Identifying the Right Replication Level to Detect and Correct Silent Errors at Scale

Anne Benoit
LIP, ENS Lyon, France
anne.benoit@ens-lyon.fr

Aurélien Cavelan
LIP, ENS Lyon, France
aurelien.cavelan@ens-lyon.fr

Franck Cappello
Argonne National Laboratory, USA
cappello@mcs.anl.gov

Padma Raghavan
Vanderbilt University, USA
padma.raghavan@vanderbilt.edu

Yves Robert
LIP, ENS Lyon, France
Univ. Tenn. Knoxville, USA
yves.robert@inria.fr

Hongyang Sun
Vanderbilt University, USA
hongyang.sun@vanderbilt.edu

ABSTRACT

This paper provides a model and an analytical study of replication as a technique to detect and correct silent errors. Although other detection techniques exist for HPC applications, based on algorithms (ABFT), invariant preservation or data analytics, replication remains the most transparent and least intrusive technique. We explore the right level (duplication, triplication or more) of replication needed to efficiently detect and correct silent errors. Replication is combined with checkpointing and comes with two flavors: *process replication* and *group replication*. Process replication applies to message-passing applications with communicating processes. Each process is replicated, and the platform is composed of process pairs, or triplets. Group replication applies to black-box applications, whose parallel execution is replicated several times. The platform is partitioned into two halves (or three thirds). In both scenarios, results are compared before each checkpoint, which is taken only when both results (duplication) or two out of three results (triplication) coincide. If not, one or more silent errors have been detected, and the application rolls back to the last checkpoint. We provide a detailed analytical study of both scenarios, with formulas to decide, for each scenario, the optimal parameters as a function of the error rate, checkpoint cost, and platform size. We also report a set of extensive simulation results that corroborates the analytical model.

ACM Reference format:
Anne Benoit, Aurélien Cavelan, Franck Cappello, Padma Raghavan, Yves Robert, and Hongyang Sun. 2017. Identifying the Right Replication Level to Detect and Correct Silent Errors at Scale. In *Proceedings of FTXS'17, Washington, DC, USA, June 26, 2017,* 8 pages.
DOI: http://dx.doi.org/10.1145/3086157.3086162

1 INTRODUCTION

Triple Modular Redundancy, or TMR [23], is the standard fault-tolerance approach for critical systems, such as embedded or aeronautical devices [1]. With TMR, computations are executed three times, and a majority voting is conducted to select the correct result out of the three available ones. Indeed, if two or more results agree, they are declared correct, because the probability of two or more errors leading to the same wrong result is assumed so low that it can be ignored. While triplication seems very expensive in terms of resources, anybody sitting in a plane would heartily agree that it is worth the price.

On the contrary, duplication, let alone triplication, has a bad reputation in the High Performance Computing (HPC) community. Who would be ready to waste half or two-thirds of precious computing resources? However, despite its high cost, several authors have been advocating the use of duplication in HPC in the recent years [15, 17, 28, 37]. In a nutshell, this is because platform sizes have become so large that fail-stop errors are likely to strike at a high rate during application execution. More precisely, the MTBF (Mean Time Between Failures) μ_P of the platform decreases linearly with the number of processors P, since $\mu_P = \frac{\mu_{\text{ind}}}{P}$, where μ_{ind} is the MTBF of each individual component (see Proposition 1.2 in [20]). Take $\mu_{\text{ind}} = 10$ years as an example. If $P = 10^5$ then $\mu_P \approx 50$ minutes and if $P = 10^6$ then $\mu_P \approx 5$ minutes: from the point of view of fault-tolerance, scale is the enemy. Given any value of μ_{ind}, there is a threshold value for the number of processors above which platform throughput will decrease [14, 17, 27, 29]: the platform MTBF becomes so small that the applications experience too many failures, hence too many recoveries and re-execution delays, to progress efficiently. All this explains why duplication has been considered for HPC applications despite its cost. The authors in [17] propose *process replication* by which each process in a parallel MPI (Message Passing Interface) application is duplicated on multiple physical processors while maintaining synchronous execution of the replicas. This approach is effective in terms of fault-tolerance because the MTBF of a set of two replicas (which is the average delay for failures to strike *both* processors in the replica set) is much larger than the MTBF of a single processor.

Process replication may not always be a feasible option. Process replication features must be provided by the application. Some prototype MPI implementations [4, 17, 18] are convincing proofs of concept and do provide such capabilities. However, many other programming frameworks (not only MPI-like libraries and runtimes, but also concurrent objects, distributed components, workflows, PGAS environments, algorithmic skeletons) do not provide an equivalent to transparent process replication for the purpose of fault-tolerance, and enhancing them with transparent replication may be non-trivial. When transparent replication is not (yet) provided by the environment or runtime system, one solution could

FTXS'17, Washington, DC, USA

DOI: http://dx.doi.org/10.1145/3086157.3086162

be to implement it explicitly within the application, but this is a labor-intensive process especially for legacy applications. Another approach introduced in [8] is *group replication*, a technique that can be used whenever process replication is not available. Group replication is agnostic to the parallel programming model, and thus views the application as an unmodified black box. The only requirement is that the application can be started from a saved checkpoint file. Group replication consists in executing multiple application instances concurrently. For example, two distinct P-process application instances could be executed on a $2P$-processor platform. At first glance, it may seem paradoxical that better performance can be achieved by using group duplication. After all, in the above example, 50% of the platform is "wasted" to perform redundant computation. The key point here is that each application instance runs at a smaller scale. As a result, each instance can use lower checkpointing frequency, and can thus have better parallel efficiency in the presence of faults, when compared to a single application instance running at full scale. In some cases, the application makespan can then be comparable to, or even shorter than that obtained when running a single application instance. In the end, the cost of wasting processor power for redundant computation can be offset by the benefit of reduced checkpointing frequency. Furthermore, in group replication, once an instance saves a checkpoint, the other instance can use this checkpoint immediately to "jump ahead" in its execution. Hence, group replication is more efficient than the mere independent execution of several instances: each time one instance successfully completes a given "chunk of work", all the other instances immediately benefit from this success. To implement group replication, the runtime system needs to perform the typical operations needed for system-assisted checkpoint/restart: determining checkpointing frequencies for each application instance, causing checkpoints to be saved, detecting application failures, and restarting an application instance from a saved checkpoint after a failure. The only additional feature is that the system must be able to stop an instance and let it resume execution from a checkpoint file produced by another instance as soon as it is available.

