International Journal of Networking and Computing – www.ijnc.org, ISSN 2185-2847
Volume X, Number Y, pages 1–26, January 20XX

Combining Checkpointing and Replication for Reliable Execution of Linear Workflows with Fail-Stop and Silent Errors

Anne Benoit
ENS Lyon, France

Aurélien Cavelan
University of Basel, Switzerland

Florina M. Ciorba
University of Basel, Switzerland

Valentin Le Fèvre
ENS Lyon, France

Yves Robert
ENS Lyon, France
University of Tennessee, Knoxville, TN, USA

Received: (received date)
Revised: (revised date)
Accepted: (accepted date)
Communicated by Editor's name

Abstract

Large-scale platforms currently experience errors from two different sources, namely fail-stop errors (which interrupt the execution) and silent errors (which strike unnoticed and corrupt data). This work combines checkpointing and replication for the reliable execution of linear workflows on platforms subject to these two error types. While checkpointing and replication have been studied separately, their combination has not yet been investigated despite its promising potential to minimize the execution time of linear workflows in error-prone environments. Moreover, combined checkpointing and replication has not yet been studied in the presence of both fail-stop and silent errors. The combination raises new problems: for each task, we have to decide whether to checkpoint and/or replicate it to ensure its reliable execution. We provide an optimal dynamic programming algorithm of quadratic complexity to solve both problems. This dynamic programming algorithm has been validated through extensive simulations that reveal the conditions in which checkpointing only, replication only, or the combination of both techniques, lead to improved performance.

Keywords: checkpoint, replication, HPC, fail-stop error, silent error, linear workflow

1 Introduction

Several high-performance computing (HPC) applications are designed as a succession of (typically large) tightly-coupled computational kernels, or tasks, that should be executed in sequence [9, 15, 26]. These parallel tasks are executed on the entire platform, and they exchange data at the end of their execution. In other words, the task graph is a linear chain, and each task (except maybe the first one and the last one) reads data from its predecessor and produces data for its successor. Such linear chains of tasks also appear in image processing applications [30], and are usually called *linear workflows* [41].

The first objective when considering linear workflows is to ensure their *efficient execution*, which amounts to minimizing the total parallel execution time, or makespan. However, a *reliable execution* is also critical to performance. Indeed, large-scale platforms are increasingly subject to errors [10, 11]. Scale is the enemy here: even if each computing resource is very reliable, with, say, a Mean Time Between Errors (MTBE) of ten years, meaning that each resource will experience an error only every 10 years on average, a platform composed of $100,000$ of such resources will experience an error every fifty minutes [25]. Hence, fault-tolerance techniques to mitigate the impact of errors are required to ensure a correct and uninterrupted execution of the application [28]. To further complicate matters, several types of errors need to be considered when computing at scale. In addition to the classical fail-stop errors (such as hardware failures or crashes), silent errors (also known as silent data corruptions) constitute another threat that can no longer be ignored [33, 53, 51, 52, 31]. 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) components as well as the 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.

The standard approach to cope with fail-stop errors is checkpoint with rollback and recovery [13, 19]: in the context of linear workflow applications, each task can decide to take a checkpoint after it has correctly executed. A checkpoint is simply a file including all intermediate results and associated data that is saved on a storage medium resilient to errors; it can be either the memory of another processor, a local disk, or a remote disk. This file can be recovered if a successor task experiences an error later in the execution. If there is an error while some task is executing, the application has to roll back to the last checkpointed task (or to start recomputing again from scratch if no checkpoint was taken). Then the checkpoint is read from the storage medium (recovery phase), and execution resumes from that task onward. If the checkpoint was taken many tasks before an error strikes, there is a lot of re-execution involved, which calls for more frequent checkpoints. However, checkpointing incurs a significant overhead, and is a mere waste of resources if no error strikes. Altogether, there is a trade-off to be found, and one may want to checkpoint only carefully selected tasks.

While checkpoint/restart [13, 19, 20] is the de-facto recovery technique for addressing fail-stop errors, there is no widely adopted general-purpose technique to cope with silent errors. The challenge with silent errors is *detection latency*: contrarily to a fail-stop error whose detection is immediate, a silent error is identified only when the corrupted data is activated and/or leads to an unusual application behavior. However, checkpoint and rollback recovery assumes instantaneous error detection, and this raises a new difficulty: if the error stroke before the last checkpoint, and is detected after that checkpoint, then the checkpoint is corrupted and cannot be used to restore the application. To address the problem of silent errors, many application-specific detectors, or verification mechanisms, have been proposed. We apply such a verification mechanism after each task in this paper. Our approach is agnostic of the nature of the verification mechanism (checksum, error correcting code, coherence test, etc.). In this context, if the verification succeeds, then the output of the task is correct, and one can safely either proceed to the next task directly, or save the result beforehand by taking a checkpoint. Otherwise, if verification fails we have to rollback to the last saved checkpoint and re-execute the work since that point on. However, and contrarily to fail-stop errors, silent errors do not cause the loss of the entire memory content of the affected processor. To account for this difference, we use a two-level checkpointing scheme: the checkpoint file is saved in the main memory of the processor before being transferred to some storage (disk) that is resilient to fail-stop errors. This allows for recovering faster after a silent error than after a fail-stop error.

Replication is a well-known, but costly, method to deal with both, fail-stop errors [22, 23, 12, 36, 49, 21, 18, 38] and silent errors [32, 3]. While both checkpointing and replication have been extensively studied separately, their combination has not yet been investigated in the context of linear workflows, despite its promising potential to minimize the execution time in error-prone environments. The contributions of this work are the following:

- We provide a detailed model for the reliable execution of linear workflows, where each task can be replicated or not, and with a two-level checkpoint/recovery mechanism whose cost depends both on the number of processors executing the task, and on whether the task is replicated or not.
- We address both fail-stop and silent errors. We perform a verification after each task to detect silent errors and recover from the last in-memory checkpoint after detecting one. We recover from the last disk checkpoint after a fail-stop error. If a task is replicated, we do not need to roll back and we can directly proceed to the next task, unless both replicas have been affected (by either error type).
- We design an optimal dynamic programming algorithm that minimizes the makespan of a linear workflow with n tasks, with a quadratic complexity, in the presence of fail-stop and silent errors.
- We conduct extensive experiments to evaluate the impact of using both replication and checkpointing during execution, and compare them to an execution without replication.
- We provide guidelines about when it is beneficial to employ checkpointing only, replication only, or to combine both techniques together.

The paper is organized as follows. Section 2 details the model and formalizes the objective function and the optimization problem. Section 3 presents a preliminary result for the dynamic programming algorithm: we explain how to compute the expected time needed to execute a single task (replicated or not), assuming that its predecessor has been checkpointed. The proposed optimal dynamic programming algorithm is outlined in Section 4. The experimental validation is provided in Section 5. Finally, related work is discussed in Section 6, and the work is concluded in Section 7.

2 Model and objective

This section details the framework of this study. We start with the application and platform models, then we detail the verification, checkpointing and replication, and finally we state the optimization problem.

2.1 Application model

We target applications whose workflows represent linear chains of parallel tasks. More precisely, for one application, consider a chain $T_1 \rightarrow T_2 \rightarrow \cdots \rightarrow T_n$ of n parallel tasks T_i, $1 \leq i \leq n$. Hence, T_1 must be completed before executing T_2, and so on.

Here, each T_i is a parallel task whose speedup profile obeys *Amdahl's law* [1]: the total work, w_i, consists of a sequential fraction $\alpha_i w_i$, $0 \leq \alpha_i \leq 1$, and the remaining fraction $(1-\alpha_i)w_i$ perfectly parallel. The (error-free) execution time, T_i, using q_i processors is thus $w_i \left(\alpha_i + \frac{1-\alpha_i}{q_i}\right)$. Without loss of generality, we assume that processors execute the tasks at unit speed, and we use time units and work units interchangeably. While our study is agnostic of task granularity, it applies primarily to frameworks where tasks represent large computational entities whose execution takes from a few minutes up to tens of minutes. In such frameworks, it may be worthwhile to replicate or to checkpoint tasks to mitigate the impact of errors.

2.2 Execution platform

We target a *homogeneous* platform with p processors P_i, $1 \leq i \leq p$. We assume that the platform is subject to fail-stop and silent errors whose inter-arrival times follow an Exponential distribution. More precisely, let λ_{ind}^F be the fail-stop error rate of each individual processor P_i: the probability

of having a fail-stop error striking P_i within T time-units is $\mathbb{P}(X \leq T) = 1 - e^{-\lambda^F_{ind} T}$. Similarly, let λ^S_{ind} be the silent error rate of each individual processor P_i: the probability of having a silent error striking P_i within T time-units is $\mathbb{P}(Y \leq T) = 1 - e^{-\lambda^S_{ind} T}$. Then, a computation on $q \leq p$ processors has an error rate $q\lambda^F_{ind}$ for fail-stop errors, and $q\lambda^S_{ind}$ for silent errors. The probability of having a fail-stop error within T time-units and with q processors becomes $1 - e^{-q\lambda^F_{ind} T}$ (and $1 - e^{-q\lambda^S_{ind} T}$ for a silent error) [25].

2.3 Verification

To detect silent errors, we add a verification mechanism at the end of each task. This ensures that the error will be detected as soon as possible. The verification following task T_i has a cost V_i. We assume that the verification mechanism has a perfect recall (it detects all errors). This guarantees that all taken checkpoints are correct, because they are always preceded by a verification. Similarly, we assume that no silent error can strike during the verification.

The cost V_i depends upon the detector and can thereby take a wide range of values. In this work, we adopt a quite general formula and use

$$V_i(q_i) = u_i + \frac{v_i}{q_i} \tag{1}$$

to model the cost of verifying task T_i when executed with q_i processors, where u_i and v_i denote the sequential and parallel cost of the verification, respectively. In the experiments (Section 5.2), we instantiate the model with two cases:

- We use $u_i = \beta w_i$ and $v_i = 0$, where β is a small parameter (around 1%). This means that the cost of the verification is proportional to the sequential cost w_i of T_i. It corresponds to the case of data-oriented kernels processing large files and checksumming for verification in a centralized location (hence sequentially) [5].
- We use $u_i = 0$ and $v_i = \beta w_i$. This means that the cost of the verification is proportional to the parallel fraction of T_i. It corresponds to the same scenario as above, but where checksumming is performed in parallel on all enrolled processors.

2.4 Checkpointing

The output of each task T_i can be checkpointed in time C_i. We use a two-level checkpoint protocol where the checkpoint is first saved locally (memory checkpoint) before being transferred to a slower but reliable storage like a filesystem (disk checkpoint). The memory checkpoint will be lost when a fail-stop error strikes a processor (and its local data), whereas the disk checkpoint will always remain available to restart the application.

When a fail-stop error strikes during the execution of T_i, we first incur a downtime D, and then we must start the execution from the task following the last checkpoint. Hence, if T_j is the last checkpointed task, the execution starts again at task T_{j+1}, and the recovery cost is R^D_{j+1}, which amounts to reading the disk checkpoint of task T_j. When a silent error is detected at the end of T_i by the verification mechanism, we also roll back to the last checkpointed task T_j, but (i) we do not pay the downtime D; and (ii) the recovery cost is R^M_{j+1}, which amounts to reading the memory checkpoint of task T_j (hence at a much smaller cost than for a fail-stop error).

The checkpoint cost C_i, and both recovery costs R^D_{j+1} and R^M_{j+1} clearly depend upon the checkpoint protocol and storage medium, as well as upon the number q_i of enrolled processors. In this work, we adopt a quite general formula for checkpoint times and use

$$C_i(q_i) = a_i + \frac{b_i}{q_i} + c_i q_i \tag{2}$$

to model the time to save a checkpoint after T_i executed with q_i processors. Here, $a_i + \frac{b_i}{q_i}$ represents the I/O overhead to write the task output file M_i to the storage medium. For in-memory

checkpointing [48], $a_i + \frac{b_i}{q_i}$ is the communication time, in which a_i denotes the latency to access the storage system; then we have $\frac{b_i}{q_i} = \frac{M_i}{\tau_{net} q_i}$, where τ_{net} is the network bandwidth (each processor stores $\frac{M_i}{q_i}$ data items). For coordinated checkpointing to stable storage, there are two cases: if the I/O bottleneck is the storage system's bandwidth, then $a_i = \beta + \frac{M_i}{\tau_{io}}$ and $b_i = 0$, where β is a start-up time and τ_{io} is the I/O bandwidth; otherwise, if the I/O bottleneck is the network latency, we retrieve the same formula as for in-memory checkpointing. Finally, $c_i q_i$ represents the message passing overhead that grows linearly with the number of processors, in order for all processors to reach a global consistent state [19, 50].

