Combining Process Replication and Checkpointing for Resilience on Exascale Systems

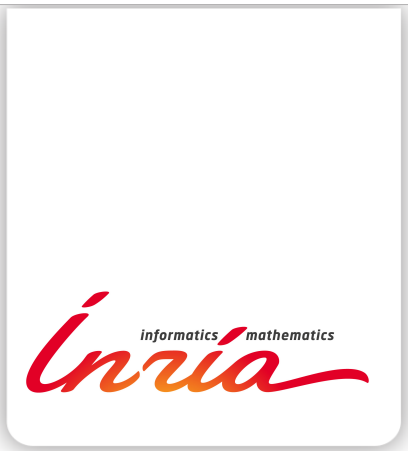
Henri Casanova, Yves Robert, Frédéric Vivien, Dounia Zaidouni

Combining Process Replication and Checkpointing for Resilience on Exascale Systems

Henri Casanova, Yves Robert, Frédéric Vivien, Dounia Zaidouni

RESEARCH REPORT
N° 7951
March 2013
Project-Team ROMA

ISSN 0249-6399 ISRN INRIA/RR--7951--FR+ENG



Combining Process Replication and Checkpointing for Resilience on Exascale Systems

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

Project-Team ROMA

Research Report n° 7951 — March 2013 — 42 pages

Abstract: Processor failures in post-petascale parallel computing platforms are common occurrences. The traditional fault-tolerance solution, checkpoint-rollback, 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, has been recently advocated in the literature. 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 for Exponential. 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, identifying scenarios in which process replication is beneficial. We also find that although the choice of the checkpointing period can have a high impact on application execution in the no-replication case, this choice is no longer critical when process replication is used.

Key-words: Fault-tolerance, parallel computing, checkpoint/restart, process replication

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

RESEARCH CENTRE GRENOBLE – RHÔNE-ALPES

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

Utilisation conjointe de la réplication et de la prise de points de sauvegarde pour la résilience sur plates-formes exascales

Résumé : Les pannes de processeurs seront des évènements courants dans les plates-formes post-petascale de calcul parallèle. La solution traditionnelle de tolérance aux pannes, la prise de points de sauvegarde et la ré-exécution, limite fortement l'efficacité des applications parallèles. Pour lever cette limitation, une solution est de répliquer les processus de l'application pour qu'une panne sur un processeur n'entraîne pas automatiquement une panne de l'application. La combinaison de la réplication de processus avec la prise de points de sauvegarde a été récemment préconisée dans la littérature. Nous dérivons d'abord des nouveaux résultats théoriques pour une distribution exponentielle des pannes: nous établissons des formules exactes pour le Nombre Moyen de Pannes avant l'Échec, et le Temps Moyen avant l'Échec. Nous étendons ensuite ces résultats à n'importe quel type de distribution, avec notamment des formules closes pour les distributions suivant des lois de Weibull. Finalement, nous évaluons la réplication de processus en utilisant des traces synthétiques et des traces réelles, et nous identifions les scénarios dans lesquels la réplication est bénéfique. Nous trouvons que le choix de la période de prise des points de sauvegarde peut avoir un fort impact sur la durée d'exécution des applications quand il n'y a pas de réplication. Par contre, avec la réplication des processus, le choix de la période n'est plus important.

Mots-clés : Tolérance aux pannes, calcul parallèle, checkpoint/redémarrage, réplication

1 Introduction

As plans are made for deploying post-petascale high performance computing (HPC) systems [1, 2], solutions need to be developed to ensure that applications on these systems are resilient to processor failures. Resilience is particularly critical for applications that enroll large numbers of processors, including those applications that are pushing the limit of current computational capabilities and that could benefit from enrolling all available processors. For such applications, processor failure 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. These 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-recovery for resuming application execution from a previously saved fault-free execution state, or *checkpoint*. 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 literature is devoted to rollback-recovery, 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 it is known that 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. While replication wastes compute resources in fault-free executions, it can alleviate the poor scalability of rollback-recovery. 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 processes so that there are two replicas of each process, each running on a distinct physical processor. This approach 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 in order to improve parallel efficiency. In [7] Ferreira et al. have presented the process replication approach along 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) 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 the choice of a good checkpointing period is no longer critical when process replication is used.

- Based on the above results, we determine in which conditions the use of process replication becomes beneficial. Importantly, 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 the literature. In [8], Daly studies periodic checkpointing policies for Exponential failures, generalizing the well-known bound obtained by Young [9]. Daly extended his work in [10] to study the impact of sub-optimal checkpointing periods. In [11], the authors develop an "optimal" checkpointing policy, based on the popular assumption that optimal checkpointing must be periodic. In [12], 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 [13], the authors show that this assumption is unreasonable for Weibull failures. They propose optimal solutions for Exponential failures and dynamic programming solutions for Weibull failures. Note that while Exponential failures are often studied in the literature due to the memoryless property of the Exponential distribution, which makes analytical developments more tractable, the Weibull distribution is recognized as a more reasonable approximation of failures in real-world systems [14, 15, 16, 17]. The work in this paper relates to checkpointing policies in the sense that we study a replication mechanism that is complementary to checkpointing.

In spite of all the above advances, several studies have questioned the feasibility of pure rollback-recovery for large-scale systems [3, 4, 5]. Replication has long been used as a fault-tolerance mechanism in distributed systems [18], and more recently in the context of volunteer computing [19]. The idea to use replication together with checkpoint-recovery has been studied in the context of grid computing [20]. One concern about the use of replication for HPC is the induced resource waste. However, given the scalability limitations of pure rollback-recovery, replication has recently received more attention in the HPC literature [21, 22, 23]. Most recently, the work by Ferreira et al. [7] 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. In this work we focus on the theoretical analysis of the problem. While some theoretical results are provided in [7], 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.

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. We use the term processor to indicate any individually scheduled compute resource (a core, a multi-core processor, a cluster node), so that our work is agnostic to the granularity of the platform. We assume that a standard checkpointing and roll-back recovery is performed at the system level. At most on 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*. The job can execute on any number $q \leq p$ processors. Letting $\mathcal{W}(q)$ be the time required for a failure-free execution on q processor, we use 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 [24], $\gamma < 1$ 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 algorithm in [25], $q = r^2$ 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*. A failure causes a *downtime* period of the failed processor. We do not distinguish between soft and hard errors, with the understanding that soft errors are handled via software rejuvenation (i.e., rebooting [26, 27]) and that hard errors are handled by the replacement of the failed processor by a spare, a commonplace approach in production systems. For simplicity we assume that the downtime of a failure is D, regardless of the failure type. After a downtime the processor is fault-free and begins a new lifetime. When a processor fails, the whole execution is stopped, and all processors must recover from the previous checkpointed state. We assume coordinated checkpointing [28] so that no message logging/replay is needed for recovery. We allow failures to happen during recovery or checkpointing, but not during a downtime (otherwise, the downtime could 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 because it makes it possible to obtain closed-form expressions for several relevant expectations, as seen in future sections. In the real world, instead, failures are bound to be correlated. Obtaining theoretical results for non-i.i.d. failures is beyond the scope of this work. But note that one cause of correlation is the hierarchical

structure of compute platforms (each rack comprises compute nodes, each compute node comprises processors, each processor comprises cores), which leads to simultaneous failures of groups of processors. Our work applies to such failures since all processors perform a coordinated recovery after a failure. This recovery lasts the time needed to restore the last checkpoint.

We let $C(q)$ denote the time needed to perform a checkpoint, and $R(q)$ the time to perform a recovery. Assuming that the application's memory footprint is V bytes, with each processor holding V/q bytes, we consider two scenarios:

- Proportional overhead: $C(q) = R(q) = \alpha V/q = C/q$ for some constant α. This is representative of cases where the bandwidth of the network card/link at each processor is the I/O bottleneck.
- Constant overhead: $C(q) = R(q) = \alpha V = C$, 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 [7], 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 checkpoint frequency.

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 copying-time would be a time during which the execution of the whole application must be paused. 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 [7], we do not consider it in this work.

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 a given 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 the other processor, in the case $g = 2$, both replicas of a same process are killed and the whole application fails. This is the option chosen in [7]. Another option is to count only failures that hit *running processors*, and thus effectively kill replicas. This approach may seem more natural as the running processors are the only ones that are important for the application execution.

We consider both options above. 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:

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

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

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

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

Proof. Let $\mathbb{E}(NFTI^{\text{rp}}|n_f)$ 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 and hence leads to an application failure, hence

$$\mathbb{E}\left(NFTI^{\mathrm{rp}}\,|n_{rg}\right) = 1.$$

For the general case $0 \leq n_f \leq n_{rg} - 1$, either the next failure hits a new replica-group with 2 still running replicas, or it hits a replica-group that had already been hit. The latter case leads to an application failure; in that case, after n_f failures, the expected number of failures needed for the whole application to fail is exactly one. The failure probability is uniformly distributed among the $2n_{rg} - n_f$ running processors, hence the probability that the next failure hits a new replica-group is $\frac{2n_{rg}-2n_f}{2n_{rg}-n_f}$. In this case, the expected number of failures needed for the whole application to fail is one (the considered failure) plus $\mathbb{E}\left(NFTI^{\mathrm{rp}}|n_f + 1\right)$. Altogether we have derived that:

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

Therefore,

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

□

Note that Theorem 2 does not make any assumption on the failure distribution; it only assumes that failures are i.i.d. However, to establish Theorem 1, an additional assumption is that the probability of failures of a node is not affected by the fact that it may have already been hit. This assumption seems to restrict this theorem to failures following Exponential (i.e., memoryless) distributions.

