Combining Process Replication and Checkpointing for Resilience on Exascale Systems

Henri Casanova
Univ. of Hawai'i at Manoa,
Honolulu, USA
henric@hawaii.edu

Yves Robert[1,2], Frédéric Vivien[1], and Dounia Zaidouni[1]
1. École Normale Supérieure de Lyon & INRIA, France
{Yves.Robert|Frederic.Vivien|Dounia.Zaidouni}@ens-lyon.fr
2. University of Tennessee Knoxville, USA

May 2012

***Abstract*—Processor failures in post-petascale settings 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 by Ferreira et al. We first identify an incorrect analogy made in their work between process replication and the birthday problem, and derive correct values for the Mean Number of Failures To Interruption and Mean Time To Interruption for Exponential failures distributions. We then extend these results to arbitrary failure distributions, including closed-form solutions for Weibull distributions. Finally, we evaluate process replication using both synthetic and real-world failure traces. Our main findings are: (i) replication is beneficial in fewer scenarios than claimed by Ferreira et al; (ii) although the choice of the checkpointing period can have a high impact on application execution in the no-replication case, with process replication this choice is no longer critical.**

I. 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 such systems are resilient to faults. 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 is the common case rather than the exception. For instance, the 45,208-processor Jaguar platform is reported to experience on the order of 1 failure per day [3], [4], and its scale is modest compared to upcoming platforms. Failures occur because not all faults can be automatically detected and corrected in hardware [5], [6], [7]. To tolerate failures rollback-recovery is used to resume job 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 efforts. The former typically rely on assumptions regarding the probability distributions of times to failure of the processors (e.g., Exponential, Weibull), while the latter rely on simulations driven by failure datasets obtained on real-world platforms. We have, ourselves, made several contributions in this context, including optimal checkpointing strategies for Exponential failures and dynamic programming solutions for Weibull failures [8].

Unfortunately, even assuming an optimal checkpointing strategy, at large scale, processors end up spending as much or even more time saving state than computing state, leading to poor parallel efficiency [5], [6], [7]. Consequently, additional mechanisms must be used. In this work we focus on *replication*: several processors perform the same computation synchronously, so that a failure on 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. The advantage of this approach is that 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. Process replication has been proposed and studied by Ferreira et al. at the SC'2011 conference [9]. In this paper we revisit and extend the results in their work. More specifically, our contributions are:

- We identify an incorrect analogy between process replication and the birthday problem in [9]. As a result, not only are the *MNFTI* (Mean Number of Failures To Interruption) and the *MTTI* (Mean Time To Interruption) values in [9] erroneous, but computing them correctly is more challenging than anticipated.

Nevertheless, we are able to derive correct values for Exponential failures.
- Following a different approach we then extend our results to arbitrary failure distributions, including closed-form solutions for Weibull distributions (the results in [9] were limited to Exponential 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.
- We revisit the "break-even point" results in [9] and obtain results less favorable for process replication. These results, unlike those in [9], are not biased by the choice of a particular checkpointing period.

The remainder of this paper is organized as follows. Section II discusses related work. Section III defines the theoretical framework and states key assumptions. Section IV presents the bulk of our theoretical contribution. Section V presents our simulation methodology. Sections VI presents our simulation results. Finally, Section VII concludes the paper with a summary of our findings.

II. Related work

Checkpointing policies have been widely studied in the literature. In [10], Daly studies periodic checkpointing policies for Exponentially distributed failures, generalizing the well-known bound obtained by Young [11]. Daly extended his work in [12] to study the impact of suboptimal checkpointing periods. In [13], the authors develop an "optimal" checkpointing policy, based on the popular assumption that optimal checkpointing must be periodic. In [14], Bouguerra et al. *prove* that the optimal checkpointing policy is periodic when checkpointing and recovery overheads are constant, for either Exponential or Weibull failures. But their results rely on the unstated assumption that all processors are rejuvenated after each failure and after each checkpoint. In our recent work [8], we have shown that this assumption is unreasonable for Weibull failures. We have developed optimal solutions for Exponential failures and dynamic programming solutions for Weibull failures, demonstrating performance improvements over checkpointing approaches proposed in the literature in the case of Weibull failures. Note that the Weibull distribution is recognized as a reasonable approximation of failures in real-world systems [15], [16], [17], [18]. 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 [5], [6], [7]. Replication has long been used as a fault-tolerance mechanism in distributed systems [19], and more recently in the context of volunteer computing [20]. The idea to use replication together with checkpoint-recovery has been studied in the context of grid computing [21]. One concern about replication in 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 [22], [23], [24]. Most recently, the work by Ferreira et al. [9] has studied the use of process replication for MPI applications. They provide a theoretical analysis of parallel efficiency, an implementation of MPI that supports transparent process replication (including failure detection, consistent message ordering among replicas, etc.), and a set of convincing experimental and simulation results. The work in [9] only considers 2 replicas per application process. The theoretical analysis, admittedly not the primary objective of the authors, is not developed in details. In this work we focus on the theoretical analysis of the problem, both correcting and extending the results in [9], as detailed in Section IV.

III. Framework

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. One application process (replica) runs on one processor, and thus we use the terms processor and process interchangeably.

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 [25], $\gamma < 1$ is the fraction of the work that is inherently sequential.
- Numerical kernels: $\mathcal{W}(q) = \mathcal{W}/q + \gamma\mathcal{W}^{2/3}/\sqrt{q}$. This is representative of a matrix product or a LU/QR factorization of size N on a 2D-processor grid, where $\mathcal{W} = O(N^3)$. In the algorithm in [26], $q = r^2$ and each processor receives $2r$ blocks of size N^2/r^2 during the execution. Here γ is the communication-to-computation ratio of the platform.

Each participating processor is subject to *failures*. A failure causes a *downtime* period of the failing processor, of duration D. When a processor fails, the whole execution is stopped, and all processors must recover from the previous checkpointed state. We let $C(q)$ denote the time needed to perform a checkpoint, and $R(q)$ the time to perform a recovery. The downtime accounts for software rejuvenation (i.e., rebooting [27], [28]) or for the replacement of the failed processor by a spare. Regardless, we assume that after a downtime the processor is fault-free and begins a new lifetime at the beginning of the recovery period. This recovery period corresponds to the time needed to restore

the last checkpoint. Assuming that the application's memory footprint is V bytes, with each processor holding V/q bytes, we consider two scenarios:

- Proportional overhead: $C(q) = R(q) = \alpha V/q = C/q$ 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.

We assume coordinated checkpointing [29] so that no message logging/replay is needed for recovery. We also assume that failures can happen during recovery or checkpointing, but not during a downtime (otherwise, the downtime could be considered part of the recovery).

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 size ω, and then checkpointing it, is $\omega + C(q)$. Finally, we assume that failure arrivals at all processors are independent and identically distributed (i.i.d).

IV. Process Replication

A parallel application consists of several application processes, each process running on a distinct processor. Process replication was recently studied in [9], 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. The results in [9] show large performance improvements due to process replication. Our objective in this section is to provide a full theoretical analysis of process replication.

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. Following [9], 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 indeed 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 we do not consider it (it was also ignored in [9]).

Two important quantities for evaluating the quality of an application execution when replication is used is the Mean Number of Failures To Interruption (*MNFTI*), i.e., the mean number of processor failures until application failure occurs, and the Mean Time To Interruption (*MTTI*), i.e., the mean time elapsed until application failure occurs. In the next two sections, we compute these two quantities, contrasting our work with that in [9], and providing a quantitative comparison in Section IV-C.

A. Computing MNFTI

Ferreira et al. [9] consider the case $g = 2$, and observe that the generalized birthday problem is related to the problem of determining the number of process failures needed to induce an application failure. The generalized birthday problem asks the following question: what is the expected number of balls $NF(m)$ to randomly put into m (originally empty) bins, so that there are two balls in a bin? In the context of process replication the bins are replica groups and the balls are processor failures, so that $m = n_{rg}$ is the number of replica-groups and $NF(m)$ denotes the number of failures. In [9] it is stated that

$$NF(n_{rg}) = 1 + \sum_{k=1}^{n_{rg}} \frac{n_{rg}!}{(n_{rg}-k)! \cdot n_{rg}^k} \approx \sqrt{\frac{\pi n_{rg}}{2}} + \frac{2}{3} \,. \qquad (1)$$

Unfortunately, the processor replication problem is not identical to the generalized birthday problem and Equation (1) does not apply. To illustrate the differences we can simply consider the case $g = 2$. There are two possible approaches to counting failures:

1) One counts each failure that hits any of the $g \cdot n_{rg}$ initial processors, including the processors *already hit* by a failure. This is the approach followed in [9]. With this approach the target problem is not identical to the generalized birthday problem because the second failure to hit a given replica-group does not necessarily induce an application interruption. Indeed, if the failure hits an already hit processor, whose replica had already been killed by the first failure, the application is not affected. If, on the contrary, the failure hits the other processor, both replicas of a same process are killed and the whole application fails.
2) One only counts 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. With this method, the problem is still not

identical to the generalized birthday problem. Let us consider the situation right after the first failure occurred. In the generalized birthday problem one assumes that all integers in the range are uniformly distributed. In our problem, the replica-group that suffered from the first failure only contains a single running replica after that failure, while all the other replica-groups still contain two running replicas. Therefore, if the probability of failures is uniformly distributed among processors that are still running[1] (which is usually assumed), then the replica-group hit by the first failure has half the probability of being hit by the second failure as that of the other replica-groups, simply because it contains half as many running replicas! Since the distribution of failure is no longer uniform, the birthday problem is not relevant.

