section 3.2.

### 3.1 With a Single Processor

We aim at computing the expected time $\mathbb{E}(T)$ to execute a period of length $\mathcal{P} = T + C$. The optimal period length will be obtained for the value of $T$, minimizing the overhead

$$\mathbb{H}(T) = \frac{\mathbb{E}(T)}{T} - 1. \tag{1}$$

We temporarily assume that fail-stop errors strike only during work $T$ and not during checkpoint $C$ nor recovery $R$. The following recursive equation is the key to most derivations:

$$\mathbb{E}(T) = (1 - F(T))(T + C) + F(T)(T_{\text{lost}}(T) + D + R + \mathbb{E}(T)). \tag{2}$$

Equation (2) reads as follows: with probability $1 - F(T)$, the execution is successful and lasts $T + C$ seconds; with probability $F(T)$, an error strikes before completion, and we need to account for time lost $T_{\text{lost}}(T)$, downtime $D$ and recovery $R$ before starting the computation anew. The expression for $T_{\text{lost}}(T)$ is the following:

$$T_{\text{lost}}(T) = \int_0^{\infty} t\mathbb{P}(X = t | X \leq T)dt = \frac{1}{F(T)} \int_0^T tf(t)dt.$$

Integrating by parts and re-arranging terms in Equation (2), we derive $\mathbb{E}(T) = T + C + \frac{F(T)}{1-F(T)}(T_{\text{lost}}(T) + D + R)$ and $\mathbb{H}(T) = \frac{C}{T} + \frac{F(T)}{T(1-F(T))}(D+R) + \frac{\int_0^T G(t)dt}{T(1-F(T))} - 1$. Now, if we instantiate the value of $F(T) = 1 - G(T) = 1 - e^{-\lambda T}$, we obtain $\mathbb{H}(T) = \frac{C}{T} + \frac{e^{\lambda T}-1}{T}(D + R + \frac{1}{\lambda}) - 1$. We can find the value $T_{opt}$ by differentiating and searching for the zero of the derivative, but the solution is complicated as it involves the Lambert function [14, 24]. Instead, we use the Taylor expansion of $e^{-\lambda T} = \sum_{i=0}^{\infty}(-1)^i \frac{(\lambda T)^i}{i!}$ and the approximation $e^{-\lambda T} = 1 - \lambda T + \frac{(\lambda T)^2}{2} + o(\lambda^2 T^2)$. This makes sense only if $\lambda T$ tends to zero. It is reasonable to make this assumption, since the length of the period $\mathcal{P}$ must be much smaller than the error MTBF $\mu = \frac{1}{\lambda}$. Hence, we look for $T = \Theta(\lambda^{-x})$, where $0 < x < 1$. Note that $x$ represents the order of magnitude of $T$ as a function of the error rate $\lambda$. We can then safely write

$$\mathbb{H}(T) = \frac{C}{T} + \frac{\lambda T}{2} + o(\lambda T). \tag{3}$$

Now, $\frac{C}{T} = \Theta(\lambda^x)$ and $\frac{\lambda T}{2} = \Theta(\lambda^{1-x})$, hence the order of magnitude of the overhead is $\mathbb{H}(T) = \Theta(\lambda^{\max(x,1-x)})$, which is minimum for $x = \frac{1}{2}$. Differentiating Equation (3), we obtain

$$T_{opt} = \sqrt{\frac{2C}{\lambda}} = \Theta(\lambda^{-\frac{1}{2}}), \text{ and } \mathbb{H}_{opt} = \sqrt{2C\lambda} + o(\lambda^{\frac{1}{2}}) = \Theta(\lambda^{\frac{1}{2}}) \tag{4}$$

which is the well-known and original Young formula [42].

Variants of Equation (4) have been proposed in the literature, such as $T_{opt} = \sqrt{2(\mu + R)C}$ in [14] or $T_{opt} = \sqrt{2(\mu - D - R)C} - C$ in [24]. All variants are approximations that collapse to Equation (4). This is because the resilience parameters $C$, $D$, and $R$ are constants and thus negligible in front of $T_{opt}$ when $\lambda$ tends to zero. This also explains that assuming that fail-stop errors may strike during checkpoint or recovery has no impact on the first-order approximation of the period given in Equation (4). For instance, assuming that fail-stop errors strike during checkpoints, we would modify Equation (2) into

$$\mathbb{E}(T{+}C) = (1{-}F(T{+}C))(T{+}C){+}F(T{+}C)(T_{\text{lost}}(T{+}C){+}D{+}R{+}\mathbb{E}(T{+}C))$$

and derive the same result as in Equation (4). Similarly, assuming that fail-stop errors strike during recovery, we would replace $R$ with $\mathbb{E}(R)$, which can be computed via an equation similar to that for $\mathbb{E}(T)$, again without modifying the final result.

Finally, a very intuitive way to retrieve Equation (4) is the following: consider a period of length $\mathcal{P} = T + C$. There is a failure-free overhead $\frac{C}{T}$, and a failure-induced overhead $\frac{1}{\mu} \times \frac{T}{2}$, because with frequency $\frac{1}{\mu}$ an error strikes, and on average it strikes in the middle of the period and we lose half of it. Adding up both overhead sources gives

$$\frac{C}{T} + \frac{T}{2\mu}, \tag{5}$$

which is minimum when $T = \sqrt{2\mu C}$. While not fully rigorous, this derivation helps understand the tradeoff related to the optimal checkpointing frequency.

### 3.2 With *N* Processors

The previous analysis can be directly extended to multiple processors. Indeed, if fail-stop errors strike each processor according to an $\exp(\lambda)$ probability distribution, then these errors strike the whole platform made of $N$ identical processors according to an $\exp(N\lambda)$ probability distribution [24]. In other words, the platform MTBF is $\mu_N = \frac{\mu}{N}$, which is intuitive: the number of failures increases linearly with the number of processors $N$, hence the mean time between two failures is divided by $N$. All previous derivations apply, and we obtain the optimal checkpointing period and overhead:

$$T_{opt} = \sqrt{\frac{2C}{N\lambda}} = \Theta(\lambda^{-\frac{1}{2}}), \text{ and } \mathbb{H}_{opt} = \sqrt{2CN\lambda} + o(\lambda^{\frac{1}{2}}) = \Theta(\lambda^{\frac{1}{2}}) \tag{6}$$

This value of $T_{opt}$ can be intuitively retrieved with the same (not fully rigorous) reasoning as before (Equation (5)): in a period of length $\mathcal{P} = T + C$, the failure-free overhead is $\frac{C}{T}$, and the failure-induced overhead becomes $\frac{1}{\mu_N} \times \frac{T}{2}$: we factor in an updated value of the failure frequency, using $\frac{1}{\mu_N} = \frac{N}{\mu}$ instead of $\frac{1}{\mu}$. Both overhead sources add up to

$$\frac{C}{T} + \frac{T}{2\mu_N} = \frac{C}{T} + \frac{NT}{2\mu}, \tag{7}$$

which is minimum when $T = \sqrt{\frac{2\mu C}{N}}$.

## 4 REPLICATION

This section deals with process replication for fail-stop errors, as introduced in [20] and recently revisited by [25]. We consider a platform with $N = 2b$ processors. Exactly as in Section 3, each processor fails according to a probability distribution $\exp(\lambda)$, and the platform MTBF is $\mu_N = \frac{\mu}{N}$. We still assume that checkpoint and recovery are error-free: it simplifies the analysis without modifying the first-order approximation of the optimal checkpointing period.

Processors are arranged by pairs, meaning that each processor has a replica. The application executes as if there were only $b$ available processors, hence with a reduced throughput. However, a single failure does not interrupt the application, because the replica of the failed processor can continue the execution. The application can thus survive many failures, until both replicas of a given pair are struck by a failure. How many failures are needed, in expectation, to interrupt the application? We compute this value in Section 4.1. Then, we proceed to deriving the optimal checkpointing period, first with one processor pair in Section 4.2, before dealing with the general case in Section 4.3.

