SC24-W: Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis | 979-8-3503-5554-3/24/$31.00 ©2024 IEEE | DOI: 10.1109/SCW63240.2024.00072

# Checkpointing strategies for a fixed-length execution

Anne Benoit
*Laboratoire LIP, ENS Lyon & Inria, Lyon, France*
Institut Universitaire de France
anne.benoit@ens-lyon.fr

Lucas Perotin
*Laboratoire LIP, ENS Lyon & Inria, Lyon, France*
Vanderbilt University, TN, USA
lucas.perotin@ens-lyon.fr

Yves Robert
*Laboratoire LIP, ENS Lyon & Inria*
Lyon, France
yves.robert@ens-lyon.fr

Frédéric Vivien
*Laboratoire LIP, ENS Lyon & Inria*
Lyon, France
frederic.vivien@inria.fr

***Abstract*—This work considers checkpointing strategies for a parallel application executing on a large-scale platform whose nodes are subject to failures. The application executes for a fixed duration, namely the length of the reservation that it has been granted. We start with small examples that show the difficulty of the problem: it turns out that the optimal checkpointing strategy neither always uses periodic checkpoints nor always takes its last checkpoint exactly at the end of the reservation. Then, we introduce a dynamic heuristic that is periodic and decides for the checkpointing frequency based upon thresholds for the time left; we determine threshold times $T_n$ such that it is best to plan for exactly $n$ checkpoints if the time left (or initially the length of the reservation) is between $T_n$ and $T_{n+1}$. Next, we use time discretization and design a (complicated) dynamic programming algorithm that computes the optimal solution, without any restriction on the checkpointing strategy. Finally, we report the results of an extensive simulation campaign that shows that the optimal solution is far more efficient than the Young/Daly periodic approach for short or mid-size reservations.**

***Index Terms*—Checkpointing, failures, fail-stop errors, fixed-length execution, fixed-size reservation.**

## I. Introduction

This work considers checkpointing strategies for a parallel application executing on a large-scale platform whose nodes are subject to failures. The application executes for a fixed duration, namely the length $T$ of the reservation that it has been granted. The large-scale platform may well experience several failures[1] per day [7], [10], [14], [15], hence, the application must be checkpointed during the reservation. After each failure, the application is interrupted and must be restarted. Without checkpointing, all the work executed for the application is lost. With checkpoint-restart [12], [16], the execution can resume from the last checkpoint, after some downtime (enroll a spare to replace the faulty processor) and a recovery (read the checkpoint).

The last checkpoint within the reservation plays a particular role, because all the work executed after that checkpoint and until the end of the reservation will be lost. The last checkpoint saves the state of the application and is usually taken at the very end of the reservation. For long-running High-Performance Computing (HPC) applications, it is common practice to make a series of reservations of the required resource: the execution is split into multiple smaller reservations, and checkpoint-restart is used to save intermediate steps of computation at the end of each reservation. For each actual reservation, the job needs to be checkpointed before the reservation time has elapsed, otherwise the progress of the execution during the reservation will be lost.

The optimal checkpointing strategy to mitigate the impact of failures has received considerable attention, but without considering fixed-size reservations. The problem is usually stated as follows: consider a parallel application executing on a platform whose processors are subject to failures. The application has a fixed amount of work $W$ to execute. For simplicity, say $W$ is expressed in seconds so that we can speak of work or time indifferently. The user has scheduled a long reservation $T$ for this amount of work, typically over-estimating the duration of the reservation by a wide margin. In other words, we have $W \ll T$. The optimal checkpointing strategy is then to decide how frequently we should checkpoint, so that the expected total execution time $\mathbb{E}(W)$ is minimized. Note that the time left in the reservation after all the work $W$ is completed is usually not billed to the user. We call this problem *fixed-work checkpointing*.

There is a well-known trade-off for *fixed-work checkpointing*: taking too many checkpoints leads to a high overhead, especially when there are few failures, while taking too few checkpoints leads to a large re-execution time after each failure. The optimal strategy to minimize the expected execution time is known when failure inter-arrival times, or IATs for short[2], obey an Exponential distribution on each processor. In that case, the optimal checkpointing strategy is periodic, the optimal period is known and can be expressed with a complicated formula that uses the Lambert function [12]. Fortunately, the optimal period can be approximated by the famous Young/Daly formula as $W_{YD} = \sqrt{2\mu C}$ [8], [17],

[1]Failures are also called *fail-stop errors.*

[2]IATs are the times elapsed between two consecutive failure events (or until the first failure at the start of the application).

where $\mu$ is the application Mean Time Between Failures (MTBF) and $C$ is the checkpoint duration. The Young/Daly formula is widely used across a variety of applications and platforms (see [2] for a survey) and represents a major progress compared to naive strategies where each application would checkpoint, say, every hour, independently of the values of its MTBF $\mu$ and checkpoint duration $C$.

While the *fixed-work checkpointing* problem is well-understood, the problem to maximize the expected work that can be achieved during a fixed-length execution, namely the duration $T$ of a reservation, has never been investigated. We call this problem *fixed-time checkpointing*. This new problem is the dual of the previous one: instead of minimizing the expected time to execute a fixed amount of work, we aim at maximizing the expected work achieved during a fixed amount of time.

This work provides the first study of *fixed-time checkpointing*, which we show to be much more difficult than *fixed-work checkpointing*. We start with examples that show that the optimal checkpointing strategy neither always uses periodic checkpoints nor always takes its last checkpoint exactly at the end of the reservation. Then, we introduce a dynamic heuristic strategy that is periodic and decides the checkpointing frequency based upon thresholds for the time left; we determine threshold times $T_k$ such that it is best to plan for exactly $k$ checkpoints if the time left (or initially the length of the reservation) is between $T_k$ and $T_{k+1}$ (letting $T_1 = 0$). Hence, the heuristic uses one (final) checkpoint for $T \leq T_2$, then two checkpoints (one intermediate and one final) for $T_2 \leq T \leq T_3$, and so on. Next, we use time discretization and design a (complicated) dynamic programming algorithm that computes the optimal solution, without any restriction on the checkpointing strategy. Finally, we report the results of an extensive simulation campaign that shows that the optimal solution can be far more efficient than the Young/Daly periodic approach for short or mid-size reservations.

The rest of the paper is organized as follows. Section II reviews related work. Section III details the framework. Section IV provides examples that show the intrinsic difficulty of the problem. Section V is devoted to the dynamic heuristic based upon thresholds. Section VI deals with the discretization of the problem into time quanta, and introduces the dynamic programming algorithm that computes the optimal solution without any restriction on the strategy. Section VII reports simulation results for a comprehensive experimental campaign. Finally, Section VIII provides concluding remarks and directions for future work.

## II. Related work

We briefly survey related work in this section, first discussing checkpoint-restart techniques, then discussing fixed-length reservations.

### A. Checkpoint-restart

Checkpoint-restart is one of the most widely used strategies to deal with failures. Several variants of this policy have been studied; see [12] for an overview. The literature focused on *fixed-work checkpointing*, for which a natural strategy is to checkpoint periodically; one must decide how often to checkpoint, i.e., derive the optimal checkpointing period. An optimal strategy is defined as a strategy that minimizes the expectation of the execution time of the application. For a preemptible application, where checkpointing can occur at any time, the classical formula due to Young [17] and Daly [8] states that the optimal checkpointing period is $W_{YD} = \sqrt{2\mu C}$, where $\mu$ is the application MTBF and $C$ the checkpoint cost. This formula is a first-order approximation. For memoryless failures, Daly provides a second-order, more accurate, approximation in [8], while our previous work [6] provides the optimal value; both [8] and [6] use the Lambert function, whose Taylor expansion is key to assess the accuracy of the Young/Daly formula. The derivation in [6] is based on Equation (1) (see Section IV-A), a formula rediscovered ten years later, with a quite different proof based on a Markov model, in [15]. Finally, we point out that non-memoryless failures are more difficult to deal with for parallel applications, and we refer to the recent paper [4] for more details.