Now that we no longer can use the analogy to the birthday problem, computing the *MNFTI* turns out to be more challenging. To cover all cases we consider both options above. We use $MNFTI^{\text{ah}}$ to denote the *MNFTI* (Mean Number of Failures To Interruption) 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 $MNFTI^{\text{ah}}$:

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

[1]Note that if the failure probability is uniformly distributed among the $g \cdot n_{rg}$ initial processors, including the processors already hit by a failure, then the probability of failure is uniformly distributed among still running processors!

The probability of failure is uniformly distributed between the two replicas, and thus between the two previous 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 processors 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* $\mathbb{E}(NFTI^{\text{rp}}|n_f) =$

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

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

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

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 $\mathbb{E}\left(NFTI^{\text{rp}}|n_f+1\right)$. Altogether we have derived that:

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

Therefore,

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

■

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*

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

Proof: We prove by induction that $\mathbb{E}(NFTI^{\text{ah}}|n_f) = 1 + \mathbb{E}(NFTI^{\text{rp}}|n_f)$, 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 $\mathbb{E}(NFTI^{\text{rp}}|n_{rg}) = 1$ and $\mathbb{E}(NFTI^{\text{ah}}|n_{rg}) = 2$. Hence the property is true for $n_f = n_{rg}$. Consider a value $n_f < n_{rg}$, and assume to have proven that $\mathbb{E}(NFTI^{\text{ah}}|i) = 1+\mathbb{E}(NFTI^{\text{rp}}|i)$, 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:

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

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:*

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

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 (2) 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:

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

which can be rewritten as Equation (2). ∎

B. Computing MTTI

In [9], for the case $g = 2$, the Mean Time To application Interruption, with the "already hit" assumption, is computed as

$$MTTI = systemMTBF(2n_{rg}) \times NF(n_{rg}), \qquad (3)$$

where the value of $NF(n_{rg})$ is given by Equation (1). $systemMTBF(n)$ denotes the mean time between failures of a platform with n processors. This expression assumes that the failures follow an Exponential distribution and becomes correct when replacing $NF(n_{rg})$ by $MNFTI^{\text{ah}}$ as given by Theorem 1. 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 (3) 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 stricken 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:

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

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 (3) 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.

These approaches for computing *MTTI* are, unfortunately, limited to exponentially distributed failures. 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. 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 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 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:*

$$R(t) = \prod_{j=1}^{n_{rg}} \left(1 - \prod_{i=1}^{g} F\left(t|\tau_{i+g(j-1)}\right)\right) \tag{4}$$

and the MTTI is given by:

$$MTTI = \int_0^{+\infty} \prod_{j=1}^{n_{rg}} \left(1 - \prod_{i=1}^{g} F\left(t|\tau_{i+g(j-1)}\right)\right) \mathrm{d}t. \tag{5}$$

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 4 and 5 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.

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:

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

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

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

Using the equation $\binom{k-1}{i} + 2\binom{k-1}{i-1} = \binom{k}{i}\frac{(k+i)}{k}$, we derive

the desired expression for $MTTI$:

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

■

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,

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

■

C. Numerical Evaluation

Table I shows the $MNFTI^{\text{ah}}$ values as computed by the formula in [9] and by Theorem 1, for various values of n_{rg} and for $g = 2$. The percentage relative difference between the two values is included in the table as well. The two values diverge significantly, with relative differences between 29% and 33%. Here again we conclude that the formula in [9] significantly under-estimates *MNFTI* for either of the failure accounting approaches (recall that $MNFTI^{\text{ah}}$ and $MNFTI^{\text{rp}}$ differ only by 1).

Table II shows the *MTTI* values as computed by the formula in [9] and by Corollary 1, for various values of n_{rg} and for $g = 2$. The percentage relative difference between the formula in [9] and our recursive formula is indicated as well. Here again the two values diverge significantly, exactly in the same proportion as for the $MNFTI^{\text{ah}}$ (see Equation (3)). We conclude that the formula in [9] significantly under-estimates *MTTI*.

As expected, using the formula from Corollary 1, plugging in the value from Theorem 1 into Equation (3), and a using the recursive computation given all lead to the same numerical values. To validate these computations we have compared the values that they produce to the *MTTI* as computed through simulations. For each studied value of n_{rg}, we have generated 200,000 random failure dates, computed the Time To application Interruption for each instance, and computed the mean of these values. This *simulated MTTI*, also reported in Table II, is in full agreement with Corollary 1.

V. Simulation framework

In this section 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/SC2012Replication.

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 [3], [4]. 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 the number of processors in the Jaguar platform. 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 and 0.5 based on the results in [16] and [17].

Log-based failure distributions – We also consider failure distributions based on failure logs from production clusters. We used logs for the largest clusters among the preprocessed logs in the *Failure trace archive* [30], i.e., for clusters at the Los Alamos National Laboratory [16]. 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 [16]). 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 (in [8] we have developed a dynamic programming strategy that leads to non-periodic checkpointing in the general case). The checkpointing period can be computed based on the *MTTI* value using Young's approximation [11] or Daly's first-order approximation [10], the latter being used in [9]. Alternately, since our experiments are in simulation, we can search numerically for the best period. To build the candidate periods, the period computed by OptExp from [8] is multiplied and divided by $1 + 0.05 \times i$ with $i \in \{1, ..., 180\}$, and by 1.1^j with $j \in \{1, ..., 60\}$. We present results with the period as given by Daly's approximation and with the best period found numerically.

Replication overhead – In [9], 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 sublinear 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

Table I
$MNFTI^{\text{ah}}$ AS COMPUTED BY THE FORMULA IN [9] AND BY THEOREM 1, FOR $n_{rg} = 2^0, \ldots, 2^{20}$, WITH $g = 2$.

Number of processes	2^0	2^1	2^2	2^3	2^4	2^5	2^6
Formula in [9]	2	2.5	3.22	4.25	5.7	7.77	10.7
Theorem 1	3	3.67	4.66	6.09	8.15	11.1	15.2
% Relative Difference	-33	-32	-31	-30	-30	-30	-30
Number of processes	2^7	2^8	2^9	2^{10}	2^{11}	2^{12}	2^{13}
Formula in [9]	14.9	20.7	29	40.8	57.4	80.9	114
Theorem 1	21.1	29.4	41.1	57.7	81.2	114	161
% Relative Difference	–30	-29	-29	-29	-29	-29	-29
Number of processes	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}
Formula in [9]	161	228	322	454	642	908	1284
Theorem 1	228	322	455	643	908	1284	1816
% Relative Difference	-29	-29	-29	-29	-29	-29	-29

Table II
$MTTI$ AS COMPUTED BY THE FORMULA IN [9] AND BY COROLLARY 1, FOR $n_{rg} = 2^0, \ldots, 2^{20}$, WITH $g = 2$.

Number of processes	2^0	2^1	2^2	2^3	2^4	2^5	2^6
Formula in [9]	1	0.625	0.402	0.265	0.178	0.121	0.0836
Corollary 1	1.5	0.917	0.582	0.381	0.255	0.173	0.119
% Relative Diff	-33.33	-31.82	-30.89	-30.32	-29.97	-29.75	-29.6
Simulated $MTTI$	1.498	0.9184	0.5831	0.3808	0.2542	0.1725	0.1188
Number of processes	2^7	2^8	2^9	2^{10}	2^{11}	2^{12}	2^{13}
Formula in [9]	0.058	0.0405	0.0284	0.0199	0.014	0.00987	0.00696
Corollary 1	0.0823	0.0574	0.0402	0.0282	0.0198	0.014	0.00985
% Relative Diff	-29.5	-29.44	-29.39	-29.36	-29.34	-29.33	-29.31
Simulated $MTTI$	0.08226	0.05738	0.0401	0.02825	0.01982	0.01399	0.009853
Number of processes	2^{14}	2^{15}	2^{16}	2^{17}	2^{18}	2^{19}	2^{20}
Formula in [9]	0.00492	0.00347	0.00245	0.00173	0.00123	0.00086	0.000612
Corollary 1	0.00695	0.00491	0.00347	0.00245	0.00173	0.00122	0.000866
% Relative Diff	-29.31	-29.3	-29.3	-29.3	-29.29	-29.29	-29.29
Simulated $MTTI$	0.00693	0.00491	0.00347	0.00245	0.00173	0.00123	0.000868

p is the number of processors. The parameters to this model are instantiated from the application in [9] that has the highest replication overhead. We use the same logarithmic model to augment our first two parallel job models in Section III:

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

Parameter values – We use the same parameters as in [8]. Namely, $C = R = 600\ s$, $D = 60\ s$ and $\mathcal{W} = 10,000$ years (except for log-base simulations for which $\mathcal{W} = 1,000$ years).

VI. Simulation Results

A. Checkpointing Period

Our first set of experiments aims at determining whether using Daly's approximation for computing the checkpointing period, as done in [9], is an effective approach. In the $g = 2$ case (two replicas per application process), we compute this period using the correct $MTTI$ expression from Corollary 1 rather than the erroneous

value given in [9]. 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 $g = 2$ results, we also present results for $g = 1$ (no replication) as a baseline (in which case the $MTTI$ is simply the processor MTBF).

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 III, augmented with the replication overhead model described in Section V: (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.

We found that for a given failure distribution all results follow the same trend regardless of the job model. We show results for the five considered parallel job models. Figures 1, 2, 3, 4, and 5 show results for each of the five considered failure distributions. Each figure shows five graphs, each graph shows average makespan vs. number of processors for one of the five parallel job models. The two curves for $g = 1$ are exactly superposed in all graphs of Figure 1, and the two curves for $g = 2$ are exactly superposed in all graphs of all 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 approximation leads to significantly suboptimal results that worsen as k decreases (as we already reported in [8]). What is perhaps less expected is that in the case $g = 2$, using Daly's approximation leads to virtually the same performance as using the best period even for Weibull failures. This is not to say that Daly's approximation yields the best checkpointing period. Application makespan is simply not sentitive to the checkpointing period, at least in a wide neighborhood around the best period. With process replication, application failures and recoveries are so infrequent, i.e., the MTBF of a pair of replicas is so large, that Daly's approximation is good enough. To quantify the frequency of application failures, Table III 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). While this low fraction showcases the benefit of process replication, it also makes the choice of the replication period non-critical.