For the cost of recovery (from memory or from disk), we assume similar formulas:

$$R_i^M(q_i) = a_i^M + \frac{b_i^M}{q_i} + c_i^M q_i; \quad R_i^D(q_i) = a_i^D + \frac{b_i^D}{q_i} + c_i^D q_i. \tag{3}$$

The coefficients depend on the type of recovery: again, a memory recovery is much faster than a disk recovery. If we further assume that reading and writing from/to the same storage medium (memory or disk) have same cost, we have

$$C_i(q_i) = R_{i+1}^D(q_i) + R_{i+1}^M(q_i)$$

since recovering for task T_{i+1} amounts to reading the checkpoint from task T_i.

Finally, we assume that there is a fictitious task T_0 of zero weight ($w_0 = 0$) that is always checkpointed, so that $R_1^D(q_1)$ represents the time for I/O input from the external world. Similarly, we systematically checkpoint the last task T_n, in order to account for the I/O output time $C_n(q_n)$.

2.5 Replication

When executing a task, we envision two possibilities: either the task is not replicated, or it is replicated. To explain the impact of replication, we momentarily assume that we consider fail-stop errors only. Then we return to the scenario with both fail-stop and silent errors.

With fail-stop errors only, consider a task T_i, and assume for simplicity that the predecessor T_{i-1} of T_i has been checkpointed. If it is not the case, i.e., if the predecessor T_{i-1} of T_i is not checkpointed, we have to roll back to the last checkpointed task, say T_k where $k < i-1$, whenever an error strikes, and re-execute the entire segment from T_{k+1} to T_i instead of just T_i.

Without replication, a single copy of T_i is executed on the entire platform, hence with $q_i = p$ processors. Then we let $\mathbb{E}^{norep}(i)$ denote the expected execution time of T_i when accounting for errors. We attempt a first execution, which takes $T_i^{norep} = w_i \left(\alpha_i + \frac{1-\alpha_i}{p}\right)$ if no fail-stop error strikes. But if a fail-stop error does strike, we must account for the time that has been lost (between the beginning of the execution and the fail-stop error), then perform a downtime D, a recovery $R_i(p)$ (since we use the entire platform for T_i), and then re-execute T_i from scratch. Similarly, if we decide to checkpoint after T_i, we need $C_i(p)$ time units. We explain how to compute $\mathbb{E}^{norep}(i)$ in Section 3.

With replication, two copies of T_i are executed in parallel, each with $q_i = \frac{p}{2}$ processors. If no fail-stop error strikes, both copies finish execution in time $T_i^{rep} = w_i \left(\alpha_i + \frac{1-\alpha_i}{\frac{p}{2}}\right)$, since each copy uses $\frac{p}{2}$ processors. If a fail-stop error strikes one copy, we proceed as before, account for the downtime D, recover (in time $R_i(\frac{p}{2})$ now), and restart execution with that copy. Then there are two cases: (i) if the second copy successfully completes its first execution, the fail-stop error has no impact and the execution time remains the same as the error-free execution time; (ii) however, if the second copy also fails to execute, we resume its execution, and iterate until one copy successfully completes. Of course, case (ii) is less likely to happen than case (i), which explains why replication can be useful. Finally, if we decide to checkpoint after T_i, the first successful copy will take the checkpoint in time $C_i(\frac{p}{2})$.

Replication raises several complications in terms of checkpoint and recovery costs. When a replicated task T_i is checkpointed, we can enforce that only one copy (the first one to complete execution) would write the output data onto the storage medium, hence with a cost $C_i(\frac{p}{2})$, as stated above. Similarly, when a single copy of a replicated task T_i performs a recovery after a fail-stop

error, the cost would be $R_i(\frac{p}{2})$. However, in the unlikely event where both copies are struck by a fail-stop error at close time instances, their recoveries would overlap, and the cost can vary anywhere between $R_i(\frac{p}{2})$ and $2R_i(\frac{p}{2})$, depending upon the amount of contention, the length of the overlap and where the I/O bottleneck lies. We will experimentally evaluate the impact of the recovery cost with replication in Section 5.1. For simplicity, in the rest of the paper, we use C_i^{rep} for the checkpoint cost of T_i when it is replicated, and C_i^{norep} when it is not. Similarly, we use R_i^{Drep} or R_i^{Mrep} for the recovery costs (disk or memory) when T_i is replicated, and R_i^{Dnorep} or R_i^{Mnorep} when it is not. Note that the recovery cost of T_i depends upon whether it is replicated or not, but does not depend upon whether the checkpointed task T_{i-1} was replicated or not, since we need to read the same file from the storage medium in both cases. The values of C_i^{rep} and C_i^{norep} can be instantiated from Equation (2) and those of R_i^{Drep}, R_i^{Dnorep}, R_i^{Mrep} and R_i^{Mnorep} can be instantiated from Equation (3). We let $\mathbb{E}^{rep}(i)$ denote the expected execution time of T_i with replication and when accounting for fail-stop errors, when T_{i-1} is checkpointed. The derivation of $\mathbb{E}^{rep}(i)$ is significantly more complicated than for $\mathbb{E}^{norep}(i)$ and represents a new contribution of this work, detailed in Section 3.2.

We now detail the impact of replication when both fail-stop and silent errors can strike. First, we have to state how the verification cost V_i of task T_i depends upon whether T_i is replicated or not. For the analysis, we keep a general model and let V_i^{rep} be the cost when T_i is replicated, and V_i^{norep} when it is not. However, as explained later in the experimental evaluation (Section 5.2), we use two different instantiations of Equation (1), which directly give the two (possibly different) values of V_i^{rep} and V_i^{norep} as a function of parameter β.

Next, consider again a task T_i, and still assume for simplicity that the predecessor T_{i-1} of T_i has been checkpointed. The impact of fail-stop errors is the same as before, and depends upon how many replicas of T_i are executed. The only difference is that the fail-stop error can now strike either during the execution of a replica or during its verification. But if no fail-stop error strikes, we still have to perform the verification to detect a possible silent error, whose probability depends upon the error-free execution time of that replica. Recall that no silent error can strike during the verification (but a fail-stop can strike). If a silent error is detected, we have to re-execute the task, in which case we recover from the memory checkpoint instead of from the disk checkpoint.

Finally, we extend the definition of $\mathbb{E}^{norep}(i)$ and $\mathbb{E}^{rep}(i)$ to account for both fail-stop and silent errors, when T_{i-1} is checkpointed. We explain how to compute both quantities in Section 3.2.

2.6 Optimization problem

The objective of this work is to *minimize the expected makespan* of the linear workflow in the presence of fail-stop and silent errors. For each task, we have four choices: either we replicate the task or not, and either we checkpoint it or not. More formally, we need to decide for each task T_i: (i) if it is checkpointed or not; and (ii) if it is replicated or not (meaning that there are 4^n combinations for the whole workflow), with the objective to minimize the total execution time of the workflow. We point out that none of these decisions can be made locally. Instead, we need to account for previous decisions and optimize globally. Our major contribution of this work is to provide an optimal dynamic programming algorithm to solve this problem, which we denote as ChainsRepCkpt.

We point out that ChainsCkpt, the simpler problem without replication, i.e., optimally placing checkpoints for a chain of tasks, has been extensively studied. The first dynamic programming algorithm to solve ChainsCkpt appears in the pioneering paper of Toueg and Babaoğlu [43] back in 1984, for the scenario with fail-stop errors only (see Section 6 on related work for further references). Adding replication significantly complicates the solution. Here is an intuitive explanation: When the algorithm recursively considers a segment of tasks from T_i to T_j, where T_{i-1} and T_j are both checkpointed and no intermediate task T_k, $i \leq k < j$ is checkpointed, there are many cases to consider to account for possible different values in: (i) execution time, since some tasks in the segment may be replicated; (ii) checkpoint, whose cost depends upon whether T_j is replicated or not; and (iii) recovery, whose cost depends upon whether T_i is replicated or not. We provide all details in Section 4.

3 Computing $\mathbb{E}^{norep}(i)$ and $\mathbb{E}^{rep}(i)$

This section details how to compute the expected time needed to execute a task T_i, assuming that the predecessor of T_i has been checkpointed. Hence, we need to re-execute only T_i when an error strikes. We explain how to deal with the general case of re-executing a segment of tasks, some of them replicated, in Section 4. Here, we start with the case where T_i is not replicated. It is already known how to compute $\mathbb{E}^{norep}(i)$ [25, 6], but we present this case to help the reader follow the derivation in Section 3.2 for the case where T_i is replicated, which is new and much more involved.

3.1 Computing $\mathbb{E}^{norep}(i)$

To compute $\mathbb{E}^{norep}(i)$, the average execution time of T_i with p processors without replication, we conduct a case analysis:

- Either a fail-stop error strikes during the execution of the task and its verification (lasting $T_i^{norep}+V_i^{norep}$), and in this case we lose some work and need to re-execute the task, recovering from a disk checkpoint;
- Either there is no fail-stop error, and in this case the verification indicates whether there has been a silent error or not:
  - If a silent error is detected, we need to re-execute the task right after the verification, recovering from a memory checkpoint;
  - Otherwise the execution has been successful.

This leads to the following recursive formula:

$$\begin{aligned}\mathbb{E}^{norep}(i) = \mathbb{P}(X_p \leq T_i^{norep} + V_i^{norep})\Big(T_{lost}^{norep}(T_i^{norep} + V_i^{norep}) + D + R_i^{Dnorep} + \mathbb{E}^{norep}(i)\Big) \\ + (1 - \mathbb{P}(X_p \leq T_i^{norep} + V_i^{norep}))\Big(T_i^{norep} + V_i^{norep} \\ + \mathbb{P}(X_p^{'} \leq T_i^{norep})(D + R_i^{Mnorep} + \mathbb{E}^{norep}(i))\Big),\end{aligned} \tag{4}$$

where $\mathbb{P}(X_p \leq t)$ is the probability of having a fail-stop error on one of the p processors before time t, i.e., $\mathbb{P}(X_p \leq t) = 1 - e^{-\lambda_{ind}^F pt}$, and $\mathbb{P}(X_p^{'} \leq t)$ is the probability of having a silent error on one of the p processors before time t, i.e., $\mathbb{P}(X_p^{'} \leq t) = 1 - e^{-\lambda_{ind}^S pt}$. The time lost when an error strikes is the expectation of the random variable X_p, knowing that the error stroke before the end of the task and its verification. We compute it as follows:

$$\begin{aligned}T_{lost}^{norep}(T_i^{norep} + V_i^{norep}) &= \int_0^{+\infty} x\mathbb{P}(X_p = x | X_p \leq T_i^{norep} + V_i^{norep})dx \\ &= \frac{1}{\mathbb{P}(X_p \leq T_i^{norep} + V_i^{norep})} \int_0^{T_i^{norep}+V_i^{norep}} x\mathbb{P}(X_p = x)dx \\ &= \frac{1}{\mathbb{P}(X_p \leq T_i^{norep} + V_i^{norep})} \int_0^{T_i^{norep}+V_i^{norep}} x\frac{d\mathbb{P}(X_p \leq x)}{dx}dx\end{aligned}$$

After integration, we get the formula:

$$T_{lost}^{norep}(T_i^{norep} + V_i^{norep}) = \frac{1}{\lambda_{ind}^F p} - \frac{T_i^{norep} + V_i^{norep}}{e^{\lambda_{ind}^F p(T_i^{norep}+V_i^{norep})} - 1}. \tag{5}$$

Replacing the left hand side term of Equation (5) in Equation (4) and solving, we derive:

$$\begin{aligned}\mathbb{E}^{norep}(i) = &\left(\frac{1}{\lambda_{ind}^{F} p} + D + R_i^{Dnorep}\right) e^{p((\lambda_{ind}^{F}+\lambda_{ind}^{S})T_i^{norep}+\lambda_{ind}^{F} V_i^{norep})} \\ &- \left(\frac{1}{\lambda_{ind}^{F} p} + (R_i^{Dnorep} - R_i^{Mnorep})\right) e^{\lambda_{ind}^{S} p T_i^{norep}} - (D + R_i^{Mnorep}).\end{aligned} \tag{6}$$

Recall that $T_i^{norep} = w_i \left(\alpha_i + \frac{1-\alpha_i}{p}\right)$ in Equation (6). Finally, if we decide to checkpoint T_i, we simply add C_i^{norep} to $\mathbb{E}^{norep}(i)$.

