On the impact of process replication on executions of large-scale parallel applications with coordinated checkpointing

Henri Casanova, Yves Robert, Frédéric Vivien, Dounia Zaidouni

On the Impact of Process Replication on Executions of Large-Scale Parallel Applications with Coordinated Checkpointing

Henri Casanova[1], Yves Robert[2,3], Frédéric Vivien[2], and Dounia Zaidouni[2]
1. Univ. of Hawai'i at Manoa, Honolulu, USA, henric@hawaii.edu
2. ENS Lyon & INRIA, France, {Yves.Robert|Frederic.Vivien|Dounia.Zaidouni}@ens-lyon.fr
3. University of Tennessee Knoxville, USA

Abstract

Processor failures in post-petascale parallel computing platforms are common occurrences. The traditional fault-tolerance solution, checkpoint-rollback-recovery, severely limits parallel efficiency. One solution is to replicate application processes so that a processor failure does not necessarily imply an application failure. Process replication, combined with checkpoint-rollback-recovery, has been recently advocated. We first derive novel theoretical results for Exponential failure distributions, namely exact values for the Mean Number of Failures To Interruption and the Mean Time To Interruption. We then extend these results to arbitrary failure distributions, obtaining closed-form solutions for Weibull distributions. Finally, we evaluate process replication in simulation using both synthetic and real-world failure traces so as to quantify average application makespan. One interesting result from these experiments is that, when process replication is used, application performance is not sensitive to the checkpointing period, provided that that period is within a large neighborhood of the optimal period. More generally, our empirical results make it possible to identify regimes in which process replication is beneficial.

Keywords: Fault-tolerance; parallel computing; checkpoint; rollback-recovery; process replication.

1 Introduction

As plans are made for deploying post-petascale high performance computing (HPC) systems [1, 2], solutions need to be developed for ensuring resilience to processor failures. Resilience is particularly critical for applications that enroll large numbers of processors. For such applications, processor failures are projected to be common occurrences [3, 4, 5]. For instance, the 45,208-processor Jaguar platform is reported to experience on the order of 1 failure per day [6], and its scale is modest compared to upcoming platforms. Failures occur because not all faults are automatically detected and corrected in current production hardware, due to both technical challenges and high cost. To tolerate failures the standard approach is to use rollback and recovery for resuming application execution from a previously saved fault-free execution state, or *checkpoint*, which is termed checkpoint-rollback-recovery (CRR). Frequent checkpointing leads to higher overhead during fault-free execution, but less frequent checkpointing leads to a larger loss when a failure occurs. A large volume of literature is devoted to CRR, including both theoretical and practical results. The former typically rely on assumptions regarding the probability distributions of times to failure of the processors (e.g., Exponential, Weibull), while the latter rely on simulations driven by failure datasets obtained on real-world platforms.

Even assuming an optimal checkpointing strategy, at large scale processors end up spending as much or even more time saving state than computing state, leading to poor parallel efficiency [3, 4, 5]. Consequently, additional resilience mechanisms must be used. In this work we focus on *replication*: several processors perform the same computation synchronously, so that a failure of one of these processors does not lead to an application failure. Replication is an age-old fault-tolerance technique, but it has gained traction in the HPC context only relatively recently [7, 8, 9].

While replication wastes compute resources in fault-free executions, it can alleviate the poor scalability of CRR. With *process replication*, a single instance of an application is executed but each application process is (transparently) replicated. For instance, instead of executing the application with $2n$ distinct processes on a $2n$-processor platform, one executes the application with n *logical* processes so that there are two replicas of each process, each running on a distinct *physical* processor.

We illustrate this approach on an example in Figure 1 for a platform with 8 processors running an application with 4 MPI processes so that each process has one replica (i.e., 8 processes in total). At time 1, one of the replicas for process #3 experiences a failure and a downtime (of 1 second). Because the other replica is still running, the application does not fail. At time 2, a replica of process #1 and a replica of process #4 each experience a failure, but once again the application can continue running. At time 4, a checkpoint is saved (perhaps as dictated by a periodic checkpointing strategy), which is possible because at least one replica of each process is running. At time 6 two other failures occur but have no impact on the application execution because enough process replicas remain (one of these two failures occurs at a processor that is no longer participating in the application execution). Another checkpoint is saved at time 8. At time 10, a failure and downtime occur for the only remaining replica of process #3. This causes an application failure, and triggers a recovery. At time 11.5, all processors resume from a previous checkpoint, each with 2 replicas. In this example, for simplicity, we do not show failures during checkpointing and/or recovery, but we do allow such failures in this work.

Process replication is sensible because the mean time to failure of a group of two replicas is larger than that of a single processor, meaning that the checkpointing frequency can be lowered thus improving parallel efficiency. In [10] Ferreira et al. have studied process replication, with a practical implementation and some analytical results. In this paper, we focus on the theoretical foundations of process replication, and we make the following novel contributions:

- We derive exact expressions for the *MNFTI* (Mean Number of Failures To Interruption)

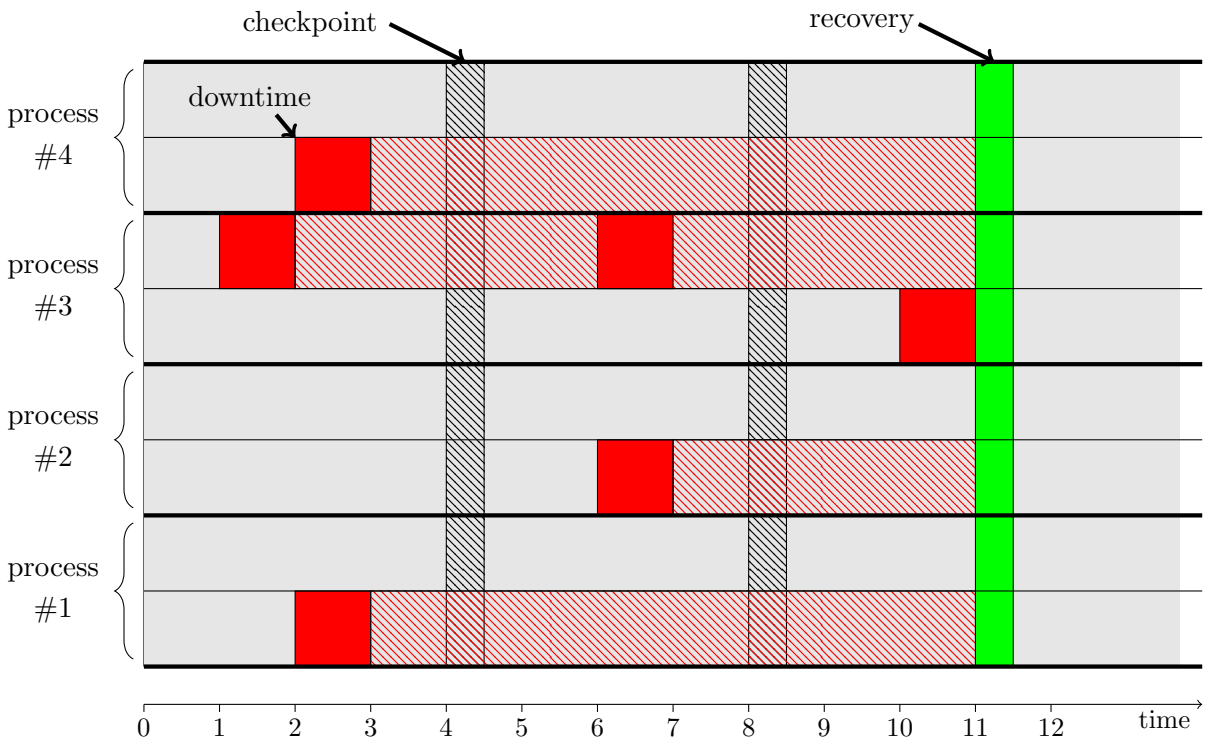


Figure 1: Gantt chart for a example application execution with process replication on 8 processors (4 logical MPI processes with one replica per process). A gray fill indicates that a processor is computing. A red fill denotes when a process is experiencing a downtime. Red hatches indicate when a processor is not longer participating in the application execution. Black hatches indicates when a processor is involved in saving a checkpoint. A green fill indicates when a processor is involved in a recovery.

and the *MTTI* (Mean Time To Interruption) for arbitrary numbers of replicas assuming Exponential failures.

- We extend these results to arbitrary failure distributions, notably obtaining closed-form solutions in the case of Weibull failures.
- We present simulation results, based on both synthetic and real-world failure traces, to compare executions with and without process replication. We find that with process replication application performance is not sensitive to the checkpointing period, provided that that period is within a large neighborhood of the optimal period.
- Based on our results, we determine in which conditions the use of process replication is beneficial. We perform a *fair* comparison between the replication and the no-replication cases, i.e., our comparison is not impacted by the (critical) choice of a particular checkpointing period in the no-replication case.

The rest of this paper is organized as follows. Section 2 discusses related work. Section 3 states our assumptions and defines the process replication approach. Section 4 presents the bulk of our theoretical contribution. Section 5 presents our simulation methodology and empirical results obtained in simulation. Finally, Section 6 concludes the paper with a summary of our findings and directions for future work.

2 Related work

Checkpointing policies have been widely studied. In [11], Daly studies periodic checkpointing for Exponential failures, generalizing the well-known bound obtained by Young [12]. In [13] he studies the impact of sub-optimal checkpointing periods. In [14], Venkatesh develops an "optimal" checkpointing policy, based on the popular assumption that optimal checkpointing must be periodic. In [15], Bouguerra et al. *prove* that the optimal checkpointing policy is periodic when checkpointing and recovery overheads are constant, for either Exponential or Weibull failures. But their results rely on the unstated assumption that all processors are rejuvenated after each failure and after each checkpoint. In [16], Bougeret et al. show that this assumption is unreasonable for Weibull failures. They propose optimal solutions for Exponential failures and dynamic programming solutions for Weibull failures. Exponential failures are often assumed due to their convenient memoryless property, i.e., the fact that the time to the next failure does not depend on when the last failure has occurred [17]. But the non-memoryless Weibull distribution is recognized as a more realistic model [18, 19, 20, 21, 22]. The work in this paper relates to CRR in the sense that we study a replication mechanism that is complementary to checkpointing.