Table III
FRACTION OF PROCESSOR FAILURES THAT LEAD TO APPLICATION FAILURES WITH PROCESS REPLICATION ($g = 2$) ASSUMING WEIBULL FAILURE DISTRIBUTIONS ($k = 0.7, 0.5$) 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

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 Daly's approximation leads to longer average makespans than when using the best checkpointing period (see Figures 6, 7, 8, 9, and 10). This is even true for exponential failures. Consider for instance a process replication scenario with Weibull failures of shape parameters $k = 0.7$, a perfect 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 20.76 days, as opposed to 18.1 days when using the best period– an increase of more than 12%.

We summarize our findings so far. Without replication, Daly's approximation produces significantly suboptimal checkpointing policies when failures are not exponentially distributed. The Weibull distribution is recognized as a reasonable approximation of failures in real-world systems [15], [16], [17], [18]. When using replication, Daly's approximation can also lead to poor periodic checkpointing. However, this never happens in practical settings because replication drastically reduces the number of failures. In fact, in practical process replication settings, the choice of the checkpointing period is not critical. Consequently, setting the checkpointing period based on Daly's approximation is a safe choice when process replication is used. This validates the unsubstantiated choice of using this approximation in [9].

B. Revisiting the Results in [9]

Figure 9 in [9] presents interesting results for the"break-even" point for process replication. More specifically, in a 2-D plane defined by the processor MTBF and the number of processors, and given a checkpointing overhead,

the figure shows 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 curves are shown for different checkpointing overheads, and as expected, the higher the overhead, the more beneficial it is to use process replication.

These results are obtained for exponential failures and using the checkpointing period given by Daly's approximation. In the previous section we have seen that in the no-replication case this approximation leads to poor results when the failure distribution is Weibull (see the $g = 1$ curves in Figures 2 and 3).

Although our results for two particular production workloads show that Daly's approximation led to reasonably good results, there is evidence that, in general, failure distributions are well approximated by Weibull distributions [15], [16], [17], [18]. Most recently, in [18], the authors show that failures observed on a production cluster, over a cumulative 42-month time period, are modeled well by a Weibull distribution. Furthermore, the shape parameter of this distribution, k, is below 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 for $k = 0.7$ to Figure 3 for $k = 0.5$). Nevertheless, in the case of process replication, Daly's approximation leads to the same results as the best checkpointing period for practical settings. We conclude that the results in [9] give a biased answer to the break-even point question, or at least an answer that is limited to the use of Daly's approximation. Such an answer is sensitive to the effect of this approximation on application execution, and turns out to be too favorable for process replication.

We now revisit the results in [9] by always using the best checkpointing period for each simulated application execution, as computed by a numerical search. Therefore, our results quantify the definitive break-even point, removing the choice of the checkpointing period from the equation. These results are shown as solid curves in Figure 11 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. For comparison, dashed curves correspond to results obtained using Daly's approximation as done in [9]. The area above a curve corresponds to settings for which replication is beneficial. As expected, the general trends are similar to that seen in Figure 9 in [9]: process 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. More importantly, the distance between each solid line and its dashed counterpart shows by how much the results in [9] are optimistic in favor of process replication due to the use of Daly's approximation. This distance increases as k decreases, which is expected since the Weibull distribution is then further away from the Exponential distribution. (For exponential distributions, the curves match.) For instance, in the case $k = 0.5$ (Figure 11(c)), 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 $C = 900s$, obtained using the best checkpointing period. Considering the solid curve for $C = 150s$ and the dashed one for $C = 300s$, one sees that the impact of the checkpointing overhead is twice as unfavorable for process replication as that indicated by results obtained with the approach in [9] when the process MTBF is over 100 years.

VII. Conclusion

In this paper we have presented a rigorous study of process replication for large-scale platforms. We have conducted a thorough analysis, providing recursive expressions for *MNFTI*, and analytical expressions for *MTTI* with arbitrary distributions, that lead to closed-form expressions for Exponential and Weibull distributions. We have explained why the *MNFTI* and *MTTI* values determined in [9] are not accurate, leading to a difference of roughly 30% with our own calculations, which are validated via simulation experiments. In addition, we have identified an unexpected relationship between two natural failure models (*already hit* and *running processors*).

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 revisited the results in [9] that quantify the break-even point between replication and no-replication for Weibull failures. Our results differ and are less favorable for process replication. This difference is because we use the best checkpointing period rather than that provided by Daly's approximation, since we have shown the latter to be detrimental to the no-replication case. Our break-even results thus provide a fairer comparison that is not impacted by the choice of a particular checkpointing period.

Altogether, our results provide a sound basis for quantifying the potential benefit of process replication for future HPC platforms. While not as favorable for replication as those in [9], 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 resource waste that it induces, 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.

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, and M. Valero, "The international exascale software project: a call to cooperative action by the global high-performance community," *Int. J. High Perform. Comput. Appl.*, vol. 23, no. 4, pp. 309–322, 2009.

[2] V. Sarkar and others, "Exascale software study: Software challenges in extreme scale systems," 2009, white paper available at: http://users.ece.gatech.edu/mrichard/ExascaleComputingStudyReports/ECSS%20report%20101909.pdf.

[3] E. Meneses, "Clustering Parallel Applications to Enhance Message Logging Protocols," https://wiki.ncsa.illinois.edu/download/attachments/17630761/INRIA-UIUC-WS4-emenese.pdf?version=1&modificationDate=1290466786000.

[4] L. Bautista Gomez, A. Nukada, N. Maruyama, F. Cappello, and S. Matsuoka, "Transparent low-overhead checkpoint for GPU-accelerated clusters," https://wiki.ncsa.illinois.edu/download/attachments/17630761/INRIA-UIUC-WS4-lbautista.pdf?version=1&modificationDate=1290470402000.

[5] E. Elnozahy and J. Plank, "Checkpointing for Peta-Scale Systems: A Look into the Future of Practical Rollback-Recovery," *IEEE Transactions on Dependable and Secure Computing*, vol. 1, no. 2, pp. 97—108, 2004.

[6] R. A. Oldfield, S. Arunagiri, P. J. Teller, S. Seelam, M. R. Varela, R. Riesen, and 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.

[7] B. Schroeder and G. A. Gibson, "Understanding Failures in Petascale Computers," *Journal of Physics: Conference Series*, vol. 78, no. 1, 2007.

[8] M. Bougeret, H. Casanova, M. Rabie, Y. Robert, and F. Vivien, "Checkpointing strategies for parallel jobs," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '11. ACM Press, 2011.

[9] K. Ferreira, J. Stearley, J. H. I. Laros, R. Oldfield, K. Pedretti, R. Brightwell, R. Riesen, P. G. Bridges, and D. Arnold, "Evaluating the Viability of Process Replication Reliability for Exascale Systems," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '11. ACM Press, 2011.

[10] J. T. Daly, "A higher order estimate of the optimum checkpoint interval for restart dumps," *Future Generation Computer Systems*, vol. 22, no. 3, pp. 303–312, 2004.

[11] J. W. Young, "A first order approximation to the optimum checkpoint interval," *Communications of the ACM*, vol. 17, no. 9, pp. 530–531, 1974.

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

[13] K. Venkatesh, "Analysis of Dependencies of Checkpoint Cost and Checkpoint Interval of Fault Tolerant MPI Applications," *Analysis*, vol. 2, no. 08, pp. 2690–2697, 2010.

[14] M.-S. Bouguerra, T. Gautier, D. Trystram, and J.-M. Vincent, "A flexible checkpoint/restart model in distributed systems," in *PPAM*, ser. LNCS, vol. 6067, 2010, pp. 206–215. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-14390-8_22

[15] T. Heath, R. P. Martin, and T. D. Nguyen, "Improving cluster availability using workstation validation," *SIGMETRICS Perf. Eval. Rev.*, vol. 30, no. 1, pp. 217–227, 2002.

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

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

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

[19] F. Gärtner, "Fundamentals of fault-tolerant distributed computing in asynchronous environments," *ACM Computing Surveys*, vol. 31, no. 1, 1999.

[20] D. Kondo, A. Chien, and H. Casanova, "Scheduling Task Parallel Applications for Rapid Application Turnaround on Enterprise Desktop Grids," *Journal of Grid Computing*, vol. 5, no. 4, pp. 379–405, 2007.

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

[22] B. Schroeder and G. Gibson, "Understanding failures in petascale computers," *Journal of Physics: Conference Series*, vol. 78, no. 1, 2007.

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

[24] C. Engelmann, H. H. Ong, and 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.

[25] G. Amdahl, "The validity of the single processor approach to achieving large scale computing capabilities," in *AFIPS Conference Proceedings*, vol. 30. AFIPS Press, 1967, pp. 483–485.

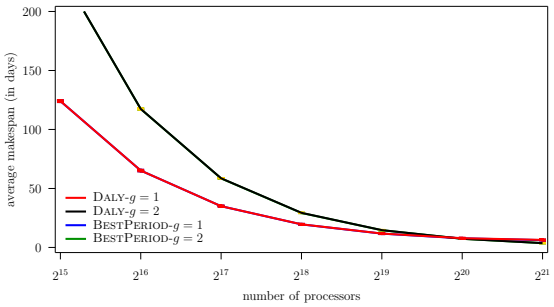
[26] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley, *ScaLAPACK Users' Guide*. SIAM, 1997.