Process or group replication has been mainly proposed in HPC to cope with fail-stop errors. However, another challenge is represented by silent errors, or silent data corruptions, whose threat can no longer be ignored [24, 26, 38]. There are several causes of silent errors, such as cosmic radiation, packaging pollution, among others. Silent errors can strike the cache and memory (bit flips) as well as CPU operations; in the latter case they resemble floating-point errors due to improper rounding, but have a dramatically larger impact because any bit of the result, not only low-order mantissa bits, can be corrupted. In contrast to a fail-stop error whose detection is immediate, a silent error is identified only when the corrupted data leads to an unusual application behavior. Such detection latency raises a new challenge: if the error struck before the last checkpoint, and is detected after that checkpoint, then the checkpoint is corrupted and cannot be used for rollback. To distinguish from fail-stop failures, we use MTBE (Mean Time Between Errors) instead of MTBF to characterize the rate of silent errors.

To address the problem of silent errors, many application-specific detectors, or verification mechanisms, have been proposed (see Section 2 for a survey). It is not clear, however, whether a special-purpose detector can be designed for each scientific application. In addition, application-specific verification mechanisms only protect from some types of error sources, and fail to provide accurate and efficient detection of all silent errors. In fact, providing such detectors for scientific applications has been identified as one of the hardest challenges[1] towards extreme-scale computing [6, 7].

Altogether, silent errors call for revisiting replication in the framework of scientific application executing on large-scale HPC platforms. Because replication is now applied at the process level, scale becomes an even harder-to-fight enemy. Existing processor count ranges to about 10^5 on the K-computer and TaihuLight systems. The number of processors could increase further to 10^6 (hence 10^6 or more processes) on future Exascale platforms, with billions of threads [12]. In addition, the probability of several errors striking during an execution can get significant, depending upon whether or not circuit manufacturers increase significantly the protection of the logic, latch/flip-flops and static arrays in the processor. In a recent paper [31], the authors consider that with significant more protection (more hardware, more power consumption), the FIT[2] rate for undetected errors on a processor circuit could be maintained to around 20. But without additional protection compared to the current situation, the FIT rate for undetected errors could be as high as 5,000 (or 1 error every 200,000 hours). Combining 10 million devices with this FIT rate would result in a silent error on the system every 72 seconds.

This work aims at providing a quantitative assessment of the potential of replication to mitigate such a threat. Specifically, the main contributions of this work are:

- an analytical model to study the performance of all replication scenarios against silent errors, namely, duplication, triplication, or more for process and group replications;
- closed-form formulas that give the optimal checkpointing period and optimal process number as a function of the error rate, checkpoint cost, and platform size;
- a set of simulation results that corroborate the analytical model.

The rest of the paper is organized as follows. Section 2 surveys the related work. We introduce the performance model in Section 3, and derive a general execution time formula in Section 4. The analysis for process replication is presented in Section 5, followed by the analysis for group replication in Section 6. Section 7 is devoted to the simulation results. Finally, we provide concluding remarks and directions for future work in Section 8.

2 RELATED WORK

Replication for fail-stop errors. Checkpointing policies have been widely studied. We refer to [20] for a survey of various protocols and the derivation of the Young's and Daly's formula [10, 35] for the optimal checkpointing periods. Recent advances include multi-level approaches, or the use of SSD or NVRAM as secondary storage [7]. Combining replication with checkpointing has been proposed in [15, 29, 37] for HPC platforms, and in [22, 34] for grid computing.

The use of redundant MPI processes is analyzed in [9, 16, 17]. In particular, the work by Ferreira et al. [17] has studied the use

[1] More generally, trustworthy computing, which aims at guaranteeing the correctness of the results of a long-lasting computation on a large-scale supercomputer, has received considerable attention recently [5].

[2] The Failures in Time (FIT) rate of a device is the number of failures that can be expected in one billion (10^9) device-hours of operation.

of process replication for MPI applications, using two replicas per MPI process. They provide a theoretical analysis of parallel efficiency, an MPI implementation that supports transparent process replication (including failure detection, consistent message ordering among replicas, etc.), and a set of experimental and simulation results. Partial redundancy is studied in [13, 32] (in combination with coordinated checkpointing) to decrease the overhead of full replication. Adaptive redundancy is introduced in [19], where a subset of processes is dynamically selected for replication.

Thread-level replication has been investigated in [36]. This work targets process-level replication, so as to detect (and correct) silent errors striking in all communication-related operations.

Ni et al. [25] introduce duplication to cope with both fail-stop and silent errors. Their pioneering paper contains many interesting results but differs from this work as follows: (i) they limit themselves to perfectly parallel applications while we investigate speedup profiles that obey Amdahl's law; (ii) they do not investigate triplication; and (iii) they compute an upper bound on the optimal period and do not determine optimal processor counts.

Finally, we note that RedMPI [18] originally deals with silent errors, and that Subasi et al. [33] introduce task replication to detect and correct silent errors for workflow applications (while we deal with general applications in this paper).

Silent error detection and correction. Application-specific information enables ad-hoc solutions, which dramatically decrease the cost of error detection. Algorithm-based fault tolerance (ABFT) [3, 21, 30] is a well-known technique, which uses checksums to detect up to a certain number of errors in linear algebra kernels. Unfortunately, ABFT can only protect datasets in linear algebra kernels, and it must be implemented for each different kernel, which incurs a large amount of work for large HPC applications.

While many application-specific detectors are proposed (see the companion research report [2] for more related work), our approach is agnostic of the application characteristics. The only information is whether we can use process replication. If not, we see the application as a black box and can use only group replication.

Table 1: List of notations.

