Fixed-Work vs. Fixed-Time Checkpointing on Large-Scale Failure-Prone Platforms

Quentin Barbut, Lucas Perotin, Anne Benoit, Thomas Herault, Yves Robert, Frédéric Vivien

Fixed-Work vs. Fixed-Time Checkpointing on Large-Scale Failure-Prone Platforms

Quentin Barbut[1], Lucas Perotin[2], Anne Benoit[1,4], Thomas Herault[3], Yves Robert[1] and Frédéric Vivien[1]

Abstract

Consider a High-Performance Computing (HPC) application executing on a large-scale failure-prone platform. The *Fixed-Work Checkpointing (*FWC*)* problem consists in minimizing the expected time to execute a fixed amount of work (namely a fraction or the totality of the application). Strategies for the FWC problem have received considerable attention and are well-understood. On the contrary, the dual problem, namely the *Fixed-Time Checkpointing (*FTC*)* problem, has been considered only very recently. The FTC problem consists in maximizing the expected work achieved during a fixed amount of time (namely the duration of a reservation granted to the application). This work provides a comparative overview of both problems. First we review existing strategies for the FWC problem and extend them to stochastic checkpoints, i.e., when the checkpoint is no longer a deterministic constant but obeys some probability distribution law instead. Then we provide a comprehensive study of the FTC problem. The problem turns out to be surprisingly difficult, even when restricting to taking one or two checkpoints. We provide a threshold-based heuristic to solve the general instance of the problem with an arbitrary number of checkpoints, and we have to resort to time discretization to provide an optimal strategy. We further extend this latter strategy to stochastic checkpoints.

Keywords

Fixed-time checkpointing, fixed-work checkpointing, fixed-length reservation, checkpoint, stochastic durations.

1 Introduction

Checkpoint/restart is the standard technique to protect applications running on High Performance Computing (HPC) platforms. Such platforms experience several failures* per day[7,10,13,15–17]. After each failure, the application executing on the faulty processor (and likely on many other processors for a large parallel application) is interrupted and must be restarted. Without checkpointing, all the work executed for the application is lost. With checkpointing, the execution can resume from the last checkpoint, recovering its state from the last checkpoint file.

There are two very natural settings to study checkpointing strategies for HPC applications. In the first setting, there is a global amount of work W_{tot} to execute, namely the total load of the application. We partition W_{tot} into several chunks, each followed by a checkpoint. The objective is to minimize the expected time to execute all these chunks (i.e., the whole application). How many chunks, and of which size, should be taken? With same-size checkpoints, this is equivalent to asking how many checkpoints should be taken. There is a well-known trade-off: 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 *Fixed-Work Checkpointing (*FWC*)* problem consists in minimizing the expected time to execute the fixed work W_{tot}. The FWC problem has received considerable attention in the recent years. We review previous work for the FWC problem in Section 2.1

In the second setting, the application executes for a fixed time T, namely the length of the reservation that it has been granted. Indeed, scheduling an application onto a computing platform typically involves making a series of reservations of the required resources. Long running applications, or applications whose total run-time are hard to predict, usually split their reservation in multiple smaller reservations and use checkpoint-restart[12,18] to save intermediate steps of computation. 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. 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 problem with a fixed-length reservation T is then to maximize the expected work that can be achieved during that reservation, given that (i) failures may strike during the execution, and

[1] ENS Lyon, Laboratoire LIP & Inria, Lyon, France
[2] Vanderbilt University, Nashville, TN, USA
[3] University Tennessee Knoxville, TN, USA
[4] Institut Universitaire de France

Corresponding author:
Yves Robert, Laboratoire LIP, ENS Lyon, 69364 Lyon Cedex 07, France.
Email: yves.robert@ens-lyon.fr

*Failures are also called *fail-stop errors*.

(ii) a final checkpoint must be taken at the end, or close to the end, of the reservation. This problem, which we call the *Fixed-Time Checkpointing (*FTC*)* problem, can be seen as the dual of the FWC problem: 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.

The FTC problem has received little attention, despite its practical significance: reservations are omnipresent in HPC, and are at the heart of batch schedulers. Building upon our previous work[2], we show that FTC turns out to be much more difficult than FWC, which is kind of unexpected, and we explain why. We then propose several heuristic strategies to solve the FTC problem.