3.2 Computing $\mathbb{E}^{rep}(i)$

We now discuss the case where T_i is replicated; each copy executes with $\frac{p}{2}$ processors. To compute $\mathbb{E}^{rep}(i)$, the expected execution time of T_i with replication, we conduct a case analysis similar to that of Section 3.1:

- Either two fail-stop errors strike before the end of the task and its verification (lasting $T_i^{rep} + V_i^{rep}$), with one fail-stop error striking each copy. Then we have lost some work and need to re-execute the task from a disk checkpoint;
- Or at least one copy is not hit by any fail-stop error. Then we need to account for two different cases in the analysis:
  - Both copies have survived: then we need to re-execute the task (recovering from a memory checkpoint) only if both copies are hit by a silent error.
  - Only one replica survived: then we need to re-execute the task if this replica is hit by a silent error.

This leads to the following formula:

$$\begin{aligned}\mathbb{E}^{rep}(i) = &\ \mathbb{P}(Y_p \le T_i^{rep} + V_i^{rep})^2 \Big(T_{lost}^{rep}(T_i^{rep} + V_i^{rep}) + D + R_i^{Drep} + \mathbb{E}^{rep}(i)\Big) \\ &+ (1 - \mathbb{P}(Y_p \le T_i^{rep} + V_i^{rep})^2)(T_i^{rep} + V_i^{rep}) \\ &+ \Big(2(1 - \mathbb{P}(Y_p \le T_i^{rep} + V_i^{rep}))\mathbb{P}(Y_p \le T_i^{rep} + V_i^{rep})\mathbb{P}(Y_p^{'} \le T_i^{rep}) \\ &+ (1 - \mathbb{P}(Y_p \le T_i^{rep} + V_i^{rep}))^2 \mathbb{P}(Y_p^{'} \le T_i^{rep})^2\Big)(D + R_i^{Mrep} + \mathbb{E}^{rep}(i)),\end{aligned} \tag{7}$$

where $\mathbb{P}(Y_p \le t)$ is the probability of having an error on one replica of $\frac{p}{2}$ processors before time t, i.e., $\mathbb{P}(Y_p \le t) = 1 - e^{-\frac{\lambda_{ind}^{F} p}{2} t}$, and $\mathbb{P}(Y_p^{'} \le t)$ is the probability of having a silent error on one replica of $\frac{p}{2}$ processors before time t, i.e., $\mathbb{P}(Y_p^{'} \le t) = 1 - e^{-\frac{\lambda_{ind}^{S} p}{2} t}$. The first line of Equation (7) corresponds to the case where both replicas are hit by a fail-stop error, the second line accounts for the time spent in case at least one replica survives. The last two lines correspond to the two cases when we need to re-execute the task after the detection of a silent error (one replica alive for line 3, two replicas alive for line 4 of Equation (7)).

The time lost when both copies fail can be computed in a similar way as before:

$$T_{lost}^{rep}(T_i^{rep}) = \frac{1}{\mathbb{P}(Y_p \le T_i^{rep} + V_i^{rep})} \int_0^{T_i^{rep}+V_i^{rep}} x \frac{d\mathbb{P}(Y_p \le x)}{dx} dx.$$

After computation and verification using a Maple sheet, we obtain the following result:

$$T_{lost}^{rep}(T_i^{rep} + V_i^{rep}) = \frac{(-2\lambda_{ind}^{F} p(T_i^{rep}+V_i^{rep})-4)e^{-\frac{\lambda_{ind}^{F} p(T_i^{rep}+V_i^{rep})}{2}} + (\lambda_{ind}^{F} p(T_i^{rep}+V_i^{rep})+1)e^{-\lambda_{ind}^{F} p(T_i^{rep}+V_i^{rep})} + 3}{(e^{-\frac{\lambda_{ind}^{F} p(T_i^{rep}+V_i^{rep})}{2}} - 1)^2 \lambda_{ind}^{F} p}. \tag{8}$$

Replacing the left hand side term of Equation (8) in Equation (7) and solving, we get:

$$\begin{aligned}\mathbb{E}^{rep}(i) = & -\frac{(4+2\lambda^F_{ind}p(R_i^{Drep}-R_i^{Mrep}))e^{p(\frac{\lambda^F_{ind}(T_i^{rep}+V_i^{rep})}{2}+\lambda^S_{ind}T_i^{rep})}}{(2e^{p\frac{\lambda^F_{ind}(T_i^{rep}+V_i^{rep})+\lambda^S_{ind}T_i^{rep}}{2}}-1)\cdot\lambda^F_{ind}p} \\ & + \frac{(1+\lambda^F_{ind}p(R_i^{Drep}-R_i^{Mrep}))e^{\lambda^S_{ind}pT_i^{rep}}}{(2e^{p\frac{\lambda^F_{ind}(T_i^{rep}+V_i^{rep})+\lambda^S_{ind}T_i^{rep}}{2}}-1)\cdot\lambda^F_{ind}p} \\ & + \frac{(3+\lambda^F_{ind}p(D+R_i^{Drep}))e^{p(\lambda^F_{ind}(T_i^{rep}+V_i^{rep})+\lambda^S_{ind}T_i^{rep})}}{(2e^{p\frac{\lambda^F_{ind}(T_i^{rep}+V_i^{rep})+\lambda^S_{ind}T_i^{rep}}{2}}-1)\cdot\lambda^F_{ind}p} - (D+R_i^{Mrep})\end{aligned} \tag{9}$$

Recall that $T_i^{rep} = w_i\left(\alpha_i + \frac{1-\alpha_i}{\frac{p}{2}}\right)$ in Equation (9). Finally, if we decide to checkpoint T_i, we simply add C_i^{rep} to $\mathbb{E}^{rep}(i)$.

4 Optimal dynamic programming algorithm

In this section, we provide an optimal dynamic programming (DP) algorithm to solve the ChainsRepCkpt problem for a linear chain of n tasks.

Theorem 1. *The optimal solution to the* ChainsRepCkpt *problem can be obtained using a dynamic programming algorithm in $O(n^2)$ time, where n is the number of tasks in the chain.*

Proof. The algorithm recursively computes the expectation of the optimal time required to execute tasks T_1 to T_i and then checkpoint T_i. As already mentioned, we need to distinguish two cases, according to whether T_i is replicated or not, because the cost of the final checkpoint depends upon this decision. Hence, we recursively compute two different functions:

- $T_{opt}^{rep}(i)$, the expectation of the optimal time required to execute tasks T_1 to T_i, knowing that T_i is replicated;
- $T_{opt}^{norep}(i)$, the expectation of the optimal time required to execute tasks T_1 to T_i, knowing that T_i is not replicated.

Note that checkpoint time is not included in $T_{opt}^{rep}(i)$ nor $T_{opt}^{norep}(i)$. The solution to ChainsRepCkpt will be given by

$$\min\left\{T_{opt}^{rep}(n)+C_n^{rep}, T_{opt}^{norep}(n)+C_n^{norep}\right\}. \tag{10}$$

We start with the computation of $T_{opt}^{rep}(j)$ for $1 \le j \le n$, hence assuming that the last task T_j is replicated. We express $T_{opt}^{rep}(j)$ recursively as follows:

$$T_{opt}^{rep}(j) = \min_{1\le i<j}\left\{\begin{array}{l} T_{opt}^{rep}(i)+C_i^{rep}+T_{NC}^{rep,rep}(i+1,j), \\ T_{opt}^{rep}(i)+C_i^{rep}+T_{NC}^{norep,rep}(i+1,j), \\ T_{opt}^{norep}(i)+C_i^{norep}+T_{NC}^{rep,rep}(i+1,j), \\ T_{opt}^{norep}(i)+C_i^{norep}+T_{NC}^{norep,rep}(i+1,j), \\ R_1^{Drep}+T_{NC}^{rep,rep}(1,j), \\ R_1^{Dnorep}+T_{NC}^{norep,rep}(1,j) \end{array}\right\} \tag{11}$$

In Equation (11), T_i corresponds to the last checkpointed task before T_j, and we try all possible locations T_i for taking a checkpoint before T_j. The first four lines correspond to the case where there is indeed an intermediate task T_i between T_1 and T_j that is checkpointed, while the last two lines correspond to the case where no checkpoint at all is taken until after T_j.

The first two lines of Equation (11) apply to the case where T_i is replicated. Line 1 is for the case when T_{i+1} is replicated, and line 2 when it is not. In the first line of Equation (11), $T_{NC}^{rep,rep}(i+1,j)$ denotes the optimal time to execute tasks T_{i+1} to T_j without any intermediate checkpoint, knowing that T_i is checkpointed, and both T_{i+1} and T_j are replicated. If T_{i+1} is not replicated, we use the second line of Equation (11), where $T_{NC}^{norep,rep}(i+1,j)$ is the counterpart of $T_{NC}^{rep,rep}(i+1,j)$, except

that it assumes that T_{i+1} is not replicated. This information on T_{i+1} (replicated or not) is needed to compute the recovery cost when executing tasks T_{i+1} to T_j and experiencing an error.

Lines 3 and 4 apply to the case where T_i is not replicated, with similar notation as before. In the first four lines, no task between T_{i+1} and T_{j-1} is checkpointed, hence the notation NC for no checkpoint.

If no checkpoint at all is taken before T_j (this corresponds to the case $i = 0$), we use the last two lines of Equation (11): we include the cost to read the initial input, which depends whether T_1 is replicated (in line 5) or not (in line 6) of Equation (11).

We have a very similar equation to express $T_{opt}^{norep}(j)$ recursively, with intuitive notation:

$$T_{opt}^{norep}(j) = \min_{1 \leq i < j} \left\{ \begin{array}{l} T_{opt}^{rep}(i) + C_i^{rep} + T_{NC}^{rep,norep}(i+1,j), \\ T_{opt}^{rep}(i) + C_i^{rep} + T_{NC}^{norep,norep}(i+1,j), \\ T_{opt}^{norep}(i) + C_i^{norep} + T_{NC}^{rep,norep}(i+1,j), \\ T_{opt}^{norep}(i) + C_i^{norep} + T_{NC}^{norep,norep}(i+1,j), \\ R_1^{Drep} + T_{NC}^{rep,norep}(1,j), \\ R_1^{Dnorep} + T_{NC}^{norep,norep}(1,j) \end{array} \right\} \tag{12}$$

To synthesize the notation, we have defined $T_{NC}^{A,B}(i+1,j)$, with $A, B \in \{rep, norep\}$, as the optimal time to execute tasks T_{i+1} to T_j without any intermediate checkpoint, knowing that T_i is checkpointed, T_{i+1} is replicated if and only if $A = rep$, and T_j is replicated if and only if $B = rep$. In a nutshell, we have to account for the possible replication of the first task T_{i+1} after the last checkpoint, and of the last task T_j, hence the four cases.

There remains to compute $T_{NC}^{A,B}(i,j)$ for all $1 \leq i, j \leq n$ and $A, B \in \{rep, norep\}$. This is still not easy, because there remains to decide which intermediate tasks should be replicated. In addition to the status of T_j (replicated or not, according to the value of B), the only thing we know so far is that the only checkpoint that we can recover from while executing tasks T_i to T_j is the checkpoint taken after task T_{i-1}, hence we need to re-execute from T_i whenever an error strikes. Furthermore, T_i is replicated if and only if $A = rep$, hence we know the corresponding cost for recovery, R_i^A. Letting $T_{NC}^{A,B}(i,j) = 0$ whenever $i > j$, we can express $T_{NC}^{A,B}(i,j)$ for $1 \leq i \leq j \leq n$ as follows:

$$\begin{array}{rl} T_{NC}^{A,B}(i,j) = & \min \left\{ T_{NC}^{A,rep}(i,j-1), T_{NC}^{A,norep}(i,j-1) \right\} \\ & + T^{A,B}(j \mid i). \end{array}$$

Here the new (and final) notation $T^{A,B}(j \mid i)$ is simply the time needed to execute task T_j, knowing that an error during T_j implies to recover from T_i. Indeed, to execute tasks T_i to T_j, we account recursively for the time to execute T_i to T_{j-1}; T_{i-1} is still checkpointed; T_i is replicated if and only if $A = rep$, T_j is replicated if and only if $B = rep$, and we consider both cases whether T_{j-1} is replicated or not. The time lost in case of an error during T_j depends whether T_j is replicated or not, and we need to restart from T_i in case of error, hence the notation $T^{A,B}(j \mid i)$, representing the expected execution time for task T_j with or without replication (depending on B), given that we need to restart from T_i if there is an error (and T_i is replicated if and only if $A = rep$).