## 4.1 Computing the Mean Time To Interruption

Let $n_{\text{fail}}(2b)$ be the expected number of failures to interrupt the application, with $b$ processor pairs. Then, the application MTTI $M_{2b}$ with $b$ processor pairs (hence $N = 2b$ processors) is given by

$$M_{2b} = n_{\text{fail}}(2b)\,\mu_{2b} = n_{\text{fail}}(2b)\frac{\mu}{2b} = \frac{n_{\text{fail}}(2b)}{2\lambda b}, \quad (8)$$

because each failure strikes every $\mu_{2b}$ seconds in expectation. Computing the value of $n_{\text{fail}}(2b)$ has received considerable attention in previous work. In [20, 34], the authors made an analogy with the birthday problem and use the Ramanujan function [21] to derive the formula $n_{\text{fail}}(2b) = 1 + \sum_{k=0}^{b} \frac{b!}{(b-k)!b;k} \approx \sqrt{\frac{\pi b}{2}}$. The analogy is not fully correct, because failures can strike either replica of a pair. A correct recursive formula is provided in [12], albeit without a closed-form expression. Recently, the authors in [25] showed that

$$n_{\text{fail}}(2b) = 2b4^b \int_0^{\frac{1}{2}} x^{b-1}(1-x)^b dx \quad (9)$$

but did not give a closed-form expression either. We provide such an expression below:

Theorem 4.1.

$$n_{fail}(2b) = 1 + 4^b \Big/ \binom{2b}{b}. \quad (10)$$

The proof, which uses the incomplete Beta function [39, 40], can be found in the companion research report [6, 7]. Using Sterling's formula, we easily derive that $n_{\text{fail}}(2b) \approx \sqrt{\pi b}$, which is 40% more than the value $\sqrt{\frac{\pi b}{2}}$ used in [20, 34].

Plugging the value of $n_{\text{fail}}(2b)$ back in Equation (8) gives the value of the MTTI $M_{2b}$. As already mentioned, previous works [12, 20, 25] all use the checkpointing period

$$T^{\text{no}}_{MTTI} = \sqrt{2M_{2b}C} \quad (11)$$

to minimize execution time overhead. This value follows from the same derivation as in Equations (5) and (7). Consider a period of length $\mathcal{P} = T + C$. The failure-free overhead is still $\frac{C}{T}$, and the failure-induced overhead becomes $\frac{1}{M_{2b}} \times \frac{T}{2}$: we factor in an updated value of the failure frequency, which now becomes the fatal failure frequency, namely $\frac{1}{M_{2b}}$. Both overhead sources add up to

$$\frac{C}{T} + \frac{T}{2M_{2b}}, \quad (12)$$

which is minimum when $T = \sqrt{2M_{2b}C}$.

In the following, we analyze the *restart* strategy. We start with one processor pair ($b = 1$) in Section 4.2, before dealing with the general case in Section 4.3.

## 4.2 With One Processor Pair

We consider two processors working together as replicas. The failure rate is $\lambda = \frac{1}{\mu}$ for each processor, and the pair MTBF is $\mu_2 = \frac{\mu}{2}$, while the pair MTTI is $M_2 = \frac{3\mu}{2}$ because $n_{\text{fail}}(2) = 3$. We analyze the *restart* strategy, which restarts a (potentially) failed processor at every checkpoint. Hence, the checkpoint has duration $C^R$ and not $C$. Consider a period of length $\mathcal{P} = T + C^R$. If one processor fails before the checkpoint but the other survives until reaching it, the period is executed successfully. The period is re-executed only when both processors fail within $T$ seconds. Let $p_1(T)$ denote the probability that both processors fail during $T$ seconds: $p_1(T) = (1 - e^{-\lambda T})^2$. We compute the expected time $\mathbb{E}(T)$ for period of duration $\mathcal{P} = T + C^R$ using the following recursive equation:

$$\mathbb{E}(T) = (1 - p_1(T))(T + C^R) + p_1(T)(T_{\text{lost}}(T) + D + R + \mathbb{E}(T)). \quad (13)$$

Here, $C^R$ denotes the time to checkpoint, and in addition, to recover whenever one of the two processors had failed during the period. As discussed in Section 2, we have $C \leq C^R \leq C + R$: the value of $C^R$ depends upon the amount of overlap between the checkpoint and the possible recovery of one processor.

Consider the scenario where one processor fails before reaching the end of the period, while the other succeeds and takes the checkpoint. The *no-restart* strategy continues execution, hence pays only for a regular checkpoint of cost $C$, and when the live processor is struck by a failure (every $M_2$ seconds on average), we roll back and recover for both processors [12, 20, 25]. However, the new *restart* strategy requires any failed processor to recover whenever a checkpoint is taken, hence at a cost $C^R$. This ensures that after any checkpoint at the end of a successful period, we have two live processors, and thus the same initial conditions. Hence, periodic checkpointing is optimal with this strategy. We compare the *restart* and *no-restart* strategies through simulations in Section 7.

As before, in Equation (13), $T_{\text{lost}}(T)$ is the average time lost, knowing that both processors have failed before $T$ seconds. While $T_{\text{lost}}(T) \sim \frac{T}{2}$ when considering a single processor, it is no longer the case with a pair of replicas. Indeed, we compute $T_{\text{lost}}(T)$ as follows:

$$\begin{aligned} T_{\text{lost}}(T) &= \int_0^\infty t\mathbb{P}(X = t | t \leq T)dt = \frac{1}{p_1(T)} \int_0^T t \frac{d\mathbb{P}(X \leq t)}{dt} dt \\ &= \frac{2\lambda}{(1 - e^{-\lambda T})^2} \int_0^T t(e^{-\lambda t} - e^{-2\lambda t})dt. \end{aligned}$$

After integration, we find that

$$T_{\text{lost}}(T) = \frac{(2e^{-2\lambda T} - 4e^{-\lambda T})\lambda T + e^{-2\lambda T} - 4e^{-\lambda T} + 3}{2\lambda(1 - e^{-\lambda T})^2} = \frac{1}{2\lambda}\frac{u(\lambda T)}{v(\lambda T)},$$

with $u(y) = (2e^{-2y} - 4e^{-y})y + e^{-2y} - 4e^{-y} + 3$ and $v(y) = (1 - e^{-y})^2$.

Assuming that $T = \Theta(\lambda^{-x})$ with $0 < x < 1$ as in Section 3.1, then Taylor expansions lead to $u(y) = \frac{4}{3}y^3 + o(y^3)$ and $v(y) = y^2 + y^3 + o(y^3)$ for $y = \lambda T = o(1)$, meaning that $T_{\text{lost}}(T) = \frac{1}{2\lambda}\frac{\frac{4\lambda T}{3} + o(\lambda T)}{1 + \lambda T + o(\lambda T)}$. Using the division rule, we obtain $T_{\text{lost}}(T) = \frac{1}{2\lambda}(\frac{4\lambda T}{3} + o(\lambda T)) = \frac{2T}{3} + o(T)$. Note that we lose two thirds of the period with a processor pair rather than one half with a single processor. Plugging back the value of $T_{\text{lost}}(T)$ and solving, we obtain:

$$\mathbb{E}(T) = T+C^R+(D+R+\frac{(2e^{-2\lambda T} - 4e^{-\lambda T})\lambda T + e^{-2\lambda T} - 4e^{-\lambda T} + 3}{2\lambda(1 - e^{-\lambda T})^2})\cdot\frac{(e^{\lambda T} - 1)^2}{2e^{\lambda T} - 1}. \tag{14}$$

We then compute the waste $\mathbb{H}^{\text{rs}}(T)$ of the *restart* strategy as follows:

$$\mathbb{H}^{\text{rs}}(T) = \frac{\mathbb{E}(T)}{T} - 1 = \frac{C^R}{T} + \frac{2}{3}\lambda^2T^2 + o(\lambda^2T^2). \tag{15}$$

Moreover, with $T = \Theta(\lambda^{-x})$, we have $\frac{C^R}{T} = \Theta(\lambda^x)$ and $\frac{2}{3}\lambda^2T^2 = \Theta(\lambda^{2-2x})$, hence $\mathbb{H}^{\text{rs}}(T) = \Theta(\lambda^{\max(x,2-2x)})$, which is minimum for $x = \frac{2}{3}$. Differentiating, we readily obtain:

$$T_{opt} = \left(\frac{3C^R}{4\lambda^2}\right)^{\frac{1}{3}} = \Theta(\lambda^{-\frac{2}{3}}), \tag{16}$$

$$\mathbb{H}^{\text{rs}}(T_{opt}) = \left(\frac{3C^R\lambda}{\sqrt{2}}\right)^{\frac{2}{3}} + o(\lambda^{\frac{2}{3}}) = \Theta(\lambda^{\frac{2}{3}}). \tag{17}$$

Note that the optimal period has the order $T_{opt} = \Theta(\lambda^{-\frac{2}{3}}) = \Theta(\mu^{\frac{2}{3}})$, while the extension $\sqrt{2M_2C}$ of the Young/Daly formula has the order $\Theta(\lambda^{-\frac{1}{2}}) = \Theta(\mu^{\frac{1}{2}})$. This means that the optimal period is much longer than the value that has been used in all previous works. This result generalizes to several processor pairs, as shown in Section 4.3. We further discuss asymptotic results in Section 6.