We also introduce a new dimension in both the FWC and FTC problems, namely that of a varying checkpoint duration. Assuming a perfect knowledge of the total work W_{tot} for FWC is quite reasonable (you know your application). Similarly, assuming a perfect knowledge of the reservation duration T for FWC is quite reasonable (you know what you paid for). On the contrary, assuming a perfect knowledge of the value of the checkpoint time C is more questionable. In particular, if the actual duration of the last checkpoint C exceeds the one planned, that checkpoint may well not complete before the end of the reservation, and all the work executed since the previous checkpoint (if any) will be lost. In fact, it is very likely that the value of C would vary from one execution to another; the range of variation of the possible values of C would typically depend upon the application. What is the best strategy then? A natural approach is to assume that C takes values within a finite range $[C_{min}, C_{max}]$, obeying a probability distribution law $\mathcal{D}_C$. We show how to accommodate such stochastic checkpoint values for both the the FWC and FTC problems.

Altogether, the major contributions of this work are the following:

- We review existing strategies for the FWC problem and extend them to stochastic checkpoints.
- We provide a comprehensive study of the FTC problem. Due to its difficulty, we have to resort to time discretization to provide an optimal strategy. We further extend this latter strategy to stochastic checkpoints.

The rest of the paper is organized as follows. We survey related work in Section 2. We detail the framework in Section 3. The next four sections provide checkpoint strategies for the following four problems: (i) FWC and constant checkpoints in Section 4.1; (ii) FWC and stochastic checkpoints in Section 4.2; (iii) FTC and constant checkpoints in Section 5; and (iv) FTC and stochastic checkpoints in Section 6. Section 6 also discusses a simpler problem, that of a fixed-length reservation and stochastic checkpoints, in a failure-free environment. This simpler problem is of interest beyond HPC applications and relevant to a wider class of applications, even shorter and sequential ones. Finally, Section 7 provides concluding remarks and directions for future work.

2 Related work

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

2.1 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[20] and Daly[8] states that the optimal checkpointing period is $W_{YD} = \sqrt{2\mu C}$, where μ 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 (8) (see Section 5.1.1), a formula rediscovered ten years later, with a quite different proof based on a Markov model, in[16]. 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 stress 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[9,14,21], 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.

2.2 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. On failure-free platforms, there is a single checkpoint close to the end of the reservation, in order to save and store output data. On failure-free platforms, more checkpoints may be taken throughout the reservation to save intermediate results, and the last checkpoint is still there to save final results.

To the best of our knowledge, the FTC 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].

3 Framework

This section details the framework used throughout the paper. We enforce standard assumptions from the literature, which we detail below.

We target a parallel application executing on a large-scale HPC platform. Failures may strike the application during execution. Failure inter-arrival times[†] (IATs) obey *Independent and Identically Distributed* probability laws, namely an Exponential distribution $Exp(\lambda_{ind})$ of parameter λ_{ind} on each processor. The Mean-Time Between Failures (MTBF) on a processor is thus $\mu_{ind} = \frac{1}{\lambda_{ind}}$. If the parallel application enrolls p processors, then the platform MTBF is $\mu = \frac{\mu_{ind}}{p}$. The superposition of p exponential distributions of parameter λ_{ind} is an exponential distribution of parameter $\lambda = p\lambda_{ind}$. 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.

After a failure, the application must roll-back to the last checkpoint. In order to restart its execution, the application enters a recovery phase, during which it reads the checkpoint file from stable storage. 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. The durations of C and R are either assumed to be fixed constants, or they can take stochastic values drawn from some probability distributions $\mathcal{D}_C$ and $\mathcal{D}_R$. In the latter case, these distributions $\mathcal{D}_C$ and $\mathcal{D}_R$ are assumed to be independent of the failure inter-arrival times with IID distribution $Exp(\lambda)$. However, $\mathcal{D}_C$ and $\mathcal{D}_R$ are not assumed to be independent in general. In fact, the size of the checkpoint file depends upon the application data, but it remains the same when writing to or reading from stable storage, so it is natural to have R proportional to C, the proportion factor being the ratio of reading from over writing to stable storage: $R = \beta C$.

The execution of the application is partitioned into chunks, and each chunk ends with a checkpoint. In particular, the last chunk always end with a checkpoint. For the FTC problem, any time that remains after the last checkpoint is lost. Symmetrically, if a failure strikes the first chunk, an initial recovery R is paid for. Finally, the application is preemptible, meaning that a checkpoint can be taken at any instant.