We point out that this work is agnostic of the checkpoint protocol in use. For instance, rather than coordinated checkpointing on stable storage, which is the choice by default, one could use in-memory checkpointing [9], [13], [18], also known as buddy checkpointing, which reduces checkpoint time at the price of increased memory requirements (several checkpoint copies must be stored in the memory of each processor). Note that this approach requires surviving processes to continue execution after a failure, necessitating the use of a fault-tolerant version of the MPI library implementing the User-Level Failure Mitigation (ULFM) extension to the MPI Standard [5]. Such a diskless checkpointing capability has been implemented over ULFM, for example in the Fault-Tolerant Programming Framework Fenix [11]. Altogether, using in-memory checkpointing only changes the value of $C$ in our study.

### B. Fixed-length reservations

Long-running HPC applications usually make a series of reservations of the required resources: the execution is split into multiple smaller reservations, and checkpoint-restart is used to save intermediate steps of computation at the end of each reservation. There are multiple advantages to this approach, but the main one is that it lowers the wait time of the application, as the job scheduler can easily place a smaller reservation. On some platforms, a maximum reservation time is imposed on applications, forcing applications that run longer than this maximum time to split their reservation and rely on a form of checkpoint-restart. These scenarios occur in large scale High Performance Computing (HPC) platforms as well as on the Cloud. For each actual reservation, the job needs to be checkpointed before the reservation time has elapsed; otherwise the progress of the execution during the reservation will be lost.

To the best of our knowledge, the fixed-time checkpointing problem, namely, dealing with the impact of failures within a fixed-length reservation, has never been addressed in the literature. This is somewhat surprising, because it is a very natural problem that must be addressed for any large-scale HPC application that is granted one or several reservations by the batch scheduler. The closest related work is our paper [1], where we consider a failure-free scenario: an application executes for a fixed-length reservation and checkpoints at the end. The checkpoint duration is a stochastic random variable that obeys some well-known probability distribution law, and the question is to decide when to checkpoint so that the expected work is maximized. Adding failures to the picture was suggested by an anonymous reviewer of [1].

## III. Framework

We consider a fixed-size reservation of length $T$. The parallel application executes and checkpoints to mitigate the impact of failures. There is always a final checkpoint, and the work after that checkpoint is lost.

*a) Failures:* Failures obey an Exponential distribution of parameter $\lambda$. The Mean-Time Between Failures (MTBF) is thus $\mu = \frac{1}{\lambda}$. If the parallel application enrolls $p$ processors, then $\mu = \frac{\mu_{ind}}{p}$, where $\mu_{ind}$ is the individual MTBF on each processor. Given an execution of duration $X$, $\mathbb{P}_{succ}(X) = e^{-\lambda X}$ is the probability of success of that execution, and $\mathbb{P}_{fail}(X) = 1 - \mathbb{P}_{succ}(X) = 1 - e^{-\lambda X}$ is the probability of (at least) one failure during that execution. Failures can strike during checkpoint and recovery, but not during downtime (otherwise we would include it in the recovery). The checkpoint cost is $C$, the recovery cost $R$, and the downtime $D$.

*b) Checkpointing strategy:* As stated above, the objective is to maximize the expected amount of work $\mathbb{E}(T)$ that can be achieved within the whole reservation of duration $T$. A checkpointing strategy is recursively defined as follows:

- Initially, the time left is $t_{left} = T$. The strategy decides how many checkpoints will be taken, and at which instants, if there is no failure during the whole execution. With $k$ checkpoints taken, let $t_{end}(i) \leq T$ be the completion time of checkpoint number $i$, with $1 \leq i \leq k$.
- If there is no failure up to time $t_{end}(k)$ (when the last checkpoint completes), then the work achieved by the strategy will be equal to $W = t_{end}(k) - kC$.
- On the contrary, if a (first) failure strikes at time $t \leq T$, let $\ell \leq k$ be the number of the last checkpoint that completed before the failure. The work done after time $t_{end}(\ell)$ and up to time $t$ is lost, and the work achieved by the strategy up to the failure will be equal to $W = t_{end}(\ell) - \ell C$.
- Now after the failure at time $t$, there is a downtime and a recovery; hence, the time left is $t_{left} = (T - t) - D - R$. if $t_{left} \geq C$, we call recursively the strategy and will add the work done then to $W$.

In other words, for any value $t_{left} \leq T$, a checkpointing strategy must decide how many checkpoints will be taken, and at which instants, if no failure strikes during the execution of length $t_{left}$. We point out that the strategy will not be the same for different values of $t_{left}$: the distances between consecutive checkpoints are recomputed after each failure: this is what renders the problem so difficult.

## IV. Difficulty of the problem

This section provides several examples and case-studies that demonstrate the difficulty of the problem.

### A. With a single checkpoint at the end of the reservation

The first example deals with a simple checkpointing strategy: given any value of time left $t_{left}$, the strategy is to always work up to the end and take a unique checkpoint at the end, at time $t_{left} - C$. We let $\mathbb{E}^{end}(T, 1)$ denote the expected amount of work achieved with this strategy. We cannot provide a closed-form expression for $\mathbb{E}^{end}(T, 1)$, but we provide a recursive formula. To do so, we need an auxiliary quantity $\mathbb{E}^{end}_R(T, 1)$, where the only difference with $\mathbb{E}^{end}(T, 1)$ is that we start the execution with a recovery.

To compute $\mathbb{E}^{end}_R(T, 1)$, we distinguish two cases:

- Either there is no failure, which happens with probability $\mathbb{P}_{succ}(T)$, and we work for $T - R - C$ seconds, accounting for the initial recovery and the final checkpoint
- Or there is a failure before time $T$, and we have a recursion depending upon the time $t$ at which the first failure strikes.

Overall, $\mathbb{E}^{end}_R(T, 1)$ can then be expressed as:

$$\begin{aligned}\mathbb{E}^{end}_R(T,1) &= e^{-\lambda T}(T - R - C) \\ &+ \int_{t=0}^{T-D-R-C} \lambda t \frac{e^{-\lambda t}}{1-e^{-\lambda T}} \mathbb{E}^{end}_R(T - t - D, 1)dt\end{aligned}$$

In the integral, we use the conditional density probability $\lambda t \frac{e^{-\lambda t}}{1-e^{-\lambda T}}$ that the first failure strikes at time $t$, given that we know that a failure will strike before time $T$. We cap the range of $t$ down to $T - D - R - C$ because no more work can be executed if there remains less than $D + R + C$ seconds after the failure.

Now the formula for $\mathbb{E}^{end}(T, 1)$ is quite similar:

$$\begin{aligned}\mathbb{E}^{end}(T,1) &= e^{-\lambda T}(T - C) \\ &+ \int_{t=0}^{T-D-R-C} \lambda t \frac{e^{-\lambda t}}{1-e^{-\lambda T}} \mathbb{E}^{end}_R(T - t - D, 1)dt\end{aligned}$$

and simply accounts for the absence of the recovery at the beginning of the execution.