For an intuitive way to retrieve Equation (16), the derivation is similar to that used for Equations (5), (7) and (12). Consider a period of length $\mathcal{P} = T + C^R$. The failure-free overhead is still $\frac{C^R}{T}$, and the failure-induced overhead becomes $\frac{1}{\mu}\frac{T}{\mu} \times \frac{2T}{3}$: we factor in an updated value of the fatal failure frequency $\frac{1}{\mu}\frac{T}{\mu}$: the first failure strikes with frequency $\frac{1}{\mu}$, and then with frequency $\frac{T}{\mu}$, there is another failure before the end of the period. As for the time lost, it becomes $\frac{2T}{3}$, because in average the first error strikes at one third of the period and the second error strikes at two-third of the period: indeed, we know that there are two errors in the period, and they are equally spaced in average. Altogether, both overhead sources add up to

$$\frac{C^R}{T} + \frac{2T^2}{3\mu^2}, \tag{18}$$

which is exactly Equation (15).

We conclude this section with a comment on the *no-restart* strategy. The intuitive derivation in Equation (12) leads to $\mathbb{H}^{\text{no}}(T) = \frac{C}{T} + \frac{T}{2M_{2b}}$. We now understand that this derivation is accurate if we have $T_{\text{lost}}(T) = \frac{T}{2} + o(T)$. While this latter equality is proven true without replication [14], it is unknown whether it still holds with replication. Hence, computing the optimal period for *no-restart* remains an open problem, even with a single processor pair.

Going further, Figure 2 shows that periodic checkpointing is not optimal for *no-restart* with a single processor pair, which provides another hint of the difficulty of the problem. In the figure, we compare four approaches: in addition to Restart($T^{\text{rs}}_{opt}$) and NoRestart($T^{\text{no}}_{MTTI}$), we use two non-periodic variants of *no-restart*, Non-Periodic($T_1$, $T_2$). In both variants, we use a first checkpointing period $T_1$ while both processors are alive, and then a shorter period $T_2$ as soon as one processor has been struck by a failure. When an application failure occurs, we start anew with periods of length $T_1$. For both variants, we only restart processors after an application failure, just as *no-restart* does. The first variant uses $T_1 = T^{\text{no}}_{MTTI} = \sqrt{3\mu C}$ (the MTTI is $M_2 = 3\frac{\mu}{2}$) and the second variant uses $T_1 = T^{\text{rs}}_{opt} = \left(\frac{3}{4}C\mu^2\right)^{\frac{1}{3}}$. We use the Young/Daly period $T_2 = \sqrt{2\mu C}$ for both variants, because there remains a single live processor when period $T_2$ is enforced. The figure shows the ratio of the time-to-solution for the two non-periodic approaches over that of periodic *no-restart* (with period $T^{\text{no}}_{MTTI}$). Note that the application is perfectly parallel, and that the only overhead is for checkpoints and re-executions after failures. Both non-periodic variants are better than *no-restart*, the first one is within 98.3% of *no-restart*, and the second one is even better (95% of *no-restart*) when the MTBF increases. We also see that *restart* is more than twice better than *no-restart* with a single processor pair. Note that results are averaged over 100,000 simulations, each lasting for 10,000 periods, so that they are statistically guaranteed to be accurate.

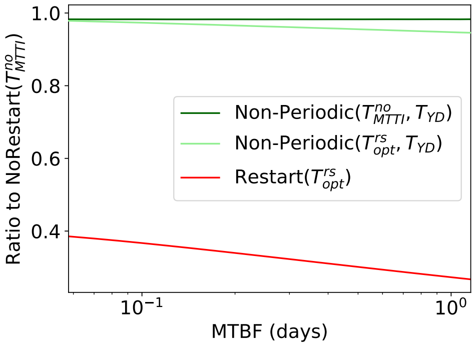


**Figure 2: Ratio of time-to-solution of two non-periodic strategies and *restart* over time-to-solution of *no-restart* (one processor pair, $C = C^R = 60$).**

## 4.3 With $b$ Processor Pairs

For $b$ pairs, the reasoning is the same as with one pair, but the probability of having a fatal error (both processors of a same pair failing) before the end of the period changes. Letting $p_b(T)$ be the probability of failure before time $T$ with $b$ pairs, we have $p_b(T) = 1-(1-(1-e^{-\lambda T})^2)^b$. As a consequence, computing the exact value of $T_{\text{lost}}(T)$ becomes complicated: obtaining a compact closed-form is not easy, because we would need to expand terms using the binomial formula. Instead, we directly use the Taylor expansion of $p_b(T)$ for $\lambda T$ close to 0. Again, this is valid only if $T = \Theta(\lambda^{-x})$ with $x < 1$. We have $p_b(T) = 1-(1-(\lambda T+o(\lambda T))^2)^b = b\lambda^2T^2+o(\lambda^2T^2)$ and compute $T_{\text{lost}}(T)$ with $b$ pairs as $T_{\text{lost}}(T) = \frac{1}{p_b(T)}\int_0^T t\frac{d\mathbb{P}(X\leq t)}{dt}dt = \frac{2T}{3}+o(T)$. As before, $T_{\text{lost}}(T) \sim \frac{2T}{3}$. Also, as in Section 4.2, we analyze the *restart* strategy, which requires any failed processor to recover whenever a checkpoint is taken. We come back to the difference with the *no-restart* strategy after deriving the period for the *restart* strategy. We compute the expected execution time of one period: $\mathbb{E}(T) = p_b(T)\big(T_{\text{lost}}(T) + D + R + \mathbb{E}(T)\big) + (1 - p_b(T))\big(T + C^R\big) = T + \frac{2b\lambda^2T^3}{3} + o(\lambda^2T^3)$, and

$$\mathbb{H}^{\text{rs}}(T) = \frac{\mathbb{E}(T)}{T} - 1 = \frac{C^R}{T} + \frac{2b\lambda^2T^2}{3} + o(\lambda^2T^2). \tag{19}$$

We finally derive the expression of the optimal checkpointing period with $b$ pairs:

$$T^{\text{rs}}_{opt} = \left(\frac{3C^R}{4b\lambda^2}\right)^{\frac{1}{3}} = \Theta(\lambda^{-\frac{2}{3}}). \tag{20}$$

When plugging it back in Equation (19), we get

$$\mathbb{H}^{\text{rs}}(T^{\text{rs}}_{opt}) = \left(\frac{3C^R\sqrt{b}\lambda}{\sqrt{2}}\right)^{\frac{2}{3}} + o(\lambda^{\frac{2}{3}}) = \Theta(\lambda^{\frac{2}{3}}). \tag{21}$$

for the optimal overhead when using $b$ pairs of processors.

The derivation is very similar to the case with a single pair, and the result is essentially the same, up to factoring in the number of pairs to account for a higher failure rate. However, the difference between the *no-restart* and the *restart* strategies gets more important. Indeed, with the *no-restart* strategy, several pairs can be struck once (and even several times if the failures always strike the failed processor) before a pair finally gets both its processors killed. While the *no-restart* strategy spares the cost of several restarts, it runs at risk with periods whose length has been estimated à la Young/Daly, thereby assuming an identical setting at the beginning of each period.

Finally, for the intuitive way to retrieve Equation (20), it goes as for Equation (18), multiplying the frequency of fatal failures $\frac{1}{\mu}\frac{T}{\mu}$ by a factor $b$ to account for each of the $b$ pairs possibly experiencing a fatal failure.

## 5 TIME-TO-SOLUTION

So far, we have focused on period length. In this section, we move to actual work achieved by the application. Following [25], we account for two sources of overhead for the application. First, the application is not perfectly parallel and obeys Amdahl's law [1], which limits its parallel speedup. Second, there is an intrinsic slowdown due to active replication related to duplicating every application message [20, 25].

First, for applications following Amdahl's law, the total time spent to compute $W$ units of computation with $N$ processors is $T_{Amdahl} = \gamma W + (1-\gamma)\frac{W}{N} = (\gamma + \frac{1-\gamma}{N})W$, where $\gamma$ is the proportion of inherently sequential tasks. When replication is used, this time becomes $T_{Amdahl} = (\gamma + \frac{2(1-\gamma)}{N})W$. Following [25], we use $\gamma = 10^{-5}$ in Section 7. Second, as stated in [20, 25], another slowdown related to active replication and its incurred increase of communications writes $T_{rep} = (1+\alpha)T_{Amdahl}$, where $\alpha$ is some parameter depending upon the application and the replication library. Following [25], we use either $\alpha = 0$ or $\alpha = 0.2$ in Section 7.