In spite of all the above advances, the feasibility of pure CRR for large-scale systems has been questioned [3, 4, 5]. Replication has long been used as a fault-tolerance mechanism in distributed systems [23]. Using replication together with CRR has been proposed in the context of grid computing [24]. One concern with replication is the induced resource waste. However, given the scalability limitations of pure CRR, replication has recently received more attention in the HPC literature [5, 8, 9]. The use of redundant MPI processes is advocated in [25] for HPC applications, and in [26] for grid computing with volatile nodes. The work by Ferreira et al. [10] has studied the use of process replication for MPI (Message Passing Interface) applications, using 2 replicas per MPI process. They provide 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. Partial redundancy is studied in [27, 28] (in combination with coordinated checkpointing) to decrease the overhead of full replication. Adaptive redundancy is introduced in [29], where a subset of processes is dynamically selected for replication.

In this work we focus on the theoretical analysis of the combination of replication and CRR. While some theoretical results are provided in [10], they are based on an analogy between the process replication problem and the birthday problem. This analogy is appealing but, as seen in Section 4.1.2, does not make it possible to compute exact $MNFTI$ and $MTTI$ values. In addition, all the papers listed above use Daly's formula for the checkpointing period, even for Weibull or other distributions: they simply use the mean of the distribution and plug it into the formula as if it were Exponential. More importantly, they do not consider that using replication will change the optimal checkpointing period, even for Exponential distributions. The corresponding approaches are thus sub-optimal. This paper provides the optimal value of the period for Exponential and Weibull distributions (either analytically or experimentally), taking into account the use of replication.

3 Framework

3.1 Models and assumptions

We consider the execution of a tightly-coupled parallel application, or *job*, on a large-scale platform composed of p processors. For simplicity, we use the term "processor" to indicate a multi-core processor socket subject to failures. We assume that standard CRR is performed (with checkpointing

either at the system level or at the application level, with some checkpointing overhead involved). At most one application process (replica) runs on one processor.

The job must complete $\mathcal{W}$ units of (divisible) work, which can be split arbitrarily into separate *chunks*. We define the work unit so that when the job is executed on a single processor one unit of work is performed in one unit of time. The job can execute on any number $q \leq p$ processors. Defining $\mathcal{W}(q)$ as the time required for a failure-free execution on q processors, we consider three models:

- Perfectly parallel jobs: $\mathcal{W}(q) = \mathcal{W}/q$.
- Generic parallel jobs: $\mathcal{W}(q) = \mathcal{W}/q + \gamma\mathcal{W}$. As in Amdahl's law [30], $\gamma < 1$ is the fraction of the work that is inherently sequential.
- Numerical kernels: $\mathcal{W}(q) = \mathcal{W}/q + \gamma\mathcal{W}^{2/3}/\sqrt{q}$, which is representative of a matrix product or a LU/QR factorization of size N on a 2D-processor grid, where $\mathcal{W} = O(N^3)$. In the algorithms in [31], $q = r^2$ and each processor receives $2r$ blocks of size N^2/r^2 during the execution; γ is the platform's communication-to-computation ratio.

Each participating processor is subject to *failures* that each cause a *downtime*. We do not distinguish between soft and hard failures, with the understanding that soft failures are handled via software rejuvenation (i.e., rebooting [32, 33]) and that hard failures are handled by processor sparing, a common approach in production systems. For simplicity we assume that a downtime lasts D time units, regardless of the failure type. After a downtime the processor is fault-free and begins a new lifetime. In the absence of replication, when a processor fails, the whole execution is stopped, and all processors must recover from the previous checkpointed state. This work is applicable to coordinated checkpointing [34]. We leave the study of uncoordinated checkpointing [35], which involves message logging/replay to perform recoveries, for future work. We allow failures to happen during recovery or checkpointing, but not during downtime (otherwise, the downtime can be considered part of the recovery). We assume that processor failures are independent and identically distributed (i.i.d.). This assumption is commonplace in the literature [12, 11, 15, 10] because it makes analysis more tractable [17]. In the real world, instead, failures are bound to be correlated. Note that one source of failure correlation is the hierarchical structure of compute platforms (each rack comprises compute nodes, each compute node comprises processor sockets, each processor comprises cores), which leads to simultaneous failures of groups of processors. Obtaining theoretical results for non-i.i.d. failures is beyond the scope of this work. The recovery lasts the time needed to restore the last checkpoint.

We let $C(q)$, resp. $R(q)$, denote the time needed to perform a checkpoint, resp. a recovery, during an application execution with q processors. Let V be the size (in bytes) of an application checkpoint. Then checkpointing, resp. recovering, requires that each processor write, resp. read, V/q bytes to, resp. from, secondary storage. We consider two scenarios:

- Proportional overhead: $C(q) = R(q) = C/q$ where C is the time needed to write/read V bytes to/from secondary storage at one processor. This is representative of cases where the bandwidth of the network card/link at each processor is the I/O bottleneck.
- Constant overhead: $C(q) = R(q) = C$, which is representative of cases where the bandwidth to/from the resilient storage system is the I/O bottleneck.

Since we consider tightly coupled parallel jobs, all q processors operate synchronously. These processors execute the same amount of work $\mathcal{W}(q)$ in parallel, chunk by chunk. The total time (on one processor) to execute a chunk of duration, or *size*, ω and then checkpointing it, is $\omega + C(q)$.

3.2 Process replication

A parallel application consists of several application processes, each process running on a distinct processor. Process replication was recently studied in [10], in which the authors propose to replicate each application process transparently on two processors. Only when both these processors fail must the job recover from the previous checkpoint. One replica performs redundant (thus wasteful) computations, but the probability that both replicas fail is much smaller than that of a single replica, thereby allowing for a drastic reduction of the checkpoint frequency. Previous work has provided MPI implementations that allow for transparent process replication, such as the RedMPI library [36]. Our work in this paper is agnostic to (but informed by) such practical developments and simply assumes that transparent process replication is available to the application, with some overhead.

We consider the general case where each application process is replicated $g \geq 2$ times. We call *replica-group* the set of all the replicas of a given process, and we denote by n_{rg} the number of replica-groups. Altogether, if there are p available processors, there are $n_{rg} \times g \leq p$ processes running on the platform. We assume that when one of the g replicas of a replica-group fails it is not restarted, and the execution of the application proceeds as long as there is still at least one running replica in each of the replica-groups. In other words, for the whole application to fail, there must exist a replica-group whose g replicas have all been "hit" by a failure. One could envision a scenario where a failed replica is restarted based on the current state of the remaining replicas in its replica-group. This would increase application resiliency but would also be time-consuming. A certain amount of time would be needed to copy the state of one of the remaining replicas. Because all replicas of a same process must have a coherent state, the execution of the still running replicas would have to be paused during this copying. In a tightly coupled application, the execution of the whole application would be paused while copying. Consequently, restarting a failed replica would only be beneficial if the restarting cost were very small, when taking in consideration the frequency of failures and the checkpoint and restart costs. The benefit of such an approach is doubtful and, like [10], we do not consider it.

4 Theoretical results

Two important quantities for evaluating the quality of an application execution, when replication is used, are: (i) the Mean Number of Failures To Interruption (*MNFTI*), i.e., the mean number of processor failures until application failure occurs; and (ii) the Mean Time To Interruption (*MTTI*), i.e., the mean time elapsed until application failure occurs. In this section, we compute exact expressions of these two quantities. We first deal with the computation of *MNFTI* values in Section 4.1. Then we proceed to computing *MTTI* values, for Exponential failures in Section 4.2, and for arbitrary failures in Section 4.3. Note that the computation of *MNFTI* applies to any failure distribution, while that of *MTTI* is strongly distribution-dependent.

4.1 Computing *MNFTI*

4.1.1 Analytical evaluation

We consider two options for "counting" failures. One option is to count each failure that hits any of the $g \cdot n_{rg}$ initial processors, including the processors *already hit* by a failure. Consequently, a failure that hits an already hit replica-group does not necessarily induce an application interruption. If the failure hits an already hit processor, whose replica had already been terminated due to an earlier failure, the application is not affected. If, on the contrary, the failure hits one of the other processors,

then the whole application fails. This is the option chosen in [10]. Another option is to count only failures that hit *running processors*, and thus effectively kill replicas. This approach seems more natural as the running processors are the only ones that are important for the application execution.

We use $MNFTI^{\text{ah}}$ to denote the *MNFTI* with the first option ("ah" stands for "already hit"), and $MNFTI^{\text{rp}}$ to denote the *MNFTI* with the second option ("rp" stands for "running processors"). The following theorem gives a recursive expression for $MNFTI^{\text{ah}}$ in the case $g = 2$ and for memoryless failure distributions.

Theorem 1. *If the failure inter-arrival times on the different processors are i.i.d. and independent from the failure history, then using process replication with $g = 2$, $MNFTI^{\text{ah}} = \mathbb{E}(NFTI^{\text{ah}}|0)$ where $\mathbb{E}(NFTI^{\text{ah}}|n_f) =$*

$$\begin{cases} 2 & \text{if } n_f = n_{rg}, \\ \frac{2n_{rg}}{2n_{rg}-n_f} + \frac{2n_{rg}-2n_f}{2n_{rg}-n_f}\mathbb{E}\left(NFTI^{\text{ah}}|n_f+1\right) & \text{otherwise.} \end{cases}$$

Proof. Let $\mathbb{E}(NFTI^{\text{ah}}|n_f)$ be the expectation of the number of failures needed for the whole application to fail, knowing that the application is still running and that failures have already hit n_f different replica-groups. Because each process initially has 2 replicas, this means that n_f different processes are no longer replicated, and that $n_{rg} - n_f$ are still replicated. Overall, there are $n_f + 2(n_{rg} - n_f) = 2n_{rg} - n_f$ processors still running.

The case $n_f = n_{rg}$ is the simplest. A new failure will hit an already hit replica-group, that is, a replica-group where one of the two initial replicas is still running. Two cases are then possible:

1. The failure hits the running processor. This leads to an application failure, and in this case $\mathbb{E}(NFTI^{\text{ah}}|n_{rg}) = 1$.
2. The failure hits the processor that has already been hit. Then the failure has no impact on the application. The $MNFTI^{\text{ah}}$ of this case is then: $\mathbb{E}(NFTI^{\text{ah}}|n_{rg}) = 1 + \mathbb{E}\left(NFTI^{\text{ah}}\,|n_{rg}\right)$.