It turns out that both failure counting options lead to very similar results:

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

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

Proof. We prove by induction that $\mathbb{E}(NFTI^{\mathrm{ah}}|n_f) = 1 + \mathbb{E}(NFTI^{\mathrm{rp}}|n_f)$, for any $n_f \in [0, n_{rg}]$. The base case is for $n_f = n_{rg}$ and the induction uses non-increasing values of n_f.

For the base case, we have $\mathbb{E}(NFTI^{\mathrm{rp}}|n_{rg}) = 1$ and $\mathbb{E}(NFTI^{\mathrm{ah}}|n_{rg}) = 2$. Hence the property is true for $n_f = n_{rg}$. Consider a value $n_f < n_{rg}$, and assume to have proven that $\mathbb{E}(NFTI^{\mathrm{ah}}|i) = 1 + \mathbb{E}(NFTI^{\mathrm{rp}}|i)$, for any value of $i \in [1 + n_f, n_{rg}]$. We now prove the equation for n_f. According to Theorem 1, we have:

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

Therefore, using the induction hypothesis, we have:

$$\begin{aligned}
&\mathbb{E}(NFTI^{\text{ah}}|n_f) \\
&= \tfrac{2n_{rg}}{2n_{rg}-n_f} + \tfrac{2n_{rg}-2n_f}{2n_{rg}-n_f}\left(1+\mathbb{E}\left(NFTI^{\text{rp}}|n_f+1\right)\right) \\
&= 2 + \tfrac{2n_{rg}-2n_f}{2n_{rg}-n_f}\mathbb{E}\left(NFTI^{\text{rp}}|n_f+1\right) \\
&= 1 + \mathbb{E}\left(NFTI^{\text{rp}}|n_f\right)
\end{aligned}$$

the last equality being established using Theorem 2. Therefore, we have proved by induction that $\mathbb{E}(NFTI^{\text{ah}}|0) = 1 + \mathbb{E}(NFTI^{\text{rp}}|0)$. To conclude, we remark that $\mathbb{E}(NFTI^{\text{ah}}|0) = MNFTI^{\text{ah}}$ and $\mathbb{E}(NFTI^{\text{rp}}|0) = MNFTI^{\text{rp}}$. □

We now show that Theorems 1 and 2 can be generalized to $g > 2$. Because the proofs are very similar, we only give the one for the $MNFTI^{\text{rp}}$ accounting approach (failures on running processors only), as it does not make any assumption on failures besides the i.i.d. assumption.

Proposition 2. *If the failure inter-arrival times on the different processors are i.i.d. then using process replication for* $g \geq 2$, $MNFTI^{\text{rp}} = \mathbb{E}\left(NFTI^{\text{rp}}|\underbrace{0,...,0}_{g-1\ zeros}\right)$ *where:*

$$\begin{aligned}
\mathbb{E}\left(NFTI^{\text{rp}}|n_f^{(1)},...,n_f^{(g-1)}\right) = 1 \\
&+\frac{g\cdot\left(n_{rg}-\sum_{i=1}^{g-1}n_f^{(i)}\right)}{g\cdot n_{rg}-\sum_{i=1}^{g-1}i\cdot n_f^{(i)}} \\
&\quad\cdot\mathbb{E}\left(NFTI^{\text{rp}}|n_f^{(1)},n_f^{(2)},...,n_f^{(g-1)}\right) \\
&+\sum_{i=1}^{g-2}\frac{(g-i)\cdot n_f^{(i)}}{g\cdot n_{rg}-\sum_{i=1}^{g-1}i\cdot n_f^{(i)}} \\
&\quad\cdot\mathbb{E}\left(NFTI^{\text{rp}}|n_f^{(1)},...,n_f^{(i-1)},n_f^{(i)}-1,\right. \\
&\qquad\left. n_f^{(i+1)}+1,n_f^{(i+2)},...,n_f^{(g-1)}\right)
\end{aligned} \tag{1}$$

Proof. Let $\mathbb{E}\left(NFTI^{\text{rp}}|n_f^{(1)},...,n_f^{(g-1)}\right)$ be the expectation of the number of failures needed for the whole application to fail, knowing that the application is still running and that, for $i \in [1..g-1]$, there are $n_f^{(i)}$ replica-groups that have already been hit by exactly i failures. Note that a replica-group hit by i failures still contains exactly $g-i$ running replicas. Therefore, in a system where $n_f^{(i)}$ replica-groups have been hit by exactly i failures, there are still overall exactly $g\cdot n_{rg}-\sum_{i=1}^{g-1}i\cdot n_f^{(i)}$ running replicas, $g\cdot\left(n_{rg}-\sum_{i=1}^{g-1}n_f^{(i)}\right)$ of which are in replica-groups that have not yet been hit by any failure. Now, consider the next failure to hit the system. There are three cases to consider.

1. The failure hits a replica-group that has not been hit by any failure so far. This happens with probability:

$$\frac{g\cdot\left(n_{rg}-\sum_{i=1}^{g-1}n_f^{(i)}\right)}{g\cdot n_{rg}-\sum_{i=1}^{g-1}i\cdot n_f^{(i)}}$$

and, in that case, the expected number of failures needed for the whole application to fail is one (the studied failure) plus $\mathbb{E}\left(NFTI^{\text{rp}}|1+n_f^{(1)},n_f^{(2)},...,n_f^{(g-1)}\right)$. Remark that we should have conditioned the above expectation with the statement "if $n_{rg} > \sum_{i=1}^{g-1} n_f^{(i)}$". In order to keep Equation (1) as simple as possible we rather do not explicitly state the condition and use the following abusive notation:

$$\frac{g \cdot \left(n_{rg} - \sum_{i=1}^{g-1} n_f^{(i)}\right)}{g \cdot n_{rg} - \sum_{i=1}^{g-1} i \cdot n_f^{(i)}} \cdot \left(1 + \mathbb{E}\left(NFTI^{\text{rp}}|1+n_f^{(1)},n_f^{(2)},...,n_f^{(g-1)}\right)\right)$$

considering than when $n_{rg} = \sum_{i=1}^{g-1} n_f^{(i)}$ the first term is null and thus that it does not matter that the second term is not defined.

2. The failure hits a replica-group that has already been hit by $g-1$ failures. Such a failure leads to a failure of the whole application. As there are $n_f^{(g-1)}$ such groups, each containing exactly one running replica, this event happens with probability:

$$\frac{n_f^{(g-1)}}{g \cdot n_{rg} - \sum_{i=1}^{g-1} i \cdot n_f^{(i)}}.$$

In this case, the expected number of failures needed for the whole application to fail is exactly equal to one (the considered failure).

3. The failure hits a replica-group that had already been hit by at least one failure, and by at most $g-2$ failures. Let i be any value in $[1..g-2]$. The probability that the failure hits a group that had previously been the victim of exactly i failures is equal to:

$$\frac{(g-i) \cdot n_f^{(i)}}{g \cdot n_{rg} - \sum_{i=1}^{g-1} i \cdot n_f^{(i)}}$$

as there are $n_f^{(i)}$ such replica-groups and that each contains exactly $g-i$ still running replicas. In this case, the expected number of failures needed for the whole application to fail is one (the studied failure) plus $\mathbb{E}\left(NFTI^{\text{rp}}|n_f^{(1)},...,n_f^{(i-1)},n_f^{(i)}-1,n_f^{(i+1)}+1,\ n_f^{(i+2)},...,n_f^{(g-1)}\right)$ as there is one less replica-group hit by exactly i failures and one more hit by exactly $i+1$ failures.

We aggregate all the cases to obtain:

$$\begin{aligned}\mathbb{E}\left(NFTI^{\mathrm{rp}}|n_f^{(1)},...,n_f^{(g-1)}\right) = \\ &\frac{g\cdot\left(n_{rg}-\sum_{i=1}^{g-1}n_f^{(i)}\right)}{g\cdot n_{rg}-\sum_{i=1}^{g-1}i\cdot n_f^{(i)}} \\ &\quad\cdot\left(1+\mathbb{E}\left(NFTI^{\mathrm{rp}}|1+n_f^{(1)},n_f^{(2)},...,n_f^{(g-1)}\right)\right) \\ &+\sum_{i=1}^{g-2}\frac{(g-i)\cdot n_f^{(i)}}{g\cdot n_{rg}-\sum_{i=1}^{g-1}i\cdot n_f^{(i)}} \\ &\quad\cdot\left(1+\mathbb{E}\left(NFTI^{\mathrm{rp}}|n_f^{(1)},...,n_f^{(i-1)},n_f^{(i)}-1,\right.\right. \\ &\qquad\left.\left.n_f^{(i+1)}+1,n_f^{(i+2)},...,n_f^{(g-1)}\right)\right) \\ &+\frac{n_f^{(g-1)}}{g\cdot n_{rg}-\sum_{i=1}^{g-1}i\cdot n_f^{(i)}}\cdot 1\end{aligned}$$

which can be rewritten as Equation (1). □

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 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 $g = 3$, $MNFTI^{\mathrm{ah}} = \mathbb{E}(NFTI^{\mathrm{ah}}|0,0)$ where*

$$\begin{aligned}\mathbb{E}\left(NFTI^{\mathrm{ah}}|n_2,n_1\right) = \\ &\frac{1}{3n_{rg}-n_2-2n_1}\left(3n_{rg}+3(n_{rg}-n_1-n_2)\mathbb{E}\left(NFTI^{\mathrm{ah}}|n_2+1,n_1\right)\right. \\ &\qquad\left.+2n_2\mathbb{E}\left(NFTI^{\mathrm{ah}}|n_2-1,n_1+1\right)\right)\end{aligned}$$

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

Proof. Let $\mathbb{E}(NFTI^{\mathrm{ah}}|n_2,n_1)$ be the expectation of the number of failures needed for the whole application to fail, knowing that the application is still running, that n_1 processes are no longer replicated (i.e., in each of thesereplica-groups exactly two of the three processors have already been hit by a failure), and that n_2 processes have only two replicas left (i.e., in each of these replica-groups exactly one processor has been hit by a failure). Overall, there are $n_1 + 2 \times n_2 + 3 \times (n_{rg} - n_1 - n_2)$ processors still running.