Unfortunately, the formulae for $\mathbb{E}^{end}(T, 1)$ and $\mathbb{E}^{end}_R(T, 1)$ do not lead to a closed-form expression. This is in sharp contrast with the corresponding instance of the dual fixed-work problem, i.e., the computation of the expected time $\mathbb{E}(W)$ to execute a given amount of work $W$ followed by a checkpoint $C$. Indeed, the recursive formula for $\mathbb{E}(W)$ writes:

$$\begin{aligned}&\mathbb{E}(W) = e^{-\lambda(W+C)}(W + C) \\ &+(1 - e^{-\lambda(W+C)})(\mathbb{E}(T_{lost}(W + C)) + \mathbb{E}(T_{rec}) + \mathbb{E}(W)),\end{aligned} \quad (1)$$

where $\mathbb{E}(T_{lost}(W + C))$ is the expected time lost before the first failure at time $t$. Similarly, $\mathbb{E}(T_{rec})$ is the expected time for the recovery (recall that failures may strike during recovery). Skipping details (see [6]), we derive the closed-form formula

$$\mathbb{E}(W) = \left(\frac{1}{\lambda} + D\right) \frac{1}{\lambda} e^{\lambda R}(e^{\lambda(W+C)} - 1).$$

The key to solving Equation (1) is that the term $\mathbb{E}(W)$ is on both sides of the recursive equation: after a failure, we resume in the same state, i.e., with the same amount $W$ of work to execute. On the contrary, when computing $\mathbb{E}_R^{end}(T,1)$, we are left with fewer time left after a failure at time $t$ and call recursively for $\mathbb{E}_R^{end}(T-t-R)$. This is the key why computing $\mathbb{E}_R^{end}(T,1)$ (or $\mathbb{E}^{end}(T,1)$) is extremely difficult.

### *B. With a single checkpoint in a short reservation*

The second example shows that the final checkpoint should not always be taken at the very end of the reservation. We consider a short reservation $T$, say $T = 6$, with $D = 0$ and $C = R = 4$: there is time for only one (final) checkpoint, because $2C > T$. Also, no extra work can be achieved after a failure, whenever it strikes, because $R + C > T$.

We compare the strategy STRAT$_1$ where we checkpoint at the end, from $t = 2$ to $t = 6$, with the strategy STRAT$_2$ where we checkpoint earlier, from $t = 1$ to $t = 5$, The expected gain GAIN of STRAT$_1$ over STRAT$_2$ can be either nonnegative or nonpositive, and depends upon whether a failure strikes and when; each case is weighted by its probability to occur:

- If there is no failure, the gain is $\mathbb{P}_{succ}(6)\times 1$: two seconds of work have been saved for STRAT$_1$, only one second for STRAT$_2$;
- If the first failure strikes after $t = 5$, the gain is $\mathbb{P}_{succ}(5)\mathbb{P}_{fail}(1) \times (-1)$: it is negative since no work is saved for STRAT$_1$, versus one second of work saved for STRAT$_2$;
- If the first failure strikes before $t = 5$, the gain is 0: no work has been saved for either strategy, and there is no time left after downtime and recovery after the failure to take a checkpoint.

Altogether, the gain is:

$$\begin{aligned}\text{GAIN} &= e^{-6\lambda} - e^{-5\lambda}(1-e^{-\lambda}) = 2e^{-6\lambda} - e^{-5\lambda}\\ &= e^{\ln(2)-6\lambda} - e^{-5\lambda},\end{aligned}$$

which is negative when $\lambda > \ln(2)$. Hence, if $\lambda$ is large enough, it is better to checkpoint before the end of the reservation!

This example is quite simple, because no extra work can be achieved after a failure, so there is no recursive call, regardless of the checkpoint strategy. We deal with a more complicated example below.

### *C. With two checkpoints*

This third example, and the heuristic approach described in Section V, both involve checkpoint strategies with recursive calls after each failure, as long as there remains sufficient time left. To compare the performance of two different checkpoint strategies STRAT$_1$ and STRAT$_2$ for a reservation of size $T$, we compare the expected work that they achieve until the first failure, or until the end of the reservation if there is no failure. In other words, we compare the expected work until the first recursive call. This is because we can always assume that both strategies will proceed identically after the recursive call, i.e., for shorter reservations of size $t_{left} < T$. Indeed, assume (by contradiction) that STRAT$_1$ achieves more expected work than STRAT$_2$ until the first failure, but globally achieves less expected work than STRAT$_2$ until the end of the reservation. Then we could use STRAT$_1$ to improve STRAT$_2$ as follows: consider strategy STRAT$_3$, a modified version of STRAT$_2$, which starts the execution using the very same checkpoints as STRAT$_1$, and is identical to STRAT$_2$ after the first failure (if any); then STRAT$_3$ has better total performance than the initial STRAT$_2$, hence than STRAT$_1$ by assumption. This explains why we can always assume that both strategies will proceed identically after the first recursive call, and why our comparison metric is the gain achieved by one strategy compared to the other until the first failure, if any.

Now, we move to the example and show that we should not always use same-size segments when provisioning two checkpoints. Consider the following problem: given a reservation $T \geq 2C$, we provision two checkpoints, the first scheduled to end at time $\alpha(T)T$, and the second one scheduled to end at time $T$. Here, $\alpha(T)$ is a scalar that depends on $T$, and we ask whether it is optimal to have $\alpha(T) = \frac{1}{2}$, i.e., two segments of same size. We have two bounds, a lower bound $C \leq \alpha(T)T$ (so that the first segment is long enough to take a checkpoint) and an upper bound $\alpha(T)T \leq T - C$ (so that the second segment is long enough to take a checkpoint). Let STRAT$_2(\alpha(T))$ be the strategy with two checkpoints, the first ending at time $\alpha(T)T$, and the second ending at time $T$. We aim at computing the best value of $\alpha(T)$. As stated above, we compare the gain (positive or negative) GAIN$(\alpha(T))$ of STRAT$_2(\alpha(T))$ over STRAT$_1$, the strategy where we execute only a single checkpoint at the end of the reservation. Recall that the gain GAIN$(\alpha(T))$ is the difference of the expected work achieved until the first failure, if any.

The gain GAIN$(\alpha(T))$ of STRAT$_2(\alpha(T))$ over STRAT$_1$ is

- $\mathbb{P}_{succ}(T) \times (-C)$ if there is no failure, because there is one more checkpoint in STRAT$_2(\alpha(T))$;
- 0 if the first failure strikes before time $\alpha(T)T$: no work saved for either strategy;
- $\mathbb{P}_{succ}(\alpha(T)T)\mathbb{P}_{fail}((1-\alpha(T))T) \times (\alpha(T)T - C)$ if the first failure strikes after time $\alpha(T)T$: no work saved for STRAT$_1$ versus $\alpha(T)T - C$ seconds of work for STRAT$_2(\alpha(T))$.

Altogether, the gain is

$$\begin{aligned}\text{GAIN}(\alpha(T)) &= e^{-\lambda T}(-C)\\ &\quad + e^{-\lambda\alpha(T)T}(1-e^{-\lambda(1-\alpha(T))T})(\alpha(T)T - C)\\ &= e^{-\lambda\alpha(T)T}(\alpha(T)T - C) - e^{-\lambda T}\alpha(T)T\end{aligned}$$

Given $T$, we consider the function $f(\alpha) = e^{-\lambda\alpha T}(\alpha T - C) - e^{-\lambda T}\alpha T$ and differentiate:

$$\begin{aligned}f'(\alpha) &= -\lambda T e^{-\lambda\alpha T}(\alpha T - C) + e^{-\lambda\alpha T}T - e^{-\lambda T}T\\ &= T e^{-\lambda\alpha T}\left(1 - \lambda(\alpha T - C) - e^{-\lambda(1-\alpha)T}\right)\end{aligned}$$

Letting $g(\alpha) = 1 - \lambda(\alpha T - C) - e^{-\lambda(1-\alpha)T}$ and differentiating again:

$$g'(\alpha) = -\lambda T(1 + e^{-\lambda(1-\alpha)T}) < 0$$

Hence, $g$ is decreasing, and $f'$ is decreasing too (product of two decreasing functions). We have

$$f'(0) = T(\lambda C + 1 - e^{-\lambda T}) > 0$$
$$f'(1) = -\lambda T e^{-\lambda T}(T - C) < 0$$

Hence, $f'$ has a unique zero $\alpha_{opt}(T)$, which is the unique solution of the equation

$$1 = \alpha T - C - e^{-\lambda(1-\alpha)T}. \quad (2)$$

We see that $\alpha_{opt}(T)$ is not equal to $\frac{1}{2}$ in general, so that having two same-size segments is not optimal. The only case it is optimal is when $T$ and $\lambda$ satisfy to $1 = \frac{T}{2} - C - e^{-\lambda\frac{T}{2}}$. This is in sharp contrast with the dual fixed-work checkpointing problem, where having same-size segments is always optimal [12]. But we point out that when $\lambda \to 0$ and with $T = \Theta(\lambda^{-\frac{1}{2}})$, then the first order expansion of $\alpha_{opt}(T)$ does give $\alpha_{opt}(T) \to \frac{1}{2}$, which is comforting.