The probability of failure is uniformly distributed between the two replicas, and thus between these two cases. Weighting the values by their probabilities of occurrence yields:

$$\mathbb{E}\left(NFTI^{\text{ah}}\,|n_{rg}\right) = \frac{1}{2} \times 1 + \frac{1}{2} \times \left(1 + \mathbb{E}\left(NFTI^{\text{ah}}\,|n_{rg}\right)\right) = 2.$$

For the general case $0 \leq n_f \leq n_{rg} - 1$, either the next failure hits a new replica-group, that is one with 2 replicas still running, or it hits a replica-group that has already been hit. The latter case leads to the same sub-cases as the $n_f = n_{rg}$ case studied above. As we have assumed that the failure inter-arrival times on the different processors are i.i.d. and *independent from the processor failure history* the failure probability is uniformly distributed among the $2n_{rg}$ processors, including the ones already hit. Hence the probability that the next failure hits a new replica-group is $\frac{2n_{rg}-2n_f}{2n_{rg}}$. In this case, the expected number of failures needed for the whole application to fail is one (the considered failure) plus $\mathbb{E}\left(NFTI^{\text{ah}}|n_f+1\right)$. Altogether we have:

$$\mathbb{E}\left(NFTI^{\text{ah}}|n_f\right) = \frac{2n_{rg}-2n_f}{2n_{rg}} \times \left(1 + \mathbb{E}\left(NFTI^{\text{ah}}|n_f+1\right)\right) + \frac{2n_f}{2n_{rg}} \times \left(\frac{1}{2} \times 1 + \frac{1}{2}\left(1 + \mathbb{E}\left(NFTI^{\text{ah}}|n_f\right)\right)\right).$$

Therefore, $\mathbb{E}\left(NFTI^{\text{ah}}|n_f\right) = \frac{2n_{rg}}{2n_{rg}-n_f} + \frac{2n_{rg}-2n_f}{2n_{rg}-n_f}\mathbb{E}\left(NFTI^{\text{ah}}|n_f+1\right)$. □

It turns out that there is a simple (and quite unexpected) relationship between both failure models:

Proposition 1. *If the failure inter-arrival times on the different processors are i.i.d. and independent from the processor failure history then, for $g = 2$,*

$$MNFTI^{\text{ah}} = 1 + MNFTI^{\text{rp}}.$$

Due to lack of space, we cannot include the proof of this and the following propositions, nor of the upcoming theoretical results in Section 4.3. These proofs span multiple pages and can be found in the companion research report [37].

Theorem 1 can be generalized to higher values of g. To give an idea of the approach, here is the recursion for $g = 3$:

Proposition 2. *If the failure inter-arrival times on the different processors are i.i.d. and independent from the failure history, then using process replication with $g = 3$, $MNFTI^{\text{ah}} = \mathbb{E}(NFTI^{\text{ah}}|0, 0)$ where*

$$\mathbb{E}\left(NFTI^{\text{ah}}|n_2, n_1\right) = \frac{1}{3n_{rg} - n_2 - 2n_1}\Big(3n_{rg} + 3(n_{rg} - n_1 - n_2)\mathbb{E}\left(NFTI^{\text{ah}}|n_2 + 1, n_1\right) + 2n_2\mathbb{E}\left(NFTI^{\text{ah}}|n_2 - 1, n_1 + 1\right)\Big)$$

One can solve this recursion using a dynamic programming algorithm of quadratic cost $O(p^2)$ (and linear memory space $O(p)$).

Proposition 3. *If the failure inter-arrival times on the different processors are i.i.d. and independent from the failure history, then using process replication with $g = 3$, $MNFTI^{\text{rp}} = \mathbb{E}(NFTI^{\text{rp}}|0, 0)$ where*

$$\mathbb{E}\left(NFTI^{\text{rp}}|n_2, n_1\right) = 1 + \frac{1}{3n_{rg} - n_2 - 2n_1}\left(3(n_{rg} - n_1 - n_2)\mathbb{E}\left(NFTI^{\text{rp}}|n_2 + 1, n_1\right) + 2n_2\mathbb{E}\left(NFTI^{\text{rp}}|n_2 - 1, n_1 + 1\right)\right)$$

Given the simple additive relationship that exists between $MNFTI^{\text{ah}}$ and $MNFTI^{\text{rp}}$ for $g = 2$ (Proposition 1), one might have expected a similar relationship for large g. However, computing numerical values for $g = 3$ shows that the difference between $MNFTI^{\text{ah}}$ and $MNFTI^{\text{rp}}$ is not constant and increases as n_{rg} increases. No simple relationship between $MNFTI^{\text{ah}}$ and $MNFTI^{\text{rp}}$ seems to exit when $g \geq 3$.

4.1.2 Numerical evaluation

In this section we evaluate our approach for computing the *MNFTI* value and include a comparison with the approach in [10]. The authors therein observe that the generalized birthday problem is related to the problem of determining the number of processor failures needed to induce an application failure. The generalized birthday problem asks the following question: what is the expected number of balls $BP(m)$ to randomly put into m (originally empty) bins so that there is a bin with two balls? This problem has a well-known closed-form solution [38]. In the context

of process replication, it is tempting to consider each replica group as a bin, and each ball as a processor failure, thus computing $MNFTI = BP(n_{rg})$. Unfortunately, this analogy is incorrect because processors in a replica group are distinguished. Let us consider the case $g = 2$, i.e., two replicas per replica group, and the two failure models described in Section 4.1.1. In the "already hit" model, which is used in [10], if a failure hits a replica group after that replica group has already been hit once (i.e., a second ball is placed in a bin) an application failure does not necessarily occur. This is unlike the birthday problem, in which the stopping criterion is for a bin to contain two balls, thus breaking the analogy. In the "running processor" model, the analogy also breaks down. Consider that one failure has already occurred. The replica group that has suffered that first failure is now twice less likely to be hit by another failure as all the other replica groups as it contains only one replica. Since probabilities are no longer identical across replica groups, i.e., bins, the problem is not equivalent to the generalized birthday problem. However, there is a direct and valid analogy between the process replication problem and another version of the birthday problem with distinguished types, which asks: what is the expected number of randomly drawn red or white balls $BT(m)$ to randomly put into m (originally empty) bins so that there is a bin that contains at least one red ball and one white ball? Unfortunately, there is no known closed-form formula for $BT(m)$, even though the results in Section 4.1.1 provide a recursive solution.

In spite of the above, [10] uses the solution of the generalized birthday problem to compute $MNFTI$. According to [39], a previous article by the authors of [10], it would seem that the value $BP(n_{rg})$ is used. While [10] does not make it clear which value is used, a recent research report by the same authors states that they use $BP(g \cdot n_{rg})$. For completeness, we include both values in the comparison hereafter.

Table 1: $MNFTI^{\text{ah}}$ computed as $BP(n_{rg})$, $BP(g \cdot n_{rg})$, and using Theorem 1, for $n_{rg} = 2^0, \ldots, 2^{20}$, with $g = 2$.

n_{rg}	2^0	2^1	2^2	2^3	2^4	2^5	2^6
Theorem 1	3.0	3.7	4.7	6.1	8.1	11.1	15.2
$BP(n_{rg})$	2.0 (-33.3%)	2.5 (-31.8%)	3.2 (-30.9%)	4.2 (-30.3%)	5.7 (-30.0%)	7.8 (-29.7%)	10.7 (-29.6%)
$BP(g \cdot n_{rg})$	2.5 (-16.7%)	3.2 (-12.2%)	4.2 (-8.8%)	5.7 (-6.4%)	7.8 (-4.6%)	10.7 (-3.3%)	14.9 (-2.3%)
n_{rg}	2^7	2^8	2^9	2^{10}	2^{11}	2^{12}	2^{13}
Theorem 1	21.1	29.4	41.1	57.7	81.2	114.4	161.4
$BP(n_{rg})$	14.9 (-29.5%)	20.7 (-29.4%)	29.0 (-29.4%)	40.8 (-29.4%)	57.4 (-29.3%)	80.9 (-29.3%)	114.1 (-29.3%)
$BP(g \cdot n_{rg})$	20.7 (-1.6%)	29.0 (-1.2%)	40.8 (-0.8%)	57.4 (-0.6%)	80.9 (-0.4%)	114.1 (-0.3%)	161.1 (-0.2%)
n_{rg}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}
Theorem 1	227.9	321.8	454.7	642.7	908.5	1284.4	1816.0
$BP(n_{rg})$	161.1 (-29.3%)	227.5 (-29.3%)	321.5 (-29.3%)	454.4 (-29.3%)	642.4 (-29.3%)	908.2 (-29.3%)	1284.1 (-29.3%)
$BP(g \cdot n_{rg})$	227.5 (-0.1%)	321.5 (-0.1%)	454.4 (-0.1%)	642.4 (-0.1%)	908.2 (-0.04%)	1284.1 (-0.03%)	1815.7 (-0.02%)

Table 1 shows the $MNFTI^{\text{ah}}$ values computed as $BP(n_{rg})$ or as $BP(g \cdot n_{rg})$, as well as the exact value computed using Theorem 1, for various values of n_{rg} and for $g = 2$. (Recall that in this case, $MNFTI^{\text{ah}}$ and $MNFTI^{\text{rp}}$ differ only by 1). The percentage relative differences between the two BP values and the exact value are included in the table as well. We see that the $BP(n_{rg})$ value leads to relative differences with the exact value between 29% and 33%. This large difference seems easily explained due to the broken analogy with the generalized birthday problem. The unexpected result is that the relative difference between the $BP(g \cdot n_{rg})$ value and the exact value is below 16% and, more importantly, decreases and approaches zero as n_{rg} increases. The implication is that using $BP(g \cdot n_{rg})$ is an effective heuristic for computing $MNFTI^{\text{ah}}$ even though the birthday problem is

not analogous to the process replication problem! These results thus provide an empirical, if not theoretical, justification for the approach in [10], whose validity was not assessed experimentally therein.

4.2 Computing *MTTI* for Exponential failures

We do not know how to compute the *MTTI* under the "running processors" approach for counting failures. However, the *MTTI* can be easily computed under the "already hit" approach, when assuming Exponential failures:

$$MTTI = systemMTBF(g \times n_{rg}) \times MNFTI^{\text{ah}} \tag{1}$$

where $systemMTBF(p)$ denotes the mean time between failures of a platform with p processors and $MNFTI^{\text{ah}}$ is given by Theorem 1. Recall that $systemMTBF(p)$ is simply equal to the MTBF of an individual processor divided by p. For completeness, note that a recursive expression for *MTTI* can also be obtained directly [37].