The case $n_2 = 0$ and $n_1 = n_{rg}$ is the simplest. A new failure will hit a replica-group where only one of the three 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:
   $\mathbb{E}(NFTI^{\mathrm{ah}}|0,n_{rg}) = 1$.
2. The failure hits one of the two processors that have already been hit. Then the failure has no impact on the application. The $MNFTI^{\mathrm{ah}}$ of this case is then: $\mathbb{E}(NFTI^{\mathrm{ah}}|0,n_{rg}) = 1 + \mathbb{E}\left(NFTI^{\mathrm{ah}}|0,n_{rg}\right)$.

The probability of failure is uniformly distributed between the three processors. Thus the second case is twice more frequent than the first one. Weighting the values by their probabilities of occurrence yields:

$$\mathbb{E}\left(NFTI^{\text{ah}}\,|0, n_{rg}\right) = \frac{1}{3} \times 1 + \frac{2}{3} \times \left(1 + \mathbb{E}\left(NFTI^{\text{ah}}\,|0, n_{rg}\right)\right) = 3.$$

For the general case, $0 \leq n_1 \leq n_{rg} - 1$, either the next failure hits a new replica-group, that is one with 3 replicas still running, or it hits a replica-group with only 2 replicas still running, or it hits a replica-group that has only one running replica left. The latter case leads to the same sub-cases as the $n_1 = n_{rg}$ 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 $3n_{rg}$ processors, including the ones already hit. Hence the probability that the next failure hits a new replica-group is $\frac{3n_{rg}-3n_2-3n_1}{3n_{rg}} = \frac{n_{rg}-n_2-n_1}{n_{rg}}$. In this case, the expected number of failures needed for the whole application to fail is one (the considered failure) plus $\mathbb{E}\left(NFTI^{\text{ah}}|n_2 + 1, n_1\right)$.

Hence the probability that the next failure hits a replica-group with exactly two running replicas is $\frac{3n_2}{3n_{rg}} = \frac{n_2}{n_{rg}}$. In this case, we have two subcases. With two chances out of three, the failure hits a running replica. Then the expected number of failures needed for the whole application to fail is one (the considered failure) plus $\mathbb{E}\left(NFTI^{\text{ah}}|n_2 - 1, n_1 + 1\right)$. With one chance out of three, the failure hits the processor that has already been hit. Then the expected number of failures needed for the whole application to fail is one (the considered failure) plus $\mathbb{E}\left(NFTI^{\text{ah}}|n_2, n_1\right)$.

Altogether we have:

$$\begin{aligned}\mathbb{E}\left(NFTI^{\text{ah}}|n_2, n_1\right) &= \frac{n_{rg} - n_2 - n_1}{n_{rg}} \times \left(1 + \mathbb{E}\left(NFTI^{\text{ah}}|n_2 + 1, n_1\right)\right) \\ + \frac{n_2}{n_{rg}} \times &\left(\frac{2}{3} \times \left(1 + \mathbb{E}\left(NFTI^{\text{ah}}|n_2 - 1, n_1 + 1\right)\right) + \frac{1}{3}\left(1 + \mathbb{E}\left(NFTI^{\text{ah}}|n_2, n_1\right)\right)\right). \\ &+ \frac{n_1}{n_{rg}}\left(\frac{1}{3} \times 1 + \frac{2}{3}\left(1 + \mathbb{E}\left(NFTI^{\text{ah}}|n_2, n_1\right)\right)\right)\end{aligned}$$

Therefore,

$$\begin{aligned}&\mathbb{E}\left(NFTI^{\text{ah}}|n_2, n_1\right) = \\ &\quad \frac{1}{3n_{rg} - n_2 - 2n_1}\left(3n_{rg} + 3(n_{rg} - n_1 - n_2)\mathbb{E}\left(NFTI^{\text{ah}}|n_2 + 1, n_1\right)\right. \\ &\qquad \left. + 2n_2\mathbb{E}\left(NFTI^{\text{ah}}|n_2 - 1, n_1 + 1\right)\right)\end{aligned}$$

□

Proposition 4. *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 $g = 3$, $MNFTI^{\mathrm{rp}} = \mathbb{E}(NFTI^{\mathrm{rp}}|0,0)$ *where*

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

Proof. Let $\mathbb{E}(NFTI^{\mathrm{rp}}|n_2,n_1)$ be the expectation of the number of failures needed for the whole application to fail, knowing that the application is still running, that n_1 processes are no longer replicated (i.e., in each of thesereplica-groups exactly two of the three processors have already been hit by a failure), and that n_2 processes have only two replicas left (i.e., in each of these replica-groups exactly one processor has been hit by a failure). Overall, there are $n_1 + 2 \times n_2 + 3 \times (n_{rg} - n_1 - n_2)$ processors still running.

The case $n_2 = 0$ and $n_1 = n_{rg}$ is the simplest: a new failure will hit an already hit replica-group and hence leads to an application failure, hence

$$\mathbb{E}\left(NFTI^{\mathrm{rp}}\,|n_{rg}\right) = 1.$$

For the general case, $0 \leq n_1 \leq n_{rg} - 1$, either the next failure hits a new replica-group, that is one with 3 replicas still running, or it hits a replica-group with only 2 replicas still running, or it hits a replica-group that has only one running replica left. The latter case leads to the same case as the $n_1 = n_{rg}$ 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 $3n_{rg} - 2n_1 - n_2$ running processors. Hence the probability that the next failure hits a new replica-group is $\frac{3n_{rg}-3n_2-3n_1}{3n_{rg}-2n_1-n_2}$. In this case, the expected number of failures needed for the whole application to fail is one (the considered failure) plus $\mathbb{E}\left(NFTI^{\mathrm{rp}}|n_2+1,n_1\right)$.

Hence the probability that the next failure hits a replica-group with exactly two running replicas is $\frac{2n_2}{3n_{rg}-2n_1-n_2}$. In this case, the expected number of failures needed for the whole application to fail is one (the considered failure) plus $\mathbb{E}\left(NFTI^{\mathrm{rp}}|n_2-1,n_1+1\right)$.

Altogether we have:

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

Therefore,

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

□

Given the simple additive relationship that exists between $MNFTI^{\text{ah}}$ and $MNFTI^{\text{rp}}$ for $g = 2$ (Proposition 1), one may expect a similar relationship for large g. Table 1 shows $MNFTI^{\text{ah}}$ and $MNFTI^{\text{rp}}$ values and the difference between them for $g = 3$. The difference is not constant and increases as n_{rg} increases, and no simple relationship seems to exist between $MNFTI^{\text{ah}}$ and $MNFTI^{\text{rp}}$.

Table 1: $MNFTI^{\text{ah}}$ and $MNFTI^{\text{rp}}$ computed using Proposition 3 and Proposition 4 and the difference between them, for $n_{rg} = 2^0, \ldots, 2^{20}$, with $g = 3$.

n_{rg}	2^0	2^1	2^2	2^3	2^4	2^5	2^6
$MNFTI^{\text{ah}}$	5.5	7.3	10.1	14.6	21.6	32.4	49.4
$MNFTI^{\text{rp}}$	3.0	4.5	6.9	10.9	17.1	27.1	42.9
($MNFTI^{\text{ah}}$-$MNFTI^{\text{rp}}$)	2.5	2.8	3.2	3.7	4.4	5.3	6.4
n_{rg}	2^7	2^8	2^9	2^{10}	2^{11}	2^{12}	2^{13}
$MNFTI^{\text{ah}}$	75.9	117.6	183.3	286.8	450.2	708.5	1117.0
$MNFTI^{\text{rp}}$	68.1	108.0	171.5	272.2	432.1	685.8	1088.7
($MNFTI^{\text{ah}}$-$MNFTI^{\text{rp}}$)	7.8	9.6	11.8	14.6	18.2	22.7	28.3
n_{rg}	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}
$MNFTI^{\text{ah}}$	1763.5	2787.6	4410.2	6982.3	11060.6	17528.6	27788.6
$MNFTI^{\text{rp}}$	1728.1	2743.2	4354.6	6912.5	10972.9	17418.4	27650.1
($MNFTI^{\text{ah}}$-$MNFTI^{\text{rp}}$)	35.4	44.3	55.6	69.8	87.7	110.2	138.6

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 [7]. 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 [29]. 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 [7], 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, [7] uses the solution of the generalized birthday problem to compute *MNFTI*. According to [30], a previous article by the authors of [7], it would seem that the value $BP(n_{rg})$ is used. While [7] 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 2: $MNFTI^{\text{ah}}$ 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 2 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 [7], whose validity was not assessed experimentally therein.