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

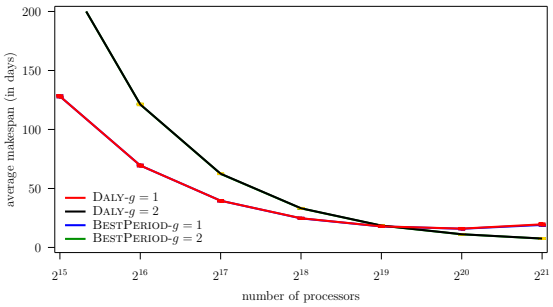
[28] V. Castelli, R. E. Harper, P. Heidelberger, S. W. Hunter, K. S. Trivedi, K. Vaidyanathan, and W. P. Zeggert, "Proactive management of software aging," *IBM J. Res. Dev.*, vol. 45, no. 2, pp. 311–332, 2001.

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

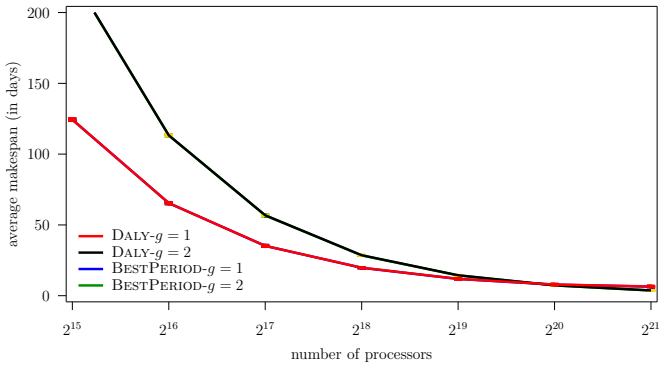
[30] D. Kondo, B. Javadi, A. Iosup, and D. Epema, "The failure trace archive: Enabling comparative analysis of failures in diverse distributed systems," *Cluster Computing and the Grid, IEEE International Symposium on*, vol. 0, pp. 398–407, 2010.



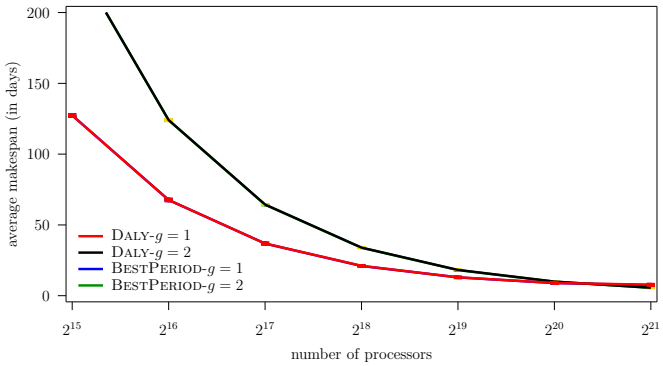
(a) Perfectly parallel jobs: $\mathcal{W}(q) = \frac{\mathcal{W}}{q}$.



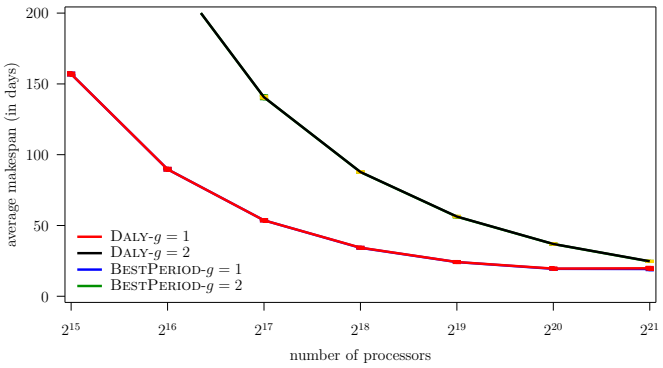
(b) Generic parallel jobs: $\mathcal{W}(q) = \frac{\mathcal{W}}{q} + 10^{-6}\ \mathcal{W}$.



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

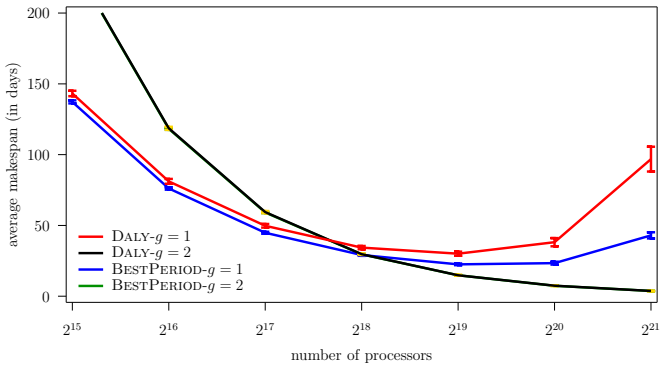


(d) Numerical kernels: $\mathcal{W}(q) = \frac{\mathcal{W}}{q} + \frac{\mathcal{W}^{2/3}}{\sqrt{q}}$.

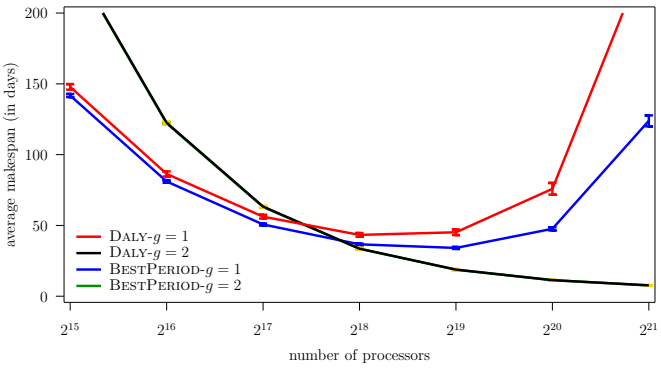


(e) Numerical kernels: $\mathcal{W}(q) = \frac{\mathcal{W}}{q} + 10\ \frac{\mathcal{W}^{2/3}}{\sqrt{q}}$.

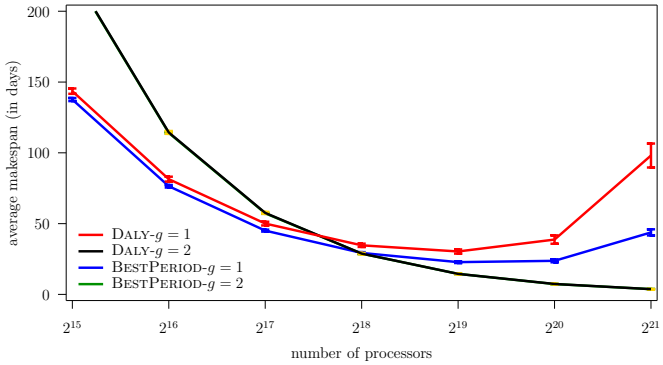
Figure 1. Average makespan vs. number of processors for two choices of the checkpointing period, without process replication (DALY-g=1 and BESTPERIOD-g=1) and with process replication (DALY-g=2 and BESTPERIOD-g=2), for Exponential failures ($MTBF$ = 125 years).



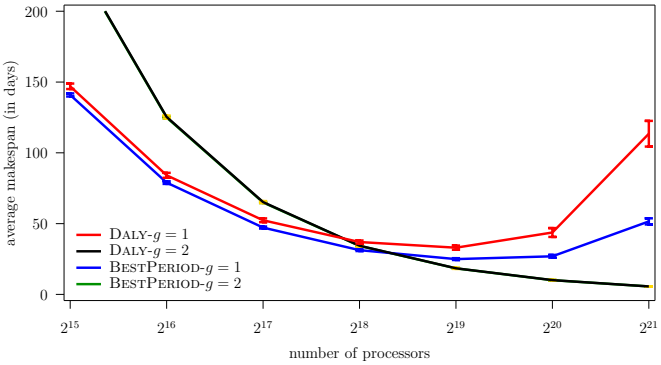
(a) Perfectly parallel jobs: $\mathcal{W}(q) = \frac{\mathcal{W}}{q}$.



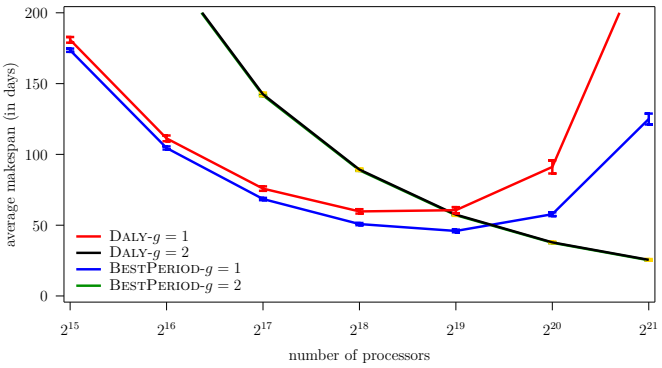
(b) Generic parallel jobs: $\mathcal{W}(q) = \frac{\mathcal{W}}{q} + 10^{-6}\ \mathcal{W}$.