The last step is hence to express these execution times. We start with the case where T_j is not replicated:

$$\begin{array}{l} T^{A,norep}(j \mid i) = \left(1 - e^{-\lambda_{ind}^F p(T_j^{norep} + V_j^{norep})}\right) \left(T_{lost}^{norep}(T_j^{norep} + V_j^{norep}) + D + R^{D}{}_i^A \right. \\ \left. + \min \left\{ T_{NC}^{A,rep}(i,j-1), T_{NC}^{A,norep}(i,j-1) \right\} + T^{A,norep}(j \mid i) \right) \\ \qquad + e^{-\lambda_{ind}^F p(T_j^{norep} + V_j^{norep})} \left(T_j^{norep} + V_j^{norep} + \left(1 - e^{-\lambda_{ind}^S p T_j^{norep}}\right) (D + R^{M}{}_i^A \right. \\ \left. + \min \left\{ T_{NC}^{A,rep}(i,j-1), T_{NC}^{A,norep}(i,j-1) \right\} + T^{A,norep}(j \mid i)) \right) \\ . \end{array}$$

The term in $e^{-\lambda_{ind}^F p(T_j^{norep} + V_j^{norep})}$ represents the case without fail-stop error, where the execution time is simply $T_j^{norep} + V_j^{norep}$. If a silent error is detected after the verification, we pay a downtime

and a memory recovery (with a cost depending on A). Next, we need to re-execute all the tasks since the last checkpoint (T_i to T_{j-1}) and take the minimal value obtained out of the execution where T_{j-1} is replicated or not; finally, we execute T_j again (with a time $T^{A,norep}(j \mid i)$) from last checkpoint. When a fail-stop error strikes, we account for $T_{lost}^{norep}(T_j^{norep} + V_j^{norep})$, the time lost within T_j, and whose value is given by Equation (5). Then we pay a downtime and a disk recovery (with a cost depending on A). Finally, we re-execute all the tasks from last checkpoint and that is similar to the previous case.

The formula is similar with replication of T_j, where the probability of error accounts for the fact that we need to recover only if both replicas fail for the fail-stop errors and accounts for the number of living replicas in the case where a silent error is detected (see Section 3.2 for the details):

$$
\begin{aligned}
T^{A,rep}(j \mid i) = & \left(1 - e^{-\frac{\lambda_{ind}^F p (T_j^{rep} + V_j^{rep})}{2}}\right)^2 \left(T_{lost}^{rep}(T_j^{rep} + V_j^{rep}) + D + R_i^{DA}\right. \\
& \left. + \min\left\{T_{NC}^{A,rep}(i, j-1), T_{NC}^{A,norep}(i, j-1)\right\} + T^{A,rep}(j \mid i)\right) \\
& + \left(1 - \left(1 - e^{-\frac{\lambda_{ind}^F p (T_j^{rep} + V_j^{rep})}{2}}\right)^2\right)\left(T_j^{rep} + V_j^{rep}\right) \\
& + \left((1 - e^{-\frac{\lambda_{ind}^F p (T_j^{rep} + V_j^{rep})}{2}}) e^{-\frac{\lambda_{ind}^F p (T_j^{rep} + V_j^{rep})}{2}} (1 - e^{-\frac{\lambda_{ind}^S p T_j^{rep}}{2}})\right. \\
& \left. + e^{-\lambda_{ind}^F p (T_j^{rep} + V_j^{rep})} (1 - e^{-\frac{\lambda_{ind}^S p T_j^{rep}}{2}})^2\right)\left(D + R_i^{MA}\right. \\
& \left. + \min\left\{T_{NC}^{A,rep}(i, j-1), T_{NC}^{A,norep}(i, j-1)\right\} + T^{A,rep}(j \mid i)\right).
\end{aligned}
$$

Note that the value of $T_{lost}^{rep}(T_j^{rep})$ is given by Equation (8). Overall, we need to compute the $O(n^2)$ intermediate values $T^{A,B}(j \mid i)$ and $T_{NC}^{A,B}(i, j)$ for $1 \leq i, j \leq n$ and $A, B \in \{rep, norep\}$, and each of these take constant time. There are $O(n)$ values $T_{opt}^{A}(i)$, for $1 \leq i \leq n$ and $A \in \{rep, norep\}$, and these perform a minimum over at most $6n$ elements, hence they can be computed in $O(n)$. The overall complexity is therefore $O(n^2)$. □

5 Experiments

In this section, we evaluate the advantages of adding replication to checkpointing in the presence of both, fail-stop and silent errors. We point out that the simulator that implements the proposed DP algorithm is publicly available at `http://graal.ens-lyon.fr/~yrobert/chainsrep.zip` so that interested readers can instantiate their preferred scenarios and repeat the same simulations for reproducibility purpose.The code is in C++ and does not use any library other than the STL.

We start by assessing scenarios with *fail-stop errors only* in Section 5.1. We first describe the evaluation framework in Section 5.1.1, then we compare *checkpoint with replication* to *checkpoint only* in Section 5.1.2. In Section 5.1.3, we assess the impact of the different model parameters on the performance of the optimal strategy. Finally, Section 5.1.4 compares the performance of the optimal solution to alternative sub-optimal solutions.

Then we assess scenarios with both fail-stop and silent errors in Section 5.2. We first describe the few modifications of the evaluation framework in Section 5.2.1, then we compare *checkpoint with replication* to *checkpoint only* in Section 5.2.2. Finally, Section 5.2.3 assesses the impact of the different model parameters on the performance of the optimal strategy.

5.1 Scenarios with fail-stop errors only

5.1.1 Experimental setup

We fix the total work in the chain to $W = 10,000$ seconds. This value does not really matter: instead, what is important is the duration of the tasks compared to the error rate. This is why we rely on five different work distributions, where all tasks are fully parallel ($\alpha_i = 0$):

- UNIFORM: every task i is of length $w_i = \frac{W}{n}$, i.e., identical tasks.

- INCREASING: the length of the tasks constantly increases, i.e., task T_i has length $w_i = i\frac{2W}{n(n+1)}$.
- DECREASING: the length of the tasks constantly decreases, i.e., task T_i has length $w_i = (n-i+1)\frac{2W}{n(n+1)}$.
- HIGHLOW: the chain is formed by long tasks followed by short tasks. The long tasks represent 60% of the total work and there are $\lceil \frac{n}{10} \rceil$ such tasks. Short tasks represent the remaining 40% of the total work and consequently there are $n - \lceil \frac{n}{10} \rceil$ small tasks.
- RANDOM: task lengths are uniformly chosen at random between $\frac{W}{2n}$ and $\frac{3W}{2n}$. If the total work of the first i tasks reaches W, the weight of each task is multiplied by $\frac{i}{n}$ so that we can continue adding the remaining tasks.

Experiments with increasing sequential part (α_i) for the tasks are available in the companion research report [4]. Setting $\alpha_i = 0$ amounts to being in the worse possible case for replication, since the tasks will fully benefit of having twice as much processors when not replicated.

For simplicity, we assume that checkpointing costs are equal to the corresponding recovery costs, assuming that read and write operations take approximately the same amount of time, i.e., $R_{i+1}^{Dnorep} = C_i^{norep}$. For replicated tasks, we set $C_i^{rep} = \alpha C_i^{norep}$ and $R_i^{Drep} = \alpha R_i^{Dnorep}$, where $1 \leq \alpha \leq 2$, and we assess the impact of parameter α in Section 5.2.3. In the following experiments, we measure the performance of a solution by evaluating the associated normalized expected makespan, i.e., the expected execution time needed to compute all the tasks in the chain, with respect to the execution time without errors, checkpoints, or replicas.

5.1.2 Comparison to checkpoint only

We start with an analysis of the solutions obtained by running the optimal dynamic programming (DP) algorithm CHAINSREPCKPT on chains of 20 tasks for the five different work distributions described in Section 5.1.1. We also run a variant of CHAINSREPCKPT that does not perform any replication, hence using a simplified DP algorithm, that is called CHAINSCKPT.

We vary the fail-stop error rate $\lambda_{ind}^F p$ from 10^{-8} to 10^{-2}. Note that when $\lambda_{ind}^F p = 10^{-3}$, we expect an average of 10 errors per execution of the entire chain (neglecting potential errors during checkpoints and recoveries). The checkpoint cost $C_i^{norep} = a_i$ is constant per task (hence $b_i = c_i = 0$) and varies from $10^{-3}T_i^{norep}$ to $10^3 T_i^{norep}$. For replicated tasks, we set $\alpha = 1$ in this experiment, i.e., $C_i^{rep} = C_i^{norep}$ and $R_i^{Drep} = R_i^{Dnorep}$.

Figure 1 presents the results of these experiments for the UNIFORM distribution. We are interested in the number of checkpoints and replicas in the optimal solution. As the optimal solution may or may not contain checkpoints and replicas we distinguish 4 cases: *None* means that no task is checkpointed nor replicated, *Checkpointing Only* means that some tasks are checkpointed but no task is replicated, *Replication Only* means that some tasks are replicated, but no task is checkpointed, and *Checkpointing+Replication* means that some tasks are checkpointed and some tasks are replicated. First, we observe that when the checkpointing cost is less than or equal to the length of a task (on the left of the black line), the optimal solution does not use replication, except when the error rate becomes very high. However, if the checkpointing cost exceeds the length of one task (on the right of the black vertical bar), replication proves useful in some cases. In particular, when the fails-stop error rate $\lambda_{ind}^F p$ is medium to high (i.e., 10^{-6} to 10^{-4}), we note that only replication is used, meaning that no checkpoint is taken and that replication alone is a better strategy to prevent any error from stopping the application. When the error rate is the highest (i.e., 10^{-4} or higher), replication is added to the checkpointing strategy to ensure maximum reliability. It may seem unusual to use replication alone when checkpointing costs increase. This is because the recovery cost has to be taken into account as well, in addition to re-executing the tasks that have failed. Replication is added to reduce this risk: if successful, there is no recovery cost to pay for, nor any task to re-execute. Finally, note that for low error rates and low checkpointing costs, only checkpoints are used, because their cost is lower than the average re-execution time in case of error. We point out that similar results are obtained when using other work distributions (see the extended version [4]).