Without loss of generality, we assume that one unit of work lasts one second, so that we can speak of work or time indifferently. For the FWC problem, we target a parallel application with W_{tot} units of work, or seconds. For the FTC problem, we have a reservation of fixed-length T seconds. The values of C, R and D are also expressed in seconds.

4 Fixed-Work Checkpointing

This section provides background on the FWC problem, first with a constant checkpoint time in Section 4.1, and then with a stochastic checkpoint time in Section 4.2.

4.1 Fixed-Work Checkpointing, Constant Checkpoint Time

This section deals with the FWC problem, with a constant checkpoint time. We recall the following result:

Lemma 1. *The expected time $\mathbb{E}(W)$ to execute a segment of W seconds of work followed by a checkpoint of C seconds is*

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

Lemma 1 comes from[6] Theorem 1. The *slowdown function* is defined as $f(W) = \frac{\mathbb{E}(W)}{W}$. We also have the following properties:

Lemma 2. *The slowdown function $W \mapsto f(W)$ has a unique minimum W_{opt} that does not depend on R, is decreasing in the interval $[0, W_{opt}]$ and is increasing in the interval $[W_{opt}, \infty)$.*

Proof. Again, see[6] Theorem 1. The exact value of W_{opt} is obtained using the Lambert W function, but a first-order approximation is the Young/Daly formula[8,20]: $W_{YD} = \sqrt{\frac{2C}{\lambda}}$.

Lemma 2 shows that infinite applications should be partitioned into segments of size W_{opt} followed by a checkpoint. What about finite applications? Back to our application of duration W_{tot}, we partition it into N segments of length W_i, $1 \leq i \leq N$, each followed by a checkpoint C. By linearity of the expectation, the expected time to execute the whole application is

$$\mathbb{E}(W_{tot}) = \sum_{i=1}^{N} \mathbb{E}(W_i) = \left(\frac{1}{\lambda} + D\right) e^{\lambda R} \sum_{i=1}^{N} \left(e^{\lambda(W_i+C)} - 1\right),$$

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

where $\sum_{i=1}^{N} W_i = W_{tot}$. By convexity of the Exponential function, or by using Lagrange multipliers, we see that $\mathbb{E}(W_{tot})$ is minimized when the W_i's take a constant value, i.e., all segments have same length. Thus, we obtain $W_i = \frac{W_{tot}}{N}$ for all i, and we aim at finding N that minimizes

$$\mathbb{E}(W_{tot}) = N\mathbb{E}\left(\frac{W_{tot}}{N}\right) = f\left(\frac{W_{tot}}{N}\right) \times W_{tot}, \quad (2)$$

where f is the slowdown function. We let $N_{opt} = \frac{W_{tot}}{W_{opt}}$, where W_{opt} achieves the minimum of the slowdown function. N_{opt} would be the optimal number of segments if we could have a non-integer number of segments. Lemma 2 shows that the optimal value N_{ME} of N is either $N_{ME} = \max(1, \lfloor N_{opt} \rfloor)$ or $N_{ME} = \lceil N_{opt} \rceil$, whichever leads to the smallest value of $\mathbb{E}(W_{tot})$. In the literature, one usually uses the simplified Young/Daly expression $N_{ME} = \left\lfloor \frac{W_{tot}}{W_{YD}} \right\rceil$, which however can be inaccurate for short values of W_{tot}.

4.2 Fixed-Work Checkpointing, Stochastic Checkpoint Time

This section extends known results for the FWC problem when checkpoints and recoveries are no longer constant but instead obey some probability distributions $\mathcal{D}_C$ and $\mathcal{D}_R$. Let $f_C(c)$ and $F_C(c)$ denote the probability density function and probability distribution function for C, with compact support $[C_{\min}, C_{\max}]$; similarly, let $f_R(r)$ and $F_R(r)$ denote the probability density function and probability distribution function for R, with compact support $[R_{\min}, R_{\max}]$. Consider some fixed work W to be executed in this framework. Conditioning the expectation $\mathbb{E}(W)$ in Equation (1) to the values of C and R, we derive that

$$\begin{aligned}\mathbb{E}(W) &= \int_{C_{\min}}^{C_{\max}} \int_{R_{\min}}^{R_{\max}} \mathbb{E}(W|C=c, R=r) dc dr \\ &= \int_{C_{\min}}^{C_{\max}} \int_{R_{\min}}^{R_{\max}} f_C(c) f_R(r) \left(\tfrac{1}{\lambda} + D\right) e^{\lambda R} \left(e^{\lambda(W+C)} - 1\right) dc\, dr.\end{aligned} \quad (3)$$