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

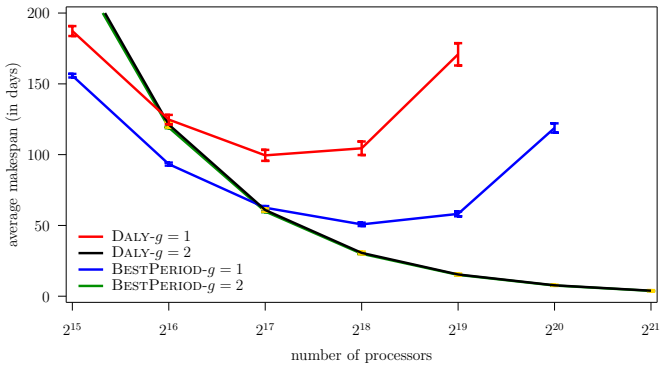


(d) Numerical kernels: $\mathcal{W}(q) = \frac{\mathcal{W}}{q} + \frac{\mathcal{W}^{2/3}}{\sqrt{q}}$.

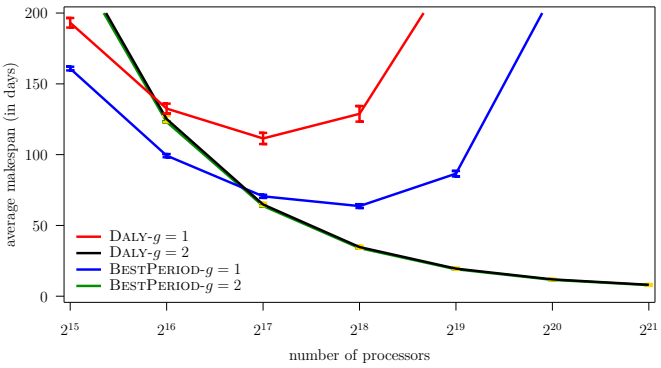


(e) Numerical kernels: $\mathcal{W}(q) = \frac{\mathcal{W}}{q} + 10\ \frac{\mathcal{W}^{2/3}}{\sqrt{q}}$.

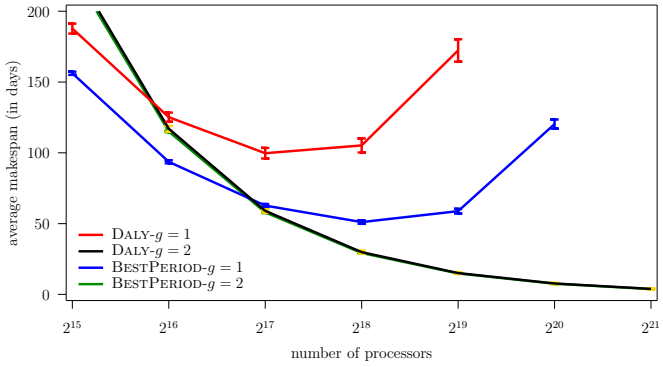
Figure 2. Average makespan vs. number of processors for two choices of the checkpointing period, without process replication (DALY-g=1 and BESTPERIOD-g=1) and with process replication (DALY-g=2 and BESTPERIOD-g=2), for Weibull failures ($MTBF$ = 125 years and $k = 0.70$).



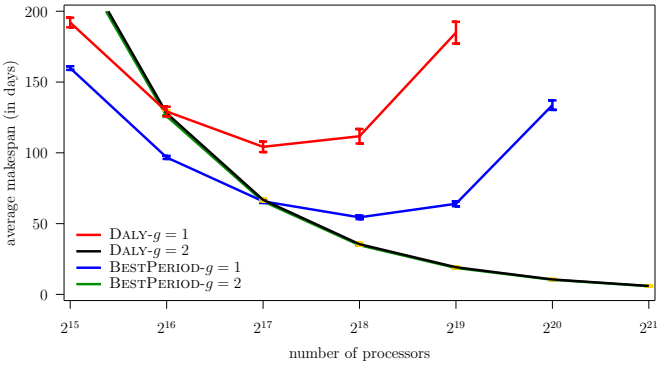
(a) Perfectly parallel jobs: $\mathcal{W}(q) = \frac{\mathcal{W}}{q}$.



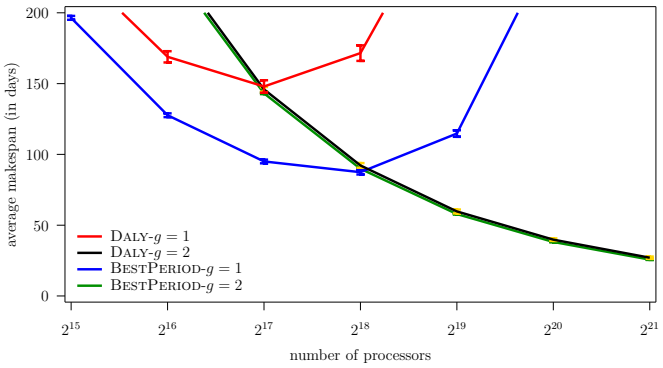
(b) Generic parallel jobs: $\mathcal{W}(q) = \frac{\mathcal{W}}{q} + 10^{-6}\ \mathcal{W}$.



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

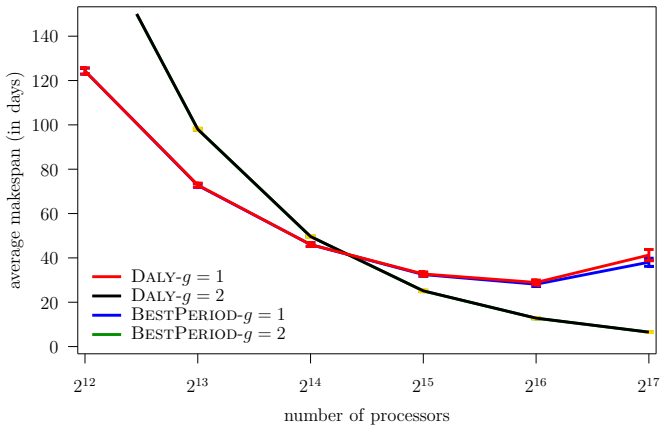


(d) Numerical kernels: $\mathcal{W}(q) = \frac{\mathcal{W}}{q} + \frac{\mathcal{W}^{2/3}}{\sqrt{q}}$.

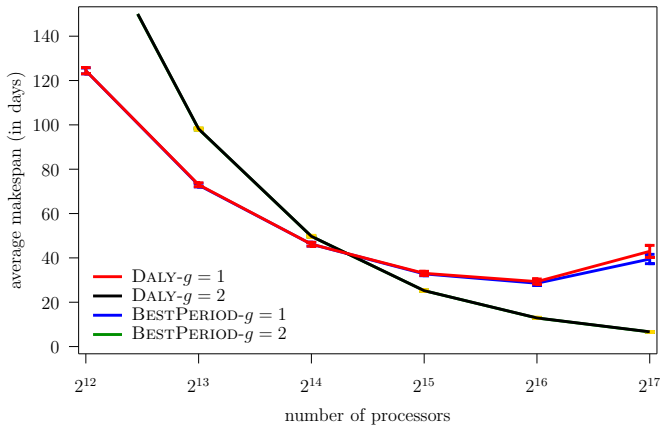


(e) Numerical kernels: $\mathcal{W}(q) = \frac{\mathcal{W}}{q} + 10\ \frac{\mathcal{W}^{2/3}}{\sqrt{q}}$.

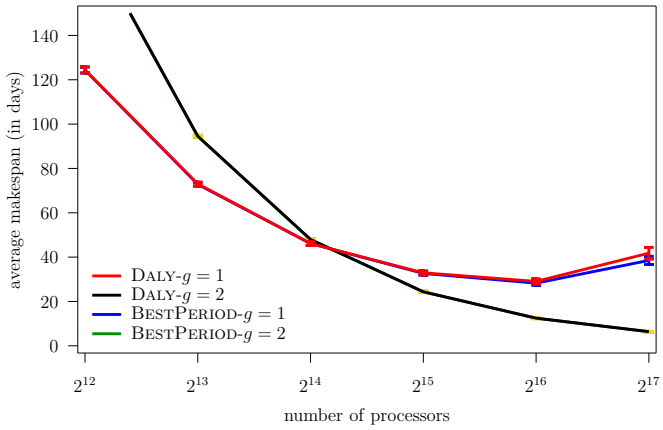
Figure 3. Average makespan vs. number of processors for two choices of the checkpointing period, without process replication (DALY-g=1 and BESTPERIOD-g=1) and with process replication (DALY-g=2 and BESTPERIOD-g=2), for Weibull failures ($MTBF$ = 125 years and $k = 0.50$).



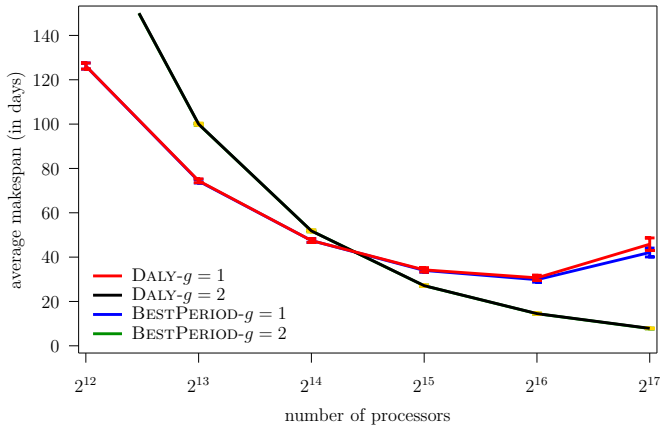
(a) Perfectly parallel jobs: $\mathcal{W}(q) = \frac{\mathcal{W}}{q}$.