Parameters	
T	Length (or period) of a pattern
P	Number of processes allocated to an application
n	Number of (process or group) replicas
$S(P)$	Speedup function of an application
$H(P) = \frac{1}{S(P)}$	Error-free execution overhead
$\mathbb{E}_n(T, P)$	Expected execution time of a pattern
$\mathbb{H}_n(T, P)$	Expected execution overhead of a pattern
$\mathbb{S}_n(T, P)$	Expected speedup function of a pattern
$\lambda = \frac{1}{\mu_{\text{ind}}}$	Silent error rate of an individual process
$\mathbb{P}_n(T, P)$	Silent error probability of a pattern
C	Checkpointing cost
R	Recovery cost
V	Verification cost (comparison of replicas)

3 MODEL

This section presents the analytical model for evaluating the performance of different replication scenarios. The model is classical, similar to those of the literature for replication [17], only with a different objective (quantifying replication for silent errors). Table 1 summarizes the main notations used in the paper. Let μ_{ind} denote the MTBE of an individual processor or process[3] of the system, so $\lambda = \frac{1}{\mu_{\text{ind}}}$ is the silent error rate of the processor.

The error rate for a collection of P processors is then given by $\lambda_P = \frac{1}{\mu_P} = \frac{P}{\mu_{\text{ind}}} = \lambda P$ [20]. Assuming that the error arrivals follow *Exponential* distribution, the probability that a computation hit by a silent error during time T on P processes is given by:

$$\mathbb{P}(T, P) = 1 - e^{-\lambda P T}.$$

Consider long-lasting HPC applications that execute for hours or even days on a large-scale platform. Resilience is enforced by the combined use of replication and periodic checkpointing. Before each checkpoint, the results of different replicas are compared. Only when both results (for duplication) or two out of three results (for triplication) coincide[4], in which case a *consensus* is said to be reached, the checkpoint is taken. Otherwise, silent errors are assumed to have been detected, and they cannot be corrected through consensus. The application then rolls back to the last checkpoint. There are two different types of replications:

(1) *Process replication*: Each process of the application is replicated, and the results of different processes are independently compared. A rollback is needed when at least one process has failed to reach a consensus;

(2) *Group replication*: The entire application (as a black box) is replicated, and the results of all replicas (as a whole) are compared. A rollback is needed when these group replicas fail to reach a consensus.

The computational chunk between two checkpoints is called a *periodic pattern*. For a replication scenario with n replicas, the objective is to minimize the expected total execution time (or makespan) of an application by finding the optimal pattern parameters:

- T: length (or period) of the pattern;
- P: number of processes allocated to the application.

Indeed, for long-lasting applications, it suffices to focus on just one pattern, since the pattern repeats itself over time. To see this, let W_{total} denote the total amount of work of the application and suppose the application has a speedup function $S(P)$ when executed on P processors. In this paper, we focus on a speedup function that obeys Amdahl's law[5]:

$$S(P) = \frac{1}{\alpha + \frac{1-\alpha}{P}}, \tag{1}$$

where $\alpha \in [0, 1]$ denotes the sequential fraction of the application that cannot be parallelized. For convenience, we also define $H(P) = \frac{1}{S(P)}$ to be the execution overhead. For a pattern of length T and run by P processes, the amount of work done in a pattern is therefore $W_{\text{pattern}} = T \cdot S(P)$, and the total number of patterns in the application can be approximated as $m = \frac{W_{\text{total}}}{W_{\text{pattern}}} = \frac{W_{\text{total}}}{T \cdot S(P)} = \frac{W_{\text{total}}}{T} H(P)$. Now, let $\mathbb{E}_n(T, P)$ denote the expected execution time of the pattern with n replicas in either replication scenario. Define

[3] We assume that each process is executed by a dedicated processor, hence use "processor" and "process" interchangeably. We also use MTBE instead of MTBF to emphasize that we deal with (silent) errors, not failures.

[4] For $n > 3$ replicas, the results of k replicas should coincide, where $2 \le k < n$ is a design parameter set by the system to control the level of reliability. $k = \lfloor \frac{n}{2} \rfloor + 1$ is a widely-used choice (majority voting).

[5] The model is generally applicable to other speedup functions as well.

$\mathbb{H}_n(T,P) = \frac{\mathbb{E}_n(T,P)}{T} H(P)$ to be the expected execution overhead of the pattern, and $\mathbb{S}_n(T,P) = \frac{1}{\mathbb{H}_n(T,P)}$ the expected speedup. The expected makespan of the application can then be written as $\mathbb{E}_{\text{total}} \approx \mathbb{E}_n(T,P)m = \mathbb{E}_n(T,P)\frac{W_{\text{total}}}{T} H(P) = \mathbb{H}_n(T,P) \cdot W_{\text{total}} = \frac{W_{\text{total}}}{\mathbb{S}_n(T,P)}$. This shows that the optimal expected makespan can be achieved by minimizing the expected execution overhead of a pattern, or equivalently, maximizing the expected speedup.

Now, we describe a model for the costs of checkpoint, recovery and consensus verification. First, the checkpoint cost clearly depends on the protocol and storage type. Note that only the result of one replica needs to be checkpointed, so the cost does not increase with the number of replicas. To save the application's memory footprint M to the storage system using P processes, we envision the following two scenarios:

- $C = \frac{M}{\tau_{io}}$: In this case, checkpoints are being written to the remote storage system, whose bandwidth is the I/O bottleneck. Here, τ_{io} is the remote I/O bandwidth.
- $C = \frac{M}{\tau_{net} P}$: This case corresponds to in-memory checkpoints, where each process stores $\frac{M}{P}$ data locally (e.g., on SSDs). Here, τ_{net} is the process network bandwidth.

The recovery cost is assumed to be the same as the checkpointing cost, i.e., $R = C$, as it involves the same I/O operations. This is a common assumption [24], although practical recovery cost can be somewhat smaller than the checkpoint cost [11]. Finally, verifying consensus is performed by communicating and comparing $\frac{M}{P}$ data stored on each process, which can be executed concurrently by all process pairs (or triplets). Hence, the verification cost satisfies $V = O(\frac{M}{P})$. Overall, we use the following general expression to account for the combined cost of verification and checkpoint/recovery:

$$V + C = c + \frac{d}{P}\,, \tag{2}$$

where c and d are constants that depend on the application memory footprint, checkpointing protocol, network or I/O bandwidth, etc. Equation (2) is convenient in terms of analysis as we will see in the subsequent sections. Here, $c = 0$ corresponds to the second checkpointing scenario discussed above.

4 EXPECTED EXECUTION TIME

In this section, we compute the expected execution time of a periodic pattern, which will be used in the next two sections to derive the optimal pattern parameters.