This formula is quite general and applies to arbitrary distributions $\mathcal{D}_C$ and $\mathcal{D}_R$. We detail the computation for an important scenario where $\mathcal{D}_C$ is Uniform in $[C_{\min}, C_{\max}]$ (hence $F_C(c) = \frac{c - C_{\min}}{C_{\max} - C_{\min}}$ for $c \in [C_{\min}, C_{\max}]$) and R is proportional to C, namely $R = \beta C$. We derive that

$$\begin{aligned}\mathbb{E}(W) \quad &= \left(\frac{\frac{1}{\lambda} + D}{C_{\max} - C_{\min}}\right) \times \\ &\int_{C_{\min}}^{C_{\max}} (c - C_{\min}) e^{\lambda \alpha c} \left(e^{\lambda(W+c)} - 1\right) dc.\end{aligned} \quad (4)$$

We know that (integrating by parts for the second equation):

$$\begin{aligned}&\int_{c_1}^{c_2} \lambda e^{-\lambda t} dt = e^{-\lambda c_1} - e^{-\lambda c_2} \\ &\int_{c_1}^{c_2} \lambda t e^{-\lambda t} dt = c_1 e^{-\lambda c_1} - c_2 e^{-\lambda c_2} + \frac{e^{-\lambda c_1} - e^{-\lambda c_2}}{\lambda}\end{aligned} \quad (5)$$

This enables us to compute a (somewhat messy) formula for $\mathbb{E}(W)$ in Equation (4). In fact, we do not need the detailed formula to proceed. Owing to Equation 5, we can write

$$\mathbb{E}(W) = A_1 e^{\lambda W} + A_2, \quad (6)$$

where A_1 and A_2 are some (complicated) constants, with $A_1 > 0$. This is all we need to solve the FWC problem with stochastic checkpoints:

- For infinite jobs, we minimize the slowdown function $f(W) = \frac{\mathbb{E}(W)}{W}$ just as with constant checkpoints, and the minimum W_{opt} is still obtained using the Lambert W function.
- For a finite job of duration W_{tot}, Equation (6) shows that $\mathbb{E}(W)$ is convex. Hence, we should divide W_{tot} into chunks of same length, and the optimal number of chunks is obtained very similarly to the case with constant checkpoints.

Interestingly, the approach to compute $\mathbb{E}(W)$ extends to the case where W itself takes stochastic values. Assume that W obeys some probability distribution law $\mathcal{D}_W$, whose support is, say, $[0, \infty[$. Let $f_W(w)$ and $F_W(w)$ denote the probability density function and probability distribution function of $\mathcal{D}_W$. We then extend Equation (3) into

$$\begin{aligned}\mathbb{E}(W) = \int_{C_{\min}}^{C_{\max}} \int_{R_{\min}}^{R_{\max}} \int_0^{\infty} f_C(c) f_R(r) f_W(w) \times \\ \left(\tfrac{1}{\lambda} + D\right) e^{\lambda R} \left(e^{\lambda(w+C)} - 1\right) dc\, dr\, dw.\end{aligned} \quad (7)$$

As a simple illustration, take $C = R = D = 0$ and $\mathcal{D}_W = Exp(\gamma)$. This means that we compute the expected time to execute a stochastic amount of work W whose values are drawn from the distribution $Exp(\gamma)$, in presence of failures whose inter-arrival times are drawn from the distribution $Exp(\lambda)$. There is no checkpoint nor recovery nor downtime, meaning that after each failure, we resume from scratch immediately. Equation (7) simplifies into

$$\mathbb{E}(W) = \tfrac{1}{\lambda} \int_0^{\infty} \gamma w e^{-\gamma w} \left(e^{\lambda w} - 1\right) dw.$$

Integrating by parts, we obtain that

$$\mathbb{E}(W) = \begin{cases} \frac{1}{\gamma - \lambda} & \text{if } \lambda < \gamma, \\ \infty & \text{otherwise.} \end{cases}$$

Without failures, the expected value of $\mathbb{E}(W)$ would be $\frac{1}{\gamma}$. With failures of rate λ, the value of $\mathbb{E}(W)$ increases, and gets infinite whenever λ is no longer smaller than γ. This little example shows the expressiveness of Equation (7), which can be easily instantiated for various distributions $\mathcal{D}_C$, $\mathcal{D}_R$ and $\mathcal{D}_W$.