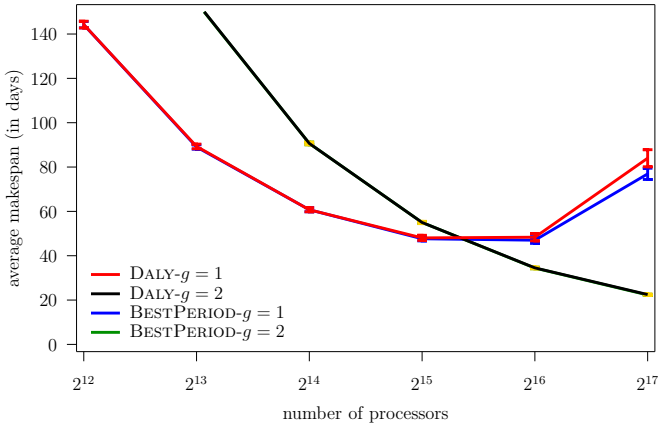
(b) Generic parallel jobs: $\mathcal{W}(q) = \frac{\mathcal{W}}{q} + 10^{-6}\ \mathcal{W}$.



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

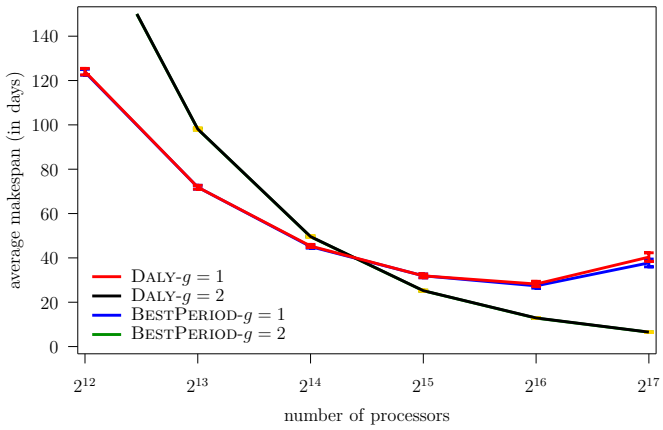


(d) Numerical kernels: $\mathcal{W}(q) = \frac{\mathcal{W}}{q} + \frac{\mathcal{W}^{2/3}}{\sqrt{q}}$.

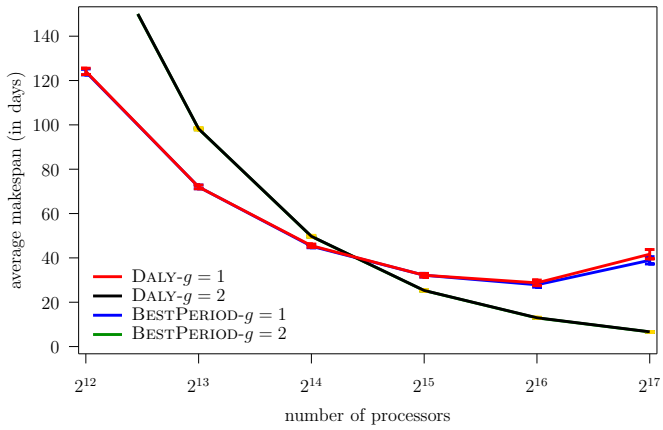


(e) Numerical kernels: $\mathcal{W}(q) = \frac{\mathcal{W}}{q} + 10\ \frac{\mathcal{W}^{2/3}}{\sqrt{q}}$.

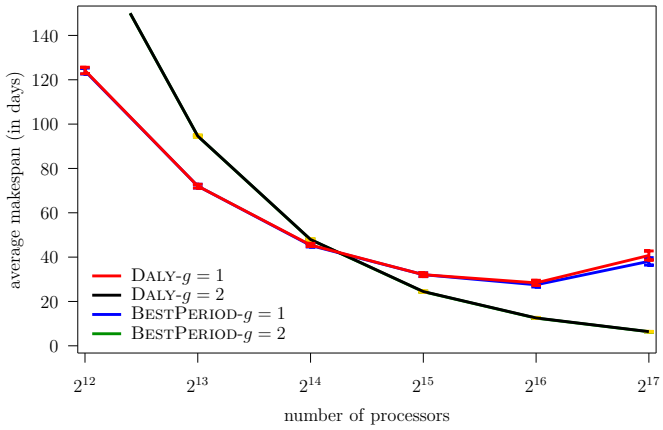
Figure 4. Average makespan vs. number of processors for two choices of the checkpointing period, without process replication (DALY-g=1 and BESTPERIOD-g=1) and with process replication (DALY-g=2 and BESTPERIOD-g=2), for failures based on the failure log of **LANL cluster 18**.



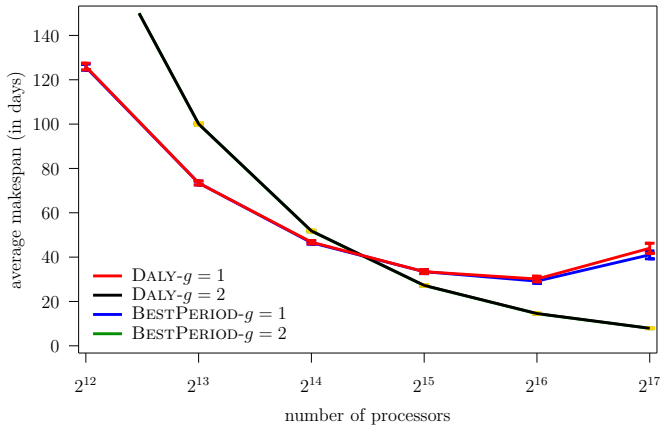
(a) Perfectly parallel jobs: $\mathcal{W}(q) = \frac{\mathcal{W}}{q}$.



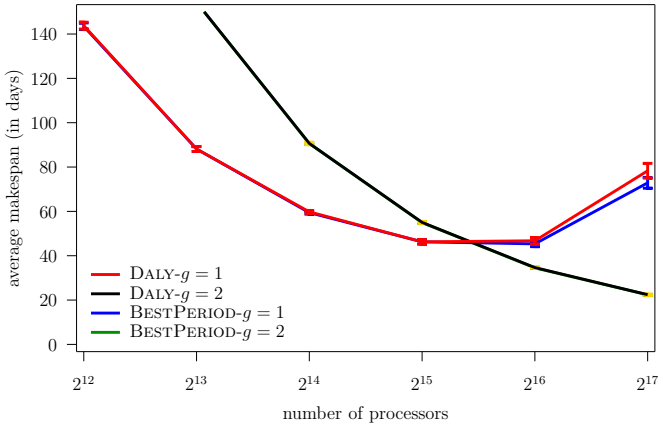
(b) Generic parallel jobs: $\mathcal{W}(q) = \frac{\mathcal{W}}{q} + 10^{-6}\ \mathcal{W}$.



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

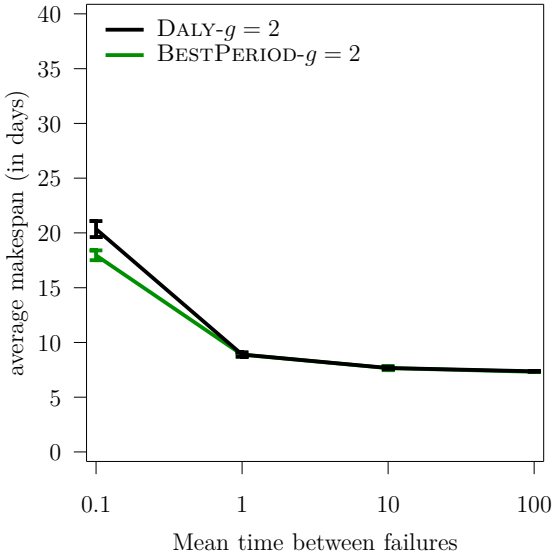


(d) Numerical kernels: $\mathcal{W}(q) = \frac{\mathcal{W}}{q} + \frac{\mathcal{W}^{2/3}}{\sqrt{q}}$.

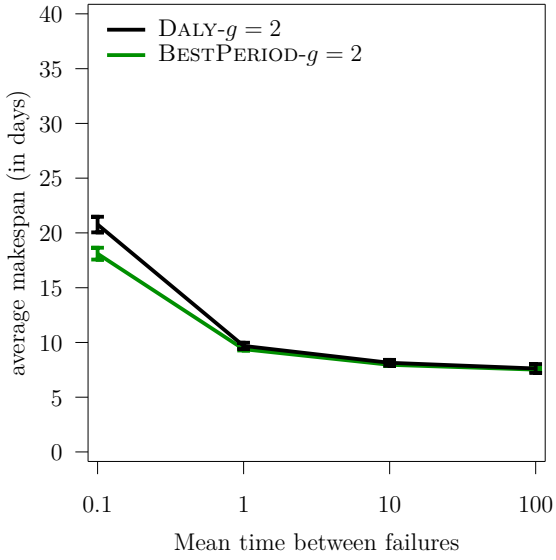


(e) Numerical kernels: $\mathcal{W}(q) = \frac{\mathcal{W}}{q} + 10\ \frac{\mathcal{W}^{2/3}}{\sqrt{q}}$.

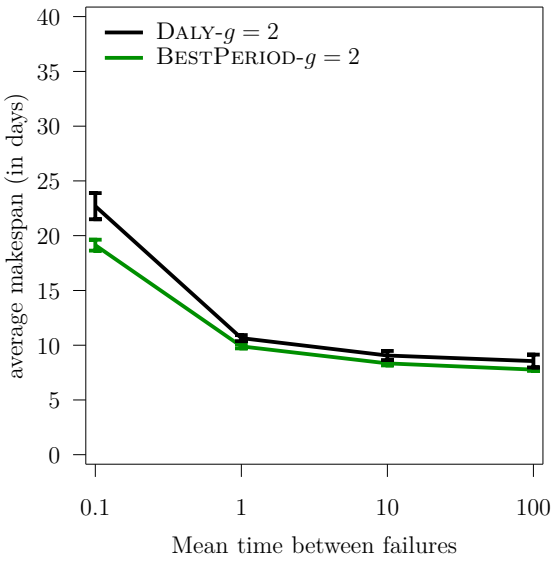
Figure 5. Average makespan vs. number of processors for two choices of the checkpointing period, without process replication (DALY-g=1 and BESTPERIOD-g=1) and with process replication (DALY-g=2 and BESTPERIOD-g=2), for failures based on the failure log of **LANL cluster 19**.