## V. Threshold-based heuristic

We introduce a dynamic heuristic that relies upon thresholds. This heuristic always uses same-size segments, each finishing with a checkpoint, and the last checkpoint is scheduled to complete exactly at the end of the reservation (or what remains of it). The major difficulty is to decide how many checkpoints to provision in case of a failure-free execution. The key of the heuristic is to provide a sequence of thresholds $(T_n)_{n\geq 1}$ so that we provision exactly $n$ checkpoints when the time left (initially the full length $T$ of the reservation) satisfies

$$T_n \leq t_{left} \leq T_{n+1}. \quad (3)$$

Intuitively, introducing such thresholds makes sense: the longer the reservation, the more checkpoints one should provision. Also, we could expect that the longer the reservation, the more work should the heuristic achieve in expectation. Unfortunately, this is not true: revisiting the example of Section IV-B, the heuristic achieves more expected work when $T = 5$ than when $T = 6$ whenever $\lambda > \ln(2)$. And the example of Section IV-C shows that relying on same-length segments is seldom optimal. Still, the heuristic is very natural, and we explain below how to determine the threshold sequence $(T_n)_{n\geq 1}$.

We construct the sequence inductively and start with $T_1 = 0$: for short reservations, we always need a checkpoint at the end. Now, given the value of the first $n$ thresholds $T_i$, for $1 \leq i \leq n$, consider a reservation of length

$$T \geq \max\left(T_n, (n+1)C\right). \quad (4)$$

We determine at which length $T_{n+1}$ it does become better to use $n+1$ checkpoints instead of $n$. To do so, we compare the gain (either positive, zero or negative) when using $(n+1)$ checkpoints (strategy $\text{STRAT}_{n+1}$) instead of using $n$ checkpoints (strategy $\text{STRAT}_n$). Recall that the gain compares the expected work until the first failure. Hence, for a reservation $T$ obeying Equation (4), we want to compare $\mathbb{E}(T, n+1)$, the expected work with $\text{STRAT}_{n+1}$, with $\mathbb{E}(T, n)$, the expected work with $\text{STRAT}_n$, until the first failure. We compute the difference $\text{GAIN}(T, n+1) = \mathbb{E}(T, n+1) - \mathbb{E}(T, n)$ via a case analysis, depending upon when the first failure (if any) strikes in the execution.

In Figure 1, we give an example for a reservation of length $T$, using either $n = 4$ or $n+1 = 5$ checkpoints. In the general case, because $n$ and $n+1$ are relatively prime, there are $n(n+1)$ chunks where the first failure may strike. For the example, there are 20 chunks. Altogether, there are several cases to analyze, depending upon which chunk is struck by the first failure, plus the last case without any failure. This last case is the easiest to analyze: with one more checkpoint for $\text{STRAT}_{n+1}$ than for $\text{STRAT}_n$, the gain is negative, and its expected value is $\mathbb{P}_{succ}(T) \times (-C)$.

Now, we discuss the gain for the case when the first failure strikes chunk number $s$, for $1 \leq s \leq n(n+1)$. Let $U = \frac{T}{n(n+1)}$ be the length of a chunk. We start with a lemma:

**Lemma 1.** *The probability that the first failure strikes chunk number $s \geq 2$ is $\mathbb{P}_{succ}((s-1)U) \times \mathbb{P}_{fail}(U)$. The probability that the first failure strikes a slice of $t$ consecutive chunks composed of chunks number $s$ to $s+t-1$ with $s \geq 1$ and $t \geq 2$ is $\mathbb{P}_{succ}((s-1)U) \times \mathbb{P}_{fail}(tU)$.*

*Proof.* This is a well-known result, due to the memoryless property of the Exponential probability distribution. It is based upon the fact that the probability that the first failure strikes a slice of $k$ consecutive chunks is the sum of the probabilities that the first failure strikes any of these chunks. □

Chunks can be partitioned into $2n$ slices $A_m$ and $B_m$ for $0 \leq m \leq n-1$:

- Slices appear in the order

$$A_0, B_0, A_1, B_1, \ldots, A_{n-1}, B_{n-1}.$$

- Slice $A_m$ is composed of chunks number $s$ with $m(n+1) < s \leq (m+1)n$: chunks after checkpoint number $m$ (or the beginning of the execution if $m = 0$) of strategy $\text{STRAT}_{n+1}$, and up to checkpoint number $m+1$ of strategy $\text{STRAT}_n$ (see Figure 1). In the figure, slice $A_0$ includes the first $n = 4$ chunks, and slice $A_1$ includes $n-1$ chunks after the first checkpoint in strategy $\text{STRAT}_{n+1}$, i.e., chunks 6 to 8. Slice $A_m$ is composed of $n - m$ chunks.
- Slice $B_m$ is composed of chunks number $s$ with $(m+1)n < s \leq (m+1)(n+1)$: chunks after checkpoint number $m+1$ of strategy $\text{STRAT}_n$ and up to checkpoint number $m+1$ of strategy $\text{STRAT}_{n+1}$. In Figure 1, slice $B_0$ includes chunk 5, and slice $B_1$ includes chunks 9 and 10. Slice $B_m$ is composed of $m+1$ chunks.

If the first failure strikes within slice $A_0$, both strategies lose everything and there is no difference. If the first failure strikes within slice $A_m$ for $m \geq 1$, then strategy $\text{STRAT}_n$ has saved more chunks than strategy $\text{STRAT}_{n+1}$, namely all the chunks of slice $B_{m-1}$. Hence, the gain is negative, and its expected value is

$$-\mathbb{P}_{succ}(m(n+1)U)\mathbb{P}_{fail}((n-m)U) \times (mU).$$

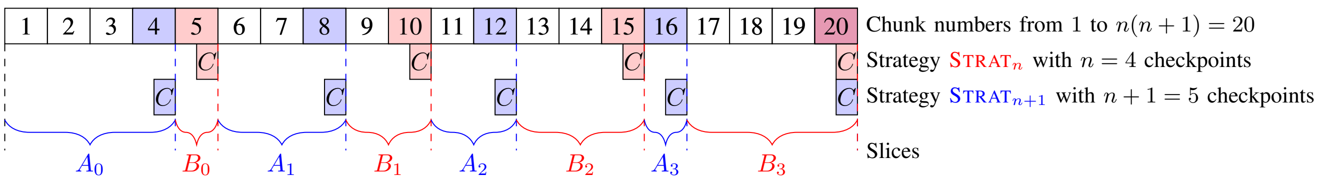
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Chunk numbers from 1 to $n(n+1) = 20$
Strategy $\textsc{Strat}_n$ with $n = 4$ checkpoints
Strategy $\textsc{Strat}_{n+1}$ with $n+1 = 5$ checkpoints
$C$
$A_0$ $B_0$ $A_1$ $B_1$ $A_2$ $B_2$ $A_3$ $B_3$
Slices

Fig. 1: Comparing strategies with $n$ and $n+1$ checkpoints for $n = 4$.

The probability is computed using the fact that slice $A_m$ starts at chunk $m(n+1)+1$, and has length $n-m$ chunks. The extra number of chunks for $\textsc{Strat}_n$ is given by the number $m$ of chunks of slice $B_{m-1}$. Indeed, since both strategies made the same number of checkpoints before the failure, the gain is exactly the difference between the completion time of the last checkpoint for both strategies, which corresponds to the number of chunks of slice $B_{m-1}$.