All in all, once we have derived $T_{opt}$, the optimal period between two checkpoints without replication (see Equation (6)), and $T^{\text{rs}}_{opt}$, the optimal period between two checkpoints with replication and *restart* (see Equation (20)), we are able to compute the optimal number of operations to be executed by an application between two checkpoints as $W_{opt} = \frac{T_{opt}}{\left(\gamma+\frac{1-\gamma}{N}\right)}$ for an application without replication, and $W^{\text{rs}}_{opt} = \frac{T^{\text{rs}}_{opt}}{(1+\alpha)\left(\gamma+\frac{1-\gamma}{b}\right)} = \frac{T^{\text{rs}}_{opt}}{(1+\alpha)\left(\gamma+\frac{2(1-\gamma)}{N}\right)}$ for an application with replication and the *restart* strategy. Finally, for the *no-restart* strategy, using $T^{\text{no}}_{MTTI}$ (see Equation (11)), the number of operations becomes $W^{\text{no}}_{MTTI} = \frac{T^{\text{no}}_{MTTI}}{(1+\alpha)\left(\gamma+\frac{1-\gamma}{b}\right)} = \frac{T^{\text{no}}_{MTTI}}{(1+\alpha)\left(\gamma+\frac{2(1-\gamma)}{N}\right)}$.

To compute the actual time-to-solution, assume that we have a total of $W_{seq}$ operations to do. With one processor, the execution time is $T_{seq} = W_{seq}$ (assuming unit execution speed). With $N$ processors working in parallel (no replication), the failure-free execution time is $T_{par} = (\gamma + \frac{1-\gamma}{N})T_{seq}$. Since we partition the execution into periods of length $T$, meaning that we have $\frac{T_{par}}{T}$ periods overall, the time-to-solution is $T_{final} = \frac{T_{par}}{T}\mathbb{E}(T) = T_{par}(\mathbb{H}(T) + 1)$, hence

$$T_{final} = \left(\gamma + \frac{1-\gamma}{N}\right)(\mathbb{H}(T) + 1)T_{seq}. \tag{22}$$

If we use replication with $b$ pairs of processors (i.e., $\frac{N}{2}$ pairs) instead, the difference is that $T_{par} = (1+\alpha)\left(\gamma + \frac{1-\gamma}{b}\right)T_{seq}$, hence

$$T_{final} = (1+\alpha)\left(\gamma + \frac{2(1-\gamma)}{N}\right)\left(\gamma + \frac{2(1-\gamma)}{N}\right)(\mathbb{H}(T)+1)T_{seq}. \tag{23}$$

Without replication, we use the optimal period $T = T_{opt}$. For the *restart* strategy, we use the optimal period $T = T^{\text{rs}}_{opt}$, and for *no-restart*, we use $T = T^{\text{no}}_{MTTI}$, as stated above.

## 6 ASYMPTOTIC BEHAVIOR

In this section, we compare the *restart* and *no-restart* strategies asymptotically. Both approaches (and, as far as we know, all coordinated rollback-recovery approaches) are subject to a design constraint: if the time between two restarts becomes of same magnitude as the time to take a checkpoint, the application cannot progress. Therefore, when evaluating the asymptotic behavior (i.e., when the number of nodes tends to infinity, and hence the MTTI tends to 0), a first consideration is to state that none of these techniques can support infinite growth, under the assumption that the checkpoint time remains constant and that the MTTI decreases with scale. Still, in that case, because the *restart* approach has a much longer checkpointing period than *no-restart*, it will provide progress for lower MTTIs (and same checkpointing cost).

However, we can (optimistically) assume that checkpointing technology will evolve, and that rollback-recovery protocols will be allowed to scale infinitely, because the checkpoint time will remain a fraction of the MTTI. In that case, assume that with any number $N$ of processors, we have $C = xM_N$ for some small constant $x < 1$ (where $M_N$ is the MTTI with $N$ processors). Consider a parallel and replicated application that would take a time $T_{app}$ to complete without failures (and with no fault-tolerance overheads). We compute the ratio $\mathcal{R}$, which is the expected time-to-solution using the *restart* strategy divided by the expected time-to-solution using the *no-restart* strategy:

$$\mathcal{R} = \frac{(\mathbb{H}^{\text{rs}}(T^{\text{rs}}_{opt}) + 1)T_{app}}{(\mathbb{H}^{\text{no}}(T^{\text{no}}_{MTTI}) + 1)T_{app}} = \frac{\sqrt[3]{\frac{9}{8}\pi x^2} + 1}{\sqrt{2x} + 1}.$$

Because of the assumption $C = xM_N$, both the number of nodes $N$ and the MTBF $\mu$ simplify out in the above ratio. Under this assumption, the *restart* strategy is up to 8.4% faster than the *no-restart* strategy if $x$ is within the range $[0, 0.64]$, i.e., as long as the checkpoint time takes less than 2/3 of the MTTI.

In the next section, we consider realistic parameters to evaluate the performance of various strategies through simulations, and we also provide results when increasing the number of processors $N$ or reducing the MTBF.

## 7 EXPERIMENTAL EVALUATION

In this section, we evaluate the performance of the *no-restart* and *restart* strategies through simulations. Our simulator is publicly available [7] so that interested readers can instantiate their preferred scenarios and repeat the same simulations for reproducibility purpose. The code is written in-house in C++ and does not use any library other than the Standard Template Library (STL).

We compare different instances of the models presented above. We let *Restart*($T$) denote the *restart* strategy with checkpointing period $T$, and *NoRestart*($T$) denote the *no-restart* strategy with checkpointing period $T$. In most figures, we present the overhead as given by Equation (1): it is a *relative* time overhead, that represents the time spent tolerating failures divided by the duration of the protected application. Recall previously introduced notations:

• For *Restart*($T$), the overhead $\mathbb{H}^{\text{rs}}(T)$ is predicted by the model according to Equation (19);

• For *NoRestart*($T$), the overhead $\mathbb{H}^{\text{no}}(T)$ is estimated in the literature according to Equation (12);

• $T^{\text{rs}}_{opt}$ denotes the optimal period for minimizing the time overhead for the *restart* strategy, as computed in Equation (20);

• $T^{\text{no}}_{MTTI}$ from Equation (11) is the standard period used in the literature for the *no-restart* strategy, after an analogy with the Young/Daly formula.

The *no-restart* strategy with overhead $\mathbb{H}^{\text{no}}(T^{\text{no}}_{MTTI})$ represents the state of the art for full replication [20]. For completeness, we also compare the *no-restart* and *restart* strategies with several levels of partial replication [17, 25].

We describe the simulation setup in Section 7.1. We assess the accuracy of our model and of first-order approximations in Section 7.2. We compare the performance of *restart* with *restart-on-failure* in Section 7.3. In Section 7.4, we show the impact of key parameters on the difference between the checkpointing periods of the *no-restart* and *restart* strategies, and on the associated time overheads. Section 7.5 discusses the impact of the different strategies on I/O pressure. Section 7.6 investigates in which scenarios a smaller time-to-solution can be achieved with full or partial replication. Section 7.7 explores strategies that restart after a given number of failures.

### 7.1 Simulation Setup

To evaluate the performance of the *no-restart* and *restart* strategies, we use a publicly available simulator [7] that generates random failures following an exponential probability distribution with a given mean time between individual node failures and number of processor pairs. Then, we set the checkpointing period, and checkpointing cost. Default values are chosen to correspond to the values used in [25], and are defined as follows. For the checkpointing cost, we consider two default values: $C = 60$ seconds corresponds to buddy checkpointing, and $C = 600$ seconds corresponds to checkpointing on remote storage. We let the MTBF of an individual node be $\mu = 5$ years, and we use $N = 200,000$, hence having $b = 100,000$ pairs when replication is used. We then simulate the execution of an application lasting for 100 periods (total execution time $100T$) and we average the results on 1000 runs. We measure two main quantities: time overhead and optimal period length. For simplicity, we always assume that $R = C$, i.e., read and write operations take (approximately) the same time. We cover the whole range of possible values for $C^R$, using either $C$, $1.5C$ or $2C$. This will show the impact of overlapping checkpoint and processor restart.

### 7.2 Model Accuracy

Figure 3 compares three different ways of estimating the time overhead of an application running on $b = 10^5$ processor pairs. Solid lines are measurements from the simulations, while dashed lines are theoretical values. The red color is for *Restart*($T^{\text{rs}}_{opt}$), the blue color is for *Restart*($T^{\text{no}}_{MTTI}$) and the green color is for *NoRestart*($T^{\text{no}}_{MTTI}$). For the *restart* strategy, $C^R = C$ in this figure.