4.2 Computing *MTTI* for Exponential failures

With the "already hit" assumption, and assuming Exponential failures, the *MTTI* can be computed easily as

$$MTTI = systemMTBF(g \times n_{rg}) \times MNFTI^{\text{ah}} \tag{2}$$

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.

A recursive expression for $MTTI$ can also be obtained directly.

While the $MTTI$ value should not depend on the way to count failures, it would be interesting for compute it with the "running processor" assumption as a sanity check. It turns out that there is no equivalent to Equation (2) for linking $MTTI$ and $MNFTI^{\text{rp}}$. The reason is straightforward. While $systemMTBF(2n_{rg})$ is the expectation of the date at which the first failure will happen, it is not the expectation of the inter-arrival time of the first and second failures when only considering failures on processors still running. Indeed, after the first failure, there only remain $2n_{rg} - 1$ running processors. Therefore, the inter-arrival time of the first and second failures has an expectation of $systemMTBF(2n_{rg} - 1)$. We can, however, use a reasoning similar to that in the proof of Theorem 2 and obtain a recursive expression for $MTTI$:

Theorem 3. *If the failure inter-arrival times on the different processors follow an Exponential distribution of parameter λ then, when using process replication with $g = 2$, $MTTI = \mathbb{E}(TTI|0)$ where $\mathbb{E}(TTI|n_f) =$*

$$\begin{cases} \frac{1}{n_{rg}}\frac{1}{\lambda} & \text{if } n_f = n_{rg} \\ \frac{1}{(2n_{rg}-n_f)}\frac{1}{\lambda} + \frac{2n_{rg}-2n_f}{2n_{rg}-n_f}\mathbb{E}(TTI|n_f+1) & \text{otherwise} \end{cases}$$

Proof. We denote by $\mathbb{E}(TTI|n_f)$ the expectation of the time an application will run before failing, knowing that the application is still running and that failures have already hit n_f different replica-groups. Since 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 thus still $n_f + 2(n_{rg} - n_f) = 2n_{rg} - n_f$ running processors.

The case $n_f = n_{rg}$ is the simplest: a new failure will hit an already hit replica-group and hence leads to an application failure. As there are exactly n_{rg} remaining running processors, the inter-arrival times of the n_{rg}-th and $(n_{rg}+1)$-th failures is equal to $\frac{1}{\lambda n_{rg}}$ (minimum of n_{rg} Exponential laws). Hence:

$$\mathbb{E}\left(TTI\,|n_{rg}\right) = \frac{1}{\lambda n_{rg}}.$$

For the general case, $0 \leq n_f \leq n_{rg} - 1$, either the next failure hits a replica-group with still 2 running processors, or it strikes a replica-group that had already been victim of a failure. The latter case leads to an application failure; then, after n_f failures, the expected application running time before failure is equal to the inter-arrival times of the n_f-th and $(n_f + 1)$-th failures, which is equal to $\frac{1}{(2n_{rg}-n_f)\lambda}$. The failure probability is uniformly distributed among the $2n_{rg} - n_f$ running processors, hence the probability that the next failure strikes a new replica-group is $\frac{2n_{rg}-2n_f}{2n_{rg}-n_f}$. In this case, the expected application running time before failure is equal to the inter-arrival times of the n_f-th and (n_f+1)-th failures plus $\mathbb{E}\left(TTI|n_f + 1\right)$. We derive that:

$$\begin{aligned}\mathbb{E}\left(TTI|n_f\right) = & \\ & \frac{2n_{rg} - 2n_f}{2n_{rg} - n_f} \times \left(\frac{1}{(2n_{rg} - n_f)\lambda} + \mathbb{E}\left(TTI|n_f + 1\right)\right) \\ & \qquad + \frac{n_f}{2n_{rg} - n_f} \times \frac{1}{(2n_{rg} - n_f)\lambda}.\end{aligned}$$

Therefore,

$$\mathbb{E}\left(TTI|n_f\right) = \frac{1}{(2n_{rg}-n_f)\lambda} + \frac{2n_{rg}-2n_f}{2n_{rg}-n_f}\mathbb{E}\left(TTI|n_f+1\right).$$

□

The above results can be generalized to $g \geq 2$. To compute $MTTI$ under the "already hit" assumption one can use Equation (2) replacing $NF(n_{rg})$ by the $MNFTI^{\text{ah}}$ value given by Theorem 1. To compute $MNFTI^{\text{rp}}$ under the "running processors," Theorem 3 can be generalized using the same proof technique as when proving Proposition 2.

The linear relationship between $MNFTI$ and $MTTI$, seen in Equation (2), allows us to use the results in Table 2 to compute $MTTI$ values. To quantify the potential benefit of replication, Table 3 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 3, 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 3.

The main and expected observation in Table 3 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 4 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 4. *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*

Table 3: *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

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

$$R(t) = \prod_{j=1}^{n_{rg}} \left(1 - \prod_{i=1}^{g} F\left(t|\tau_{i+g(j-1)}\right) \right), \tag{3}$$

and the MTTI is given by:

$$MTTI = \int_0^{+\infty} \prod_{j=1}^{n_{rg}} \left(1 - \prod_{i=1}^{g} F\left(t|\tau_{i+g(j-1)}\right) \right) \mathrm{d}t. \tag{4}$$

While failure independence is necessary to prove Theorem 4, the assumption that failures are i.i.d. can be removed. Nevertheless, we include this assumption here so as to simplify the writing of Equations 3 and 4 above.

Proof. The probability that processor P_i suffers from a failure during the next t units of time, knowing that the time elapsed since its last failure is τ_i, is equal by definition to $F_i(t) = F(t|\tau_i)$. Then the probability that the g processors running the replicas of process j, $1 \leq j \leq n_{rg}$, all suffer from a failure during the next t units of time is then equal to:

$$F_j^{(g)}(t) = \prod_{i=1}^{g} F_{i+g(j-1)}(t) = \prod_{i=1}^{g} F\left(t|\tau_{i+g(j-1)}\right).$$

Therefore, the probability that at least one of the g duplicates of process j is still running after t units of time is equal to:

$$R_j^{(g)}(t) = 1 - F_j^{(g)}(t) = 1 - \prod_{i=1}^{g} F\left(t|\tau_{i+g(j-1)}\right).$$

For the whole application to still be running after t units of time, each of the n_{rg} application processes must still be running (i.e., each must have at least one

of its g initial replicas still running). So, the probability that the application is still running after t units of time is:

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

We can then compute the Mean Time To Interruption of the whole application:

$$\begin{aligned} MTTI &= \int_0^{+\infty} R(t)\mathrm{d}t \\ &= \int_0^{+\infty} \prod_{j=1}^{n_{rg}} \left(1 - \prod_{i=1}^{g} F\left(t|\tau_{i+g(j-1)}\right)\right) \mathrm{d}t. \end{aligned}$$

□

We now consider the case of the Exponential law.

This theorem can then be used to obtain a closed-form expression for $MTTI$ when the failure distribution is Exponential (Theorem 5) or Weibull (Theorem 6):

Theorem 5. *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:*

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

Proof. According to Theorem 4, the probability that the application is still running after t units of time is:

$$R(t) = \left(1 - \left(1 - e^{-\lambda t}\right)^g\right)^{n_{rg}}$$

and the Mean Time To Interruption (MTTI) of the whole application is:

$$
\begin{aligned}
&\int_0^{+\infty} R(t)\mathrm{d}t \\
&= \int_0^{+\infty} \left(1-\left(1-e^{-\lambda t}\right)^g\right)^{n_{rg}} \mathrm{d}t \\
&= \int_0^{+\infty} \sum_{i=0}^{n_{rg}} \binom{n_{rg}}{i} (-1)^i \left(1-e^{-\lambda t}\right)^{i\cdot g} \mathrm{d}t \\
&= \int_0^{+\infty} \sum_{i=0}^{n_{rg}} \binom{n_{rg}}{i} (-1)^i \left(\sum_{j=0}^{i\cdot g} \binom{i\cdot g}{j} (-1)^j e^{-\lambda j t}\right) \mathrm{d}t \\
&= \int_0^{+\infty} \sum_{i=0}^{n_{rg}} \binom{n_{rg}}{i} (-1)^i \left(1+\sum_{j=1}^{i\cdot g} \binom{i\cdot g}{j} (-1)^j e^{-\lambda j t}\right) \mathrm{d}t \\
&= \int_0^{+\infty} \left[\sum_{i=0}^{n_{rg}} \binom{n_{rg}}{i} (-1)^i \right. \\
&\qquad \left. + \sum_{i=0}^{n_{rg}} \left(\binom{n_{rg}}{i} (-1)^i \sum_{j=1}^{i\cdot g} \binom{i\cdot g}{j} (-1)^j e^{-\lambda j t}\right)\right] \mathrm{d}t \\
&= \int_0^{+\infty} \sum_{i=0}^{n_{rg}} \left[\binom{n_{rg}}{i} (-1)^i \sum_{j=1}^{i\cdot g} \binom{i\cdot g}{j} (-1)^j e^{-\lambda j t}\right] \mathrm{d}t \\
&= \sum_{i=0}^{n_{rg}} \left[\binom{n_{rg}}{i} (-1)^i \sum_{j=1}^{i\cdot g} \binom{i\cdot g}{j} (-1)^j \int_0^{+\infty} e^{-\lambda j t}\mathrm{d}t\right] \\
&= \sum_{i=0}^{n_{rg}} \left[\binom{n_{rg}}{i} (-1)^i \sum_{j=1}^{i\cdot g} \frac{\binom{i\cdot g}{j}(-1)^j}{\lambda j}\right] \\
&= \sum_{i=1}^{n_{rg}} \left[\binom{n_{rg}}{i} (-1)^i \sum_{j=1}^{i\cdot g} \frac{\binom{i\cdot g}{j}(-1)^j}{\lambda j}\right]
\end{aligned}
$$