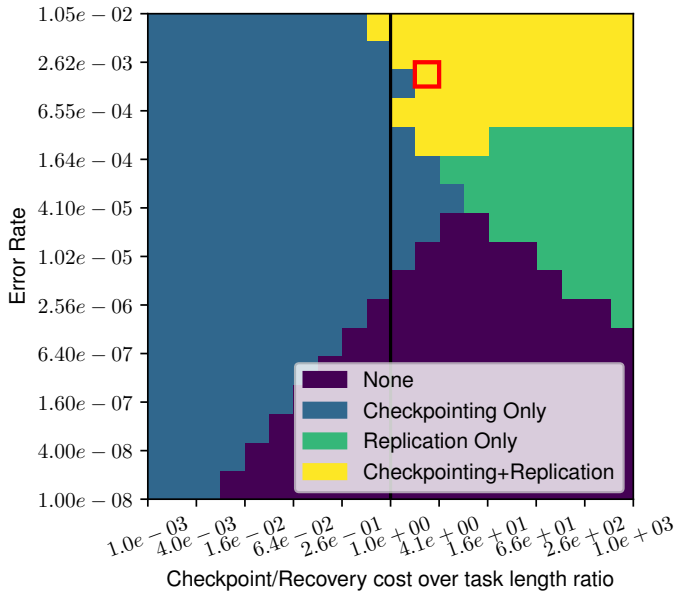
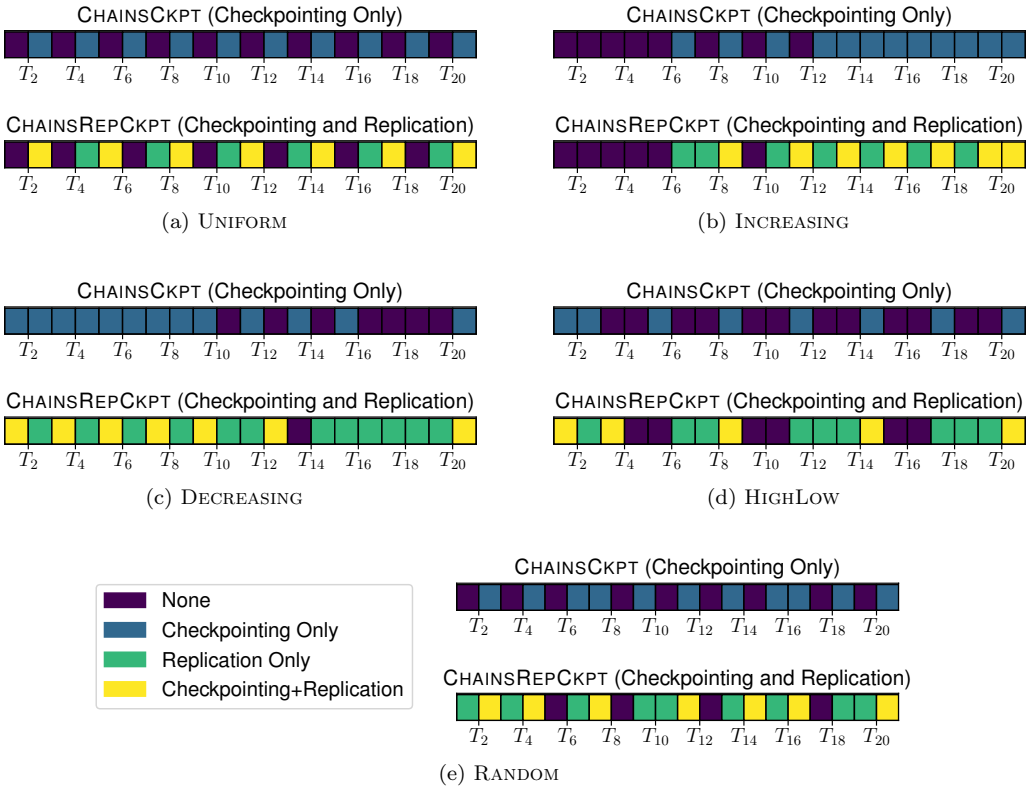


Figure 1: Impact of checkpoint/recovery cost and error rate on the usage of checkpointing and replication. Total work is fixed to 10,000s and is distributed uniformly among $n = 20$ tasks (i.e., $T_1 = T_2 = \cdots = T_{20} = 500$s). Each color shows the presence of checkpoints and/or replicas in the optimal solution. Results corresponding to the case highlighted with a red square are presented in Figure 2.

In the next experiment, we focus on scenarios where both checkpointing and replication are useful, i.e., we set the checkpointing cost to be twice the length of a task (i.e., $C_i^{norep} = a_i = 2T_i^{norep}$), and we set the fail-stop error rate $\lambda_{ind}^F p$ to 10^{-3}, which corresponds to the case highlighted by the red box in Figure 1. Figure 2 presents the optimal solutions obtained with the ChainsCkpt and ChainsRepCkpt algorithms for the Uniform, Increasing, Decreasing, HighLow and Random work distributions, respectively. First, for the Uniform work distribution, it is clear that the ChainsRepCkpt strategy leads to a decrease in the number of checkpoints compared to the ChainsCkpt strategy. Under the ChainsCkpt strategy, a checkpoint is taken every two tasks, while under the ChainsRepCkpt strategy, a checkpoint is instead taken every three tasks, while two out of three tasks are also replicated. Then, for the Increasing and Decreasing work distributions, the results show that most tasks should be replicated, while only the longest tasks are also checkpointed. A general rule of thumb is that *replication only* is preferred for short tasks while *checkpointing and replication* is reserved for longer tasks, where the probability of error and the re-execution cost are the highest. Finally, we observe a similar trend for the HighLow work distribution, where two of the first four longer tasks are checkpointed and replicated.

Figure 3 compares the performance of ChainsRepCkpt to the *checkpoint-only* strategy ChainsCkpt. First, we observe that the expected normalized makespan of ChainsCkpt remains almost constant at ≈ 4.5 for any number of tasks and for any work distribution. Indeed, in our scenario, checkpoints are expensive and the number of checkpoints that can be used is limited to ≈ 17 in the optimal solution, as shown in the middle plot. However, the ChainsRepCkpt strategy can take advantage of the increasing number of shorter tasks by replicating them. In this scenario (high error rate and high checkpoint cost), this is clearly a winning strategy. The normalized expected makespan decreases with increasing n, as the corresponding number of tasks that are replicated increases almost linearly. The ChainsRepCkpt strategy reaches a normalized makespan of ≈ 2.6 for $n = 100$, i.e., a reduction of 35% compared to the normalized expected makespan of the ChainsCkpt strategy. This is because replicated tasks tend to decrease the global probability of having an error, thus reducing even more the number of checkpoints needed as seen previously. Regarding the HighLow

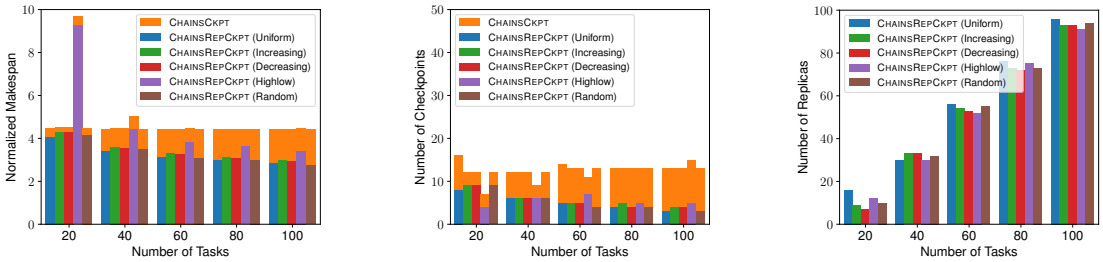


(a) Uniform (b) Increasing

(c) Decreasing (d) HighLow

(e) Random

Figure 2: Optimal solutions obtained with the ChainsCkpt algorithm (top) and the ChainsRepCkpt algorithm (bottom) for the five work distributions.



Uniform, Increasing, Decreasing, HighLow, Random distributions

Figure 3: Comparison of the ChainsCkpt and ChainsRepCkpt strategies for different numbers of tasks: impact on the makespan (left), number of checkpoints (middle) and number of replicas (right) with a fail-stop error rate of $\lambda_{ind}^{F} p = 10^{-3}$ and a constant chekpointing/recovery cost $C_i^{norep} = C_i^{rep} = 1000$s.

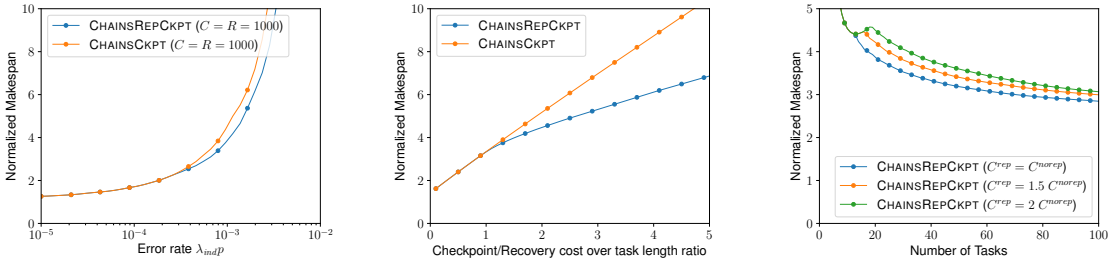


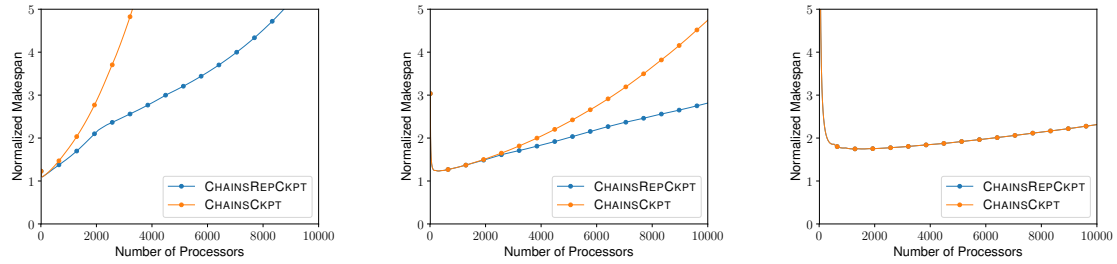
Figure 4: Impact of fail-stop error rate $\lambda_{ind}^{F} p$ (left), checkpoint cost (middle), and ratio α between the checkpointing cost for replicated task C_i^{rep} over non-replicated tasks C_i^{norep} (right).

work distribution, we observe a higher optimal expected makespan for both the ChainsCkpt and the ChainsRepCkpt strategies. Indeed, in this scenario, the first tasks are very long (60% of the total work), which greatly increases the probability of error and the associated re-execution cost.

5.1.3 Impact of error rate and checkpoint cost on the performance

Figure 4 shows the impact of three of the model parameters on the optimal expected normalized makespan of both ChainsCkpt and ChainsRepCkpt. First, we show the impact of the fail-stop error rate $\lambda_{ind}^{F} p$ on the performance. The ChainsRepCkpt algorithm improves the ChainsCkpt strategy for large values of $\lambda_{ind}^{F} p$: replication starts to be used for $\lambda_{ind}^{F} p > 2.6 \times 10^{-4}$ and it reduces the makespan by $\approx 16\%$ for $\lambda_{ind}^{F} p = 10^{-3}$ and by up to $\approx 40\%$ when $\lambda_{ind}^{F} p = 10^{-2}$, where all tasks are checkpointed and replicated.

Then, we investigate the impact of the checkpointing cost with respect to the task length. As



(a) $(100, 10000, 1)$ (b) $(100, 100000, 0.1)$ (c) $(100, 1000000, 0.01)$

Figure 5: Comparison of the ChainsCkpt and ChainsRepCkpt strategies for different numbers of processors, with different model parameter values for the checkpointing cost (a_i, b_i, c_i).

shown in Figure 1, replication is not needed for low checkpointing costs, i.e., when the checkpointing cost is between 0 and 0.8 times the cost of one task: in this scenario, all tasks are checkpointed and both strategies lead to the same makespan. When the checkpointing cost is between 0.9 and 1.6 times the cost of one task, ChainsRepCkpt checkpoints and replicates half of the tasks. Overall, the ChainsRepCkpt strategy improves the optimal normalized expected makespan by $\approx 11\%$ for a checkpointing cost ratio of 1.6, and by as much as $\approx 36\%$ when the checkpointing cost is five times the length of one task.

We now investigate the impact of the ratio between the checkpointing and recovery cost for replicated tasks and non-replicated tasks α and we present the results for $\alpha = 1$ ($C_i^{rep} = R_i^{Drep} = C_i^{norep} = R_i^{Dnorep}$), $\alpha = 1.5$ ($C_i^{rep} = R_i^{Drep} = 1.5C_i^{norep} = 1.5R_i^{Dnorep}$) and $\alpha = 2$ ($C_i^{rep} = R_i^{Drep} = 2C_i^{norep} = 2R_i^{Dnorep}$). As expected, the makespan increases with α, but it is interesting to note that the makespan converges towards a same lower-bound as the number of (shorter) tasks increases. As shown previously, when tasks are smaller, ChainsRepCkpt favors replication over checkpointing, especially when the checkpointing cost is high, which means less checkpoints, recoveries and re-executions.