The linear relationship between *MNFTI* and *MTTI*, seen in Equation (1), allows us to use the results in Table 1 to compute *MTTI* values. To quantify the potential benefit of replication, Table 2 shows these values as the total number of processors increases. For a given total number of processors, we show results for $g = 1$, 2, and 3. As a safety check, we have compared these predicted values with those computed through simulations, using an individual processor MTBF equal to 125 years. For each value of n_{rg} in Table 2, we have generated $1,000,000$ random failure dates, computed the Time To application Interruption for each instance, and computed the mean of these values. This *simulated MTTI*, is in full agreement with the predicted *MTTI* in Table 2 (see [37] for full methodological details).

The main and expected observation in Table 2 is that increasing g, i.e., the level of replication, leads to increased *MTTI*. The improvement in *MTTI* due to replication increases as n_{rg} increases, and increases when the level of replication, g, increases. Using $g = 2$ leads to large improvement over using $g = 1$, with an *MTTI* up to 3 orders of magnitude larger for $n_{rg} = 2^{20}$. Increasing the replication level to $g = 3$ leads to more moderate improvement over $g = 2$, with an *MTTI* only about 10 times larger for $n_{rg} = 2^{20}$. Overall, these results show that, at least in terms of *MTTI*, replication is beneficial. Although these results are for a particular MTBF value, they lead us to believe that moderate replication levels, namely $g = 2$, are sufficient to achieve drastic improvements in fault-tolerance.

4.3 Computing *MTTI* for arbitrary failures

The approach that computes *MTTI* from $MNFTI^{\text{ah}}$ is limited to memoryless (i.e., Exponential) failure distributions. To encompass arbitrary distributions, we use another approach based on the failure distribution density at the platform level. Theorem 2 quantifies the probability of successfully completing an amount of work of size $\mathcal{W}$ when using process replication for any failure distribution, which makes it possible to compute *MTTI* via numerical integration:

Theorem 2. *Consider an application with n_{rg} processes, each replicated g times using process replication, so that processor P_i, $1 \leq i \leq g \cdot n_{rg}$, executes a replica of process $\left\lceil \frac{i}{g} \right\rceil$. Assume that the failure inter-arrival times on the different processors are i.i.d, and let τ_i denote the time elapsed since the last failure of processor P_i. Let F denote the cumulative distribution function of the failure probability, and $F(t|\tau)$ be the probability that a processor fails in the next t units of time, knowing*

Table 2: *MTTI* values achieved for Exponential failures and a given number of processors using different replication factors (total of $p = 2^0, \ldots, 2^{20}$ processors, with $g = 1$, 2, and 3). The individual processor MTBF is 125 years, and MTTIs are expressed in hours.

p	2^0	2^1	2^2	2^3	2^4	2^5	2^6
$g=1$	1 095 000	547 500	273 750	136 875	68 438	34 219	17 109
$g=2$		1 642 500	1 003 750	637 446	416 932	278 726	189 328
$g=3$			1 505 625	999 188	778 673	565 429	432 102
p	2^7	2^8	2^9	2^{10}	2^{11}	2^{12}	2^{13}
$g=1$	8555	4277	2139	1069	535	267	134
$g=2$	130 094	90 135	62 819	43 967	30 864	21 712	15 297
$g=3$	326 569	251 589	194 129	151 058	117 905	92 417	72 612
p	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}
$g=1$	66.8	33.4	16.7	8.35	4.18	2.09	1.04
$g=2$	10 789	7615	5378	3799	2685	1897	1341
$g=3$	57 185	45 106	35 628	28 169	22 290	17 649	13 982

that its last failure happened τ units of time ago. Then the probability that the application will still be running after t units of time is:

$$R(t) = \prod_{j=1}^{n_{rg}} \left(1 - \prod_{i=1}^{g} F\left(t|\tau_{i+g(j-1)}\right)\right), \tag{2}$$

Let f denote the probability density function of the entire platform (f is the derivative of the function $1 - R$): the MTTI is given by:

$$MTTI = \int_0^{+\infty} tf(t)dt = \int_0^{+\infty} R(t)dt = \int_0^{+\infty} \prod_{j=1}^{n_{rg}} \left(1 - \prod_{i=1}^{g} F\left(t|\tau_{i+g(j-1)}\right)\right) \mathrm{d}t. \tag{3}$$

This theorem can then be used to obtain a closed-form expression for *MTTI* when the failure distribution is Exponential (Theorem 3) or Weibull (Theorem 4):

Theorem 3. *Consider an application with n_{rg} processes, each replicated g times using process replication. If the probability distribution of the time to failure of each processor is Exponential with parameter λ, then the MTTI is given by:*

$$MTTI = \frac{1}{\lambda} \sum_{i=1}^{n_{rg}} \sum_{j=1}^{i \cdot g} \left(\frac{\binom{n_{rg}}{i}\binom{i \cdot g}{j}(-1)^{i+j}}{j} \right).$$

The following corollary gives a simpler expression for the case $g = 2$:

Corollary 1. *Consider an application with n_{rg} processes, each replicated 2 times using process replication. If the probability distribution of the time to failure of each processor is Exponential with parameter λ, then the MTTI is given by:*

$$\begin{aligned} MTTI &= \frac{1}{\lambda} \sum_{i=1}^{n_{rg}} \sum_{j=1}^{i \cdot 2} \left(\frac{\binom{n_{rg}}{i}\binom{i \cdot 2}{j}(-1)^{i+j}}{j} \right) \\ &= \frac{2^{n_{rg}}}{\lambda} \sum_{i=0}^{n_{rg}} \left(\frac{-1}{2}\right)^i \frac{\binom{n_{rg}}{i}}{(n_{rg}+i)}. \end{aligned}$$

Theorem 4. *Consider an application with n_{rg} processes, each replicated g times using process replication. If the probability distribution of the time to failure of each processor is Weibull with scale parameter λ and shape parameter k, then the MTTI is given by:*

$$MTTI = \frac{\lambda}{k}\,\Gamma\left(\frac{1}{k}\right)\,\sum_{i=1}^{n_{rg}}\sum_{j=1}^{i\cdot g}\frac{\binom{n_{rg}}{i}\binom{i\cdot g}{j}(-1)^{i+j}}{j^{\frac{1}{k}}}.$$

While Theorem 3 is yet another approach to computing the *MTTI* for Exponential distributions, Theorem 4 is the first analytical result (to the best of our knowledge) for Weibull distributions. Unfortunately, due to the limits of floating point arithmetic, the formula in Theorem 4 is not numerically stable for large values of n_{rg}. As a result, we resort to numerical simulation to compute *MTTI* values. Table 3, which is the counterpart of Table 2 for Weibull failures, shows *MTTI* results obtained as averages computed on the first 100,000 application failures of each simulated scenario. The results are similar to those in Table 2. The *MTTI* with $g = 2$ is much larger than that using $g = 1$, up to more than 3 orders of magnitude at large scale ($n_{rg} = 2^{20}$). The improvement in *MTTI* with $g = 3$ compared to $g = 2$ is more modest, reaching about a factor 10. The conclusions are thus similar: replication leads to large improvements, and a moderate replication level ($g = 2$) may be sufficient.

Table 3: Simulated *MTTI* values achieved for Weibull failures with shape parameter 0.7 and a given number of processors p using different replication factors (total of $p = 2^0, \ldots, 2^{20}$ processors, with $g = 1$, 2, and 3). The individual processor MTBF is 125 years, and *MTTIs* are expressed in hours.

p	2^0	2^1	2^2	2^3	2^4	2^5	2^6
$g=1$	1 091 886	549 031	274 641	137 094	68 812	34 383	17 202
$g=2$		2 081 689	1 243 285	769 561	491 916	321 977	214 795
$g=3$			2 810 359	1 811 739	1 083 009	763 629	539 190
p	2^7	2^8	2^9	2^{10}	2^{11}	2^{12}	2^{13}
$g=1$	8603	4275	2132	1060	525	260	127
$g=2$	144 359	98 660	67 768	46 764	32 520	22 496	15 767
$g=3$	398 410	296 301	223 701	170 369	131 212	101 330	78 675
p	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}
$g=1$	60.1	27.9	12.2	5.09	2.01	0.779	0.295
$g=2$	11 055	7766	5448	3843	2708	1906	1345
$g=3$	61 202	47 883	37 558	29 436	23 145	18 249	14 391

5 Empirical evaluation

In the previous section, we have obtained exact expressions for the *MNFTI* and the *MTTI*, which are of direct relevance to the performance of the application. The main performance metric of interest to end-users, however, is the application *makespan*, i.e., the time elapsed between the launching of the application and its successful completion. Since it is not tractable to derive a closed-form expression of the expected makespan, in this section we compute the makespan empirically via simulation experiments.

The makespan, the *MNFTI*, and the *MTTI* are inter-related but impacted differently by the use of process replication. As the number of replicas per process, g, increases, the *MNFTI* and the

MTTI increase because with more replicas more individual processor failures can occur before an application failure occurs. These increases are quantified in Sections 4.2 and 4.3. The makespan is driven by the checkpointing overhead (a higher overhead implies a higher makespan) and the number of logical application processes (a higher number implies a lower makespan). As g increases, precisely because the *MNFTI* and the *MTTI* increase, a lower checkpointing frequency can be used, leading to a lower checkpointing overhead. However, as g increases, the number of logical application processes decreases since more processors are "wasted" to perform redundant computation. Consequently, as g increases the makespan is subject to a trade-off between the decreased checkpointing overhead and the decreased number of logical processes. Our goal in this section is to explore this trade-off, and to identify the conditions in which the *MNFTI* and *MTTI* increases due to using $g > 1$ seen in previous sections do translate to decreases in makespan.

5.1 Simulation framework and models

Here we detail our simulation methodology for evaluating the benefits of process replication. We use both synthetic and real-world failure distributions. The source code and all simulation results are publicly available at `http://graal.ens-lyon.fr/~fvivien/DATA/ProcessReplication.tar.bz2`.