Thus,

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

□

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:

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

Proof. The first expression is a simple corollary of Theorem 5 for the case $g = 2$. The second expression is obtained through direct computation. Let $f(t)$ be the probability density function associated to the cumulative distribution function $F(t)$. Then, we have:

$$\begin{aligned} & MTTI \\ &= \int_0^{+\infty} t \cdot f(t) \mathrm{d}t \\ \\ &= \int_0^{+\infty} t \; 2^{n_{rg}} n_{rg} \lambda \left(1 - e^{-\lambda t}\right) e^{-\lambda n_{rg} t} \left(1 - \frac{e^{-\lambda t}}{2}\right)^{n_{rg}-1} \mathrm{d}t \\ \\ &= 2^{n_{rg}} n_{rg} \lambda \int_0^{+\infty} t \left(1 - e^{-\lambda t}\right) e^{-\lambda n_{rg} t} \\ &\qquad \sum_{i=0}^{n_{rg}-1} \binom{n_{rg}-1}{i} \left(\frac{-1}{2}\right)^i e^{-\lambda i t} \mathrm{d}t \\ \\ &= 2^{n_{rg}} n_{rg} \lambda \sum_{i=0}^{n_{rg}-1} \binom{n_{rg}-1}{i} \left(\frac{-1}{2}\right)^i \\ &\qquad \int_0^{+\infty} t \left(1 - e^{-\lambda t}\right) e^{-\lambda (n_{rg}+i) t} \mathrm{d}t \\ \\ &= 2^{n_{rg}} n_{rg} \lambda \sum_{i=0}^{n_{rg}-1} \binom{n_{rg}-1}{i} \left(\frac{-1}{2}\right)^i \\ &\qquad \int_0^{+\infty} \left(t e^{-\lambda (n_{rg}+i) t} - t e^{-\lambda (n_{rg}+i+1) t} \right) \mathrm{d}t. \end{aligned}$$

As $\int_0^{+\infty} t e^{-\lambda t} = \frac{1}{\lambda^2}$, the expression of $MTTI$ can be further refined as

follows:

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

Using the equation $\binom{n_{rg}-1}{i} + 2\binom{n_{rg}-1}{i-1} = \binom{n_{rg}}{i}\frac{(n_{rg}+i)}{n_{rg}}$, we derive the desired expression for *MTTI*:

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

□

We now consider the case of the Weibull law.

Theorem 6. *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:*

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

Proof. According to Theorem 4, the probability that the application is still running after t units of time is:

$$R(t) = \left(1 - \left(1 - e^{-\left(\frac{t}{\lambda}\right)^k}\right)^g\right)^{n_{rg}}$$

and the Mean Time To Interruption of the whole application is:

$$
\begin{aligned}
&MTTI \\
&= \int_0^{+\infty} R(t)\mathrm{d}t \\
\\
&= \int_0^{+\infty} \sum_{i=0}^{n_{rg}} \binom{n_{rg}}{i} (-1)^i \left(1 - e^{-\left(\frac{t}{\lambda}\right)^k}\right)^{i \cdot g} \mathrm{d}t \\
\\
&= \int_0^{+\infty} \sum_{i=0}^{n_{rg}} \binom{n_{rg}}{i} (-1)^i \\
&\qquad \left(\sum_{j=0}^{i \cdot g} \binom{i \cdot g}{j} (-1)^j \, e^{-j\left(\frac{t}{\lambda}\right)^k} \right) \mathrm{d}t \\
\\
&= \int_0^{+\infty} \sum_{i=0}^{n_{rg}} \binom{n_{rg}}{i} (-1)^i \\
&\qquad \left(1 + \sum_{j=1}^{i \cdot g} \binom{i \cdot g}{j} (-1)^j \, e^{-j\left(\frac{t}{\lambda}\right)^k} \right) \mathrm{d}t \\
\\
&= \int_0^{+\infty} \left[\sum_{i=0}^{n_{rg}} \binom{n_{rg}}{i} (-1)^i \right. \\
&\left. + \sum_{i=0}^{n_{rg}} \binom{n_{rg}}{i} (-1)^i \left(\sum_{j=1}^{i \cdot g} \binom{i \cdot g}{j} (-1)^j \, e^{-j\left(\frac{t}{\lambda}\right)^k} \right) \right] \mathrm{d}t \\
\\
&= \int_0^{+\infty} \sum_{i=0}^{n_{rg}} \binom{n_{rg}}{i} (-1)^i \\
&\qquad \left[\sum_{j=1}^{i \cdot g} \binom{i \cdot g}{j} (-1)^j \, e^{-j\left(\frac{t}{\lambda}\right)^k} \right] \mathrm{d}t \\
\\
&= \sum_{i=1}^{n_{rg}} \binom{n_{rg}}{i} (-1)^i \\
&\qquad \left[\sum_{j=1}^{i \cdot g} \binom{i \cdot g}{j} (-1)^j \int_0^{+\infty} e^{-j\left(\frac{t}{\lambda}\right)^k} \mathrm{d}t \right]
\end{aligned}
$$

We consider any value $j \in [0..n_{rg} \cdot g]$ and we make the following change of variable: $u = \frac{j}{\lambda^k} t^k$. This is equivalent to $t = \lambda \left(\frac{u}{j}\right)^{\frac{1}{k}}$ and thus $\mathrm{d}t =$

$\frac{\lambda}{k}\left(\frac{1}{j}\right)^{\frac{1}{k}} u^{\left(\frac{1}{k}-1\right)}\mathrm{d}u$. With this notation,

$$\int_0^{+\infty} e^{-j\left(\frac{t}{\lambda}\right)^k}\mathrm{d}t \quad = \frac{\lambda}{kj^{\frac{1}{k}}}\Gamma\left(\frac{1}{k}\right).$$

Therefore, $MTTI$ is equal to:

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

Thus,

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

□

While Theorem 5 is yet another approach to computing the $MTTI$ for Exponential distributions, Theorem 6 is the first analytical result (to the best of our knowledge) for Weibull distributions. Unfortunately, the formula in Theorem 6 is not numerically stable for large values of n_{rg}. As a result, we resort to simulation to compute $MTTI$ values. Table 4, which is the counterpart of Table 3 for Weibull failures, show $MTTI$ results obtained as averages computed on the first $100,000$ application failures of each simulated scenario. The results are similar to those in Table 3. The $MTTI$ with 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 4: 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 $MTTI$ quantities, which are of direct relevance to the performance of the

application and are amenable to analytical derivations. 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. But 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. One of our goals here is to verify that the performance advantage of process replication seen in Sections 4.2 and 4.3 in terms of $MTTI$ are also seen when considering the 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://perso.ens-lyon.fr/frederic.vivien/Data/Resilience/JPDCReplication`.

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 0.7 or 0.5 based on the results in [15] and [16].

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* [31], i.e., for clusters at the Los Alamos National Laboratory [15]. 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 [15]). 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.

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 [9] or Daly's first-order approximation [8], the latter being used in [7]. We use Daly's approximation in this work because it is classical, often used in practice, and used in previous work [7]. 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 [32]. 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 [13] (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 (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).

Replication overhead – In [7], 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 [7] 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**: $W(p) = \frac{W}{p} \times (1 + \frac{1}{100} \times (\frac{log(p)}{10} + 3.67))$.
- **Generic parallel jobs**: $W(p) = (\frac{W}{p} + \gamma W) \times (1 + \frac{1}{100} \times (\frac{log(p)}{10} + 3.67))$.

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**: $W(p) = \frac{W}{p} + \frac{\gamma \times W^{\frac{2}{3}}}{\sqrt{p}} \times 4$.

When $g = 3$, we (somewhat arbitrarily) multiply by 9/4 the overhead for perfectly parallel and generic parallel jobs, because the number and volume of communications are multiplied by 4 when $g = 2$ and by 9 when $g = 3$. When $g = 3$, we multiply the communication component by a factor 9 for numerical kernels.

Parameter values – We use the same parameter values as in the recent article [13]: $C = R = 600$ s, $D = 60$ s and $\mathcal{W} = 10,000$ years (except for log-based simulations for which $\mathcal{W} = 1,000$ years).