Finally, we evaluate the efficiency of both strategies when the number of processors increases. For this experiment, we instantiate the model using variable checkpointing costs, i.e., we do not use $b_i = c_i = 0$ anymore, so that the checkpointing/recovery cost depends on the number of processors. We set $n = 50$, $\lambda_{ind}^F = 10^{-7}$ and we make p vary from 10 to 10,000 (i.e., the global error rate varies between 10^{-6} and 10^{-3}). Figure 5 presents the results of the experiment using three different sets of values for a_i, b_i and c_i. We see that when b_i increases while c_i decreases, the replication becomes useless, even for the larger error rate values. However, when the term $c_i p$ becomes large in front of $\frac{b_i}{p}$, we see that ChainsRepCkpt is much better than ChainsCkpt, as the checkpointing costs tend to decrease, in addition to all the other advantages investigated in the previous sections. With $p = 10,000$, the three different experiments show an improvement of 80.5%, 40.7% and 0% (from left to right, respectively).

5.1.4 Impact of the number of checkpoints and replicas

Figure 6 shows the impact of the number of checkpoints and replicas on the normalized expected makespan for different checkpointing costs and fail-stop error rates $\lambda_{ind}^F p$ under the Uniform work distribution. We show that the optimal solution with ChainsRepCkpt (highlighted by the green box) always matches the minimum value obtained in the simulations, i.e., the optimal number of checkpoints, number of replicas, and expected execution times are consistent. In addition, we show that in scenarios where both the checkpointing cost and the error rate are high, even a small deviation from the optimal solution can quickly lead to a large overhead.

5.2 Scenarios with both fail-stop and silent errors

In this section we evaluate the power of replication in addition to checkpointing on platforms subject to both fail-stop and silent errors.

5.2.1 Experimental setup

All the model parameters are instantiated as before, with the following changes to account for the presence of silent errors. Unless stated otherwise, the fail-stop error rate has been set to 1.28e-3s^{-1} and the silent error rate has been set to 5.48e-3s^{-1}. The silent error rate has been computed from real measures [2]: we derived a non-corrected silent error rate per core of 5.48e-9s^{-1}. Similarly, the fail-stop error rate per core considered was 1.28e-9s^{-1}, which corresponds to a core lifetime of 25 years. Finally, we considered a platform of 1 million cores which tends to be the trend for current Top500 machines [42].

As for other parameters, we considered a verification cost of 1% of the corresponding task length. The cost of memory recovery was set to 5% of that of a disk recovery, considering an average between different measured values from [31].

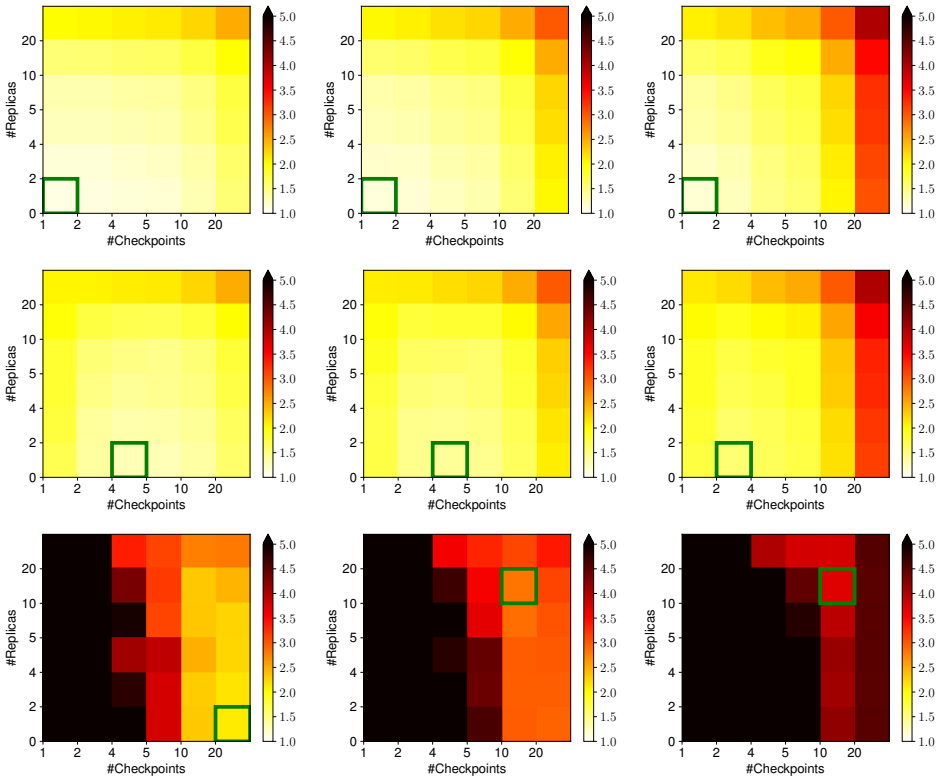


Figure 6: Impact of the number of checkpoints and replicas on the normalized expected makespan for fail-stop error rates of $\lambda_{ind}^{F} = 10^{-4}$ (top), $\lambda = 10^{-3}$ (middle) and $\lambda = 10^{-2}$ (bottom) and for checkpointing costs of $0.5 \times T_i^{norep}$ (left), $1 \times T_i^{norep}$ (middle) and $2 \times T_i^{norep}$ (right), with $C_i^{norep} = C_i^{rep}$ under UNIFORM work distribution. The optimal solution obtained with CHAINSREPCKPT always matches the minimum simulation value and is highlighted by the green box.

For simplicity, we assume that checkpointing costs are equal to the sum of the corresponding recovery costs, assuming that read and write operations take approximately the same amount of time, i.e., $R_{i+1}^{Dnorep} + R_{i+1}^{Mnorep} = C_i^{norep}$. For replicated tasks, we set $C_i^{rep} = \alpha C_i^{norep}$, $R_i^{Drep} = \alpha R_i^{Dnorep}$ and $R_i^{Mrep} = \alpha R_i^{Mnorep}$, where $1 \leq \alpha \leq 2$, and we assess the impact of parameter α in Section 5.2.3. As in the previous section, we measure the performance of a solution by evaluating the associated normalized expected makespan, i.e., the expected execution time needed to compute all the tasks in the chain, with respect to the execution time without errors, checkpoints, or replicas.

5.2.2 Comparison to checkpoint only

We start with an analysis of the solutions obtained by running the optimal dynamic programming (DP) algorithm CHAINSREPCKPT on chains with 20 tasks for the five different work distributions described in Section 5.1.1. We also run a variant of CHAINSREPCKPT that does not perform any replication, hence using a simplified DP algorithm, that is called CHAINSCKPT.

We vary the fail-stop error rate $\lambda_{ind}^{F}p$ from 10^{-8} to 10^{-2}, without changing the silent error rate λ_{ind}^{S}. The disk checkpoint/recovery cost is constant per task and varies from $10^{-3}T_i^{norep}$ to $10^{3}T_i^{norep}$ (hence, the memory checkpoint/recovery cost varies from $5 \times 10^{-5}T_i^{norep}$ to $50T_i^{norep}$). Overall, all checkpoints have a cost from $1.05 \times 10^{-3}T_i^{norep}$ to $1.05 \times 10^{3}T_i^{norep}$ as we always perform both types of checkpoints. For replicated tasks, we set $\alpha = 1$ in this experiment, i.e., $C_i^{rep} = C_i^{norep}$, $R_i^{Drep} = R_i^{Dnorep}$ and $R_i^{Mrep} = R_i^{Mnorep}$. In another experiment, we also make the silent error rate $\lambda_{ind}^{S}p$ vary from 10^{-8} to 10^{-2} without changing the fail-stop error rate of 1.28e-3, with the same range for the checkpoint cost.

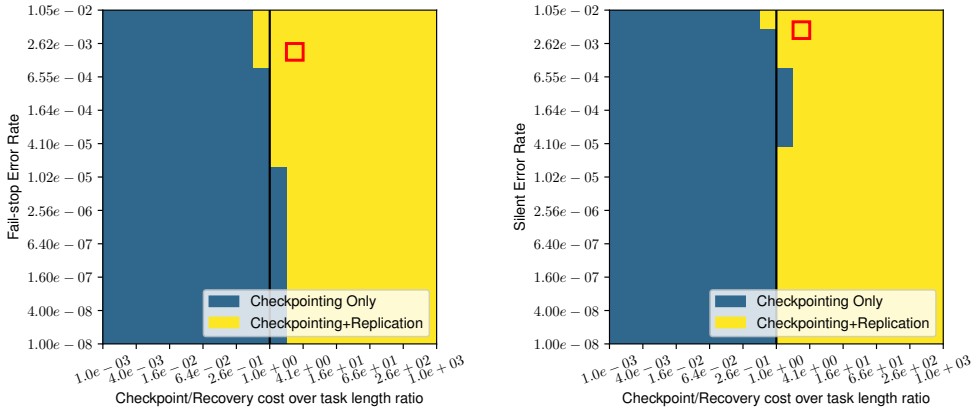


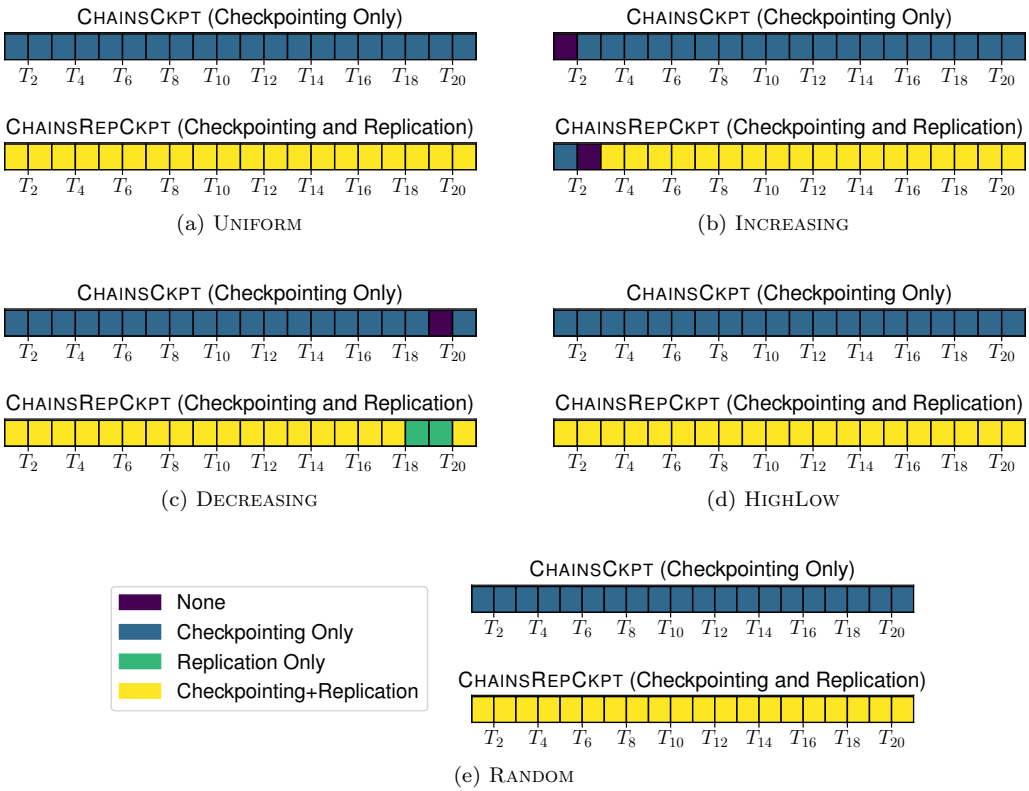
Figure 7: Impact of checkpoint/recovery cost and error rates on the usage of checkpointing and replication. Total sequential work is fixed to 10,000s and is distributed uniformly among $n = 20$ tasks (i.e., $T_1 = T_2 = \cdots = T_{20} = 500$s). Each color shows the presence of checkpoints and/or replicas in the optimal solution.

Figure 7 presents the results of these experiments for the UNIFORM distribution. The colors are the same as in Figure 1, with *Checkpointing Only* meaning that some tasks are checkpointed but no task is replicated and *Checkpointing+Replication* meaning that some tasks are checkpointed and some tasks are replicated. The left figure presents the results when the silent error rate is fixed but the fail-stop error rate varies. The right figure presents the results of the other experiment with a fixed fail-stop error rate and different silent error rates.