For the *restart* strategy, the results from simulation match the results from the theory quite accurately. Because our formula is an approximation valid when $T \gg C$, the difference between simulated time overhead and $\mathbb{H}^{\text{rs}}(T^{\text{rs}}_{opt})$ slightly increases when the checkpointing cost becomes greater than 1500 seconds. We also verify that *Restart*($T^{\text{rs}}_{opt}$) has smaller overhead than *Restart*($T^{\text{no}}_{MTTI}$) in the simulations, which nicely corroborates the model.

We also see that $\mathbb{H}^{\text{no}}(T^{\text{no}}_{MTTI})$ is a good estimate of the actual simulated overhead of *NoRestart*($T^{\text{no}}_{MTTI}$) only for $C < 500$. Larger values of $C$ induce a significant deviation between the prediction and the simulation. Values given by $\mathbb{H}^{\text{no}}(T)$ underestimate the overheads for lower values of $C$ more than $\mathbb{H}^{\text{rs}}(T)$, even when using the same $T^{\text{no}}_{MTTI}$ period to checkpoint. As described at the end of Section 4.1, the $\mathbb{H}^{\text{no}}(T)$ formula is an approximation whose accuracy is unknown, and when $C$ scales up, some elements that were neglected by the approximation become significant. The formula for $T^{\text{rs}}_{opt}$, on the contrary, remains accurate for higher values of $C$.

Figure 4 is the exact counterpart of Figure 3 when using log traces from real platforms instead of randomly generated failures with an exponential distribution. We use the two traces featuring the largest number of failures from the LANL archive [26, 27], namely LANL#2 and LANL#18. According to the detailed study in [2], failures in LANL#18 are not correlated while those in LANL#2 are correlated, providing perfect candidates to experimentally study the impact of failure distributions. LANL#2 has an MTBF of 14.1 hours and is composed of 5350 failures, while LANL#18 has an MTBF of 7.5 hours and is composed of 3899 failures. For the sake of comparing with Figure 3 that used a processor MTBF of 5 years (and an exponential distribution), we scale both traces as follows:

• We target a platform of 200,000 processors with an individual MTBF of 5 years. Thus the global platform MTBF needs to be 64 times smaller than the MTBF of LANL#2, and 32 times smaller than the MTBF of LANL#18. Hence we partition the global platform into 64 groups (of 3,125 processors) for LANL#2, and into 32 groups (of 6,250 processors) for LANL#18;

• Within each group, the trace is rotated around a randomly chosen date, so that each trace starts independently;

• We generate 200 sets of failures for each experiment and report the average time overhead.

We observe similar results in Figure 3 and Figure 4. For LANL#18, the experimental results are quite close to the model. For LANL#2,

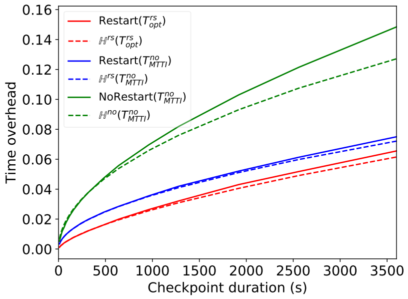


**Figure 3: Evaluation of model accuracy for time overhead.** $\mu = 5$ **years,** $b = 10^5$**.**

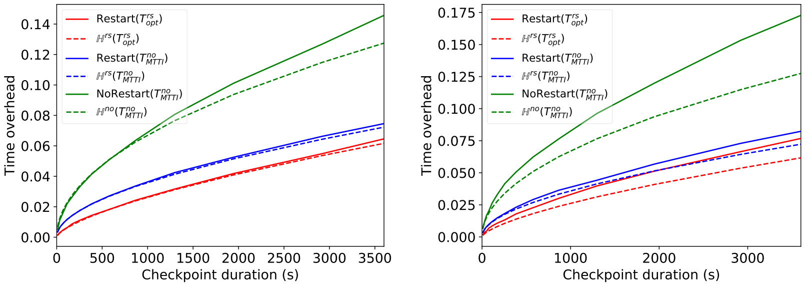


**Figure 4: Evaluation of model accuracy for time overhead with two trace logs (LANL#18 on the left, and LANL#2 on the right).**

the model is slightly less accurate because of some severely degraded intervals with failure cascades. However, the *restart* strategy still grants lower time overheads than the *no-restart* strategy. For an exponential distribution, only 15% of the runs where an application failure was experienced did experience two or more failures. This ratio increases to 20% for LANL#18 and reaches 50% for LANL#2; this leads to a higher overhead than estimated for IID failures, but this is true for all strategies, and *restart* remains the best one.

Next, on both graphs in Figure 5, we present the details of the evolution of the time overhead as a function of the period length for $C = 60$s and $C = 600$s. Here, we compare the overhead of the *restart* strategy obtained through simulations (solid red, orange and yellow lines for different values of $C^R$), the overhead of the *restart* strategy obtained through the theoretical model with $C^R$ $=C$ (dashed blue line), and the overhead of the *no-restart* strategy obtained through simulations (solid green line). In each case, a circle denotes the optimal period, while $T^{\text{no}}_{MTTI}$ (the MTTI extension of the Young/Daly formula for *no-restart*) is shown with a vertical bar.

$\mathbb{H}^{\text{rs}}(T^{\text{rs}}_{opt})$ perfectly matches the behavior of the simulations, and the optimal value is very close to the one found through simulations. The simulated overhead of *NoRestart*($T$) is always larger than for *Restart*($T$), with a significant difference as $T$ increases. Surprisingly, the optimal value for the simulated overhead of *NoRestart*($T$) is obtained for a value of $T$ close to $T^{\text{no}}_{MTTI}$, which shows *a posteriori* that the approximation worked out pretty well in this scenario. The figure also shows that the *restart* strategy is much more robust than the *no-restart* one: in all cases, *Restart*($T$) provides a lower overhead than *NoRestart*($T$) throughout the spectrum, even when $C^R = 2C$. More importantly, this overhead remains close to the minimum for a large range of values of $T$: when $C^R = C = 60s$, for values of $T$ between 21,000s and 25,000s, the overhead remains between 0.39% (the optimal), and 0.41%. If we take the same tolerance (overhead increased by 5%), the checkpointing period must be between 6,000s and 9,000s, thus a range that is 1/3rd larger than for the *restart* strategy. When considering $C^R = C = 600s$, this range is 18,000s (40,000s to 58,000s) for the *restart* strategy, and 7,000s (22,000s to 29,000s) for the *no-restart* one. This means that a user has a much higher chance of obtaining close-to-optimum performance by using the *restart* strategy than if she was relying on the *no-restart* one, even if some key parameters that are used to derive $T^{\text{rs}}_{opt}$ are misevaluated. If $C^R = 1.5C$ or $C^R = 2C$, the same trends are observed: the optimal values are obtained for longer periods, but they remain similar in all cases, and significantly lower than for the *no-restart* strategy. Moreover, the figures show the same plateau effect around the optimal, which makes the *restart* strategy robust.

### 7.3 *Restart-on-failure*

Figures 3 to 5 showed that the *restart* strategy is more efficient than the *no-restart* one. Intuitively, this is due to the rejuvenation introduced by the periodical restarts: when reaching the end of a period, failed processes are restarted, even if the application could continue progressing in a more risky configuration. A natural extension would be to consider the *restart-on-failure* strategy described in Section 1. This is the scenario evaluated in Figure 6: we compare the time overhead of *Restart*($T^{\text{rs}}_{opt}$) with *restart-on-failure*, which restarts each processor after each failure.

Compared to *Restart*($T^{\text{no}}_{MTTI}$), the *restart-on-failure* strategy grants a significantly higher overhead that quickly grows to high values as the MTBF decreases. The *restart-on-failure* strategy works as designed: no rollback was ever needed, for any of the simulations (i.e., failures never hit a pair of replicated processors within the time needed to checkpoint). However, the time spent checkpointing after each failure quickly dominates the execution. This reflects the issue with this strategy, and the benefit of combined replication and checkpointing: as failures hit the system, it is necessary for performance to let processors fail and the system absorb most of the failures using the replicates. Combining this result with Figure 5, we see that it is critical for performance to find the optimal rejuvenation period: restarting failed processes too frequently is detrimental to performance, as is restarting them too infrequently.