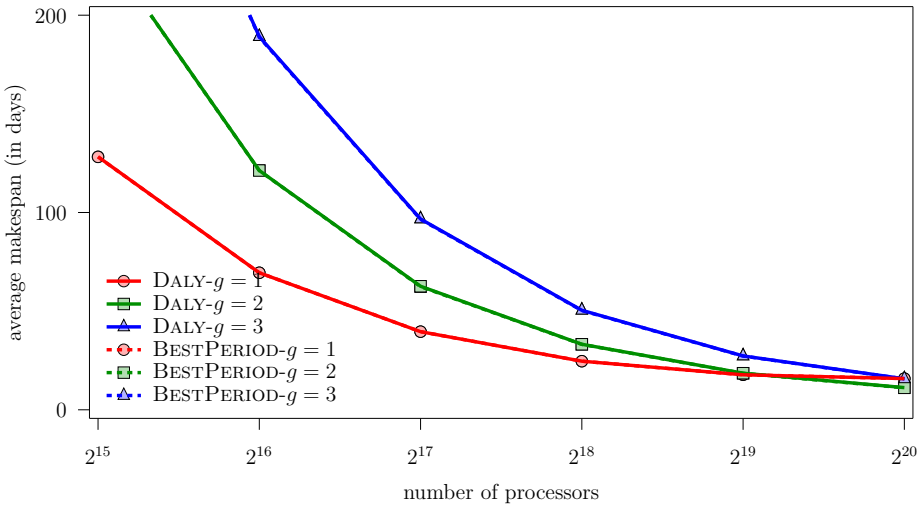
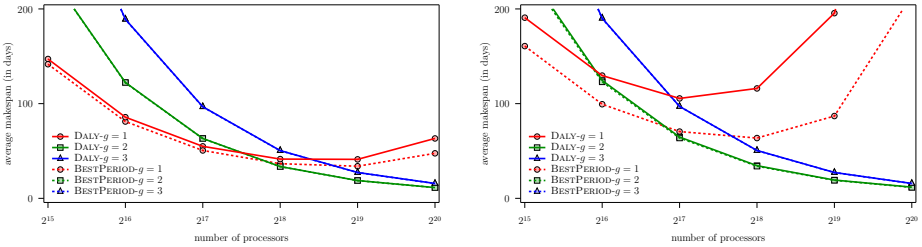


Figure 1: Average makespan vs. number of processors for two choices of the checkpointing period, without process replication (DALY-g=1 and BESTPERIOD-g=1) and with process replication (DALY-g=2 or 3 and BESTPERIOD-g=2 or 3), for generic parallel jobs subject to Exponential failures ($MTBF = 125$ years).

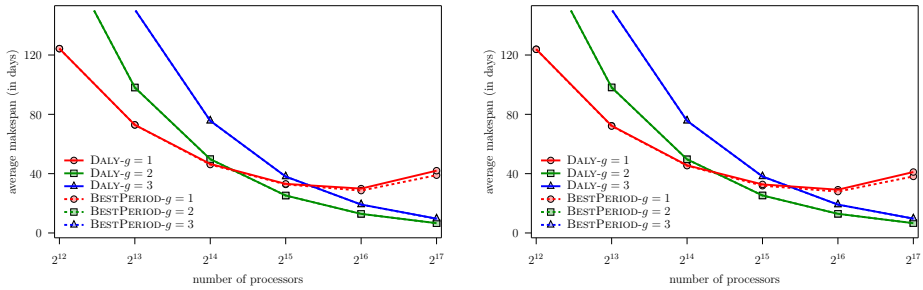


(a) $k = 0.70$ (b) $k = 0.50$

Figure 2: Same as Figure 1 (generic parallel jobs) but for Weibull failures ($MTBF = 125$ years).

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 [7], is a reasonable idea when replication is used. In the $g = 2$ case (two replicas per application process), we compute this period using the exact $MTTI$ expression from Corollary 1. 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. We present results using the best period found via a numerical search in a similar manner. 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.,



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

Figure 3: Same as Figure 1 (generic parallel jobs) but for real-world failures.

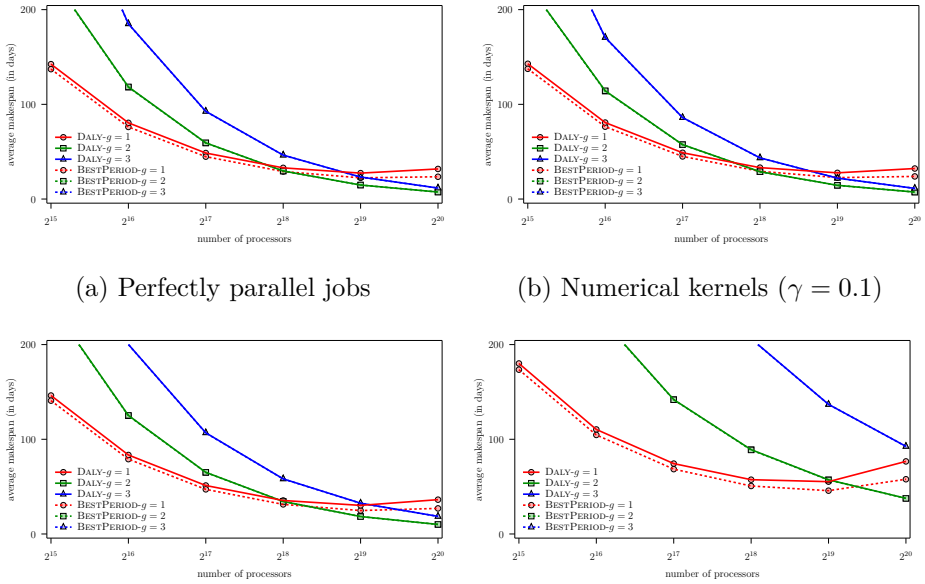
$g \times n/g$.

We ran experiments for five 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) Failures drawn from the failure log of LANL cluster 18; and (v) Failures drawn from the failure 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 $5 \times 5 = 25$ sets of results.

Figures 1 through 4 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 4 for Weibull failures with $k = 0.7$. Due to lack of space, we only show results for the generic parallel job model in Figure 5 and Table 5 below.

Figures 1, 2, and 3 show average makespan vs. number of processors for generic parallel jobs subject to each of the five considered failure distributions. We first remark that, except when failures follow an Exponential distribution, the minimum makespan is not achieved on the largest platform. The fact that in most cases the makespan with 2^{19} processors is lower than the makespan with 2^{20} processors suggests that duplicating processes 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 none of the configurations, using a replication with f $g = 3$ is more beneficial than with$g = 2$. More importantly, in each configuration, the minimum makespan is always achieved while duplicating the processes($g = 2$) and using the maximum number of processors.

The two curves for $g = 1$ are exactly superposed in Figure 1. For $g = 2$ and for $g = 3$ the two curves are exactly superposed in all three figures. 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 approxi-



(a) Perfectly parallel jobs

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

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

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

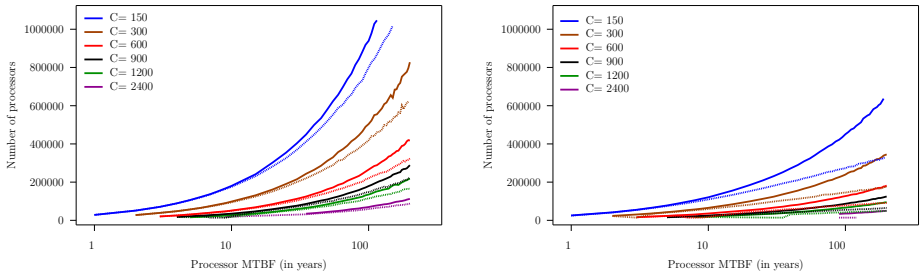
Figure 4: Same as Figure 2(a) (Weibull failures with $k = 0.7$, $MTBF = 125$ years) but for other job types.

mation leads to significantly suboptimal results that worsen as k decreases (as expected and already reported in [13]). What is perhaps less expected is that 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 replication, 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 5 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 $k = 0.5$ and $k = 0.7$, $C = 600s$, and for various numbers of processors. We see that very few application failures, and thus recoveries, occur throughout application execution (recall that makespans are measured in days in our experiments). This is because a very small fraction of processor failures manifest themselves as application failures (below 0.4% in our experiments). This also explains why using $g = 3$ replicas does not lead to any further performance improvements (recall that the expectation was that further improvement would be low anyway given the results in Tables 3 and 4) . While this low number of application failures demonstrates the benefit of process replication, the interesting result is that it also makes the choice of the checkpointing period not critical.

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. Consider for instance a process replication scenario

Table 5: Number of application failures and fraction of processor failures that cause application failures with process replication ($g = 2$) assuming Weibull failure distributions ($k = 0.7$ or 0.5) for various numbers of processors and C=600s. Results are averaged over 100 experiments.

	# of app. failures		% of proc. failures	
# of proc.	$k = 0.7$	$k = 0.5$	$k = 0.7$	$k = 0.5$
2^{14}	1.95	4.94	0.35	0.39
2^{15}	1.44	3.77	0.25	0.28
2^{16}	0.88	2.61	0.15	0.19
2^{17}	0.45	1.67	0.075	0.12
2^{18}	0.20	1.11	0.034	0.076
2^{19}	0.13	0.72	0.022	0.049
2^{20}	0.083	0.33	0.014	0.023



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

Figure 5: Break-even point curves for replication ($g = 2$) vs. no-replication for various checkpointing overheads, as computed using the best checkpointing periods (solid lines) and Daly's approximation (dashed lines), assuming Weibull failure distributions.

with Weibull failures of shape parameters $k = 0.7$, 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. When using replication, a poor choice can also theoretically lead to poor results, but this is very unlikely in practice because replication drastically reduces the number of failures. In fact, in practical settings, the choice of the checkpointing period is simply not critical when replication is used. Consequently, 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 dupli-

cation ($g = 2$) as we have seen that the case $g = 3$ was never beneficial with respect to the case $g = 2$.

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., the resource waste due to replication is not worthwhile because the processor MTBF is too large or the number of processors is too low. Several such curves are shown in [7] (Figure 9 therein) for different checkpointing overheads, and, as expected, the higher the overhead the more beneficial it is to use process replication.