Theorem 4.1. *The expected time to execute a periodic pattern of length T using P processes and n replicas can be expressed as*

$$\mathbb{E}_n(T,P) = T + V + C + \frac{\mathbb{P}_n(T,P)}{1 - \mathbb{P}_n(T,P)}\,(T + V + R)\,, \tag{3}$$

where $\mathbb{P}_n(T,P)$ denotes the probability that the execution fails due to silent errors striking during the pattern and we have to roll back to the last checkpoint.

Proof. Since replicas are synchronized, we can generally express the expected execution time as follows:

$$\mathbb{E}_n(T,P) = T + V + \mathbb{P}_n(T,P)\big(R + \mathbb{E}_n(T,P)\big) + \big(1 - \mathbb{P}_n(T,P)\big)C\,. \tag{4}$$

First, the pattern of length T is executed followed by the verification (through comparison and/or voting), which incurs cost V. With probability $\mathbb{P}_n(T,P)$, the pattern fails due to silent errors. In this case, we need to re-execute the pattern after performing a recovery from the last checkpoint with cost R. Otherwise, with probability $1 - \mathbb{P}_n(T,P)$, the execution succeeds and the checkpoint with cost C is taken at the end of the pattern. Now, solving for $\mathbb{E}_n(T,P)$ from Equation (4), we can obtain the expected execution time of the pattern as shown in Equation (3). □

Remarks. Theorem 4.1 is applicable to both process replication and group replication. The only difference lies in the computation of the failure probability $\mathbb{P}_n(T,P)$, which depends not only on the replication scenario but also on the number of replicas n.

5 PROCESS REPLICATION

In this section, we consider process replication. We first derive the optimal computing patterns when each process of the application is duplicated (Section 5.1) and triplicated (Section 5.2), respectively. We also generalize the results to an arbitrary but constant number of replications per process under a general process replication framework. Proofs are available in the companion research report [2].

5.1 Process duplication

We start with process duplication, that is, each process has two replicas. The following lemma shows the failure probability of a given computing pattern in this case.

Lemma 5.1. *Using process duplication, the failure probability of a computing pattern of length T and with P processes is given by*

$$\mathbb{P}_2^{\text{prc}}(T,P) = 1 - e^{-2\lambda T P}\,. \tag{5}$$

Using the failure probability in Lemma 5.1, we derive the optimal computing pattern for process duplication as shown in the following theorem. Recall that the application speedup follows Amdahl's law as shown in Equation (1) and the cost of verification and checkpoint is modeled by Equation (2).

Theorem 5.2. *A first-order approximation to the optimal number of processes for an application with 2 replicas per process is given by*

$$P_{\text{opt}} = \min\left\{\frac{Q}{2}, \left(\frac{1}{2}\left(\frac{1-\alpha}{\alpha}\right)^2 \frac{1}{c\lambda}\right)^{\frac{1}{3}}\right\}\,, \tag{6}$$

where Q denotes the total number of available processes in the system. The associated optimal checkpointing period and the expected speedup function of the application are

$$T_{\text{opt}}(P_{\text{opt}}) = \left(\frac{V + C}{2\lambda P_{\text{opt}}}\right)^{\frac{1}{2}}, \quad \mathbb{S}_2^{\text{prc}}(P_{\text{opt}}) = \frac{S(P_{\text{opt}})}{1 + 2\big(2\lambda(V+C)P_{\text{opt}}\big)^{\frac{1}{2}}} \tag{7}$$

5.2 Process triplication and general replication

Now, we consider process triplication, that is, each process has three replicas. This is the smallest number of replicas that allows an application to recover from silent errors through majority voting instead of rolling back to the last checkpoint.

Lemma 5.3. *Using process triplication, the failure probability of a computing pattern of length T and with P processes is given by*

$$\mathbb{P}_3^{\text{prc}}(T,P) = 1 - \left(3e^{-2\lambda T} - 2e^{-3\lambda T}\right)^P\,. \tag{8}$$

Theorem 5.4. *A first-order approximation to the optimal number of processes for an application with 3 replicas per process is given by*

$$P_{\text{opt}} = \min\left\{\frac{Q}{3}, \left(\frac{4}{3}\left(\frac{1-\alpha}{\alpha}\right)^3\left(\frac{1}{c\lambda}\right)^2\right)^{\frac{1}{4}}\right\}, \tag{9}$$

where Q denotes the total number of available processes in the system. The associated optimal checkpointing period and the expected speedup function of the application are

$$T_{\text{opt}}(P_{\text{opt}}) = \left(\frac{V+C}{6\lambda^2 P_{\text{opt}}}\right)^{\frac{1}{3}}, \quad \mathbb{S}_3^{\text{prc}}(P_{\text{opt}}) = \frac{S(P_{\text{opt}})}{1+3\left(\frac{3}{4}\left(\lambda(V+C)\right)^2 P_{\text{opt}}\right)^{\frac{1}{3}}} \tag{10}$$

Compared with duplication, the ability to correct errors in triplication allows checkpoints to be taken less frequently. In terms of expected speedup, triplication suffers from a smaller error-free speedup ($\frac{Q}{3}$ vs $\frac{Q}{2}$ for perfectly parallel applications, i.e., $\alpha = 0$) due to the use of fewer concurrent processes to perform useful work, but also has a smaller error-induced denominator, especially on platforms with a large number of processes Q. In Section 7, we conduct simulations to evaluate this trade-off and compare the performance of duplication and triplication.

Formulas for a general process replication framework with n replica groups, out of which k of them must agree to avoid a rollback, are given in the extended version [2]. Results apply for any k, including the two natural choices $k = 2$ and $k = \lfloor \frac{n}{2} \rfloor + 1$.

6 GROUP REPLICATION

In this section, we consider group replication. Recall that, unlike process replication where the results of each process from different replicas are independently compared, group replication compares the outputs of the different groups viewed as independent black-box applications. First, we make the following technical observation, which establishes the relationship between the two replication mechanisms from the resilience point of view.

Observation 1. *Running an application using group replication with n replicas, where each replica has P processes and each process has error rate λ, has the same failure probability as running it using process replication with one process, which has error rate λP and is replicated n times.*

The above observation allows us to compute the failure probability for group replication by deriving from the corresponding formulas under process replication while setting $P = 1$ and $\lambda = \lambda P$. The rest of this section shows the results for duplication, triplication, and a general group replication framework. Proofs are similar to those in process replication, and are hence omitted.