Simulator – We have developed a simple simulator in C++ that takes as input a number of processor (p), a parallel job model (see Section 3.1), a specification of the failure distribution, a specification of the overhead due to replication, and values for C (checkpointing overhead), R (recovery overhead), D (down time duration), and $\mathcal{W}$ (total amount of work). The simulator then generates failure occurrences for each processor based on the failure distribution and construct Gantt charts as that shown in Figure 1, for various checkpointing policies. The simulator outputs the application makespan. Hereafter we describe how we produce input to the simulator, and which checkpointing policies it implements.

Synthetic failure distributions – To choose failure distribution parameters that are representative of realistic systems, we use failure statistics from the Jaguar platform. Jaguar is said to experience on the order of 1 failure per day [6]. Assuming a 1-day platform MTBF gives us a processor MTBF equal to $\frac{p_{total}}{365} \approx 125$ years, where $p_{total} = 45,208$ is Jaguar's number of processors. We then compute the parameters of Exponential and Weibull distributions so that they lead to this MTBF value. Namely, for the Exponential distribution we set $\lambda = \frac{1}{MTBF}$ and for the Weibull distribution, which requires two parameters k and λ, we set $\lambda = MTBF/\Gamma(1 + 1/k)$. We fix k to 0.156, 0.5, and 0.7, based on the results in [21], [20], and [19].

Log-based failure distributions – We also consider failure distributions based on failure logs from production clusters. We use logs from the largest clusters among the preprocessed logs in the *Failure trace archive* [40], i.e., for clusters at the Los Alamos National Laboratory [19]. In these logs, each failure is tagged by the node —and not just the processor— on which the failure occurred. Among the 26 possible clusters, we opted for the logs of the only two clusters with more than 1,000 nodes. The motivation is that we need a sample history sufficiently large to simulate platforms with more than ten thousand nodes. The two chosen logs are for clusters 18 and 19 in the archive (referred to as 7 and 8 in [19]). For each log, we record the set $\mathcal{S}$ of availability intervals. The discrete failure distribution for the simulation is generated as follows: the conditional probability $P(X \geq t \mid X \geq \tau)$ that a node stays up for a duration t, knowing that it has been up for a duration τ, is set to the ratio of the number of availability durations in $\mathcal{S}$ greater than or equal to t, over the number of availability durations in $\mathcal{S}$ greater than or equal to τ.

Scenario generation – Given a p-processor job, a failure trace is a set of failure dates for each processor over a fixed time horizon h set to 2 years. The job start time is assumed to be one

year to avoid side-effects related to the synchronous initialization of all nodes/processors. Given the distribution of inter-arrival times at a processor, for each processor we generate a trace via independent sampling until the target time horizon is reached.

The two clusters used for computing our log-based failure distributions consist of 4-processor nodes. Hence, to simulate a 45,208-processor platform we generate 11,302 failure traces, one for each four-processor node.

Replication overhead – In [10], the authors consider that the communication overhead due to replication is proportional to the application's communication demands. Arguing that, to be scalable, an application must have sub-linear communication costs with respect to increasing processor counts, they consider an approximate logarithmic model for the percentage replication overhead: $\frac{log(p)}{10} + 3.67$, where p is the number of processors. The parameters to this model are instantiated from the application in [10] that has the highest replication overhead. When $g = 2$, we use the same logarithmic model to augment our first two parallel job models in Section 3.1:

- **Perfectly parallel jobs**: $\mathcal{W}(q) = \frac{\mathcal{W}}{q} \times (1 + \frac{1}{100} \times (\frac{log(q)}{10} + 3.67))$.
- **Generic parallel jobs**: $\mathcal{W}(q) = (\frac{\mathcal{W}}{q} + \gamma\mathcal{W}) \times (1 + \frac{1}{100} \times (\frac{log(q)}{10} + 3.67))$.

For the numerical kernel job model, we can use a more accurate overhead model that does not rely on the above logarithmic approximation. Our original model in Section 3.1 comprises a computation component and a communication component. Using replication ($g = 2$), for each point-to-point communication between two original application processes, now a communication occurs between each process pair, considering both original processors and replicas, for a total of 4 communications. We can thus simply multiply the communication component of the model by a factor 4 and obtain the augmented model:

- **Numerical kernels**: $\mathcal{W}(q) = \frac{\mathcal{W}}{q} + \frac{\gamma \times \mathcal{W}^{\frac{2}{3}}}{\sqrt{q}} \times 4$.

No overhead model for $g = 3$ is given in [10], but in a subsequent article overhead measurements for $g = 2$ and $g = 3$ are reported for several benchmarks [36]. Considering the strong scaling benchmark results, which correspond to our job models, the maximum ratio between the percentage replication overhead for $g = 3$ to that for $g = 2$ is 3.18 (ignoring one outlier ratio that is above 60). Given these results, and given the lack of an analytical model, we conservatively multiply the $g = 2$ overhead by 3.18 in our experiments with $g = 3$ for the perfectly parallel and generic parallel job models. For the numerical kernel job model we simply multiply the communication component by a factor g^2 (as done in the $g = 2$ case).

Parameter values – We use the same parameter values as in the recent article [16]: $C = R = 600$ s, $D = 60$ s and $\mathcal{W} = 10,000$ years (except for log-based simulations for which $\mathcal{W} = 1,000$ years).

Checkpointing policy – Replication dramatically reduces the number of application failures, so that standard periodic checkpointing strategies can be used. The checkpointing period can be computed based on the $MTTI$ value using Young's approximation [12] or Daly's first-order approximation [11]. We use Daly's approximation in this work because it is classical, often used in practice, and used in previous work [10]. It would be also interesting to present results obtained with the optimal checkpointing period, so as to evaluate the impact of the choice of the checkpointing period on our results. However, deriving the optimal period is not tractable [41]. However, since our experiments are in simulation, we can search numerically for the best period among a sensible set of candidate periods. To build the candidate periods, we use the period computed in [16]

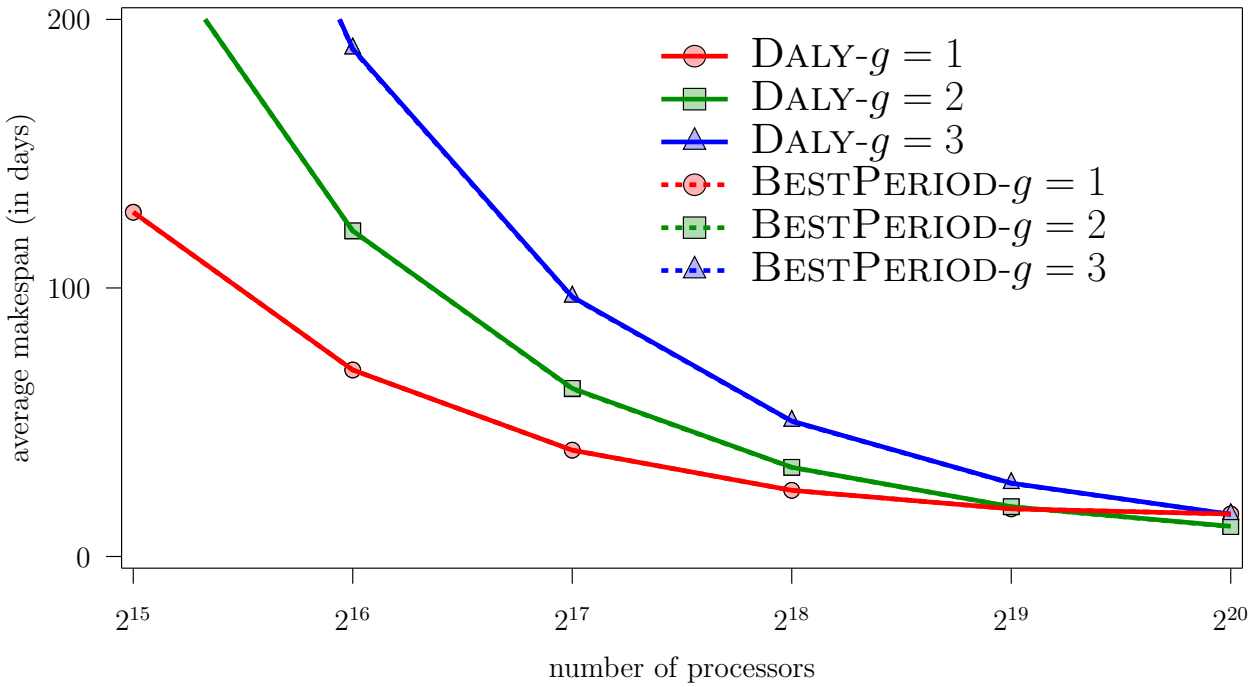


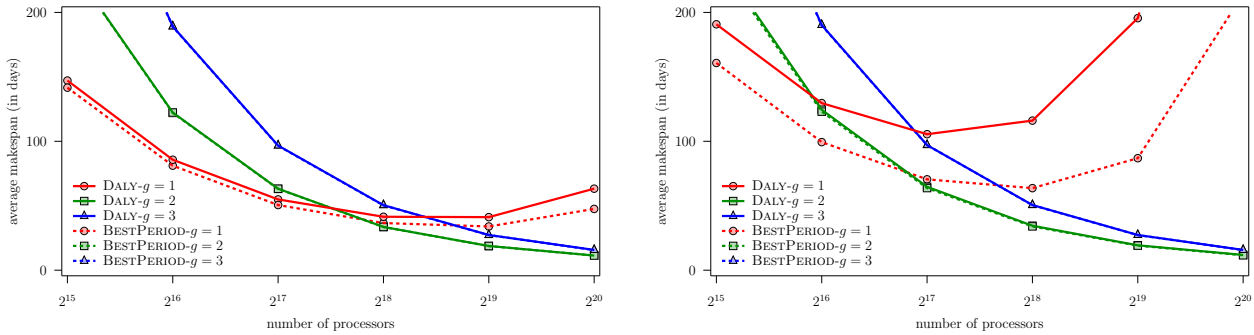
Figure 2: Average makespan vs. number of processors for two choices of the checkpointing period, without process replication (DALY-g=1 and BESTPERIOD-g=1) and with process replication (DALY-g=2 or 3 and BESTPERIOD-g=2 or 3), for generic parallel jobs subject to Exponential failures ($MTBF = 125$ years). Each solid curve overlaps its dotted counterpart.