### 7.4 Impact of Parameters

The graphs in Figure 7 describe the impact of the individual MTBF of the processors on the time overhead. We compare *Restart*($T^{\text{rs}}_{opt}$), *Restart*($T^{\text{no}}_{MTTI}$) (both in the most optimistic case when $C^R = C$ and in the least optimistic case when $C^R = 2C$) and *NoRestart*($T^{\text{no}}_{MTTI}$). As expected, when $C^R$ increases, the time overhead increases. However, even in the case $C^R = 2C$, both *restart* strategies outperform the *no-restart* strategy. As the MTBF increases, the overhead of all strategies tends to be negligible, since a long MTBF has the cumulated effect that the checkpointing period increases and the

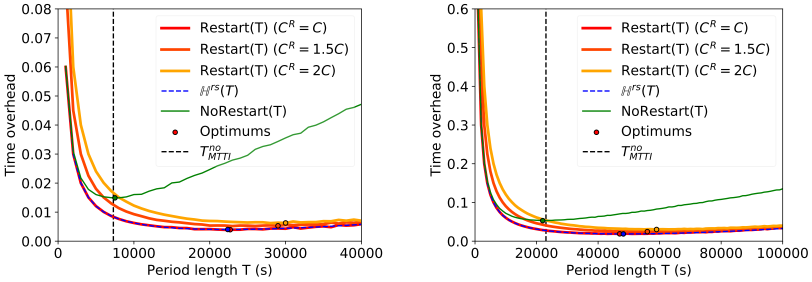


**Figure 5: Time overhead as a function of the checkpointing period $T$ for $C = 60$ seconds (left) or $C = 600$ seconds (right), MTBF of 5 years, IID failures and $b = 10^5$ processor pairs.**

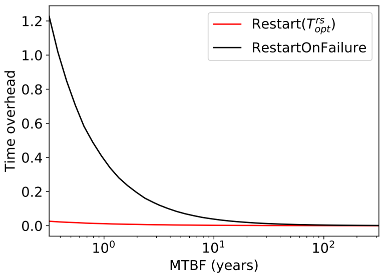


**Figure 6: Comparison with *restart-on-failure.***

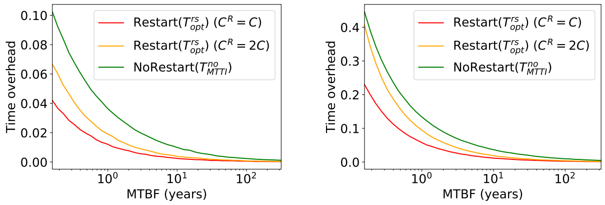


**Figure 7: Time overhead as a function of MTBF, with $C = 60$s (left) or $C = 600$s (right), $b = 10^5$ processor pairs.**

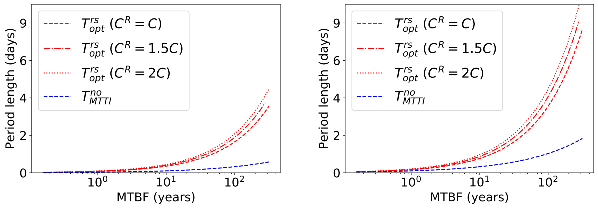


**Figure 8: Period length $T$ as function of MTBF, with $C = 60$s (left) or $C = 600$s (right), $b = 10^5$ processor pairs.**

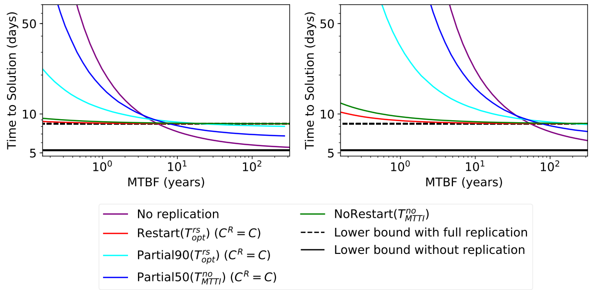


**Figure 9: Time-to-solution for $N = 2 \times 10^5$ standalone proc. against full and partial replication approaches, as a function of MTBF, with $C^R = C = 60$s (left) or $C^R = C = 600$s (right), $\gamma = 10^{-5}, \alpha = 0.2$.**

risk of needing to re-execute decreases. The longer the checkpoint time $C$, the higher the overheads, which is to be expected; more interestingly, with higher $C$, the *restart* strategy needs $C^R$ to remain close to $C$ to keep its advantage against the *no-restart* strategy. This advocates for a buddy checkpointing approach with *restart* strategy when considering replication and checkpointing over unreliable platforms.

## 7.5 I/O Pressure

Figure 8 reports the difference between $T^{\text{rs}}_{opt}$ and $T^{\text{no}}_{MTTI}$. We see that $T^{\text{rs}}_{opt}$ increases faster than $T^{\text{no}}_{MTTI}$ when the MTBF decreases. This is due to the fact that the processors are restarted at each checkpoint, hence reducing the probability of failure for each period; it mainly means that using the *restart* strategy (i) decreases the total application time, and (ii) decreases the I/O congestion in the machine, since checkpoints are less frequent. This second property is critical for machines where a large number of applications are running concurrently, and for which, with high probability, the checkpoint times are longer than expected because of I/O congestion.

## 7.6 Time-To-Solution

Looking at the time overhead is not sufficient to evaluate the efficiency of replication. So far, we only compared different strategies that all use full process replication. We now compare the *restart* and *no-restart* strategies to the approach without replication, and also to the approach with partial replication [17, 25]. Figure 9 shows the corresponding time-to-solution for $\gamma = 10^{-5}$ and $\alpha = 0.2$ (values used in [25]), and $C^R = C$ when the individual MTBF varies. Recall that the time-to-solution is computed using Equation (22) without replication (where $\mathbb{H}(T)$ is given by Equation (7)), and using Equation (23) with replication (where $\mathbb{H}(T)$ is given by Equation (12) for *no-restart*, and by Equation (19) for *restart*). In the simulations, $T_{seq}$ is set so that the application lasts one week with 100,000 processors (and no replication).

In addition to the previously introduced approaches, we evaluate $Partial90(T^{\text{rs}}_{opt})$ and $Partial50(T^{\text{no}}_{MTTI})$. $Partial90$ represents a partial replication approach where 90% of the platform is replicated (there are 90,000 processor pairs and 20,000 standalone processors).

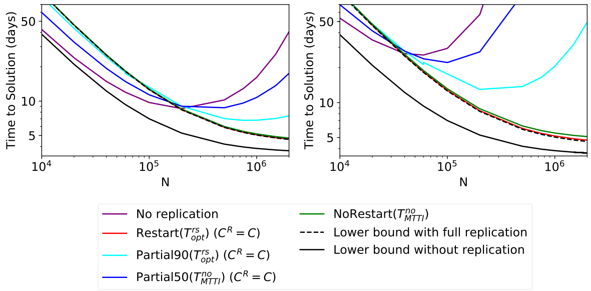


**Figure 10: Time-to-solution with MTBF of 5 years against full and partial replication approaches, as a function of $N$, with $C^R = C = 60$s (left) or $C^R = C = 600$s (right), $\gamma = 10^{-5}$, $\alpha = 0.2$.**

Similarly, 50% of the platform is replicated for *Partial*50 (there are 50,000 processor pairs and 100,000 standalone processors). Figure 9 illustrates the benefit of full replication: when the MTBF becomes too short, replication becomes mandatory. Indeed, in some cases, simulations without replication or with partial replication would not complete, because one fault was (almost) always striking before a checkpoint, preventing progress. For $C = 60$s and $N = 2 \times 10^5$, $\gamma = 10^{-5}$ and $\alpha = 0.2$, full replication grants the best time-to-solution for an MTBF shorter than $1.8 \times 10^8$. However, when the checkpointing cost increases, this value climbs up to $1.9 \times 10^9$, i.e., roughly 10 times higher than with 60 seconds. As stated before, $T^{\text{rs}}_{opt}$ gives a better overhead, thus a better execution time than $T^{\text{no}}_{MTTI}$. If machines become more unreliable, the *restart* strategy allows us to maintain the best execution time. Different values of $\gamma$ and $\alpha$ give the same trend as in our example, with large values of $\gamma$ making replication more efficient, while large values of $\alpha$ reduce the performance. Similarly to what was observed in [25], for a homogeneous platform (i.e., if all processors have a similar risk of failure), partial replication (at 50% or 90%) exhibits lower performance than no replication for long MTBF, and lower performance than the *no-restart* strategy (hence even lower performance than the *restart* strategy) for short MTBF. This confirms that partial replication has potential benefit only for heterogeneous platforms, which is outside the scope of this study.