6.1 Group duplication

By applying Observation 1 on Lemma 5.1, we can get the failure probability for a given pattern under group duplication as follows.

Lemma 6.1. *Using group duplication, the failure probability of a computing pattern of length T and with P processes is given by*

$$\mathbb{P}_2^{\text{grp}}(T,P) = 1 - e^{-2\lambda TP}. \tag{11}$$

This leads us to the following theorem on the optimal pattern:

Theorem 6.2. *A first-order approximation to the optimal number of processes for an application with 2 replica groups is given by*

$$P_{\text{opt}} = \min\left\{\frac{Q}{2}, \left(\frac{1}{2}\left(\frac{1-\alpha}{\alpha}\right)^2\frac{1}{c\lambda}\right)^{\frac{1}{3}}\right\}, \tag{12}$$

where Q denotes the total number of available processes in the system. The associated optimal checkpointing period and the expected speedup function of the application are

$$T_{\text{opt}}(P_{\text{opt}}) = \left(\frac{V+C}{2\lambda P_{\text{opt}}}\right)^{\frac{1}{2}}, \quad \mathbb{S}_2^{\text{grp}}(P_{\text{opt}}) = \frac{S(P_{\text{opt}})}{1+2\left(2\lambda(V+C)P_{\text{opt}}\right)^{\frac{1}{2}}} \tag{13}$$

Remarks. The result is identical to that of process duplication. Indeed, in both cases, a single silent error that strikes any of the running processes will cause the whole application to fail.

6.2 Group triplication and general replication

Again, applying Observation 1 on Lemma 5.3, we can get the failure probability for a given pattern under group triplication, and then determine the optimal pattern.

Lemma 6.3. *Using group triplication, the failure probability of a computing pattern of length T and with P processes is given by*

$$\mathbb{P}_3^{\text{grp}}(T,P) = 1 - \left(3e^{-2\lambda TP} - 2e^{-3\lambda TP}\right). \tag{14}$$

Theorem 6.4. *A first-order approximation to the optimal number of processes for an application with 3 replica groups is given by*

$$P_{\text{opt}} = \min\left\{\frac{Q}{3}, \left(\frac{1}{6}\left(\frac{1-\alpha}{\alpha}\right)^3\left(\frac{1}{c\lambda}\right)^2\right)^{\frac{1}{5}}\right\}, \tag{15}$$

where Q denotes the total number of available processes in the system. The associated optimal checkpointing period and the expected execution overhead are

$$T_{\text{opt}}(P_{\text{opt}}) = \left(\frac{V+C}{6(\lambda P_{\text{opt}})^2}\right)^{\frac{1}{3}}, \quad \mathbb{S}_3^{\text{grp}}(P_{\text{opt}}) = \frac{S(P_{\text{opt}})}{1+3\left(\frac{3}{4}\left(\lambda(V+C)P_{\text{opt}}\right)^2\right)^{\frac{1}{3}}} \tag{16}$$

Remarks. Compared to the result of process triplication (Theorem 5.4) and under the same condition (e.g., $\alpha = 0$ so both scenarios allocate the same number of $P_{\text{opt}} = \frac{Q}{3}$ processes to each replica), group triplication needs to place checkpoints more frequently yet enjoys a smaller execution speedup. This provides a theoretical explanation to the common understanding that group replication in general cannot recover from some error combinations that its process counterpart is capable of, making the latter a superior replication mechanism provided that it can be feasibly implemented.

Formulas for a general group replication framework with n replica groups, out of which k of them must agree to avoid a rollback, are given in the extended version [2].

7 SIMULATIONS

We conduct a set of simulations whose goal is twofold: (i) validate the accuracy of the theoretical study; and (ii) evaluate the efficiency of both process and group replication under different scenarios at extreme scale. The simulator is publicly available at

http://perso.ens-lyon.fr/aurelien.cavelan/replication.zip so that interested readers can instantiate their preferred scenarios and repeat the same simulations for reproducibility purpose.

7.1 Simulation setup

The simulator has been designed to simulate each process individually, and each process has its own error trace. A simulation works as follows: we feed the simulator with the model parameters μ_{ind}, Q, C, V, R, and α, and we compute the associated optimal number of processes P_{opt} and the optimal checkpointing period $T_{\text{opt}}(P_{\text{opt}})$ using the corresponding model equations. For each run, the simulator outputs the efficiency, defined as $\frac{\mathbb{S}(P_{\text{opt}})}{Q}$, as well as the average number of errors and the average number of recoveries per million CPU hours of work. Then, for each of the following scenarios, we compare the simulated efficiency to the theoretical value, obtained using the model equations for $\mathbb{S}(P_{\text{opt}})$.

As suggested by Observation 1, process and group replications with $n = 2$ lead to identical results, so we have merged them together. Intuitively, this is explained by the fact that in both cases, a single error cannot be corrected, and requires to recover from the last checkpoint. In the following, we set the cost of recovery to be the same as the checkpoint cost (as discussed in Section 3), and we set the cost $V + C$ according the values of c and d as in Equation (2). We consider different Mean Time Between Errors (MTBE), ranging from 10^6 seconds ($\approx$ 11 days) down to 10^2 seconds ($<$ 2 minutes) for $Q = 10^6$ processes, matching the numbers in [31].

7.2 Impacts of $MTBE$ and checkpoint cost

Figure 1 presents the impact of the $MTBE$ on the efficiency of both duplication and triplication for three different checkpoint costs, but using the same value $\alpha = 10^{-6}$ for the sequential fraction of the application (see next section for the impact of varying α). The first row of plots is obtained with a cost of 30 minutes (i.e. $c = 1,800, d = 0$), the second row with a cost of 60 seconds (i.e. $c = 60, d = 0$), and the last row with $c = 0, d = 10^7$, which correspond to a checkpoint cost of 20 seconds for duplication with $\frac{Q}{2}$ processes and 30 seconds for triplication with $\frac{Q}{3}$ processes. In addition to the efficiency, we provide the average number of errors and recoveries per million hours of work, the optimal checkpointing period $T_{\text{opt}}(P_{\text{opt}})$ and the optimal number of processes P_{opt}.