If the first failure strikes within slice $B_m$, then strategy $\textsc{Strat}_n$ has saved fewer chunks than strategy $\textsc{Strat}_{n+1}$, namely all the chunks of slice $A_m$, but had to do one less checkpoint. Hence, the expected value of the gain is $\mathbb{P}_{succ}((m+1)nU)\mathbb{P}_{fail}((m+1)U) \times ((n-m)U - C)$. The probability is computed using the fact that slice $B_m$ starts at chunk $m(n+1)+1$, and has length $m+1$ chunks. The extra number of chunks for $\textsc{Strat}_{n+1}$ is given by the number $n-m$ of chunks of slice $A_m$, but this time we need to deduce the additional checkpoint taken by $\textsc{Strat}_{n+1}$ at the end of $A_m$. Altogether, we have derived that

$$\begin{aligned}
&\textsc{Gain}(T, n+1) = \mathbb{E}(T, n+1) - \mathbb{E}(T, n)\\
&= -\mathbb{P}_{succ}(T) \times C\\
&\quad \text{(loss if no failure)}\\
&- \textstyle\sum_{m=1}^{n-1} \mathbb{P}_{succ}(m(n+1)U)\mathbb{P}_{fail}((n-m)U) \times (mU)\\
&\quad \text{(gain if first failure strikes slice } A_m)\\
&+ \textstyle\sum_{m=0}^{n-1} \mathbb{P}_{succ}((m+1)nU)\mathbb{P}_{fail}((m+1)U) \times ((n-m)U - C)\\
&\quad \text{(loss if first failure strikes slice } B_m)\ .
\end{aligned}$$

We are unable to prove that there exists a single solution $T_{n+1} \geq (n+1)C$ to the equation $\textsc{Gain}(T_{n+1}, n+1) = 0$, but we conjecture that this is always the case (as one can intuitively expect, there is no reason to use more checkpoints in a shorter reservation than in a longer one). In the experiments reported in Section VII, we have solved the equation numerically and we constructed a unique sequence of thresholds $(T_n)_{n\geq 1}$.

We proceed to a first-order approximation of $T_{n+1}$ when $\lambda \to 0$. As for the Young/Daly approximation [12], the failure-free overhead and the failure-induced overhead must be of the same order of magnitude for this approximation. The failure-free overhead is the relative cost of checkpoints $\Theta(\frac{C}{T})$. The failure-induced overhead is the re-execution cost and is of the order $\Theta(\lambda T)$: indeed, we re-execute a fraction of the reservation every time a failure occurs, which happens a fraction $\frac{1}{\lambda}$ (the MTBF) of the time. This gives that $T = \Theta(\frac{1}{\sqrt{\lambda}})$, hence, $T \to \infty$ when $\lambda \to 0$, but also $\lambda T = \Theta(\sqrt{\lambda}) \to 0$ when $\lambda \to 0$, and we can safely use the first order approximation $\mathbb{P}_{succ}(T) = e^{-\lambda T} = 1 - \lambda T + o(\frac{1}{\lambda})$. We rewrite the gain $\textsc{Gain}(T, n+1)$ as

$$\begin{aligned}
&\textsc{Gain}(T, n+1) = -C(1 - n(n+1)\lambda U)\\
&- \textstyle\sum_{m=1}^{n-1}(1 - m(n+1)\lambda U)(n-m)\lambda U m U\\
&+ \textstyle\sum_{m=0}^{n-1}(1 - (m+1)n\lambda U)(m+1)\lambda U((n-m)U - C) + O(\lambda)
\end{aligned}$$

We have $\lambda^2 U^2 = O(\lambda)$ while $\lambda^2 U = O(\sqrt{\lambda})$ because $T$ (and $U$) are $\Theta(\frac{1}{\sqrt{\lambda}})$. Keeping only first-order terms, we derive that

$$\begin{aligned}
\textsc{Gain}(T, n+1) &= -C + \tfrac{n(n+1)}{2}\lambda U^2 + O(\sqrt{\lambda})\\
&= -C + \tfrac{\lambda T^2}{2n(n+1)} + O(\sqrt{\lambda}),
\end{aligned}$$

which leads to the first-order approximation

$$T_{n+1} \approx \sqrt{\frac{2n(n+1)C}{\lambda}}. \tag{5}$$

It is interesting to compare with the Young/Daly formula:

- When $n = 1$; introducing the MTBF $\mu = \frac{1}{\lambda}$, we have $T_2 \approx \sqrt{4C\mu}$ while the optimal Young/Daly period is $W_{YD} = \sqrt{2C\mu}$: in limited time, better keep a unique checkpoint up to $T_2$, whose value is $\sqrt{2}$ larger than $W_{YD}$.
- More generally, the length of $n$ Young/Daly segments is $f(n) = n\sqrt{2\mu C}$; hence, a first approximation for $T_{n+1}$, the threshold limit between $n$ or $n+1$ segments, is the geometric mean of $f(n)$ and $f(n+1)$, which gives
$$\sqrt{f(n)f(n+1)} = \sqrt{n(n+1)2\mu C} \approx T_{n+1}.$$

## VI. Discretization with Time Quanta

In this section, we show that time discretization enables us to derive the optimal solution, without any restriction on the solution: we can indeed have segments with different lengths and we can checkpoint a few moments before the end of the reservation if it is better (in expectation) than working up to the very last moment. The main idea is to discretize time into small quanta, and to use dynamic programming to compute the best checkpointing strategy. Obviously, the smaller the quanta, the more accurate the results, but the more costly the dynamic programming algorithm.

Technically, we introduce a time quantum $u$, and we discretize time into quanta. This means that all quantities (reservation length, segment sizes, checkpoint and recovery times, downtime) are integer multiples of $u$, and that failures strike at the end of a quantum. More precisely, if a variable $y$ is defined in work units, $y^* = y/u$ is the corresponding number of quanta, which we always suppose integer. The reservation length becomes the total number of quanta $T^* = T/u$, the checkpoint time is $C^* = C/u$ quanta, the recovery time is $R^* = R/u$ quanta and the downtime $D^* = D/u$ quanta. The probability that a failure happens during (at the end of) quantum $i$, $1 \leq i \leq T^*$, is $p_i^* = \int_{(i-1)u}^{iu} \lambda e^{-\lambda x} dx = e^{-\lambda(i-1)u} - e^{-\lambda iu}$. We denote the probability that a failure happens during the next $j$ quanta of execution as $\mathbb{P}^*_{fail}(j) = 1 - e^{-\lambda ju}$. We define $\mathbb{P}^*_{succ}(j) = e^{-\lambda ju}$ similarly. And we retrieve that the probability that the first failure strikes during quantum $i$ is $\mathbb{P}^*_{succ}(i-1)\mathbb{P}^*_{fail}(1) = p_i^*$. This discretization restricts the search for an optimal execution to a finite set of possible executions. The trade-off is that a smaller value of $u$ leads to a more accurate solution, but also to a higher number of states in the algorithm, hence, to a higher computing time.