We now further focus on discussing when replication should be used. Figure 10 shows the execution time of an application when the number of processors $N$ varies. Each processor has an individual MTBF of 5 years. The same general comments can be made: *Restart*($T^{\text{rs}}_{opt}$) always grants a slightly lower time-to-solution than *NoRestart*($T^{\text{no}}_{MTTI}$), because it has a smaller overhead. As before, when $N$ is large, the platform is less reliable and the difference between *Restart*($T^{\text{rs}}_{opt}$) and *NoRestart*($T^{\text{no}}_{MTTI}$) is higher compared to small values of $N$. We see that replication becomes mandatory for large platforms: without replication, or even with 50% of the platform replicated, the time-to-solution is more than 10 times higher than the execution time without failures. With $\gamma = 10^{-5}$ and $\alpha = 0.2$, replication becomes more efficient than no replication for $N \geq 2 \times 10^5$ processors when $C = 60$s. However, when $C = 600$s, it starts being more efficient when $N \geq 2.5 \times 10^4$, i.e., roughly 10 times less processors when $C$ is 10 times longer. This study further confirms that partial replication never proved to be useful throughout our experiments.

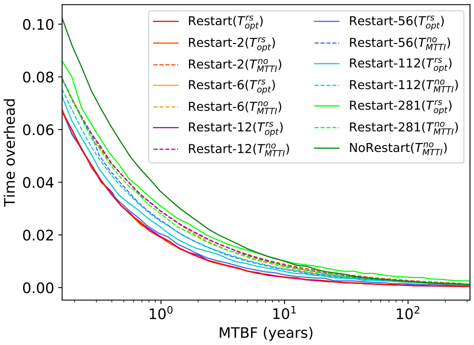


**Figure 11: Comparison of *restart* strategy with *restart* only every 2, 6, 12, 56, 112, or 281 dead proc., with $T^{\text{rs}}_{opt}$ and $T^{\text{no}}_{MTTI}$.**

## 7.7 When to Restart

In this section, we consider a natural extension of the *restart* approach: instead of restarting failed processors at each checkpoint, the restart can be delayed until the next checkpoint where the number of accumulated failures reaches or exceeds a given bound $n_{\text{bound}}$, thereby reducing the frequency of the restarts.

The *restart* strategy assumes that after a checkpoint, the risk of any processor failing is the same as in the initial configuration. For the extension, there is no guarantee that $T^{\text{rs}}_{opt}$ remains the optimal interval between checkpoints; worse, there is no guarantee that periodic checkpointing remains optimal. To evaluate the potential gain of reducing the restart frequency, we consider the two proposed intervals: $T^{\text{rs}}_{opt}$ and $T^{\text{no}}_{MTTI}$. And, since most checkpoints will not incur a restart, we assume $C^R = C$ when computing $T^{\text{rs}}_{opt}$. However, checkpoints where processes are restarted have a cost of twice the cost of a simple checkpoint in the simulation: this is the worst case for the *restart* strategy. We then simulate the execution, including restarts due to reaching $n_{\text{bound}}$ failures and due to application crashes. With $b = 100,000$ processor pairs, we expect $n_{\text{fail}}(2b) = 561$ failures before the application is interrupted; so we will consider a large range of values for $n_{\text{bound}}$: from 2, 6, 12, to cover cases where few failures are left to accumulate, to 56, 112, or 281, that represent respectively 10%, 20% and 50% of $n_{\text{fail}}(2b)$, to cover cases where many failures can accumulate.

The results are presented in Figure 11, for a variable node MTBF. The time overhead of the extended versions is higher than the time overhead of the *restart* approach using $T^{\text{rs}}_{opt}$ as a checkpointing (and restarting) interval. The latter is also lower than the overhead of the *no-restart* strategy, which on average corresponds to restarting after $n_{\text{bound}} = n_{\text{fail}}(2b) = 561$ failures. This shows that restarting the processes after each checkpoint consistently decreases the time overhead. Using the optimal checkpointing period for *restart* $T^{\text{rs}}_{opt}$, increasing $n_{\text{bound}}$ also increases the overhead. Moreover, when using small values (such as 2 and 6) for $n_{\text{bound}}$, we obtain exactly the same results as for the *restart* strategy. This is due to the fact that between two checkpoints, the *restart* strategy usually looses around 6 processors, meaning that *restart* is already the same strategy as accumulating errors up to 6 (or less) before restarting. With $n_{\text{bound}}$ =12, on average the restart happens every two checkpoints, and the performance is close, but slightly slower than the *restart* strategy.

Finally, an open problem is to determine the optimal checkpointing strategy for the extension of *restart* tolerating $n_{\text{bound}}$ failures before restarting failed processors. This optimal strategy could render the extension more efficient than the baseline *restart* strategy. Given the results of the simulations, we conjecture this optimal number to be 0, i.e., *restart* would be the optimal strategy.

**Summary.** Overall, we have shown that the *restart* strategy with period $T_{opt}^{\text{rs}}$ is indeed optimal and that our model is realistic. We showed that *restart* decreases time overhead, hence time-to-solution, compared to using *no-restart* with period $T_{MTTI}^{\text{no}}$. The extended version [6, 7] shows similar gains in energy overheads. The main decision is still to decide whether the application should be replicated or not. However, whenever it should be (which is favored by a large ratio of sequential tasks $\gamma$, a large checkpointing cost $C$, or a short MTBF), we are now able to determine the best strategy: use full replication, restart dead processors at each checkpoint (overlapped if possible), and use $T_{opt}^{\text{rs}}$ for the checkpointing period.

## 8 RELATED WORK

Checkpoint-restart is the most widely used strategy to deal with fail-stop errors. Several variants of this policy have been studied, see [24] for a survey. The natural strategy is to checkpoint periodically, and one must derive the optimal checkpointing period. For a divisible application, results were first obtained by Young [42] and Daly [14]. The original strategy has been extended to deal with a multi-level checkpointing scheme [5, 15, 29], or by using SSD or NVRAM as secondary storage [10].

Combining replication with checkpointing has been proposed in [18, 35, 45] for HPC platforms, and in [28, 41] for grid computing.

If the error rate and/or checkpoint cost is too important, and hence the overhead induced by the checkpointing strategy is large, checkpointing can be combined with replication. Hence, some redundant MPI processes are used to execute a replica of the work [12, 19, 20]. For instance, Ferreira et al. [20] used two replicas per MPI process, and they provided a theoretical analysis of parallel efficiency, an MPI implementation that supports transparent process replication (including failure detection, consistent message ordering among replicas, etc.), and a set of experimental and simulation results. Hence, they demonstrate that replication outperforms traditional checkpoint/restart approach in several scenarios.

Partial redundancy is studied in [17, 36, 37] (in combination with coordinated checkpointing) to decrease the overhead of full replication. Recently, Hussain et al. [25] have demonstrated the usefulness of partial redundancy for platforms where individual node failure distributions are not identical. They numerically determine the optimal partial replication degree. For malleable applications, adaptive redundancy is discussed in [22], where a subset of processes is dynamically selected for replication. Furthermore, the number of processors on which the applications execute is changed at runtime, yielding significant improvement in application performance.

In contrast to fail-stop errors whose detection is immediate, *silent errors* are identified only when the corrupted data leads to an unusual application behavior, and several works use replication to detect and/or correct silent errors. For instance, thread-level replication has been investigated in [13, 33, 43], which target process-level replication in order to detect (and correct) silent errors striking in all communication-related operations. Also, Ni et al. [30] introduce process duplication to cope with both fail-stop and silent errors. Recently, Benoit et al. [4] extended these work to general applications, and compare traditional process replication with *group replication*, where the whole application is replicated as a black box. They analyze several scenarios with duplication or triplication.

To the best of our knowledge, all related works use the *no-restart* strategy: in a replicated execution, failed processes are not restarted until the application experiences a fatal failure.

## 9 CONCLUSION

In this work, we have revisited process replication combined with checkpointing, an approach that has received considerable attention from the HPC community in recent years. Opinion is divided about replication. By definition, its main drawback is that 50% of platform resources will not contribute to execution progress, and such a reduced throughput does not seem acceptable in many scenarios. However, checkpoint/restart alone cannot ensure full reliability in heavily failure-prone environments, and must be complemented by replication in such unreliable environments. Previous approaches all used the *no-restart* strategy. In this work, we have introduced a new rollback/recovery strategy, the *restart* strategy, which consists of restarting all failed processes at the beginning of each period. Thanks to this rejuvenation, the system remains in the same conditions at the beginning of each checkpointing period, which allowed us to build an accurate performance model and to derive the optimal checkpointing period for this strategy. This period turns out to be much longer than the one used with the *no-restart* strategy, hence reducing significantly the I/O pressure introduced by checkpoints, and improving the overall time-to-solution. To validate this approach, we have simulated the behavior of realistic large-scale systems, with failures either IID or from log traces. We have compared the performance of *restart* with the state-of-the-art strategies. Another key advantage of the *restart* strategy is its robustness: the range of periods in which its performance is close to optimal is much larger than for the *no-restart* strategy, making it a better practical choice to target unreliable platforms where the key elements (MTBF and checkpoint duration) are hard to estimate. In the future, we plan to evaluate, at least experimentally, non-periodic checkpointing strategies that rejuvenate failed processors after a given number of failures is reached or after a given time interval is exceeded.