First, we observe that with silent errors, checkpointing becomes mandatory. Too many failures can strike during the execution, and checkpointing helps reducing the time spent on rollbacks and re-executions. However, as soon as the cost of a checkpoint exceeds the length of a task, replication becomes useful and this remains true even for low error rates. This holds for both fail-stop errors (left) and silent errors (right). There is one exception: when the fail-stop error rate is lower than 10^{-5} and the checkpointing cost is less than twice the length of a task, checkpoints are sufficient and is replication is not needed. Replication is overall not needed under good conditions, however for our real setup, indicated by the red box, using both checkpointing and replication is a better solution. We point out that similar results are obtained when using other work distributions (see the extended version [4]).

In the next experiment, we focus on scenarios where both checkpointing and replication are useful, i.e., we set the checkpointing cost to be twice the length of a task (i.e., $C_i^{norep} = a_i = 2T_i^{norep}$), keeping $\lambda_{ind}^F p = 1.28\text{e-}3$ and $\lambda_{ind}^S p = 5.48\text{e-}3$, for the fail-stop and silent error rates, respectively, which corresponds to the case highlighted by the red box in Figure 7. Figure 8 presents the optimal solutions obtained with the CHAINSCKPT and CHAINSREPCKPT algorithms for the UNIFORM, INCREASING, DECREASING, HIGHLOW and RANDOM work distributions, respectively. With two sources of errors, the solution is straightforward: almost every task must be checkpointed, with the exception of one (short) task for the DECREASING and INCREASING distributions. However almost every task is also replicated (20 tasks out of 20 for the UNIFORM distribution compared to only 13 in the experiments of Section 5.1.2), showing once more that replication grants better protection to failures even if it increases the failure-free execution time. Checkpoints are being taken the same way as in our previous experiments: long tasks are systematically checkpointed while shorter tasks are either unprotected or replicated, as can be seen with the first tasks of the INCREASING distribution and the last task of the DECREASING distributions.

Figure 9 compares the performance of CHAINSREPCKPT to the *checkpoint-only* strategy CHAINSCKPT with fail-stop and silent errors. First, we observe that long tasks, being more likely to fail than shorter tasks, introduce a high overhead. As a consequence, with 20 tasks, the normalized makespan is too high and the execution of such applications is not possible, independently of the work distribution and the chosen checkpointing strategy. With more tasks however, the CHAINS-

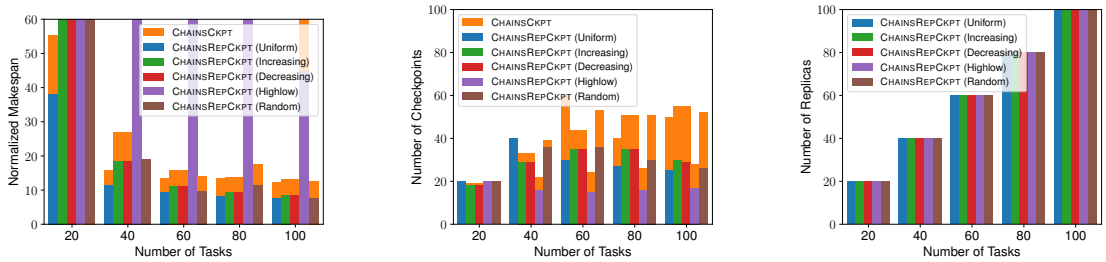


(a) Uniform (b) Increasing

(c) Decreasing (d) HighLow

(e) Random

Figure 8: Optimal solutions obtained with the ChainsCkpt algorithm (top) and the ChainsRepCkpt algorithm (bottom) for the five work distributions.



Uniform, Increasing, Decreasing, HighLow, Random distributions

Figure 9: Comparison of the ChainsCkpt and ChainsRepCkpt strategies for different numbers of tasks: impact on the makespan (left), number of checkpoints (middle) and number of replicas (right) with a fail-stop error rate of $\lambda^F_{ind}p$ = 1.28e-3, a silent error rate of $\lambda^S_{ind}p$ = 5.48e-3 and a constant chekpointing/recovery cost $C_i^{norep} = C_i^{rep} = 1000$s.

RepCkpt strategy always yield a shorter makespan compared to the ChainsCkpt strategy. For example, with 100 tasks, the normalized makespan obtained with the ChainsRepCkpt strategy is as high as ≈ 8.5 (and much more for the HighLow distribution), compared to ≈ 13 for ChainsCkpt. Indeed, with such high error rates, all tasks are replicated under the ChainsRepCkpt strategy, as can be seen on the right plot, but fewer tasks need to be be checkpointed (up to 50% fewer checkpoints with 100 tasks and the Uniform distribution).

The improvement is comparable to the 35% improvement observed with only fail-stop errors. Once again, replicated tasks tend to decrease the global probability of having an error, thus slightly reducing the number of checkpoints needed, while reducing the re-execution costs that can be very important with late-detected silent errors. Regarding the HighLow work distribution, we again observe a higher optimal expected makespan for both the ChainsCkpt and the ChainsRepCkpt strategies. Indeed, in this scenario, the first tasks are very long (60% of the total work), which greatly increases the error probability and the associated re-execution cost. Overall, for such applications on platforms subject to both, fail-stop and silent errors, replication appears to be mandatory and allows a reduction of the makespan of at least 30% if tasks are not too large (i.e. the probability of completing the task is not close to 1).

5.2.3 Impact of error rate and checkpoint cost on the performance

Figure 10 shows the impact of four of the model parameters on the optimal expected normalized makespan of both ChainsCkpt and ChainsRepCkpt, using the Uniform distribution. First, we show the impact of the fail-stop error rate $\lambda^F_{ind}p$ on the performance. The ChainsRepCkpt strategy always yields shorter makespans compared to the ChainsCkpt strategy. All tasks are always replicated, reducing the probability of having an error for each task, and each task is also checkpointed. The normalized makespan for ChainsCkpt is 19.5 for $\lambda^F_{ind}p = 10^{-5}$, compared to 19.2 for ChainsRepCkpt, i.e. a reduction of only 1.7%, but this goes up to 50.3 for $\lambda^F_{ind}p$ = 1.14e-3 compared to 35.2 when using replication, i.e. a reduction of 30%. The results are similar when we vary the silent error rate: when $\lambda^S_{ind}p = 10^{-5}$, ChainsRepCkpt results in a normalized makespan of 4.60 compared to 5.55 with ChainsCkpt, i.e. a reduction of 17%, and this goes up to more than 30% when $\lambda^S_{ind}p > 5 \times 10^{-3}$.

Then, we investigate the impact of the checkpointing cost with respect to the task length. The results are slightly different now that we have silent errors: ChainsCkpt and ChainsRepCkpt behave similarly only for small values of checkpoint cost. ChainsRepCkpt becomes better than ChainsCkpt for $C \geq 0.525$, thus reducing the makespan obtained using only checkpoints. Both strategies yield a makespan that increases linearly with the checkpointing cost, however the ChainsRepCkpt strategy needs less checkpoints, and the makespan increases slower. This means that the costlier the checkpoints the better the improvement thanks to replication. Overall, the execution under the ChainsRepCkpt strategy is 1.17 times faster than ChainsCkpt for a checkpointing cost

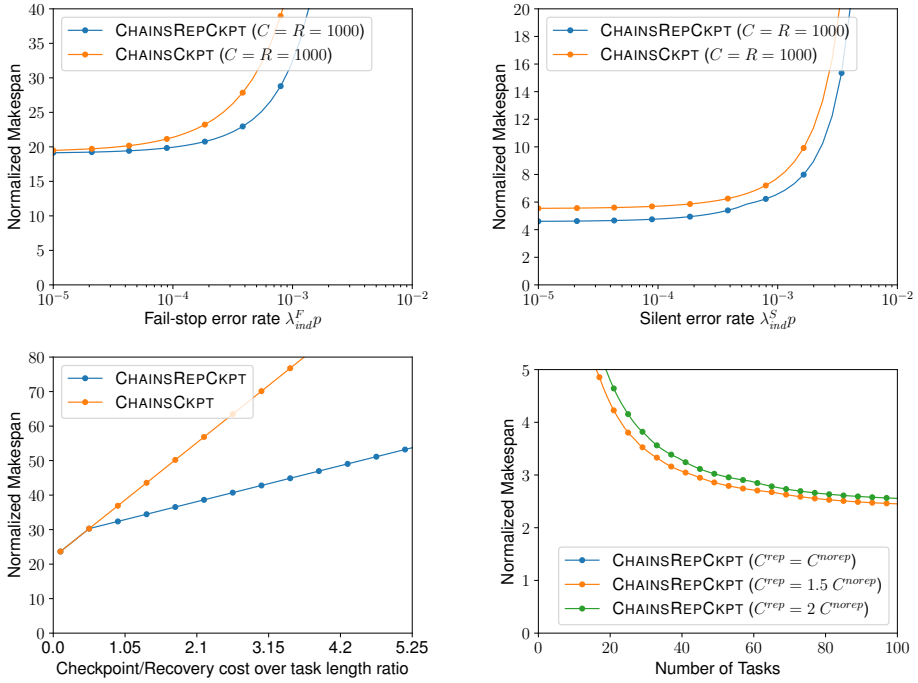
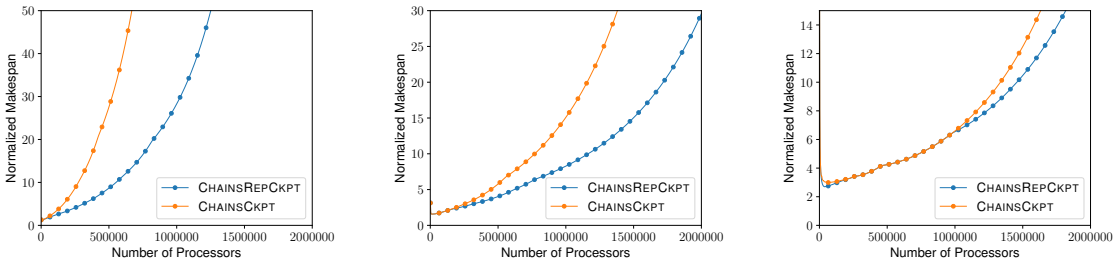


Figure 10: Impact of fail-stop error rate $\lambda^F_{ind}p$ (left), checkpoint cost (middle) and ratio α between the checkpointing cost for replicated task C_i^{rep} over non-replicated tasks C_i^{norep} (right) for the Uniform distribution.



(a) $(105, 1050000, 0.0105)$ (b) $(105, 10500000, 0.00105)$ (c) $(105, 105000000, 0.000105)$

Figure 11: Comparison of the ChainsCkpt and ChainsRepCkpt strategies for different numbers of processors, with different model parameter values for the checkpointing cost (a_i, b_i, c_i).

of $1.05T_i^{norep}$, 1.66 times faster for a checkpoint cost of $3.15T_i^{norep}$, and this goes up to 1.95 times faster when the checkpointing cost is $5.25T_i^{norep}$.

We now investigate the impact of the ratio α between the checkpointing and recovery cost for replicated tasks and non-replicated tasks and we present the results for $\alpha = 1$, $\alpha = 1.5$ and $\alpha = 2$. As expected, the makespan increases with α, but it is interesting to note that the makespan converges towards a same lower-bound as the number of (shorter) tasks increases. As shown previously, when tasks are smaller, ChainsRepCkpt favors replication over checkpointing, especially when the checkpointing cost is high, which means fewer checkpoints, recoveries and re-executions.

Finally, we evaluate the efficiency of both strategies when the number of processors increases. For this experiment, we instantiate the model using variable checkpointing costs, i.e., we do not use $b_i = c_i = 0$ anymore, so that the checkpointing/recovery cost depends on the number of processors. We set $n = 50$, $\lambda_{ind}^F = 1.28 \times 10^{-9}$, $\lambda_{ind}^S = 5.48 \times 10^{-9}$ and we make p vary from 1000 to 2,000,000 (i.e., the error rates vary between 10^{-6} and 10^{-2} approximately). Figure 11 presents the results of the experiment using three different sets of values for a_i, b_i and c_i. The trend is the same as previously with fail-stop errors: when b_i increases and c_i decreases, the advantage of using replication becomes less clear. However, on every plot, ChainsCkpt and ChainsRepCkpt grants the same makespan only when using a few cores. Every plot shows that, with the increasing number of cores on nowadays platforms, ChainsRepCkpt will behave better and better compared to ChainsCkpt. In particular, the improvement for each set of parameters (from left to right) is 69%, 30% and 0% for $p = 500000$, and is 76%, 60% and 16% for $p = 1500000$.