Efficiency. First, we observe in the first column that the difference between the theoretical efficiency and the simulated efficiency remains small ($<$ 5% absolute difference), which shows the accuracy of the first-order approximation. Then, with very few errors ($MTBE = 10^6$), we observe that duplication is always better than triplication. This is as expected, since the maximum efficiency for duplication is 0.5 (assuming $\alpha = 0$ and no error), while the maximum efficiency for triplication is 0.33. However, as the $MTBE$ decreases, triplication becomes more attractive and eventually outperforms duplication. With a checkpoint cost of 30 minutes (first row), the $MTBE$ required is around 28 hours for process triplication to win and 20 hours for group triplication to win. With smaller checkpoint costs, such as 60 seconds (second row) and 30 seconds (third row), checkpoints can be more frequent and the $MTBE$ required for triplication to win is pushed down to a couple of hours and a couple of minutes, respectively.

Number of errors and recoveries. The second column presents the number of errors and the corresponding number of recoveries per million hours of work. The number of errors is always higher than the number of recoveries, because multiple errors can occur during a period (before the checkpoint, which is the point of detection), causing a single recovery. At $MTBE = 10^2$, almost half of the errors that occurred with duplication were actually hidden behind another error. Even more errors were hidden with group triplication, since one more error (in a different replica) is required to cause a recovery. Finally, (almost) all errors were hidden with process replication, which is able to handle many errors, as long as they strike in different processes.

Optimal checkpointing period. The third column shows the optimal length of the pattern. In order to cope with the increasing number of errors and recoveries, the length of the optimal period becomes smaller. Note that the length of the period for group triplication is comparable to that for duplication, around one day when $MTBE = 10^6$ down to a couple of minutes when $MTBE = 10^2$. However, the length of the pattern for process triplication is always higher by several orders of magnitude, from more than 10 days when $MTBE = 10^6$ down to a couple of hours when $MTBE = 10^2$.

Optimal number of processes. With $\alpha = 10^{-6}$, the application has ample parallelism, so the optimal number of processes to use is always $\frac{Q}{2} = 5 \cdot 10^5$ for duplication and $\frac{Q}{3} \approx 3.3 \cdot 10^5$ for triplication, except when $MTBE = 10^2$ and $c = 1,800$, where the optimal number of processes for duplication is $\approx 3 \cdot 10^5$ and the optimal number of processes for group triplication is $\approx 2 \cdot 10^5$.

7.3 Impact of sequential fraction (Amdahl)

Figure 2 presents two additional simulation results for $\alpha = 10^{-7}$ and $\alpha = 10^{-5}$. With a small fraction of sequential work (left plots), the efficiency is improved ($\approx$ 85% of the maximum efficiency for duplication and $\approx$ 95% for triplication at $MTBE = 10^6$), and both duplication and triplication use all processors available. On the contrary, with a higher sequential fraction of work (right plots), the efficiency drops ($<$ 20% of the maximum efficiency for duplication and $<$ 30% for triplication at $MTBE = 10^6$), and using more processes does not improve the efficiency and only contributes to increasing the number of errors. Therefore, these results suggest that even when using replication or triplication, there comes a point where it is no longer beneficial to use all processors available. In this example, when $MTBE = 10^2$, duplication and group triplication would use fewer than $2 \cdot 10^5$ processes (one fifth of the available resources). Process triplication, on the other hand, still utilizes all the resources and outperforms the other two schemes in terms of the efficiency across the whole range of system MTBE.

7.4 Impact of number of processes

Figure 3 shows the impact of the number of processes on the simulated efficiency of different replication scenarios. In addition, we also show (as big dots) the theoretical efficiency obtained with the optimal number of processes from Theorems 5.2, 5.4 and 6.4. By varying the number of processes, we find that the simulated optimum (that yields the best efficiency) matches our theoretical optimal number of processes closely. We can also see that process triplication scales very well with increasing number of processes