To express the dynamic programming algorithm, we need to distinguish whether the current segment starts with a recovery or not. Let $\mathbb{E}(n, k, \delta)$ be the optimal work that can be achieved (in expectation) during $n$ quanta and when taking exactly $k$ checkpoints initially, with $\delta = 0$ if there is no initial recovery with the work, and $\delta = 1$ otherwise. Let $k_{max} = \lfloor \frac{T^*}{C^*} \rfloor$ be the maximum number of checkpoints that can be taken within $T^*$ quanta, i.e., if we checkpoint all the time. The objective is to use a dynamic programming algorithm to determine

$$\mathbb{E}_{opt}(T^*) = \max_{1 \leq k \leq k_{max}} \mathbb{E}(T^*, k, 0). \quad (6)$$

In Equation (6), we have $k \geq 1$ because we need at least one checkpoint. We start with $k = 1$, i.e., having exactly one checkpoint. We derive the recursion formula, where $i$ is the index of the quantum when the (unique) checkpoint completes:

$$\begin{aligned}\mathbb{E}(n, 1, \delta) &= \max_{\delta R^* + C^* + 1 \leq i \leq n} \mathbb{P}^*_{succ}(i)(i - C^* - \delta R^*) \\ &+ \textstyle\sum_{f=1}^{i-D^*-R^*-C^*} p_f^* \mathbb{E}(n - f - D^*, 1, 1).\end{aligned} \quad (7)$$

The formula reads as follows: we look for every quantum $i$ where to complete the checkpoint, meaning that if no failure strikes, the actual execution would have last $i - C^*$ quanta, minus again $R^*$ quanta if there was an initial recovery; this gives the lower bound for $i$, because we aim at doing at least one quantum of work. Now, if the execution does succeed, which happens with probability $\mathbb{P}^*_{succ}(i)$, the remaining quanta will be lost, since we cannot take a second checkpoint by hypothesis. However, if the first failure strikes during quantum $f$ (which happens with probability $p_f^*$), we have to restart from scratch. There will remain $n - f - D^*$ quanta for execution, starting with a recovery, and still aiming at taking one checkpoint. The upper bound for $f$ comes from the fact that we will need a downtime, a recovery and a checkpoint before saving any work.

To understand how the algorithm is initialized, we point out that we have

$$\mathbb{E}(n, 1, 0) = \max_{C^*+1 \leq i \leq n} \mathbb{P}^*_{succ}(i)(i - C^*)$$

for $n \leq 1 + C^* + D^* + R^*$. Indeed, without an initial recovery, we try all possible quanta $i$ where to end the checkpoint. If there is a failure, we will need a downtime, a recovery and a checkpoint before saving any more work; in the best case the failure strikes during the first quantum, hence, the upper bound for $n$. The case with an initial recovery is similar, we have $\mathbb{E}(n, 1, 1) = \max_{R^*+C^*+1 \leq i \leq n} \mathbb{P}^*_{succ}(i)(i - C^* - R^*)$ for $n \leq 1 + C^* + D^* + R^*$.

**Algorithm 1:** Dynamic programming algorithm

```
1 for $k = 1$ to $k_{max}$ do
2     for $n = 1$ to $T^*$ do
3         Compute $\mathbb{E}(n,k,0)$ and $\mathbb{E}(n,k,1)$
4         using Equation (7) if $k = 1$,
5         and Equation (8) if $k \geq 2$
```

In fact, we can use the simpler initialization

| | |
|---|---|
| $\mathbb{E}(n, 1, 0) = 0$ | for $n \leq C^*$ (where possibly $n \leq 0$) |
| $\mathbb{E}(n, 1, 1) = 0$ | for $n \leq R^* + C^*$ (where possibly $n \leq 0$) |

and recursively compute the values $\mathbb{E}(n, 1, 0)$ and $\mathbb{E}(n, 1, 1)$ for larger values of $n$ using Equation (7).

We now detail the recursion for arbitrary values of the number $k \geq 2$ of checkpoints:

$$\begin{aligned}&\mathbb{E}(n, k, \delta) = \\ &\max_{\delta R^* + C^* + 1 \leq i \leq n-(k-1)C^*} \Big(\mathbb{P}^*_{succ}(i)\big(i - C^* - \delta R^* + \mathbb{E}(n-i, k-1, 0)\big) \\ &\qquad + \textstyle\sum_{f=1}^{i} p_f^* \max_{1 \leq m \leq k} \mathbb{E}(n - f - D^*, m, 1)\Big).\end{aligned} \quad (8)$$

The main differences with the case $k = 1$ are the following:

- After completing a checkpoint at quantum $i$, if no failure has struck by then, there remains the possibility to save more work, because we are still planning for $k - 1$ more checkpoints, hence, the recursive call to $\mathbb{E}(n-1, k-1, 0)$.
- On the contrary, if a failure has struck at quantum $f$, there is no reason to continue with $k$ checkpoints, we can try using fewer checkpoints instead, hence the recursive call to the maximum value $\mathbb{E}(n - f - D^*, m, 1)$ achieved with $m$ checkpoints, where $1 \leq m \leq k$.

The initialization now writes exactly as for $k = 1$:

| | |
|---|---|
| $\mathbb{E}(n, k, 0) = 0$ | for $n \leq kC^*$ (where possibly $n \leq 0$) |
| $\mathbb{E}(n, k, 1) = 0$ | for $n \leq R^* + kC^*$ (where possibly $n \leq 0$) |

Algorithm 1 informally presents the dynamic programming algorithm. Its complexity is $O((T^*)^2(k_{max})^2)$ because there are $k_{max}T^*$ iterations, each calling up to $k_{max}T^*$ already computed values.

## VII. Experiments

In this section, we assess through simulations the performance of four checkpointing strategies:

- DynamicProgramming is the quantum-based dynamic programming algorithm (Algorithm 1).
- FirstOrder uses the first-order approximation of the thresholds $T_n$ defined by Equation (5).

- NumericalOptimum uses a numerical approximation of the thresholds $T_n$, with a numerical search for the zero of $\text{Gain}(T, n)$.
- YoungDaly is the baseline reference, which takes checkpoints periodically following the Young/Daly formula $W_{YD}$. If the remaining reservation length after the last checkpoint or last recovery after a failure is shorter than $W_{YD}$, then a checkpoint is taken right at the end of the reservation.

## A. *Simulation framework*

To assess the performance of the different checkpointing policies, we considered the following set of parameters:

- $C \in \{10, 20, 40, 80, 160\}$, and $R = C$;
- $D \in \{0, 5\}$;
- $\lambda \in \{0.01, 0.001, 0.0001\}$;
- $T \in [C, 2000\}$.

All these values are expressed in an arbitrary time-unit (seconds, minutes or hours), thereby allowing for variation in the granularity of the scenarios.

For each combination of parameter values, we randomly generate 1000 failure traces. On any given instance, the work achieved by any heuristic is upper bounded by $T - C$. In every figure, we report the *proportion of work* achieved by each heuristic, i.e., the amount of work achieved by this heuristic divided by the $T - C$ upper bound. Hence, the proportion of work takes values between 0 and 1, and the higher the better.

We first report performance for DynamicProgramming when the quantum is set at 1. Then we study the influence of the quantum value.

## B. *Simulation results*

Due to space limitations, we only present here a subset of the results. Complete results can be found in the companion research report [3]. In Figure 2, we compare the performance of the different strategies when $\lambda = 0.001$. In Figure 3, we illustrate an extreme case where $C \in \{80, 160\}$ and $\lambda = 0.0001$. For both figures, we use $D = 0$, since the value of $D$ has no significant impact on the performance of all strategies (see [3] for details).

Below is a summary of the main observations from these figures, which hold true for the entire simulation campaign (see [3]):

- NumericalOptimum delivers better, or equivalent, performance than FirstOrder. This is natural because NumericalOptimum uses a finer approximation for the thresholds $T_n$. In most cases, the difference is negligible. It can nevertheless be easily spotted on Figure 3 when $C = 160$.
- DynamicProgramming delivers better, or equivalent, performance than NumericalOptimum. If we temporarily forget about the issue of the quantum size (which will be discussed later), DynamicProgramming is supposed to achieve better results than NumericalOptimum because it searches for the best solution in a larger set of potential solutions. Indeed, DynamicProgramming is not mandated to complete its last checkpoint exactly at the end of the reservation. Once again, in most cases, the difference of performance with NumericalOptimum is negligible. It can nevertheless be seen on Figure 3 when $C = 160$. Note that, even on this figure, when $C = 80$ and when $C = 160$, there are reservation lengths for which NumericalOptimum achieves a better performance than DynamicProgramming. This can be explained by the limitation of the quantization and/or by the variability of random instances.
- The higher the failure rate and the shorter the reservation, the more significant the differences between the strategies. On the contrary, when the reservation length grows towards infinity or when the failure rate is small, then the problem more closely resembles that with an infinite reservation, and the approximations made by YoungDaly and FirstOrder are fully justified. When the length of the reservation grows, the performance of all strategies converge toward the asymptotic performance of YoungDaly.
- YoungDaly achieves significantly lower performance when only a handful of checkpoints are required. The most significant loss of performance happens when the reservation length becomes slightly larger than $W_{YD}$: then YoungDaly uses a second checkpoint, while the other strategies still use a single checkpoint. The second most significant loss of performance happens when YoungDaly starts performing a second checkpoint, e.g., when $1.2W_{YD} \leq T \leq 1.6W_{YD}$. Its first checkpoint completes at time $W_{YD}$ and the second one at the end of the reservation length, so they are not equally spaced. In such circumstances, the three other strategies achieve significantly better performance by using two segments of equal size.