6 Related work

In this section, we discuss the work related to checkpointing and replication. Each of these mechanisms has been studied for coping with fail-stop errors and/or with silent errors. The present work combines checkpointing and replication for linear workflows in the presence of fail-stop and silent errors.

6.1 Checkpointing

The de-facto general-purpose recovery technique in high-performance computing is checkpointing and rollback recovery [13, 20]. Checkpointing policies have been widely studied and we refer to [25] for a survey of various protocols.

For divisible load applications where checkpoints can be inserted at any point in the execution for a nominal cost C, there exist well-known formulas proposed by Young [46] and Daly [16] to determine the optimal checkpointing period. For applications expressed as linear workflows, such as considered in the present work, the problem of finding the optimal checkpointing strategy, i.e., of determining which tasks to checkpoint, to minimize the expected execution time, has been solved by Toueg and Babaoğlu [43].

Single-level checkpointing schemes suffer from the intrinsic limitation that the cost of checkpointing and recovery grows with the error probability, and becomes unsustainable at large scale [23, 8] (even with diskless or incremental checkpointing [34]). Recent advances in decreasing the cost of checkpointing include multi-level checkpointing approaches, or the use of SSD or NVRAM as secondary storage [11]. To reduce the I/O overhead, various two-level checkpointing protocols have been studied. Vaidya [44] proposed a two-level recovery scheme that tolerates a single node error using a local checkpoint stored on a partner node. If more than one error occurs during any local checkpointing interval, the scheme resorts to the global checkpoint. Silva and Silva [37] advocated for a similar scheme by using memory protected by XOR encoding to store local checkpoints. Di et al. [17] analyzed a two-level computational pattern, and proved that equal-length checkpointing segments constitute the optimal solution. Benoit et al. [7] relied on disk checkpoints to cope with fail-stop errors and used memory checkpoints coupled with error detectors to handle silent data corruptions. They derived first-order approximation formulas for the optimal pattern length as well as the number of memory checkpoints between two disk checkpoints. The present work employs single-level checkpointing (in memory or on stable storage) for individual tasks in linear workflows.

6.2 Replication

As mentioned earlier, this work only considers *task duplication*. *Triplication* [29] (three replicas per task) is also possible yet only useful with extremely high error rates, which are unlikely in HPC systems. The use of redundant MPI processes is analyzed in [12, 22, 23]. In particular, Ferreira et al. [23] studied the use 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 error detection, consistent message ordering among replicas, etc.), and a set of experimental and simulation results. Thread-level replication has also been investigated [47, 14, 35]. The present work targets selective task replication as opposed to full task replication in conjunction with selective task checkpointing to cope with fail-stop and silent errors and minimize makespan.

Partial redundancy was also studied (in combination with coordinated checkpointing) to decrease the overhead of full replication [18, 38, 39]. Adaptive redundancy is introduced in [24], where a subset of processes is dynamically selected for replication. Earlier work [3] considered replication in the context of divisible load applications. In the present work, task replication (including work and data) is studied in the context of linear workflows, which represent a harder case than that of divisible load applications as tasks cannot arbitrarily be divided and are executed non-preemptively.

Ni et al. [32] introduce process duplication to cope both with fail-stop and silent errors. Their pioneering paper contains many interesting results. It differs from this work in that they only consider perfectly parallel applications while we investigate herein per task speedup profiles that obey Amdahl's law. More recently, Subasi et al. [40] proposed a software-based selective replication of task-parallel applications with both, fail-stop and silent errors. In contrast, the present work (i) considers dependent tasks such as found in applications consisting of linear workflows; and (ii) proposes an optimal dynamic programming algorithm to solve the combined selective replication *and* checkpointing problem. Combining replication with checkpointing has also been proposed in [36, 49, 21] for HPC platforms, and in [27, 45] for grid computing.

7 Conclusion

In this work, we studied the combination of checkpointing and replication to minimize the execution time of linear workflows in environments prone to both fail-stop and silent errors. We introduced a sophisticated dynamic programming algorithm that solves the combined problem optimally, by determining which tasks to checkpoint and which tasks to replicate, in order to minimize the total execution time. This dynamic programming algorithm was validated through extensive simulations that reveal the conditions in which checkpointing, replication, or both lead to improved performance. We have observed that the gain over the checkpoint-only approach is quite significant, in particular when checkpointing is costly and error rates are high.

Future work will address workflows whose dependence graphs are more complex than linear chains of tasks. Although an optimal solution seems hard to reach, the design of efficient heuristics that decide where to locate checkpoints and when to use replication, would prove highly beneficial for the efficient and reliable execution of HPC applications on current and future large-scale platforms.

Acknowledgement

This work has been partially supported by the Swiss Platform for Advanced Scientific Computing (PASC) project SPH-EXA.

References

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

[2] L. Bautista-Gomez, F. Zyulkyarov, O. Unsal, and S. McIntosh-Smith. Unprotected computing: A large-scale study of dram raw error rate on a supercomputer. In Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC '16, pages 55:1–55:11, Piscataway, NJ, USA, 2016. IEEE Press.

[3] A. Benoit, F. Cappello, A. Cavelan, P. Raghavan, Y. Robert, and H. Sun. Identifying the right replication level to detect and correct silent errors at scale. In FTXS'2017, the Workshop on Fault-Tolerance for HPC at Extreme Scale, in conjunction with HPDC'2017. IEEE Computer Society Press, 2017.

[4] A. Benoit, A. Cavelan, F. Ciorba, V. L. Fèvre, and Y. Robert. Combining checkpointing and replication for reliable execution of linear workflows. Research report RR-9152, INRIA, 2018.

[5] A. Benoit, A. Cavelan, Y. Robert, and H. Sun. Assessing general-purpose algorithms to cope with fail-stop and silent errors. ACM Trans. Parallel Comput., 3(2):13:1–13:36, July 2016.

[6] A. Benoit, A. Cavelan, Y. Robert, and H. Sun. Assessing general-purpose algorithms to cope with fail-stop and silent errors. ACM Trans. Parallel Computing, 3(2), 2016.

[7] A. Benoit, A. Cavelan, Y. Robert, and H. Sun. Optimal resilience patterns to cope with fail-stop and silent errors. In IPDPS. IEEE, 2016.

[8] G. Bosilca, A. Bouteiller, E. Brunet, F. Cappello, J. Dongarra, A. Guermouche, T. Herault, Y. Robert, F. Vivien, and D. Zaidouni. Unified model for assessing checkpointing protocols at extreme-scale. Concurrency and Computation: Practice and Experience, 2013.

[9] E. S. Buneci. Qualitative Performance Analysis for Large-Scale Scientific Workflows. PhD thesis, Duke University, 2008.

[10] F. Cappello, A. Geist, W. Gropp, S. Kale, B. Kramer, and M. Snir. Toward Exascale Resilience. Int. J. High Performance Computing Applications, 23(4):374–388, 2009.

[11] F. Cappello, A. Geist, W. Gropp, S. Kale, B. Kramer, and M. Snir. Toward Exascale Resilience: 2014 update. Supercomputing frontiers and innovations, 1(1), 2014.

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

[13] K. M. Chandy and L. Lamport. Distributed snapshots: Determining global states of distributed systems. ACM Transactions on Computer Systems, 3(1):63–75, 1985.

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

[15] V. Cuevas-Vicenttín, S. C. Dey, S. Köhler, S. Riddle, and B. Ludäscher. Scientific workflows and provenance: Introduction and research opportunities. Datenbank-Spektrum, 12(3):193–203, 2012.

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

[17] S. Di, Y. Robert, F. Vivien, and F. Cappello. Toward an optimal online checkpoint solution under a two-level HPC checkpoint model. IEEE Trans. Parallel & Distributed Systems, 2016.

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

[19] E. Elnozahy and J. Plank. Checkpointing for peta-scale systems: a look into the future of practical rollback-recovery. IEEE Trans. Dependable and Secure Computing, 1(2):97–108, 2004.

[20] E. N. M. Elnozahy, L. Alvisi, Y.-M. Wang, and D. B. Johnson. A survey of rollback-recovery protocols in message-passing systems. ACM Computing Survey, 34:375–408, 2002.

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

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

[23] K. Ferreira, J. Stearley, J. H. I. Laros, R. Oldfield, K. Pedretti, R. Brightwell, R. Riesen, P. G. Bridges, and D. Arnold. Evaluating the viability of process replication reliability for exascale systems. In SC'11. ACM, 2011.

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

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

[26] G. Kandaswamy, A. Mandal, and D. A. Reed. Fault tolerance and recovery of scientific workflows on computational grids. In Proc. of CCGrid'2008, pages 777–782, 2008.

[27] T. Leblanc, R. Anand, E. Gabriel, and J. Subhlok. VolpexMPI: An MPI Library for Execution of Parallel Applications on Volatile Nodes. In 16th European PVM/MPI Users' Group Meeting, pages 124–133. Springer-Verlag, 2009.

[28] R. Lucas, J. Ang, K. Bergman, S. Borkar, W. Carlson, L. Carrington, G. Chiu, R. Colwell, W. Dally, J. Dongarra, et al. Top ten exascale research challenges. DOE ASCAC subcommittee report, pages 1–86, 2014.

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

[30] D. P. Mehta, C. Shetters, and D. W. Bouldin. Meta-Algorithms for Scheduling a Chain of Coarse-Grained Tasks on an Array of Reconfigurable FPGAs. VLSI Design, 2013.

[31] A. Moody, G. Bronevetsky, K. Mohror, and B. R. d. Supinski. Design, Modeling, and Evaluation of a Scalable Multi-level Checkpointing System. In SC. ACM, 2010.

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

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

[34] J. Plank, K. Li, and M. Puening. Diskless checkpointing. IEEE Trans. Parallel Dist. Systems, 9(10):972–986, 1998.

[35] M. W. Rashid and M. C. Huang. Supporting highly-decoupled thread-level redundancy for parallel programs. In Proc. HPCA'2008, pages 393–404. IEEE, 2008.

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

[37] L. Silva and J. Silva. Using two-level stable storage for efficient checkpointing. IEE Proceedings - Software, 145(6):198–202, 1998.

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

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

[40] O. Subasi, G. Yalcin, F. Zyulkyarov, O. Unsal, and J. Labarta. Designing and Modelling Selective Replication for Fault-Tolerant HPC Applications. In Proc. CCGrid'2017, pages 452–457, May 2017.

[41] D. Talia. Workflow Systems for Science: Concepts and Tools. ISRN Software Engineering, 2013.

[42] Top500. Top500 Supercomputer Sites. `http://www.top500.org`.

[43] S. Toueg and Ö. Babaoglu. On the optimum checkpoint selection problem. SIAM J. Comput., 13(3):630–649, 1984.

[44] N. H. Vaidya. A case for two-level distributed recovery schemes. SIGMETRICS Perform. Eval. Rev., 23(1):64–73, 1995.

[45] S. Yi, D. Kondo, B. Kim, G. Park, and Y. Cho. Using replication and checkpointing for reliable task management in computational grids. In SC. ACM, 2010.

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

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

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

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

[50] Z. Zheng, L. Yu, and Z. Lan. Reliability-aware speedup models for parallel applications with coordinated checkpointing/restart. IEEE Trans. Computers, 64(5):1402–1415, 2015.

[51] J. Ziegler, H. Muhlfeld, C. Montrose, H. Curtis, T. O'Gorman, and J. Ross. Accelerated testing for cosmic soft-error rate. IBM J. Res. Dev., 40(1):51–72, 1996.

[52] J. Ziegler, M. Nelson, J. Shell, R. Peterson, C. Gelderloos, H. Muhlfeld, and C. Montrose. Cosmic ray soft error rates of 16-Mb DRAM memory chips. IEEE Journal of Solid-State Circuits, 33(2):246–252, 1998.

[53] J. F. Ziegler, H. W. Curtis, H. P. Muhlfeld, C. J. Montrose, and B. Chin. IBM experiments in soft fails in computer electronics. IBM J. Res. Dev., 40(1):3–18, 1996.