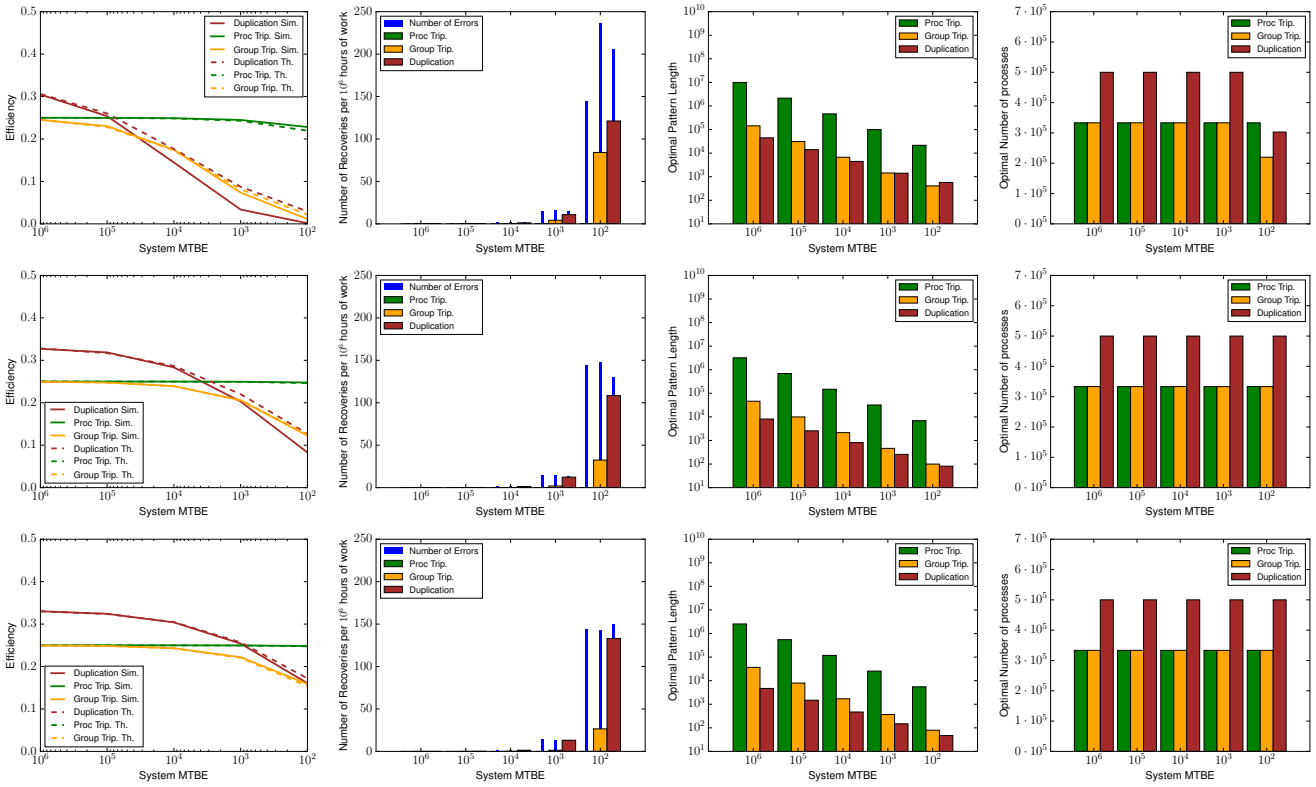

Figure 1: Impact of System MTBE on the efficiency with $c = 1,800, d = 0$ (top), $c = 60, d = 0$ (middle), $c = 0, d = 10^7$ (bottom) and $\alpha = 10^{-6}$.

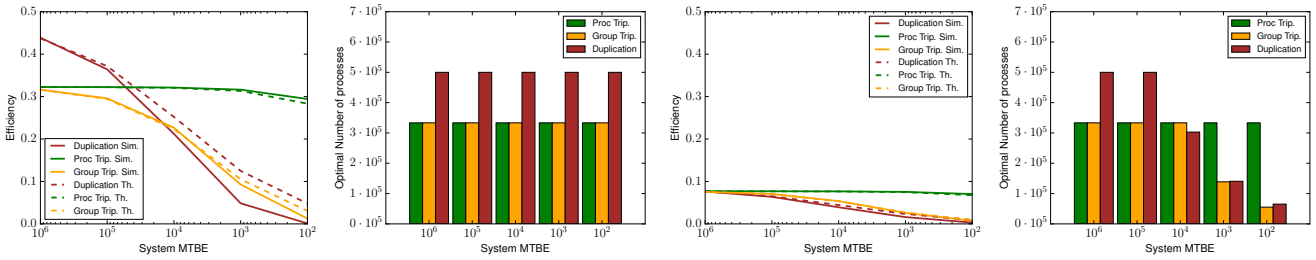

Figure 2: Impact of sequential fraction (in Amdahl's Law) on efficiency and optimal number of processes with $\alpha = 10^{-7}$ (left) and $\alpha = 10^{-5}$ (right).

up to $Q = 10^6$. As opposed to group triplication, which has to recover from a checkpoint if just two errors strike in two different replicas, process triplication benefits from having an additional process: from a resilience point of view, each replica acts as a buffer to handle one more error, and the probability that two errors strike two replicas of the same process decreases, thereby improving the efficiency.

7.5 Summary

Results suggest that duplication is more efficient than triplication for high *MTBE*s (e.g., 10^6 seconds). Process triplication, when available, is always more efficient for smaller *MTBE*s: its efficiency

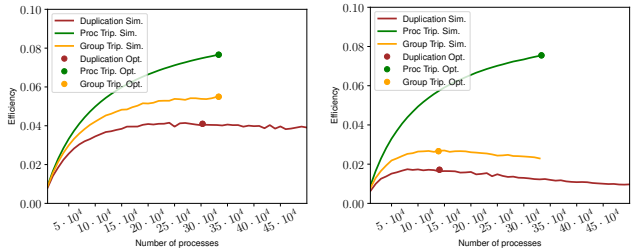

Figure 3: Impact of the number of processes on the efficiency with $MTBE = 10^4$ (left), $MTBE = 10^3$ (right), $Q = 10^6$, $c = 1n800, d = 0$, and $\alpha = 10^{-5}$.

remains stable despite increasing numbers of errors. If process triplication is not available, group triplication is slightly more efficient than duplication for small $MTBE$s, but the gain is marginal.

Furthermore, the impact of the sequential fraction α of the application (in Amdahl's Law) is twofold: (i) it limits the efficiency (e.g., $< 30\%$ of the maximum with $\alpha = 10^{-5}$ for both duplication and triplication); and (ii) it is a major factor in limiting the optimal number of processes (e.g., one fifth of the platform for duplication with $\alpha = 10^{-5}$ and $Q = 10^6$ at $MTBE = 10^2$).

8 CONCLUSION

Silent errors represent a major threat to the HPC community. In the absence of application-specific detectors, replication is the only solution. Unfortunately, it comes with a high cost: by definition, the efficiency is upper-bounded by 0.5 for duplication, and by 0.33 for triplication. Are these upper bounds likely to be achieved? If yes, it means that duplication should always be preferred to triplication. If not, it means that in some scenarios, the striking of errors is so frequent that duplication is not the right choice.

The major contribution of this paper is to provide an in-depth analysis of process and group duplication, and of process and group triplication. Given the replication scenario, and a set of application/platform parameters (speedup profile, total number or processors, process MTBE, checkpoint cost, etc.), we derive closed-form formulas for the optimal checkpointing interval, the optimal resource usage, and the overall speedup/efficiency of the approach. The results allow us to identify the right replication level to cope with silent errors.

A set of simulations demonstrate the accuracy of the model and analysis. Our simulator is made publicly available, so that interested readers can instantiate their preferred scenario. Altogether, this paper has laid the foundations for a better understanding of replication and its impact on silent errors while using HPC at scale.

Future work will be devoted to combining replication and checkpointing to mitigate both fail-stop failures and silent errors. Partial replication is another topic to explore: if the application comes as a workflow whose tasks are atomic components, one could assign different replication levels (duplication, triplication or more) to the different tasks, depending upon their criticality in terms of longest paths, number of successors, etc.

REFERENCES

[1] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl E. Landwehr. 2004. Basic Concepts and Taxonomy of Dependable and Secure Computing. *IEEE Trans. Dependable Sec. Comput.* 1, 1 (2004), 11–33.

[2] Anne Benoit, Aurélien Cavelan, Franck Cappello, Padma Raghavan, Yves Robert, and Hongyang Sun. 2017. *Identifying the right replication level to detect and correct silent errors at scale.* Research report RR-9047. INRIA.

[3] George Bosilca, Rémi Delmas, Jack Dongarra, and Julien Langou. 2009. Algorithm-based fault tolerance applied to high performance computing. *J. Parallel Distrib. Comput.* 69, 4 (2009), 410–416.