## ACKNOWLEDGEMENT

We thank the reviewers for very helpful comments and suggestions. This work was supported, in part, by the National Science Foundation under Grant No. 1563744 (SHF: Medium: Collaborative Research: Toward Extreme Scale Fault-Tolerance: Exploration Methods, Comparative Studies and Decision Processes).

## REFERENCES

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

[2] G. Aupy, Y. Robert, and F. Vivien. Assuming failure independence: are we right to be wrong? In *FTS'2017, the Workshop on Fault-Tolerant Systems, in conjunction with Cluster'2017*. IEEE Computer Society Press, 2017.

[3] 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.

[4] A. Benoit, A. Cavelan, F. Cappello, P. Raghavan, Y. Robert, and H. Sun. Coping with silent and fail-stop errors at scale by combining replication and checkpointing. *J. Parallel and Distributed Computing*, 122:209–225, 2018.

[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, T. Herault, V. L. Fèvre, and Y. Robert. Replication is more efficient than you think. Research report RR-9278, INRIA, 2019.

[7] A. Benoit, T. Herault, V. L. Fèvre, and Y. Robert. Replication is more efficient than you think: Code and technical report, August 2019. https://doi.org/10.5281/zenodo.3366221.

[8] W. Bland, A. Bouteiller, T. Herault, J. Hursey, G. Bosilca, and J. J. Dongarra. An evaluation of User-Level Failure Mitigation support in MPI. *Computing*, 95(12):1171–1184, 2013.

[9] R. Brightwell, K. Ferreira, and R. Riesen. Transparent redundant computing with mpi. In *EuroMPI*. Springer, 2010.

[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] F. Cappello, K. Mohror, et al. VeloC: very low overhead checkpointing system. https://veloc.readthedocs.io/en/latest/, march 2019.

[12] H. Casanova, Y. Robert, F. Vivien, and D. Zaidouni. On the impact of process replication on executions of large-scale parallel applications with coordinated checkpointing. *Future Generation Computer Systems*, 51:7–19, 2015.

[13] S. P. Crago, D. I. Kang, M. Kang, R. Kost, K. Singh, J. Suh, and J. P. Walters. Programming models and development software for a space-based many-core processor. In *4th Int. Conf. on Space Mission Challenges for Information Technology*, pages 95–102. IEEE, 2011.

[14] J. T. Daly. A higher order estimate of the optimum checkpoint interval for restart dumps. *Future Generation Comp. Syst.*, 22(3):303–312, 2006.

[15] 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.

[16] N. El-Sayed and B. Schroeder. Reading between the lines of failure logs: Understanding how HPC systems fail. In *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 1–12, June 2013.

[17] J. Elliott, K. Kharbas, D. Fiala, F. Mueller, K. Ferreira, and C. Engelmann. Combining partial redundancy and checkpointing for HPC. In *ICDCS*. IEEE, 2012.

[18] C. Engelmann, H. H. Ong, and S. L. Scorr. The case for modular redundancy in large-scale high performance computing systems. In *PDCN*. IASTED, 2009.

[19] C. Engelmann and B. Swen. Redundant execution of HPC applications with MR-MPI. In *PDCN*. IASTED, 2011.

[20] 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.

[21] P. Flajolet, P. J. Grabner, P. Kirschenhofer, and H. Prodinger. On Ramanujan's Q-Function. *J. Computational and Applied Mathematics*, 58:103–116, 1995.

[22] C. George and S. S. Vadhiyar. ADFT: An adaptive framework for fault tolerance on large scale systems using application malleability. *Procedia Computer Science*, 9:166 – 175, 2012.

[23] R. Guerraoui and A. Schiper. Fault-tolerance by replication in distributed systems. In A. Strohmeier, editor, *Reliable Software Technologies — Ada-Europe '96*, pages 38–57, 1996.

[24] T. Herault and Y. Robert, editors. *Fault-Tolerance Techniques for High-Performance Computing*, Computer Communications and Networks. Springer Verlag, 2015.

[25] Z. Hussain, T. Znati, and R. Melhem. Partial redundancy in HPC systems with non-uniform node reliabilities. In *SC '18*. IEEE, 2018.

[26] D. Kondo, B. Javadi, A. Iosup, and D. Epema. The failure trace archive: Enabling comparative analysis of failures in diverse distributed systems. *Cluster Computing and the Grid, IEEE International Symposium on*, pages 398–407, 2010.

[27] LANL. Computer failure data repository. https://www.usenix.org/cfdr-data, 2006.

[28] T. Leblanc, R. Anand, E. Gabriel, and J. Subhlok. VolpexMPI: An MPI library for execution of parallel applications on volatile nodes. In *16th European PVM/MPI Users' Group Meeting*, pages 124–133. Springer-Verlag, 2009.

[29] A. Moody, G. Bronevetsky, K. Mohror, and B. R. d. Supinski. Design, modeling, and evaluation of a scalable multi-level checkpointing system. In *SC*. ACM, 2010.

[30] X. Ni, E. Meneses, N. Jain, and L. V. Kalé. ACR: Automatic Checkpoint/Restart for Soft and Hard Error Protection. In *Proc. SC'13*. ACM, 2013.

[31] X. Ni, E. Meneses, and L. V. Kalé. Hiding checkpoint overhead in HPC applications with a semi-blocking algorithm. In *Cluster Computing (CLUSTER), 2012 IEEE International Conference on*, pages 364–372. IEEE Computer Society, 2012.

[32] R. Oldfield, S. Arunagiri, P. Teller, S. Seelam, M. Varela, R. Riesen, and P. Roth. Modeling the impact of checkpoints on next-generation systems. In *Proc. of IEEE MSST*, pages 30–46, 2007.

[33] M. W. Rashid and M. C. Huang. Supporting highly-decoupled thread-level redundancy for parallel programs. In *14th Int. Conf. on High-Performance Computer Architecture (HPCA)*, pages 393–404. IEEE, 2008.

[34] R. Riesen, K. Ferreira, and J. Stearley. See applications run and throughput jump: The case for redundant computing in HPC. In *Proc. of the Dependable Systems and Networks Workshops*, pages 29–34, 2010.

[35] B. Schroeder and G. A. Gibson. Understanding Failures in Petascale Computers. *Journal of Physics: Conference Series*, 78(1), 2007.

[36] J. Stearley, K. B. Ferreira, D. J. Robinson, J. Laros, K. T. Pedretti, D. Arnold, P. G. Bridges, and R. Riesen. Does partial replication pay off? In *FTXS*. IEEE, 2012.

[37] O. Subasi, J. Arias, O. Unsal, J. Labarta, and A. Cristal. Programmer-directed partial redundancy for resilient HPC. In *Computing Frontiers*. ACM, 2015.

[38] Top500. Top 500 Supercomputer Sites, November 2018. https://www.top500.org/lists/2018/11/.

[39] E. Weisstein. Gauss hypergeometric function. From MathWorld–A Wolfram Web Resource. http://functions.wolfram.com/HypergeometricFunctions/Hypergeometric2F1/03/04/02/.

[40] E. Weisstein. Incomplete Beta Function. From MathWorld–A Wolfram Web Resource. http://mathworld.wolfram.com/IncompleteBetaFunction.html.

[41] S. Yi, D. Kondo, B. Kim, G. Park, and Y. Cho. Using Replication and Checkpointing for Reliable Task Management in Computational Grids. In *SC'10*. ACM, 2010.

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

[43] J. Yu, D. Jian, Z. Wu, and H. Liu. Thread-level redundancy fault tolerant CMP based on relaxed input replication. In *ICCIT*. IEEE, 2011.

[44] G. Zheng, L. Shi, and L. V. Kale. FTC-Charm++: an in-memory checkpoint-based fault tolerant runtime for Charm++ and MPI. In *Cluster Computing, 2004 IEEE International Conference on*, pages 93–103. IEEE Computer Society, 2004.

[45] Z. Zheng and Z. Lan. Reliability-aware scalability models for high performance computing. In *Cluster Computing*. IEEE, 2009.