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 the choice of the period has little impact and that Daly's approximation can be used safely. 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 (see the $g = 1$ curves in Figure 2). 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 3), there is evidence that, in general, failure distributions are well approximated by Weibull distributions [14, 15, 16, 17], while not at all by exponential distributions. Most recently, in [17], 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 $k < 0.5$. In other words, the failure distribution is far from being Exponential and thus Daly's approximation would be far from the best period (compare Figure 2(a) for $k = 0.7$ to Figure 2(b) for $k = 0.5$).

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. 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. These results, for $g = 2$, are shown as solid curves in Figure 5, for exponential failures and for Weibull failures with $k = 0.7$ and $k = 0.5$, each curve corresponding to a different checkpointing overhead (C) value. Each curve corresponds to the break-even point and the area above the curve corresponds to settings for which replication is beneficial. As expected, replication becomes 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. For comparison purposes, the figure also shows a set 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. 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 5(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. Note that the results presented in [7] are obtained using Daly's approximation as the checkpointing period.

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 have shown that although the choice of a good checkpointing period can be important in the no-replication case, namely for Weibull failure distributions, this choice is not critical when process replication is used. 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 no longer challenging. 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 could 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. The 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, others, Exascale software study: Software challenges in extreme scale systems, white paper available at: `http://users.ece.gatech.edu/mrichard/ExascaleComputingStudyReports/ECSS%20report%20101909.pdf` (2009).

[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] 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.

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

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

[10] 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.

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

[12] 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`

[13] 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.

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

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

[16] 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.

[17] 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.

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

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

[20] 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.

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

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

[23] 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.

[24] 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.

[25] 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.

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

[27] 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.

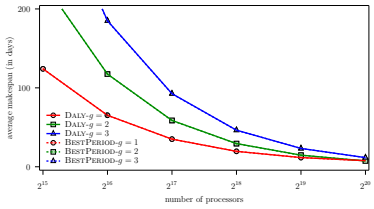
[28] L. Wang, P. Karthik, Z. Kalbarczyk, R. Iyer, L. Votta, C. Vick, A. Wood, Modeling Coordinated Checkpointing for Large-Scale Supercomputers, in: Proc. of the International Conference on Dependable Systems and Networks, 2005, pp. 812–821.

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

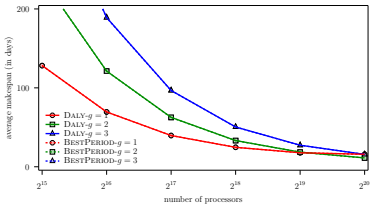
[30] R. Riesen, K. Ferreira, J. Stearley, See applications run and throughput jump: The case for redundant computing in HPC, in: Proc. of the Dependable Systems and Networks Workshops, 2010, pp. 29–34.

[31] D. Kondo, B. Javadi, A. Iosup, D. Epema, The failure trace archive: Enabling comparative analysis of failures in diverse distributed systems, Cluster Computing and the Grid, IEEE International Symposium on 0 (2010) 398–407. doi:http://doi.ieeecomputersociety.org/10.1109/CCGRID.2010.71.

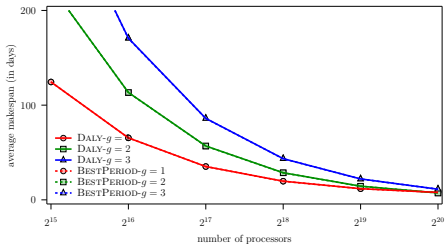
[32] M. Bougeret, H. Casanova, Y. Robert, F. Vivien, D. Zaidouni, Using replication for resilience on exascale systems, Research Report RR-7830, INRIA (2011).
URL `http://hal.inria.fr/hal-00650325`



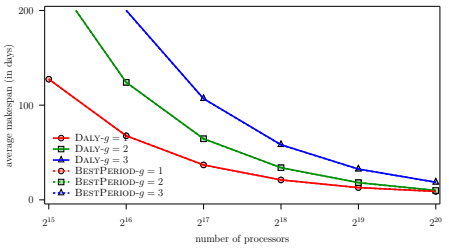
(a) Perfectly parallel jobs: $\mathcal{W}(q) = \frac{\mathcal{W}}{q}$.



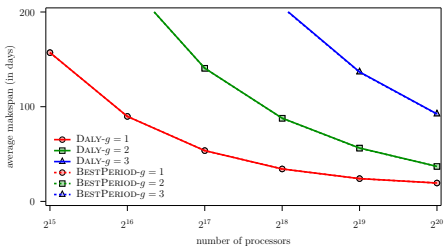
(b) Generic parallel jobs: $\mathcal{W}(q) = \frac{\mathcal{W}}{q} + 10^{-6}\ \mathcal{W}$.



(c) Numerical kernels: $\mathcal{W}(q) = \frac{\mathcal{W}}{q} + 0.1\ \frac{\mathcal{W}^{2/3}}{\sqrt{q}}$.

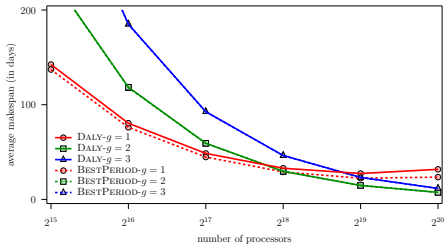


(d) Numerical kernels: $\mathcal{W}(q) = \frac{\mathcal{W}}{q} + \frac{\mathcal{W}^{2/3}}{\sqrt{q}}$.

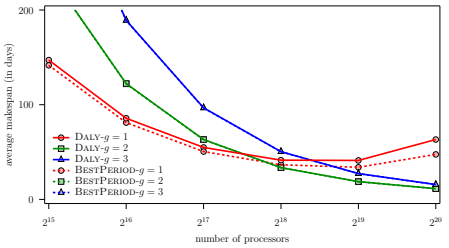


(e) Numerical kernels: $\mathcal{W}(q) = \frac{\mathcal{W}}{q} + 10\ \frac{\mathcal{W}^{2/3}}{\sqrt{q}}$.

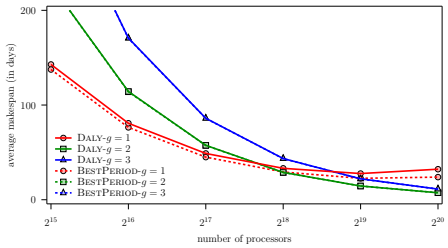
Figure 6: Average makespan vs. number of processors for two choices of the checkpointing period, without process replication (DALY-g=1 and BESTPERIOD-g=1) and with process replication (DALY-g=2 or 3 and BESTPERIOD-g=2 or 3), for Exponential failures ($MTBF = 125$ years).



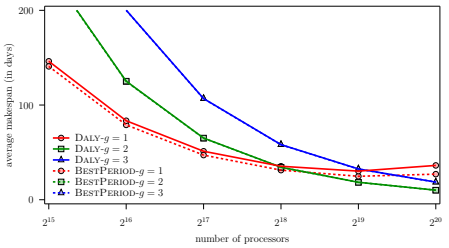
(a) Perfectly parallel jobs: $\mathcal{W}(q) = \frac{\mathcal{W}}{q}$.



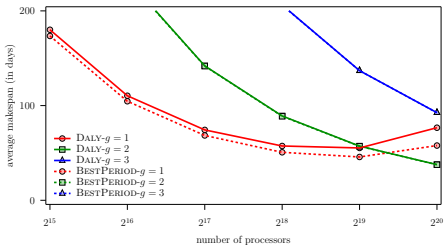
(b) Generic parallel jobs: $\mathcal{W}(q) = \frac{\mathcal{W}}{q} + 10^{-6}\ \mathcal{W}$.



(c) Numerical kernels: $\mathcal{W}(q) = \frac{\mathcal{W}}{q} + 0.1\ \frac{\mathcal{W}^{2/3}}{\sqrt{q}}$.

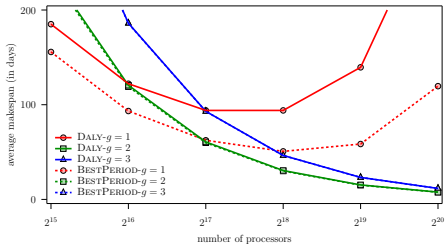


(d) Numerical kernels: $\mathcal{W}(q) = \frac{\mathcal{W}}{q} + \frac{\mathcal{W}^{2/3}}{\sqrt{q}}$.

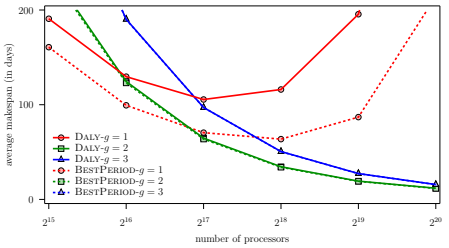


(e) Numerical kernels: $\mathcal{W}(q) = \frac{\mathcal{W}}{q} + 10\ \frac{\mathcal{W}^{2/3}}{\sqrt{q}}$.

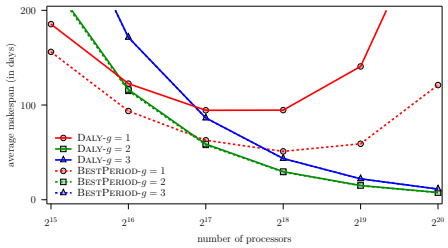
Figure 7: Average makespan vs. number of processors for two choices of the checkpointing period, without process replication (DALY-g=1 and BESTPERIOD-g=1) and with process replication (DALY-g=2 or 3 and BESTPERIOD-g=2 or 3), for Weibull failures ($MTBF$ = 125 years and k = 0.70).