[4] Ron Brightwell, Kurt Ferreira, and Rolf Riesen. 2010. Transparent Redundant Computing with MPI. In *EuroMPI*. Springer.

[5] Franck Cappello, Emil M. Constantinescu, Paul D. Hovland, Tom Peterka, Carolyn Phillips, Marc Snir, and Stefan M. Wil. 2015. *Improving the trust in results of numerical simulations and scientific data analytics.* White paper MCS-TM-352. ANL.

[6] Franck Cappello, Al Geist, Bill Gropp, Laxmikant Kale, Bill Kramer, and Marc Snir. 2009. Toward Exascale Resilience. *Int. J. High Performance Computing Applications* 23, 4 (2009), 374–388.

[7] Franck Cappello, Al Geist, William Gropp, Sanjay Kale, Bill Kramer, and Marc Snir. 2014. Toward Exascale Resilience: 2014 update. *Supercomputing frontiers and innovations* 1, 1 (2014).

[8] Henri Casanova, Marin Bougeret, Yves Robert, Frédéric Vivien, and Dounia Zaidouni. 2014. Using group replication for resilience on exascale systems. *Int. Journal of High Performance Computing Applications* 28, 2 (2014), 210–224.

[9] Henri Casanova, Yves Robert, Frédéric Vivien, and Dounia Zaidouni. 2015. On the impact of process replication on executions of large-scale parallel applications with coordinated checkpointing. *Future Generation Comp. Syst.* 51 (2015), 7–19.

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

[11] Sheng Di, Mohamed Slim Bouguerra, Leonardo Bautista-Gomez, and Franck Cappello. 2014. Optimization of multi-level checkpoint model for large scale HPC applications. In *IPDPS*. IEEE.

[12] J. Dongarra and et al. 2011. The International Exascale Software Project Roadmap. *Int. J. High Perform. Comput. Appl.* 25, 1 (2011), 3–60.

[13] James Elliott, Kishor Kharbas, David Fiala, Frank Mueller, Kurt Ferreira, and Christian Engelmann. 2012. Combining partial redundancy and checkpointing for HPC. In *ICDCS*. IEEE.

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

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

[16] Christian Engelmann and Böhm Swen. 2011. Redundant execution of HPC applications with MR-MPI. In *PDCN*. IASTED.

[17] K. Ferreira, J. Stearley, J. H. III Laros, R. Oldfield, K. Pedretti, R. Brightwell, R. Riesen, P. G. Bridges, and D. Arnold. 2011. Evaluating the Viability of Process Replication Reliability for Exascale Systems. In *SC'11*. ACM.

[18] David Fiala, Frank Mueller, Christian Engelmann, Rolf Riesen, Kurt Ferreira, and Ron Brightwell. 2012. Detection and correction of silent data corruption for large-scale high-performance computing. In *SC*. ACM.

[19] Cijo George and Sathish S. Vadhiyar. 2012. ADFT: An Adaptive Framework for Fault Tolerance on Large Scale Systems using Application Malleability. *Procedia Computer Science* 9 (2012), 166 – 175.

[20] Thomas Hérault and Yves Robert (Eds.). 2015. *Fault-Tolerance Techniques for High-Performance Computing*. Springer Verlag.

[21] Kuang-Hua Huang and J. A. Abraham. 1984. Algorithm-Based Fault Tolerance for Matrix Operations. *IEEE Trans. Comput.* 33, 6 (1984), 518–528.

[22] Troy Leblanc, Rakhi Anand, Edgar Gabriel, and Jaspal Subhlok. 2009. VolpexMPI: An MPI Library for Execution of Parallel Applications on Volatile Nodes. In *16th European PVM/MPI Users' Group Meeting*. Springer-Verlag, 124–133.

[23] R. E. Lyons and W. Vanderkulk. 1962. The use of triple-modular redundancy to improve computer reliability. *IBM J. Res. Dev.* 6, 2 (1962), 200–209.

[24] Adam Moody, Greg Bronevetsky, Kathryn Mohror, and Bronis R. de Supinski. 2010. Design, Modeling, and Evaluation of a Scalable Multi-level Checkpointing System. In *SC*. ACM.

[25] Xiang Ni, Esteban Meneses, Nikhil Jain, and Laxmikant V. Kalé. 2013. ACR: Automatic Checkpoint/Restart for Soft and Hard Error Protection. In *SC*. ACM.

[26] T.J. O'Gorman. 1994. The effect of cosmic rays on the soft error rate of a DRAM at ground level. *IEEE Trans. Electron Devices* 41, 4 (1994), 553–557.

[27] R. A. Oldfield, S. Arunagiri, P. J. Teller, S. Seelam, M. R. Varela, R. Riesen, and P. C. Roth. 2007. Modeling the Impact of Checkpoints on Next-Generation Systems. In *24th IEEE Conf. Mass Storage Systems and Technologies*. IEEE.

[28] B. Schroeder and G. Gibson. 2007. Understanding failures in petascale computers. *Journal of Physics: Conference Series* 78, 1 (2007).

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

[30] Manu Shantharam, Sowmyalatha Srinivasmurthy, and Padma Raghavan. 2012. Fault Tolerant Preconditioned Conjugate Gradient for Sparse Linear System Solution. In *ICS*. ACM.

[31] M. Snir and et al. 2014. Addressing Failures in Exascale Computing. *Int. J. High Perform. Comput. Appl.* 28, 2 (2014), 129–173.

[32] Jon Stearley, Kurt B. Ferreira, David J. Robinson, Jim Laros, Kevin T. Pedretti, Dorian Arnold, Patrick G. Bridges, and Rolf Riesen. 2012. Does partial replication pay off?. In *FTXS*. IEEE.

[33] Omer Subasi, Javier Arias, Osman Unsal, Jesus Labarta, and Adrian Cristal. 2015. Programmer-directed Partial Redundancy for Resilient HPC. In *Computing Frontiers*. ACM.

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

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

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

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

[38] J. F. Ziegler, H. W. Curtis, H. P. Muhlfeld, C. J. Montrose, and B. Chin. 1996. IBM Experiments in Soft Fails in Computer Electronics. *IBM J. Res. Dev.* 40, 1 (1996), 3–18.