(a) Exponential failure distribution

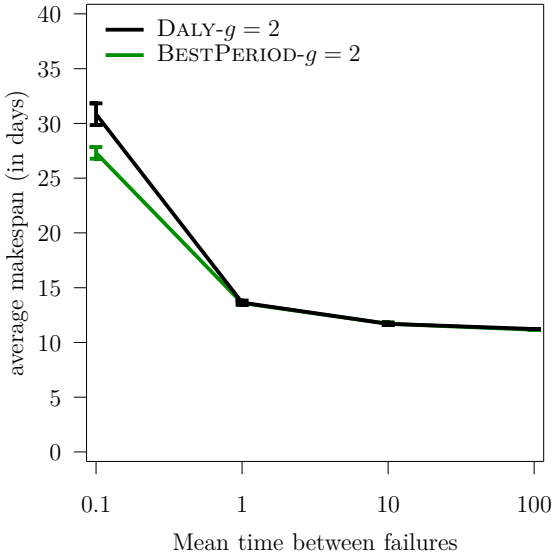


(b) Weibull failure distribution ($k = 0.7$)

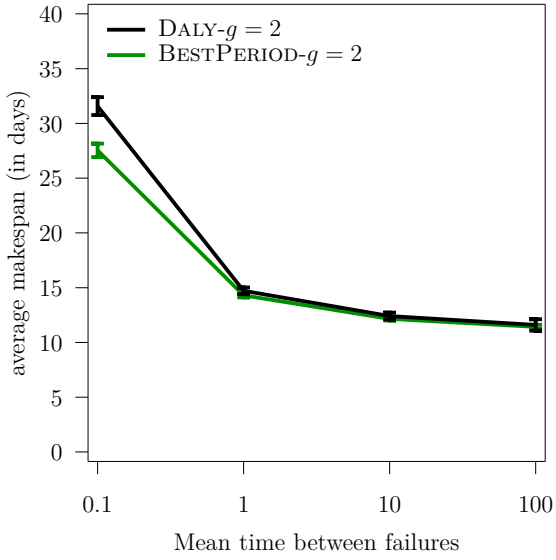


(b) Weibull failure distribution ($k = 0.5$)

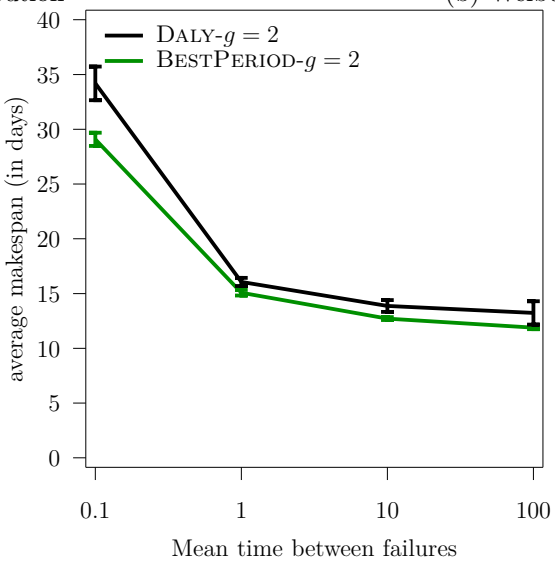
Figure 6. Average makespan vs. processor MTBF using group replication ($g = 2$) and using Perfectly parallel jobs: $\mathcal{W}(q) = \frac{\mathcal{W}}{q}$, showing that Daly's approximation can be suboptimal.



(a) Exponential failure distribution

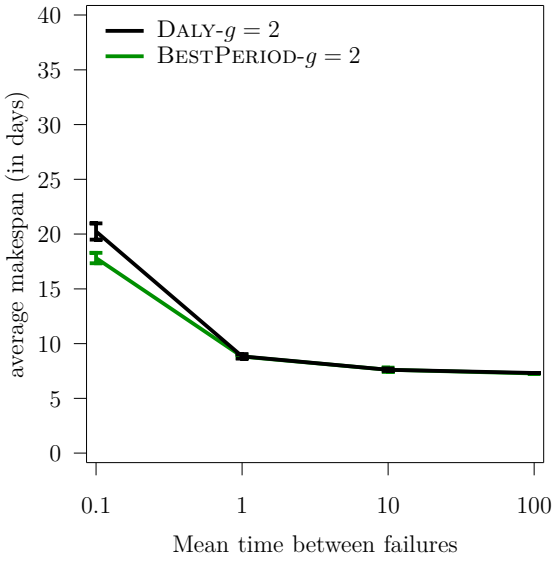


(b) Weibull failure distribution ($k = 0.7$)

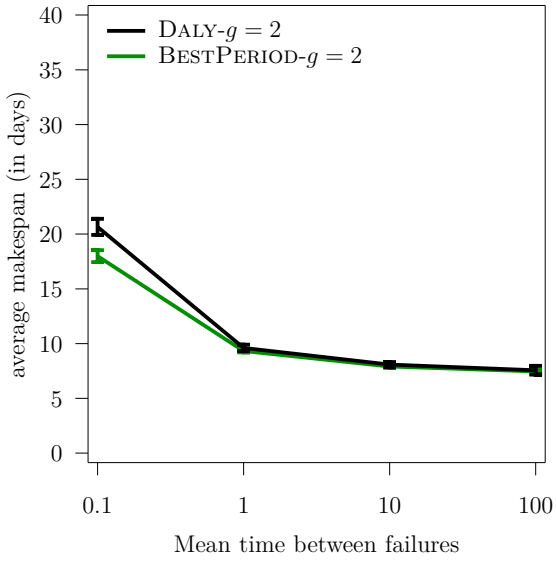


(b) Weibull failure distribution ($k = 0.5$)

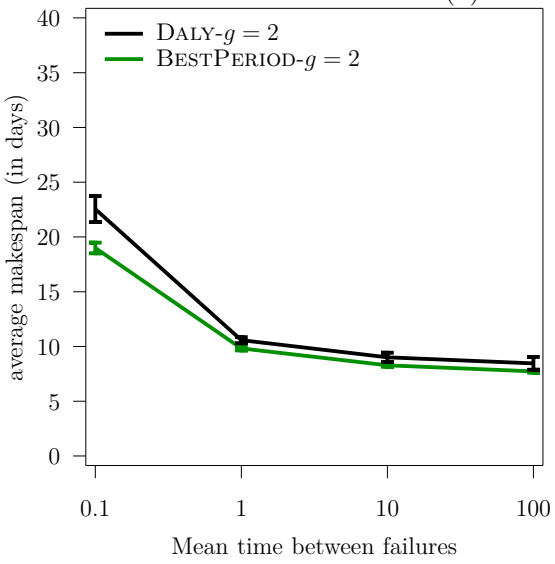
Figure 7. Average makespan vs. processor MTBF using group replication ($g = 2$) and using Generic parallel jobs: $\mathcal{W}(q) = \frac{\mathcal{W}}{q} + 10^{-6}\ \mathcal{W}$, showing that Daly's approximation can be suboptimal.



(a) Exponential failure distribution

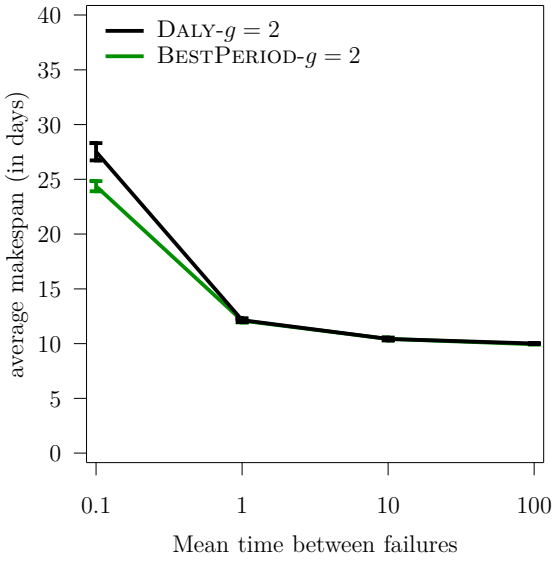


(b) Weibull failure distribution ($k = 0.7$)

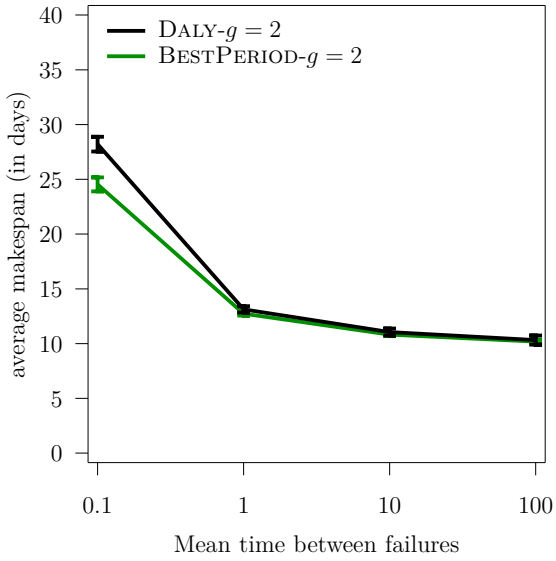


(b) Weibull failure distribution ($k = 0.5$)

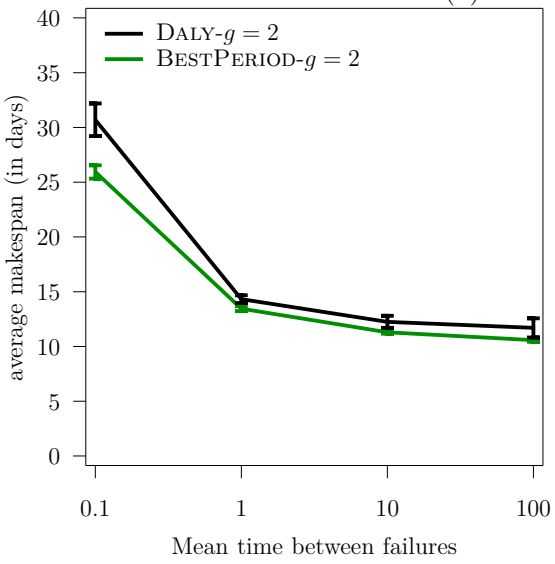
Figure 8. Average makespan vs. processor MTBF using group replication ($g = 2$) and using Numerical kernels model: $\mathcal{W}(q) = \frac{\mathcal{W}}{q} + 0.1\,\frac{\mathcal{W}^{2/3}}{\sqrt{q}}$, showing that Daly's approximation can be suboptimal.