(called OPTEXP) as a starting point. We then multiply and divide this period by $1 + 0.05 \times i$ with $i \in \{1, ..., 180\}$, and by 1.1^j with $j \in \{1, ..., 60\}$ and pick among these the value that leads to the lowest makespan. For a given replication level $x \geq 1$ (g=x), we present results with the period computed using Daly's approximation (DALY-g=x) and with the best candidate period found numerically (BESTPERIOD-g=x).

5.2 Choice of the checkpointing period

Our first set of experiments aims at determining whether using Daly's approximation for computing the checkpointing period, as done in [10], is a reasonable idea when replication is used. In the $g = 2$ case we compute this period using the exact $MTTI$ expression from Corollary 1. In the $g = 3$ case, since the formula from Theorem 4 is not numerically stable, we compute the $MTTI$ using numerical simulation (as explained in Section 4.3). Given a failure distribution and a parallel job model, we compute the average makespan over 100 sample simulated application executions for a range of numbers of processors. Each sample is obtained using a different seed for generating random failure events based on the failure distribution. In addition to the $g = 2$ and $g = 3$ results, we also present results for $g = 1$ (no replication) as a baseline, in which case the $MTTI$ is simply the processor MTBF. In the three options the total number of processors is the same, i.e., $g \times p/g$.

We run experiments for 6 failure distributions: (i) Exponential with a 125-year MTBF; (ii) Weibull with a 125-year MTBF and shape parameter $k = 0.70$; (iii) Weibull with a 125-year MTBF and shape parameter $k = 0.50$; (iv) Weibull with a 125-year MTBF and shape parameter $k = 0.156$; (v) Failures drawn from the failure log of LANL cluster 18; and (vi) Failures drawn from the failure



(a) $k = 0.70$ (b) $k = 0.50$

Figure 3: Same as Figure 2 (generic parallel jobs) but for Weibull failures ($MTBF = 125$ years). The solid curves for $g = 2$ and $g = 3$ overlap their dotted counterparts.

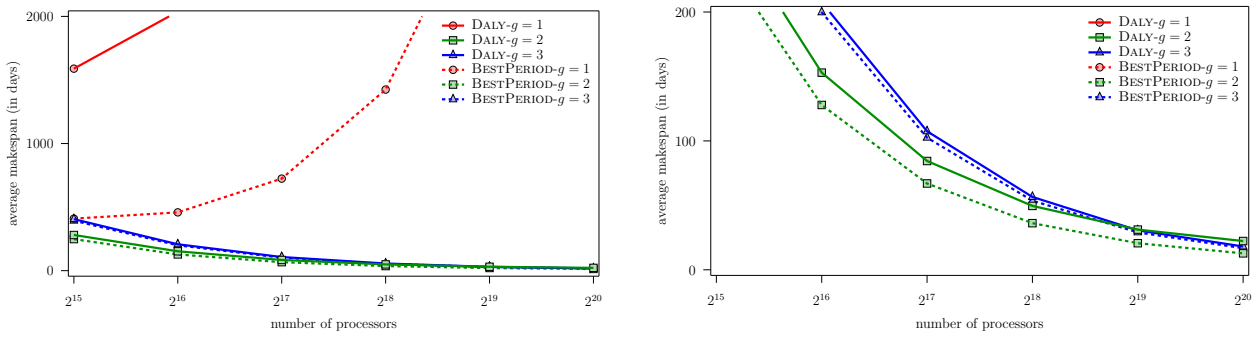
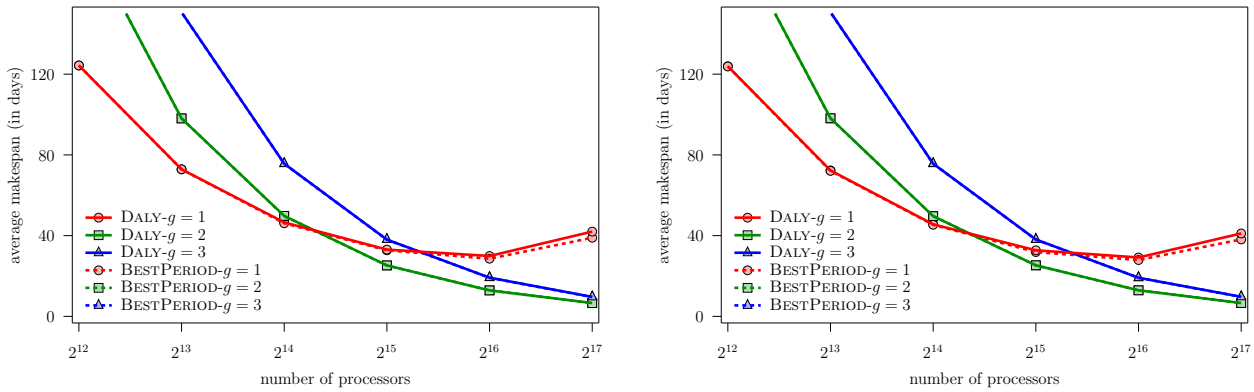


Figure 4: Same as Figure 2 (generic parallel jobs) but for Weibull failures ($k = 0.156$ and $MTBF = 125$ years). The right-hand side is a zoomed-in version of the right-hand side.

log of LANL cluster 19. For each failure distribution, we use five parallel job models as described in Section 3.1, augmented with the replication overhead model described in Section 5.1: (i) perfectly parallel; (ii) generic parallel jobs with $\gamma = 10^{-6}$; (iii) numerical kernels with $\gamma = 0.1$; (iv) numerical kernels with $\gamma = 1$; and (v) numerical kernels with $\gamma = 10$. We thus have $6 \times 5 = 30$ sets of results. Figures 2 through 6 show average makespan vs. number of processors. It turns out that, for a given failure distribution, all results follow the same trend regardless of the job model, as illustrated in Figure 6 for Weibull failures with $k = 0.7$. Due to lack of space, with the exception of Figure 6 we only show results for the generic parallel job model.

Figures 2 to 5 show results for generic parallel jobs subject to each of the 6 considered failure distributions. The two curves for $g = 1$ are exactly superposed in Figure 2. For $g = 2$ and for $g = 3$ the two curves are exactly superposed in all figures but for Figure 4. We first remark that when no replication is used (i.e., $g = 1$), the minimum makespan is not achieved on the largest platform unless failures follow an Exponential distribution. The fact that in most cases the makespan with 2^{19} processors is lower than the makespan with 2^{20} processors suggests that process duplicating should be beneficial. This is indeed always the case for the largest platforms: when using 2^{20} processors, the makespan without replication is always larger than the makespan with replication, the replication factor being either $g = 2$ or $g = 3$. However, in almost all cases using a replication with $g = 2$ is more beneficial than with $g = 3$. The exception is for Weibull failures with $k = 0.156$. Regardless, in each tested configuration, the minimum makespan is always achieved while duplicating the processes and using the maximum number of processors.



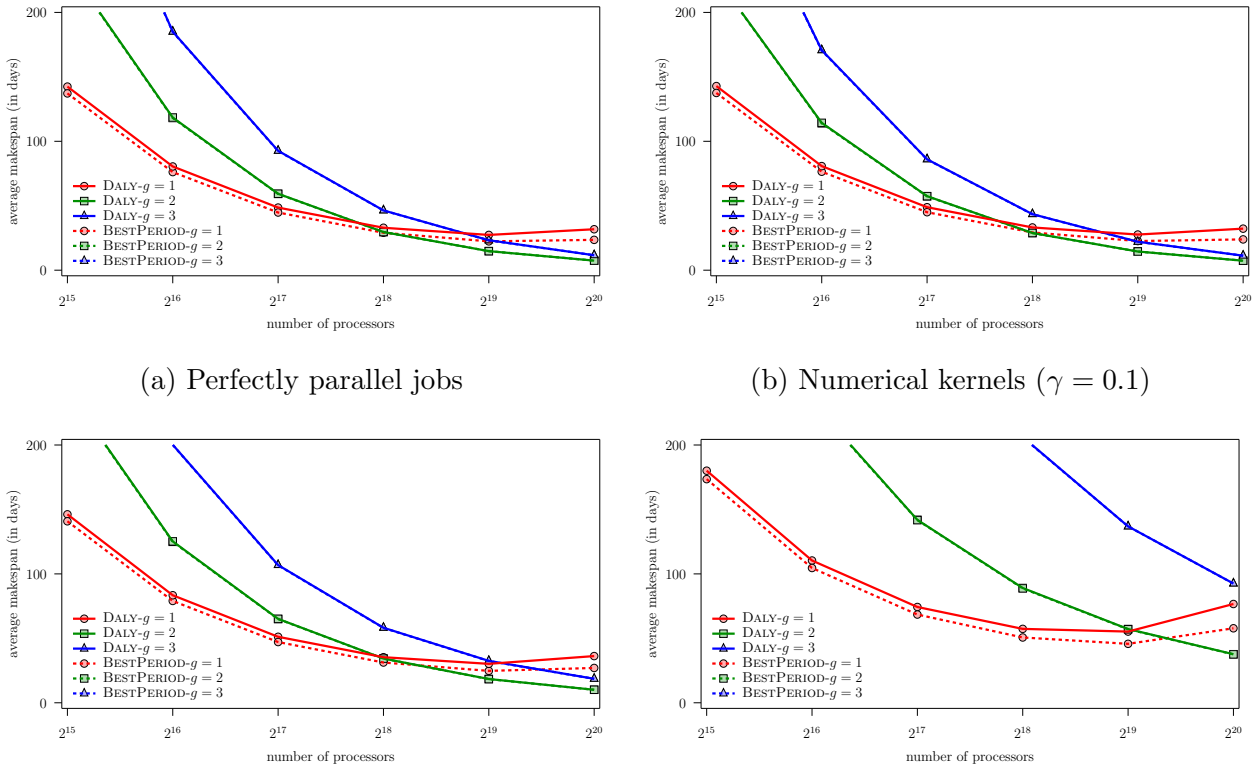
(a) LANL cluster 18 (b) LANL cluster 19

Figure 5: Same as Figure 2 (generic parallel jobs) but for real-world failures. The solid curves for $g = 2$ and $g = 3$ overlap their dotted counterparts.