5 Fixed-Time Checkpointing, Constant Checkpoint Time

This section focuses on the FTC problem. We consider a parallel application executing for a fixed-time T on a parallel platform. The application experiences failures which follow an Exponential probability distribution $Exp(\lambda)$. The objective is to maximize the expected work $\mathbb{E}(T)$ achieved during T seconds. In this section, we assume that the checkpoint time is a constant C, and we deal with a stochastic checkpoint time in Section 6.

A checkpointing strategy for the FTC problem 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.

5.1 Difficulty of the Problem

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

5.1.1 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, starting 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 $\mathbb{E}_R^{end}(T, 1)$, where the only difference with $\mathbb{E}^{end}(T, 1)$ is that we start the execution with a recovery.

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

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

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:

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

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 $\mathbb{E}_R^{end}(T, 1)$ 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:

$$\begin{aligned}\mathbb{E}(W) &= e^{-\lambda(W+C)}(W + C) \\ &+ (1 - e^{-\lambda(W+C)})\big(\mathbb{E}(T_{lost}(W + C)) + \mathbb{E}(T_{rec}) + \mathbb{E}(W)\big),\end{aligned} \tag{8}$$

where $\mathbb{E}(T_{lost}(W + C))$ is the expected time lost before the first failure. Similarly, $\mathbb{E}(T_{rec})$ is the expected time for the recovery (recall that failures may strike during recovery). Skipping details (see[6]), we obtain Equation (1) (see Lemma 1). The key to solving Equation (8) 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.

5.1.2 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:

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

which is negative when $\lambda > \ln(2)$. Hence, if λ 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.

5.1.3 With Two Checkpoints. This third example, and the heuristic approach described in Section 5.2, 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 $\text{STRAT}_2(\alpha(T))$ 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) $\text{GAIN}(\alpha(T))$ of $\text{STRAT}_2(\alpha(T))$ over STRAT_1, the strategy where we execute only a single checkpoint at the end of the reservation. Recall that the gain $\text{GAIN}(\alpha(T))$ is the difference of the expected work achieved until the first failure, if any.

The gain $\text{GAIN}(\alpha(T))$ of $\text{STRAT}_2(\alpha(T))$ over STRAT_1 is

- $\mathbb{P}_{succ}(T) \times (-C)$ if there is no failure, because there is one more checkpoint in $\text{STRAT}_2(\alpha(T))$;
- 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 $\text{STRAT}_2(\alpha(T))$.

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:

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

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

$$\begin{aligned}f'(0) &= T(\lambda C + 1 - e^{-\lambda T}) > 0 \qquad \text{and} \\ f'(1) &= -\lambda T e^{-\lambda T}(T - C) < 0\end{aligned}$$

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

$$1 = \lambda(\alpha T - C) + e^{-\lambda(1-\alpha)T}. \tag{9}$$

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 λ satisfy to $1 = \lambda(\frac{T}{2} - C) + e^{-\lambda\frac{T}{2}}$. 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.

5.2 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

$$T_n \leq t_{left} \leq T_{n+1}. \tag{10}$$

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 achieved by the heuristic in expectation. Unfortunately, this is not true: revisiting the example of Section 5.1.2, the heuristic achieves more expected work when $T = 5$ than when $T = 6$ whenever $\lambda > \ln(2)$. And the example of Section 5.1.3 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

$$T \geq \max\left(T_n, (n+1)C\right). \tag{11}$$

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 STRAT_{n+1}) instead of using n checkpoints (strategy STRAT_n). Recall that the gain compares the expected work until the first failure. Hence,

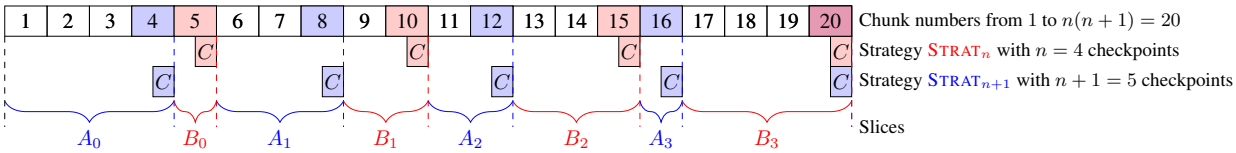


Figure 1. Comparing strategies with n and $n+1$ checkpoints for $n = 4$.