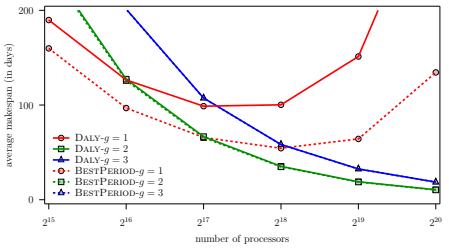
(a) Perfectly parallel jobs: $\mathcal{W}(q) = \frac{\mathcal{W}}{q}$.



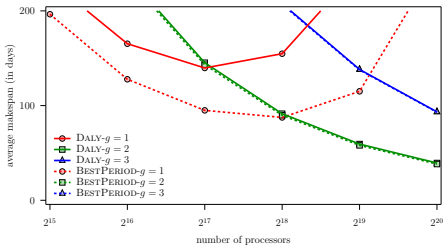
(b) Generic parallel jobs: $\mathcal{W}(q) = \frac{\mathcal{W}}{q} + 10^{-6}\ \mathcal{W}$.



(c) Numerical kernels: $\mathcal{W}(q) = \frac{\mathcal{W}}{q} + 0.1\ \frac{\mathcal{W}^{2/3}}{\sqrt{q}}$.

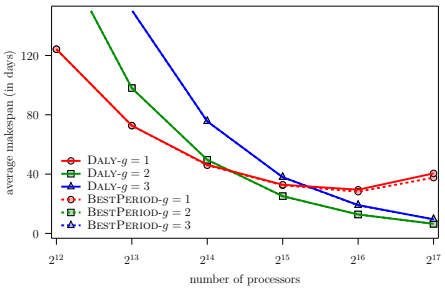


(d) Numerical kernels: $\mathcal{W}(q) = \frac{\mathcal{W}}{q} + \frac{\mathcal{W}^{2/3}}{\sqrt{q}}$.

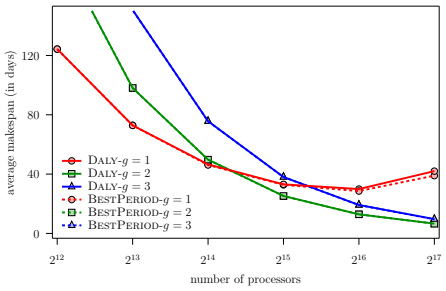


(e) Numerical kernels: $\mathcal{W}(q) = \frac{\mathcal{W}}{q} + 10\ \frac{\mathcal{W}^{2/3}}{\sqrt{q}}$.

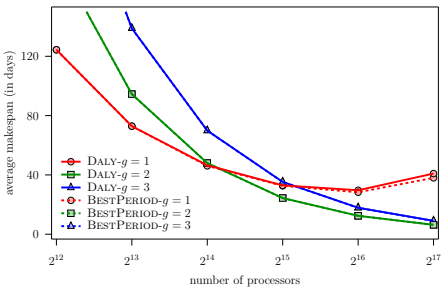
Figure 8: Average makespan vs. number of processors for two choices of the checkpointing period, without process replication (DALY-g=1 and BESTPERIOD-g=1) and with process replication (DALY-g=2 or 3 and BESTPERIOD-g=2 or 3), for Weibull failures ($MTBF$ = 125 years and k = 0.50).



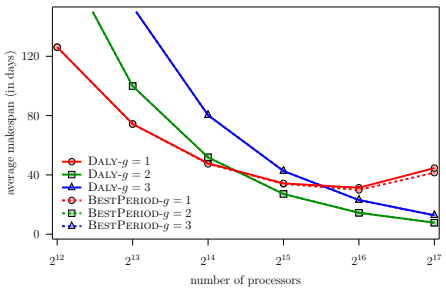
(a) Perfectly parallel jobs: $\mathcal{W}(q) = \frac{\mathcal{W}}{q}$.



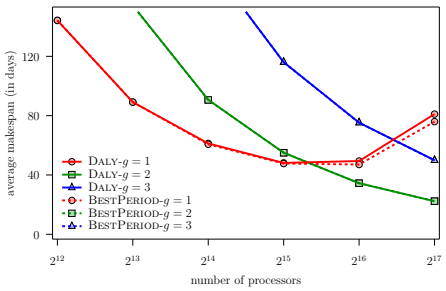
(b) Generic parallel jobs: $\mathcal{W}(q) = \frac{\mathcal{W}}{q} + 10^{-6}\ \mathcal{W}$.



(c) Numerical kernels: $\mathcal{W}(q) = \frac{\mathcal{W}}{q} + 0.1\ \frac{\mathcal{W}^{2/3}}{\sqrt{q}}$.

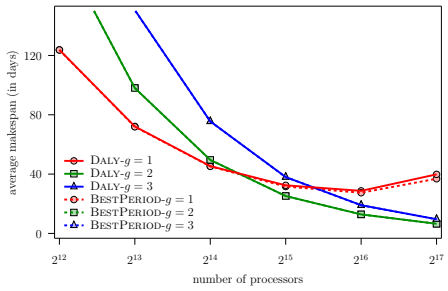


(d) Numerical kernels: $\mathcal{W}(q) = \frac{\mathcal{W}}{q} + \frac{\mathcal{W}^{2/3}}{\sqrt{q}}$.

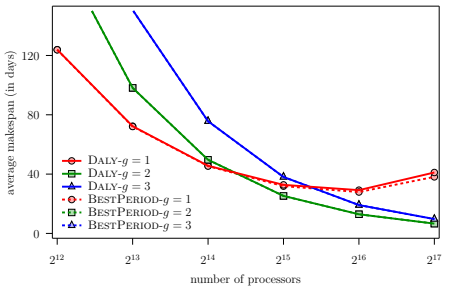


(e) Numerical kernels: $\mathcal{W}(q) = \frac{\mathcal{W}}{q} + 10\ \frac{\mathcal{W}^{2/3}}{\sqrt{q}}$.

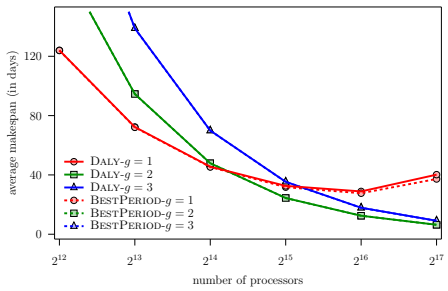
Figure 9: Average makespan vs. number of processors for two choices of the checkpointing period, without process replication (DALY-g=1 and BESTPERIOD-g=1) and with process replication (DALY-g=2 or 3 and BESTPERIOD-g=2 or 3), for failures based on the failure log of **LANL cluster 18**.



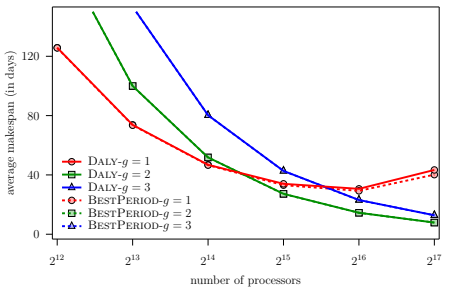
(a) Perfectly parallel jobs: $\mathcal{W}(q) = \frac{\mathcal{W}}{q}$.



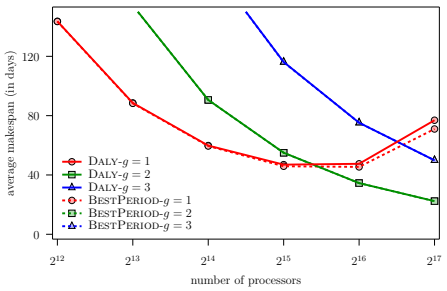
(b) Generic parallel jobs: $\mathcal{W}(q) = \frac{\mathcal{W}}{q} + 10^{-6}\ \mathcal{W}$.



(c) Numerical kernels: $\mathcal{W}(q) = \frac{\mathcal{W}}{q} + 0.1\ \frac{\mathcal{W}^{2/3}}{\sqrt{q}}$.

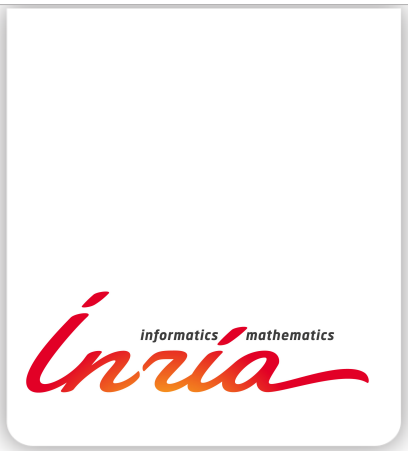


(d) Numerical kernels: $\mathcal{W}(q) = \frac{\mathcal{W}}{q} + \frac{\mathcal{W}^{2/3}}{\sqrt{q}}$.



(e) Numerical kernels: $\mathcal{W}(q) = \frac{\mathcal{W}}{q} + 10\ \frac{\mathcal{W}^{2/3}}{\sqrt{q}}$.

Figure 10: 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 failures based on the failure log of **LANL cluster 19**.



RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

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

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

ISSN 0249-6399