Results for the case $g = 1$ (no replication) show that Daly's approximation achieves the same performance as the best periodic checkpointing policy for Exponential failures. For our two real-world failure datasets, using the approximation also does well, deviating from the best periodic checkpointing policy only marginally as the platform becomes large. For Weibull failures, however, Daly's approximation leads to significantly suboptimal results that worsen as k decreases (as expected and already reported in [16]). However, in the cases $g = 2$ and $g = 3$, using Daly's approximation leads to virtually the same performance as using the best period even for Weibull failures with $k = 0.7$ and $k = 0.5$. The only significant difference is seen for $k = 0.156$ and $g = 2$, which shows the limit of approximating a Weibull distribution with such a small shape parameter k by an Exponential distribution of same mean. For higher values of k, with replication the application makespan is simply not sensitive to the checkpointing period, at least in a wide neighborhood around the best period. This is because application failures and recoveries are infrequent, i.e., the MTBF of a pair of replicas is large.

To quantify the frequency of application failures, Table 4 shows the percentage of processor failures that actually lead to failure recoveries when using process replication. Results are shown in the case of Weibull failures for $k = 0.156$, $k = 0.5$, and $k = 0.7$, and for $C = 600s$ and various numbers of processors. We see that for $k = 0.5$ and $k = 0.7$, very few application failures, and thus recoveries, occur throughout application execution (recall that makespans are measured in days in our experiments). With $k = 0.156$, the number of application failures is more than one order of magnitude larger. Table 4 also shows the fraction of processor failures that manifest themselves as application failures. This fraction is low, below 0.3% in our experiments, which explains why using $g = 3$ replicas does not lead to any improvement over using $g = 2$ (recall that the expectation was that further improvement would be low anyway given the results in Tables 2 and 3). Note that when setting the processor MTBF to a lower value so that the MTBF of a pair of replicas is not as large, one does observe that the choice of the checkpointing period matters even for large values of k with Weibull failures. Consider for instance a process replication scenario with Weibull failures with $k = 0.7$, a generic parallel job, and a platform with 2^{20} processors. When setting the MTBF to an unrealistic 0.1 year, using Daly's approximation yields an average makespan of 22.7 days, as opposed to 19.1 days when using the best period–an increase of more than 18%. Similar examples can be found for Exponential failures.

We summarize our findings so far as follows. Without replication, a poor choice of checkpointing period produces significantly suboptimal performance. With replication, the number of application



(a) Perfectly parallel jobs

(b) Numerical kernels ($\gamma = 0.1$)

(c) Numerical kernels ($\gamma = 1$)

(d) Numerical kernels ($\gamma = 10$)

Figure 6: Same as Figure 3(a) (Weibull failures with $k = 0.7$, $MTBF = 125$ years) but for other job types. The solid curves for $g = 2$ and $g = 3$ overlap their dotted counterparts.

failures is drastically reduced. As a result, unless failures are very far from being Exponentially distributed (i.e., Weibull distribution with very low k) or the MTBF is very low, the choice of the checkpointing period is not critical and it is sufficient to pick a period in a large neighborhood of the optimal period. In these conditions setting the checkpointing period based on an approximation, Daly's being the most commonplace and oft referenced, is appropriate.

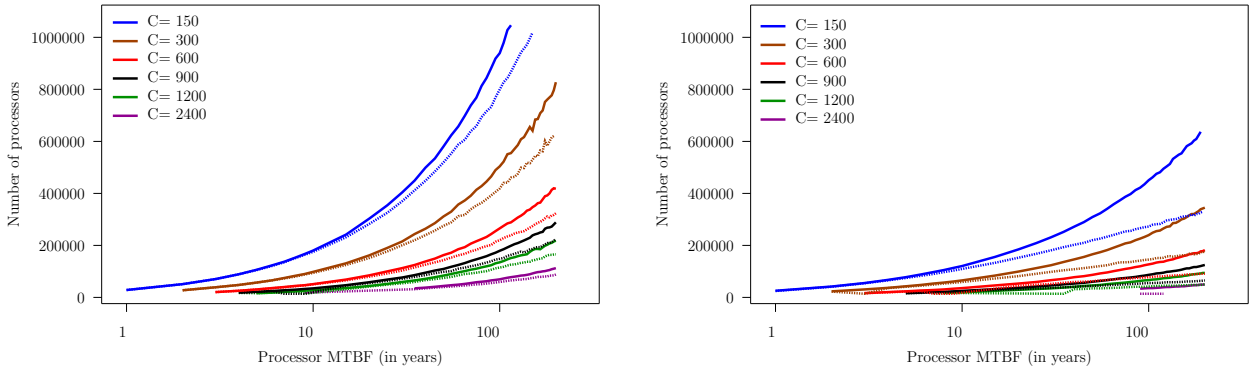
5.3 When is process replication beneficial?

In this section we determine under which conditions process replication is beneficial, i.e., leads to a lower makespan, when compared to a standard application execution that uses only checkpoint-rollback. We restrict our study to duplication ($g = 2$) as we have seen that the case $g = 3$ is rarely beneficial in our results.

One question when comparing the replication and the no-replication cases is that of the checkpointing period. We have seen in the previous section that when using process replication Daly's approximation can be used safely in most cases. In the no-replication case, however, Daly's approximation should only be used in the case of exponentially distributed failures as it leads to poor results when the failure distribution is Weibull (e.g., see the $g = 1$ curves in Figure 3). Although our results for two particular production workloads show that Daly's approximation leads to reasonably good results in the no-replication case (see the $g = 1$ curves in Figures 5), there is evidence that, in general, failure distributions are well approximated by Weibull distributions [18, 19, 20, 22, 21],

Table 4: Number of application failures and fraction of processor failures that cause application failures with process replication ($g = 2$) assuming Weibull failure distributions ($k = 0.7$, 0.5, or 0.156) for generic parallel jobs with various numbers of processors and C=600s. Results are averaged over 100 experiments.

	# of app. failures			% of proc. failures		
# of proc.	$k = 0.7$	$k = 0.5$	$k = 0.156$	$k = 0.7$	$k = 0.5$	$k = 0.156$
2^{15}	1.44	3.77	37.1	0.25	0.28	0.23
2^{16}	0.88	2.61	30.6	0.15	0.19	0.17
2^{17}	0.45	1.67	24.2	0.075	0.12	0.12
2^{18}	0.20	1.11	19.6	0.034	0.076	0.089
2^{19}	0.13	0.72	16.1	0.022	0.049	0.063
2^{20}	0.083	0.33	14.8	0.014	0.023	0.046



(a) $k = 0.70$ (b) $k = 0.50$

Figure 7: Break-even point curves for replication ($g = 2$) vs. no-replication for generic parallel jobs with various checkpointing overheads, as computed using the best checkpointing periods (solid lines) and Daly's approximation (dashed lines), assuming Weibull failure distributions.

while not at all by exponential distributions. Most recently, in [22], the authors show that failures observed on a production cluster, over a cumulative 42-month time period, are modeled well by a Weibull distribution with shape parameter $k < 0.5$. And in this work we have used the $k = 0.156$ value from [21]. Given that Daly's approximation is based on the exponential distribution, the further the failure distribution from the exponential distribution the further Daly's approximation from the best period. This is illustrated in our results by the widening gap between the DALY-$g = 1$ and BESTPERIOD-$g = 1$ curves as k decreases in Figures 3(a) ($k = 0.7$), 3(b) ($k = 0.5$), and 4 ($k = 0.156$).

Given the above, comparing the replication case to the no-replication case with Weibull failure distributions and using Daly's approximation as the checkpointing period gives an unfair advantage to process replication. This is how the comparison is done in [10]. In this work, instead, to isolate the effect of replication from checkpointing period effects we opt for the following method: we always use the best checkpointing period for each simulated application execution, as computed by a numerical search over a range of simulated executions each with a different checkpointing period.

In a 2-D plane defined by the processor MTBF and the number of processors, and given a checkpointing overhead, simulation results can be used to construct a curve that divides the plane into two regions. Points above the curve correspond to cases in which process replication is beneficial. Points below the curve correspond to cases in which process replication is detrimental, i.e.,

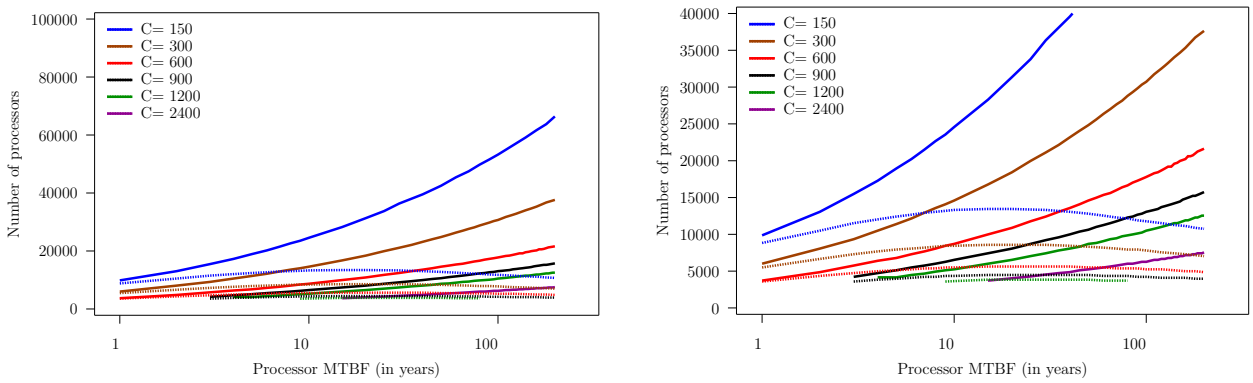


Figure 8: Break-even point curves for replication ($g = 2$) vs. no-replication for generic parallel jobs with various checkpointing overheads, as computed using the best checkpointing periods (solid lines) and Daly's approximation (dashed lines), assuming Weibull failure distributions with shape parameter $k = 0.156$. The right-hand side is a zoomed-in version of left-hand side.

the resource waste due to replication is not worthwhile because the processor MTBF is too large or the number of processors is too low. These results, for $g = 2$, are shown as solid curves in Figure 7, for Weibull failures with $k = 0.7$ and $k = 0.5$, and in Figure 8 for Weibull failures with $k = 0.156$, each curve corresponding to a different checkpointing overhead (C) value (see [37] for results with exponentially distributed failures). Several such curves are shown in [10] (Figure 9 therein) for different checkpointing overheads. For comparison purposes, Figures 7 and 8 also show sets of dashed curves that correspond to results obtained when using Daly's approximation as the checkpointing period instead of using our numerical search for the best such period.