(a) Exponential failure distribution

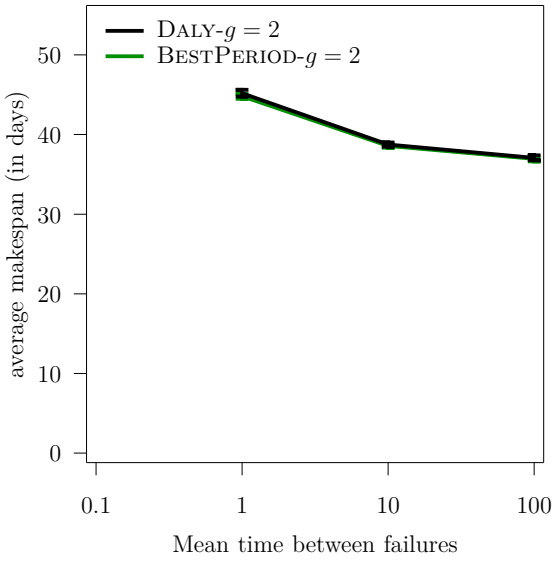


(b) Weibull failure distribution ($k = 0.7$)

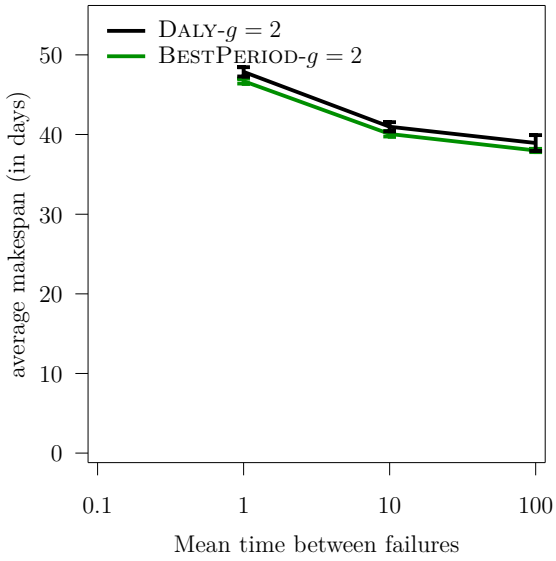


(b) Weibull failure distribution ($k = 0.5$)

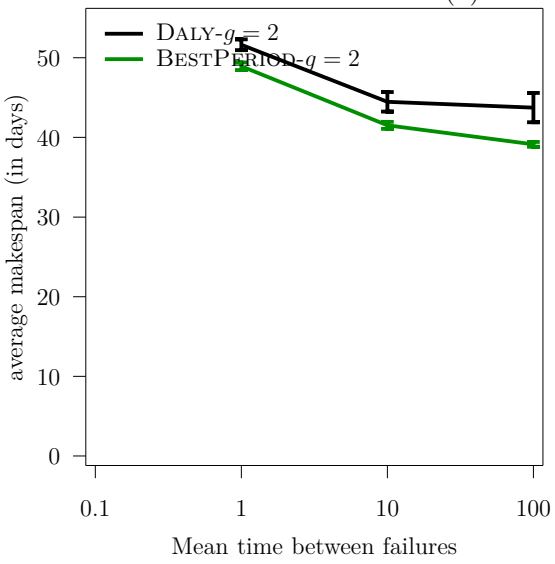
Figure 9. Average makespan vs. processor MTBF using group replication ($g = 2$) and uusing Numerical kernels model: $\mathcal{W}(q) = \frac{\mathcal{W}}{q} + \frac{\mathcal{W}^{2/3}}{\sqrt{q}}$, showing that Daly's approximation can be suboptimal.



(a) Exponential failure distribution

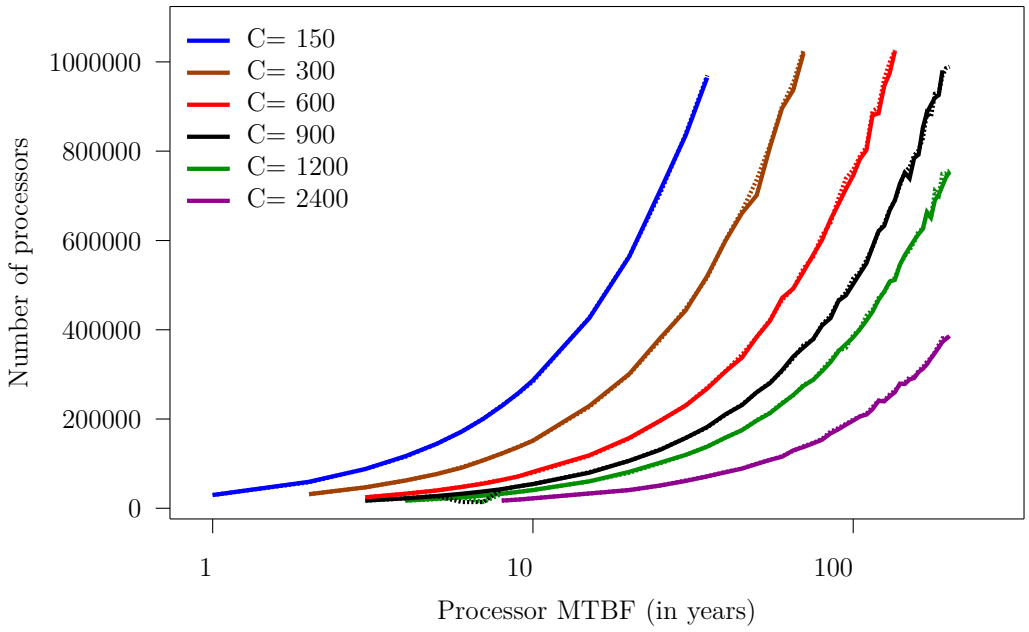


(b) Weibull failure distribution ($k = 0.7$)

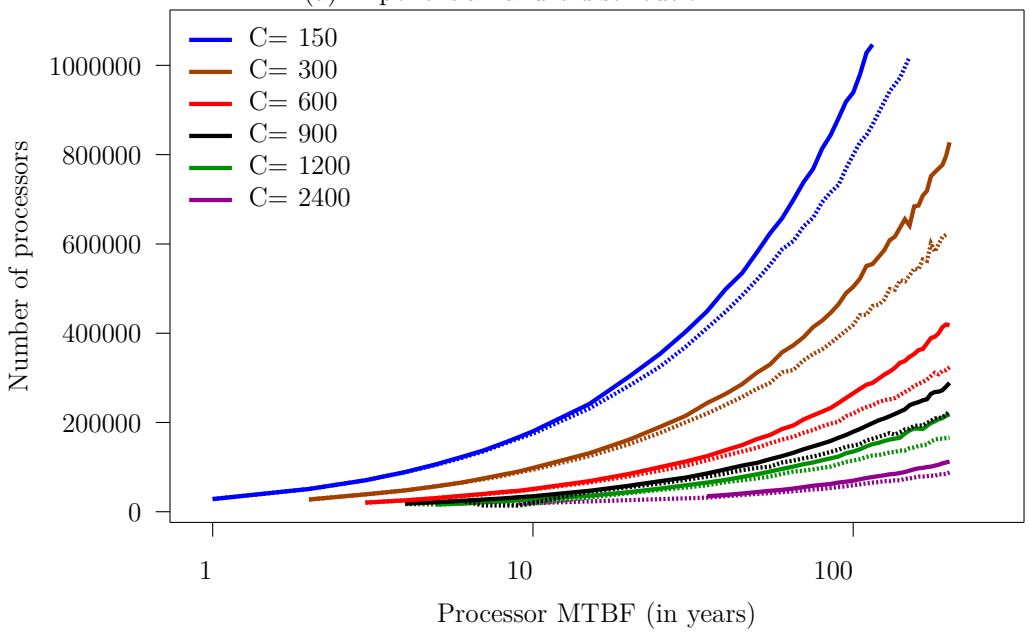


(b) Weibull failure distribution ($k = 0.5$)

Figure 10. Average makespan vs. processor MTBF using group replication ($g = 2$) and using Numerical kernels model: $\mathcal{W}(q) = \frac{\mathcal{W}}{q} + 10\,\frac{\mathcal{W}^{2/3}}{\sqrt{q}}$, showing that Daly's approximation can be suboptimal.



(a) Exponential failure distribution.



(b) Weibull failure distribution with $k = 0.70$