Figure 4 presents the impact of the choice of the quantum size on DynamicProgramming when $C = 20$. The choice of the quantum size only has a significant impact on the performance of DynamicProgramming when the reservation length is quite small, being of the order of a handful of quanta. This can be seen on Figure 5, which presents a subset of the data of Figure 4 corresponding to $T \leq 100$. When the quantum does not exceed 2, the differences become quickly negligible. We do not observe any difference when the quantum is equal to 1, and choosing a smaller quantum (say, 0.5) does not lead to better performance. This justifies our choice of quantum for Figures 2 and 3.

Overall, when the reservation length is of the order of a few Young/Daly periods (say, less than half a dozen such periods), using the Young/Daly period can lead to significant loss of performance. One should rather use evenly distributed checkpoints whose number is derived using a numerical approximation of the thresholds $T_n$ (the NumericalOptimum policy).

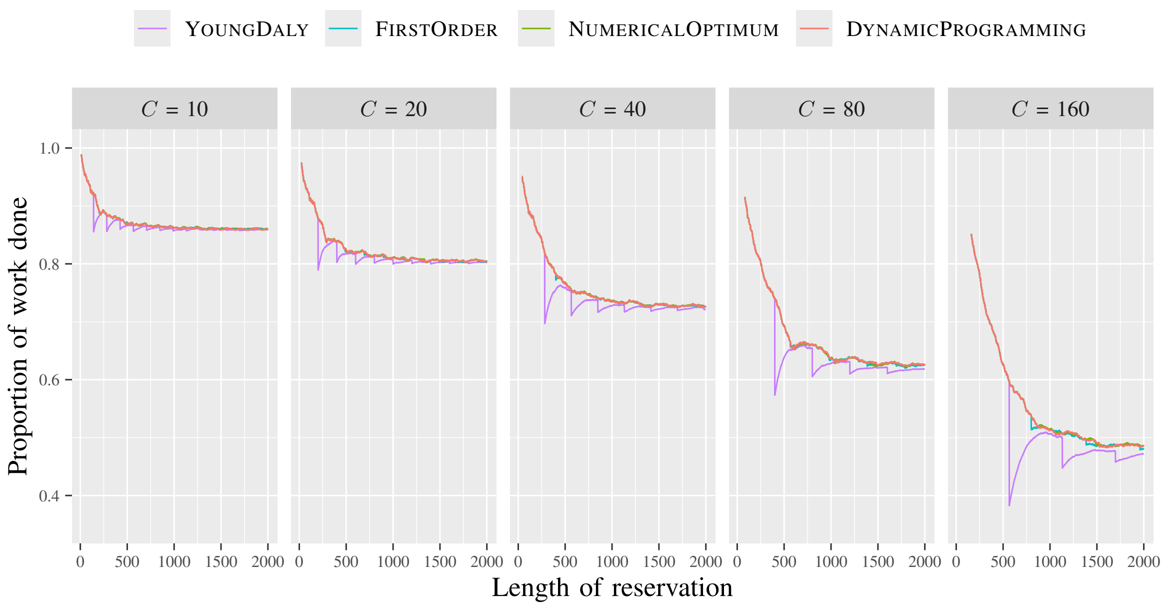

Fig. 2: Proportion of work completed by the four checkpointing strategies when $\lambda = 0.001$ and $D = 0$.

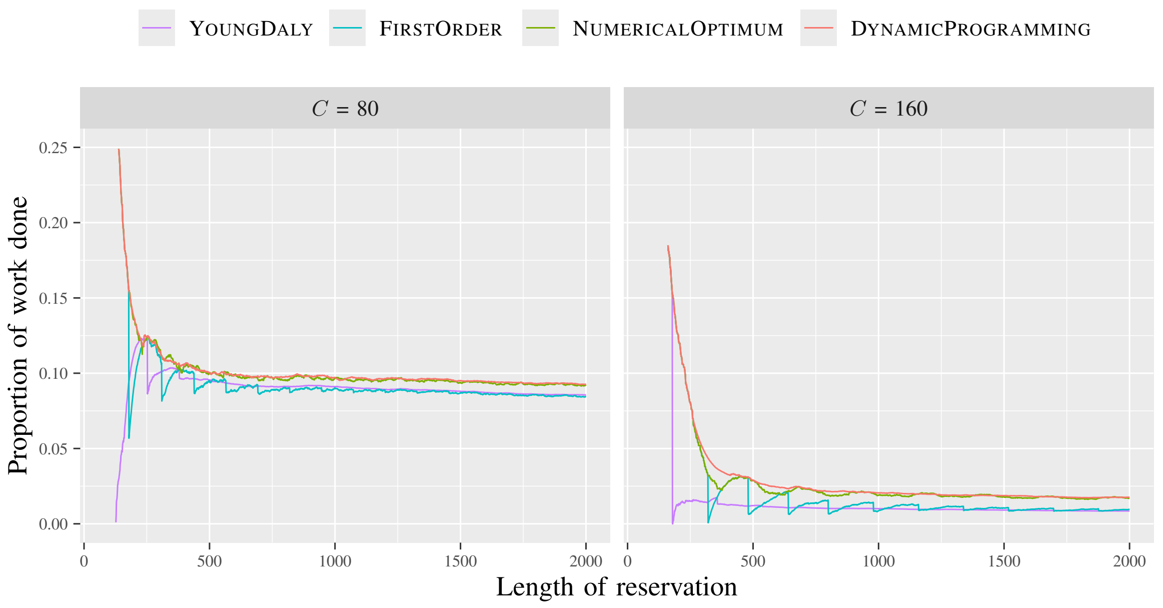

Fig. 3: Proportion of work completed by the four checkpointing strategies when $\lambda = 0.01$, $D = 0$, and $C \in \{80, 160\}$.

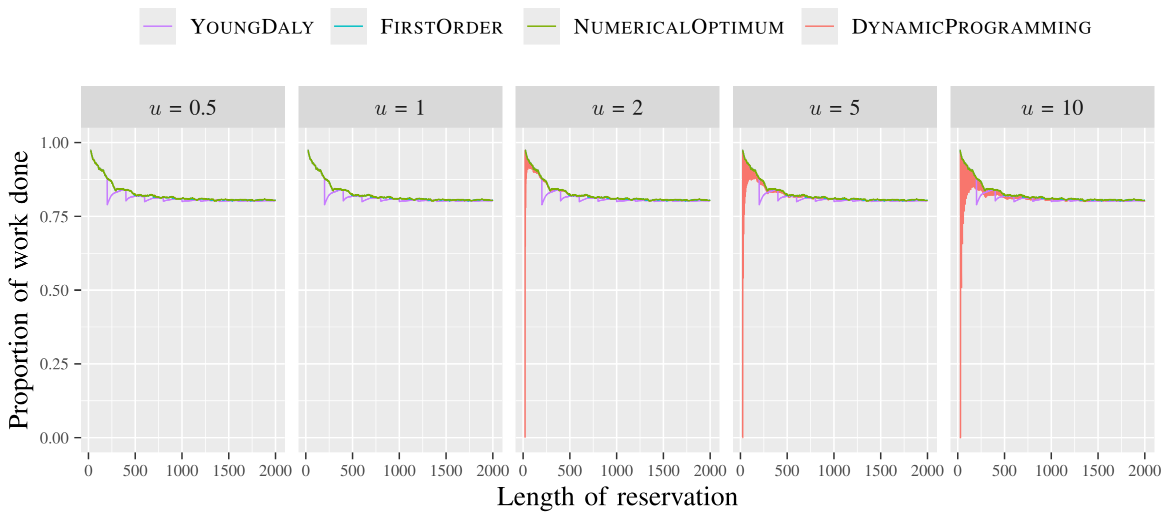

Fig. 4: Impact of quantum size on the performance of DynamicProgramming when $\lambda = 0.001$, $D = 0$, and $C = 20$.