We see that, as expected, using Daly's approximation gives an unfair advantage to process replication. In other terms, in Figure 7, each dashed curve is below its solid counterpart. This advantage increases as k decreases, since the Weibull distribution is then further away from the Exponential distribution. (For exponential distributions, the curves match.) For instance, for $k = 0.5$ (Figure 7(b)), the break-even curve for $C = 600s$ as obtained using Daly's approximation is in fact, for most values of the MTBF, below the break-even curve for $C = 900s$ as obtained using the best checkpointing period. For a low k value of 0.156, the Weibull distribution is very far from an exponential distribution. As seen in Figure 8, for $k = 0.156$, using Daly's approximation leads to such an unfair advantage for process replication that the break-even curves exhibit incorrect trends, i.e., decreasing as the MTBF increases (which would indicate that the use of process replication is increasingly beneficial as processors become more reliable).

Considering the solid curves, we find that replication is detrimental when the number of processors is too small, when the checkpointing overhead is too low, and/or when the processor MTBF is too large. While these trends are expected, quantitative results such as those in Figures 7 and 8 make it possible to determine the regimes in which process replication should be used.

6 Conclusion

In this paper we have presented a rigorous study of process replication for large-scale platforms. We have obtained recursive expressions for *MNFTI*, and analytical expressions for *MTTI* with arbitrary distributions, which lead to closed-form expressions for Exponential and Weibull distributions. We have also identified an unexpected relationship between two natural failure models (*already hit* and *running processors*) in the case of process duplication ($g = 2$). We have conducted an extensive

set of simulations for Exponential, Weibull, and trace-based failure distributions. These results show that the choice of a good checkpointing period can be critical in the no-replication case, namely for Weibull failure distributions. By contrast, this choice is no longer critical when process replication is used unless the MTBF is (unrealistic) low or the failure distribution is very non-exponential (i.e., Weibull with low k). This is because with process replication few processor failures lead to application failures (i.e., rollback and recovery). This effect is essentially the reason why process replication was proposed in the first place. But a surprising and interesting side-effect is that choosing a good checkpointing period is likely no longer challenging in many practical situations. Finally, we have determined the break-even point between replication and no-replication for Weibull failures. Unlike that in previous work, this determination is agnostic to the choice of the checkpointing period, leading to results that are not as favorable for process replication. Our results nevertheless point to relevant scenarios, defined by instantiations of the platform and application parameters, in which replication is worthwhile when compared to the no-replication case. This is in spite of the induced resource waste, and even if the best checkpointing period is used in the no-replication case. Finally, our results also have laid the necessary theoretical foundations for future studies of process replication.

Further work should investigate the impact of *partial* replication instead of full replication as studied in this paper. In this approach, replication would be used only for critical components (e.g., message loggers in uncoordinated checkpoint protocols), while traditional checkpointing would be used for non-critical components. The goal would be to reduce the overhead of replication while still achieving some of its benefits in terms of resilience.

Another direction for future work is to study the impact of resilience techniques on energy consumption. Together with fault-tolerance, energy consumption is expected to be a major challenge for exascale machines [1, 2]. A promising next step in this search is the study of the interplay between checkpointing, replication, and energy consumption. By definition, both checkpointing and replication induce additional power consumption, but both techniques lead to faster executions in expectation. There are thus various energy trade-offs to achieve. A key question is to determine the best execution strategy given both an energy budget and a maximum admissible application makespan.

Acknowledgment

We would like to thank Kurt Ferreira and Leonardo Bautista Gomez for discussions. This work was supported in part by the French ANR White Project *RESCUE*. Yves Robert is with Institut Universitaire de France.

References

[1] J. Dongarra, P. Beckman, P. Aerts, F. Cappello, T. Lippert, S. Matsuoka, P. Messina, T. Moore, R. Stevens, A. Trefethen, M. Valero, The international exascale software project: a call to cooperative action by the global high-performance community, Int. J. High Perform. Comput. Appl. 23 (4) (2009) 309–322. doi:http://dx.doi.org/10.1177/1094342009347714.

[2] V. Sarkar, W. Harrod, A. Snavely, Software challenges in extreme scale systems, J. Phys.: Conf. Ser. 180 (1).

[3] E. Elnozahy, J. Plank, Checkpointing for Peta-Scale Systems: A Look into the Future of Practical Rollback-Recovery, IEEE Transactions on Dependable and Secure Computing 1 (2) (2004) 97—108.

[4] R. A. Oldfield, S. Arunagiri, P. J. Teller, S. Seelam, M. R. Varela, R. Riesen, P. C. Roth, Modeling the Impact of Checkpoints on Next-Generation Systems, in: Proc. of the 24th IEEE Conference on Mass Storage Systems and Technologies, 2007, pp. 30—46.

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

[6] G. Zheng, X. Ni, L. Kale, A scalable double in-memory checkpoint and restart scheme towards exascale, in: Dependable Systems and Networks Workshops (DSN-W), 2012. doi:10.1109/DSNW.2012.6264677.

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

[8] Z. Zheng, Z. Lan, Reliability-aware scalability models for high performance computing, in: Proc. of the IEEE Conference on Cluster Computing, 2009.

[9] C. Engelmann, H. H. Ong, S. L. Scorr, The case for modular redundancy in large-scale highh performance computing systems, in: Proc. of the 8th IASTED Infernational Conference on Parallel and Distributed Computing and Networks (PDCN), 2009, pp. 189–194.

[10] K. Ferreira, J. Stearley, J. H. I. Laros, R. Oldfield, K. Pedretti, R. Brightwell, R. Riesen, P. G. Bridges, D. Arnold, Evaluating the Viability of Process Replication Reliability for Exascale Systems, in: Proc. 2011 Int. Conf. High Performance Computing, Networking, Storage and Analysis SC '11, ACM Press, 2011.

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

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

[13] W. Jones, J. Daly, N. DeBardeleben, Impact of sub-optimal checkpoint intervals on application efficiency in computational clusters, in: HPDC'10, ACM, 2010, pp. 276–279.

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

[15] M.-S. Bouguerra, T. Gautier, D. Trystram, J.-M. Vincent, A flexible checkpoint/restart model in distributed systems, in: PPAM, Vol. 6067 of LNCS, 2010, pp. 206–215.
URL http://dx.doi.org/10.1007/978-3-642-14390-8_22

[16] M. Bougeret, H. Casanova, M. Rabie, Y. Robert, F. Vivien, Checkpointing strategies for parallel jobs, in: Proc. 2011 Int. Conf. High Performance Computing, Networking, Storage and Analysis SC '11, ACM Press, 2011.

[17] K. Balakrishnan, A. P. Basu (Eds.), Exponential Distribution: Theory, Methods and Applications, CRC Press, 1996.

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

[19] B. Schroeder, G. A. Gibson, A large-scale study of failures in high-performance computing systems, in: Proc. of DSN, 2006, pp. 249–258.

[20] Y. Liu, R. Nassar, C. Leangsuksun, N. Naksinehaboon, M. Paun, S. Scott, An optimal checkpoint/restart model for a large scale high performance computing system, in: IPDPS 2008, IEEE, 2008, pp. 1–9.

[21] T. J. Hacker, F. Romero, C. D. Carothers, An Analysis of Clustered Failures on Large Supercomputing Systems, Journal of Parallel and Distributed Computing 69 (7) (2009) 652–665.

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

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

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

[25] C. Engelmann, B. Swen, Redundant execution of HPC applications with MR-MPI, in: PDCN, IASTED, 2011.

[26] T. Leblanc, R. Anand, E. Gabriel, J. Subhlok, Volpexmpi: An mpi library for execution of parallel applications on volatile nodes, in: 16th European PVM/MPI Users' Group Meeting, Springer-Verlag, 2009, pp. 124–133.

[27] J. Elliott, K. Kharbas, D. Fiala, F. Mueller, K. Ferreira, C. Engelmann, Combining partial redundancy and checkpointing for HPC, in: ICDCS'12, IEEE, 2012.

[28] J. Stearley, K. B. Ferreira, D. J. Robinson, J. Laros, K. T. Pedretti, D. Arnold, P. G. Bridges, R. Riesen, Does partial replication pay off?, in: FTXS (a DSN workshop), IEEE, 2012.

[29] C. George, S. S. Vadhiyar, Adft: An adaptive framework for fault tolerance on large scale systems using application malleability, Procedia Computer Science 9 (2012) 166 – 175.

[30] G. Amdahl, The validity of the single processor approach to achieving large scale computing capabilities, in: AFIPS Conf. Proceedings, Vol. 30, AFIPS Press, 1967, pp. 483–485.

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

[32] N. Kolettis, N. D. Fulton, Software rejuvenation: Analysis, module and applications, in: FTCS '95, IEEE CS, Washington, DC, USA, 1995, p. 381.

[33] V. Castelli, R. E. Harper, P. Heidelberger, S. W. Hunter, K. S. Trivedi, K. Vaidyanathan, W. P. Zeggert, Proactive management of software aging, IBM J. Res. Dev. 45 (2) (2001) 311–332.

[34] L. Wang, P. Karthik, Z. Kalbarczyk, R. Iyer, L. Votta, C. Vick, A. Wood, Modeling Coordinated Checkpointing for Large-Scale Supercomputers, in: Proc. of the International Conference on Dependable Systems and Networks, 2005, pp. 812–821.

[35] A. Guermouche, T. Ropars, M. Snir, F. Cappello, HydEE: Failure Containment without Event Logging for Large Scale Send-Deterministic MPI Applications, in: IPDPS'12, IEEE, 2012.

[36] D. Fiala, F. Mueller, C. Engelmann, R. Riesen, K. Ferreira, R. Brightwell, Detection and correction of silent data corruption for large-scale high-performance computing, in: Proc. of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC), 2012.

[37] H. Casanova, Y. Robert, F. Vivien, D. Zaidouni, Combining process replication and checkpointing for resilience on exascale systems, Research report RR-7951, INRIA (May 2012).

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

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

[40] D. Kondo, B. Javadi, A. Iosup, D. Epema, The failure trace archive: Enabling comparative analysis of failures in diverse distributed systems, in: Cluster Computing, IEEE Computer Society, 2010.

[41] M. Bougeret, H. Casanova, Y. Robert, F. Vivien, D. Zaidouni, Using replication for resilience on exascale systems, Research Report RR-7830, INRIA (2011).
URL `http://hal.inria.fr/hal-00650325`