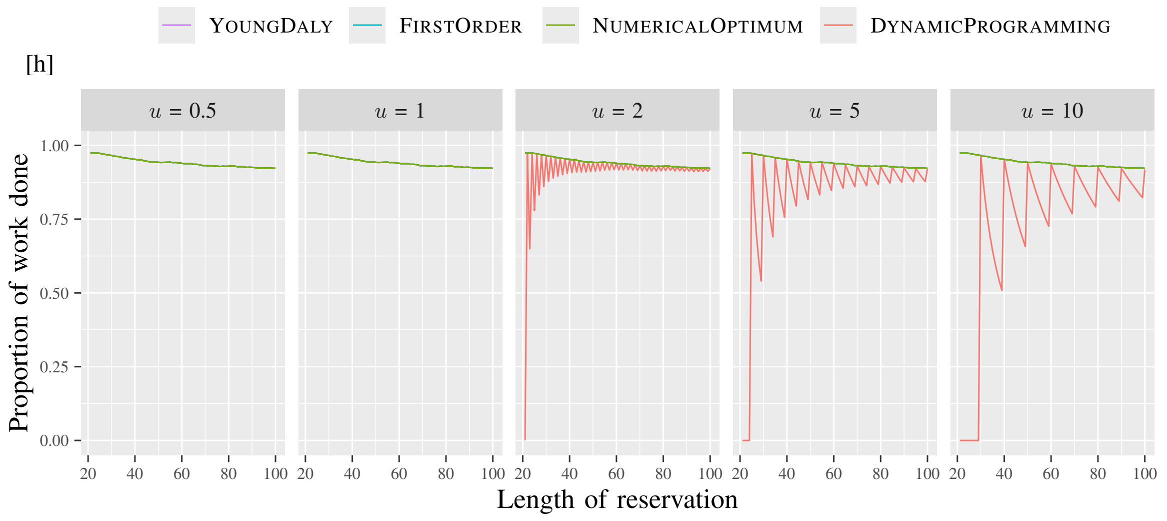

Fig. 5: Impact of quantum size on the performance of DynamicProgramming when $\lambda = 0.001$, $D = 0$, and $C = 20$: detail from Figure 4.

The DYNAMICPROGRAMMING policy is more general and more expensive to run but does not lead to significantly better performance. However, it fully validates using the NUMERICALOPTIMUM policy as a lightweight and efficient checkpointing strategy.

## VIII. CONCLUSION

This work has focused on the *fixed-time checkpointing* problem, where the objective is to maximize the expected work achieved during a fixed-length reservation. The *fixed-time checkpointing* problem is the dual of the classical *fixed-work checkpointing* problem, where the objective is to minimize the expected time to execute a fixed amount of work. To the best of our knowledge, this is the first work to study the *fixed-time checkpointing* problem, despite its importance for scientific applications that execute on large-scale, hence, failure-prone platforms; typically, these applications split their execution into a series of fixed-length reservations, a policy that is safer and more friendly to the batch scheduler than using a single (necessarily over-provisioned) reservation.

Our first contribution is to show that the *fixed-time checkpointing* problem is much more difficult than the classical and well-studied dual problem. The optimal solution seems out of reach, and we have provided several examples to explain why. Our second contribution is to provide two variants of a dynamic threshold-based strategy that outperform the standard Young/Daly approach for short-size or medium-size reservations, as demonstrated by a comprehensive simulation campaign. Finally, our third contribution is to use time discretization, and to design a dynamic programming algorithm that computes the optimal solution, without any restriction on the checkpointing strategy. The dynamic programming algorithm is optimal for small quanta, and achieves only slightly better performance than the dynamic threshold strategy, which assesses the quality of the latter strategy.

Altogether, we can conclude with two good news. The first one is that the Young/Daly approach performs well for long reservations lasting at least, say, a dozen Young/Daly periods $W_{YD} = \sqrt{2\mu C}$. The second one is that the dynamic threshold strategy is very easy to use and performs remarkably well for shorter reservations, as demonstrated by its performance on par with the optimal (discrete) solution.

For future work, it would be interesting to correlate all these results to real numbers and real error rates, maybe with a case study of a particular cluster. In addition, it would be compelling to extend the study of *fixed-time checkpointing* to a stochastic framework, where both checkpoint durations and application progress rate (think of iterations in sparse linear algebra problems) are no longer fully deterministic.

### *Acknowledgment*

We would like to thank the anonymous reviewer of our FTXS'2023 paper [1], whose question motivated this work on the fixed-time checkpointing problem.

## REFERENCES

[1] Q. Barbut, A. Benoit, T. Herault, Y. Robert, and F. Vivien. When to checkpoint at the end of a fixed-length reservation? In *FTXS'2023, the 13th Workshop on Fault-Tolerance for HPC at Extreme Scale, in conjunction with SC'2023*. ACM Press, 2023.

[2] L. Bautista-Gomez, A. Benoit, S. Di, T. Herault, Y. Robert, and H. Sun. A survey on checkpointing strategies: Should we always checkpoint à la Young/Daly? *Future Generation Computer Systems*, 161:315–328, 2024.

[3] A. Benoit, L. Perotin, Y. Robert, and F. Vivien. Checkpointing strategies for a fixed-length execution. Research report RR-9552, Inria, July 2024. Available at https://inria.hal.science/hal-04668191.

[4] A. Benoit, L. Perotin, Y. Robert, and F. Vivien. Checkpointing strategies to tolerate non-memoryless failures on HPC platforms. *ACM Trans. Parallel Computing*, 11(1), 2024.

[5] W. Bland, A. Bouteiller, T. Herault, J. Hursey, G. Bosilca, and J. J. Dongarra. An evaluation of User-Level Failure Mitigation support in MPI. *Computing*, 95(12):1171–1184, 2013.

[6] M. Bougeret, H. Casanova, M. Rabie, Y. Robert, and F. Vivien. Checkpointing strategies for parallel jobs. In *Proceedings of SC'11*, 2011.

[7] 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.

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

[9] J. Dongarra, T. Hérault, and Y. Robert. Performance and reliability trade-offs for the double checkpointing algorithm. *Int. J. of Networking and Computing*, 4(1):23–41, 2014.

[10] 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.

[11] M. Gamell, R. F. V. der Wijngaart, K. Teranishi, and M. Parashar. Specification of fenix MPI fault tolerance library, version 1.0.1. Technical Report SAND2016-10522, Sandia National Laboratory, September 2016. https://www.osti.gov/servlets/purl/1330192.

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

[13] J. S. Plank, K. Li, and M. A. Puening. Diskless checkpointing. *IEEE Trans. Parallel and Distributed Systems*, 9(10):972–986, 1998.

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

[15] P. Sigdel, X. Yuan, and N. Tzeng. Realizing best checkpointing control in computing systems. *IEEE TPDS*, 32(2):315–329, 2021.

[16] Wikipedia. Application checkpointing, 2023. https://en.wikipedia.org/wiki/Application_checkpointing.

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

[18] G. Zheng, X. Ni, and L. V. Kalé. A scalable double in-memory checkpoint and restart scheme towards exascale. In *IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)*, pages 1–6. IEEE, 2012.