for a reservation T obeying Equation (11), we want to compare $\mathbb{E}(T, n+1)$, the expected work with STRAT$_{n+1}$, with $\mathbb{E}(T, n)$, the expected work with 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 STRAT$_{n+1}$ than for 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 3. *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 STRAT$_n$, and up to checkpoint number $m+1$ of strategy STRAT$_{n+1}$ (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 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 STRAT$_{n+1}$ and up to checkpoint number $m+1$ of strategy STRAT$_n$. 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 STRAT$_n$ has saved more chunks than strategy STRAT$_{n+1}$, namely all the chunks of slice B_{m-1}. Hence, the gain is negative, and its expected value is

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

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 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 STRAT$_n$ has saved fewer chunks than strategy 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 $(m+1)n+1$, and has length $m+1$ chunks. The extra number of chunks for 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 STRAT$_{n+1}$ at the end of A_m.

Altogether, we have derived that

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

We are unable to prove that there exists a single solution $T_{n+1} \geq (n+1)C$ to the equation $\text{GAIN}(T_{n+1}, n+1) = 0$, 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 5.4, 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

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

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

$$T_{n+1} \approx \sqrt{\frac{2n(n+1)C}{\lambda}}. \tag{12}$$

It is interesting to compare it 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

$$\sqrt{f(n)f(n+1)} = \sqrt{n(n+1)2\mu C} \approx T_{n+1}.$$

5.3 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, $1 \le i \le T^*$, 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, 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

$$\mathbb{E}_{opt}(T^*) = \max_{1 \le k \le k_{max}} \mathbb{E}(T^*, k, 0). \tag{13}$$

In Equation (13), we have $k \ge 1$ 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:

$$\begin{aligned}\mathbb{E}(n, 1, \delta) = &\max_{\delta R^* + C^* + 1 \le i \le n} \mathbb{P}^*_{succ}(i)(i - C^* - \delta R^*) \\ &+ \sum_{f=1}^{i - D^* - R^* - C^*} p_f^* \mathbb{E}(n - f - D^*, 1, 1).\end{aligned} \tag{14}$$

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

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

for $n \le 1 + C^* + D^* + R^*$. 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 $\mathbb{E}(n, 1, 1) = \max_{R^* + C^* + 1 \le i \le n} \mathbb{P}^*_{succ}(i)(i - C^* - R^*)$ for $n \le 1 + C^* + D^* + R^*$.

In fact, we can use the simpler initialization

$$\begin{array}{ll}\mathbb{E}(n, 1, 0) = 0 & \text{for } n \le C^* \text{ (where possibly } n \le 0) \\ \mathbb{E}(n, 1, 1) = 0 & \text{for } n \le R^* + C^* \text{ (where possibly } n \le 0)\end{array}$$

Algorithm 1: Dynamic programming algorithm

```
for k = 1 to k_max do
    for n = 1 to T* do
        Compute E(n, k, 0) and E(n, k, 1)
        using Equation (14) if k = 1,
        and Equation (15) if k ≥ 2
```

and recursively compute the values $\mathbb{E}(n,1,0)$ and $\mathbb{E}(n,1,1)$ for larger values of n using Equation (14).

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

$$\mathbb{E}(n,k,\delta) = \max_{\delta R^*+C^*+1 \leq i \leq n-(k-1)C^*} \mathbb{E}(n,k,\delta,i) \quad (15)$$

where

$$\mathbb{E}(n,k,\delta,i) = \mathbb{P}^*_{succ}(i)\big(i - C^* - \delta R^* + \mathbb{E}(n-i,k-1,0)\big) + \sum_{f=1}^{i} p^*_f \max_{1\leq m\leq k} \mathbb{E}(n-f-D^*,m,1).$$

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 $k=1$:

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

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.

5.4 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 (12).
- 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.

5.4.1 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.

5.4.2 Simulation Results. We only present here a subset of the results. Complete results can be found in the research report[3]. We initially report performance for DYNAMICPROGRAMMING when the quantum is set at 1 (see below for the influence of the quantum value).

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.

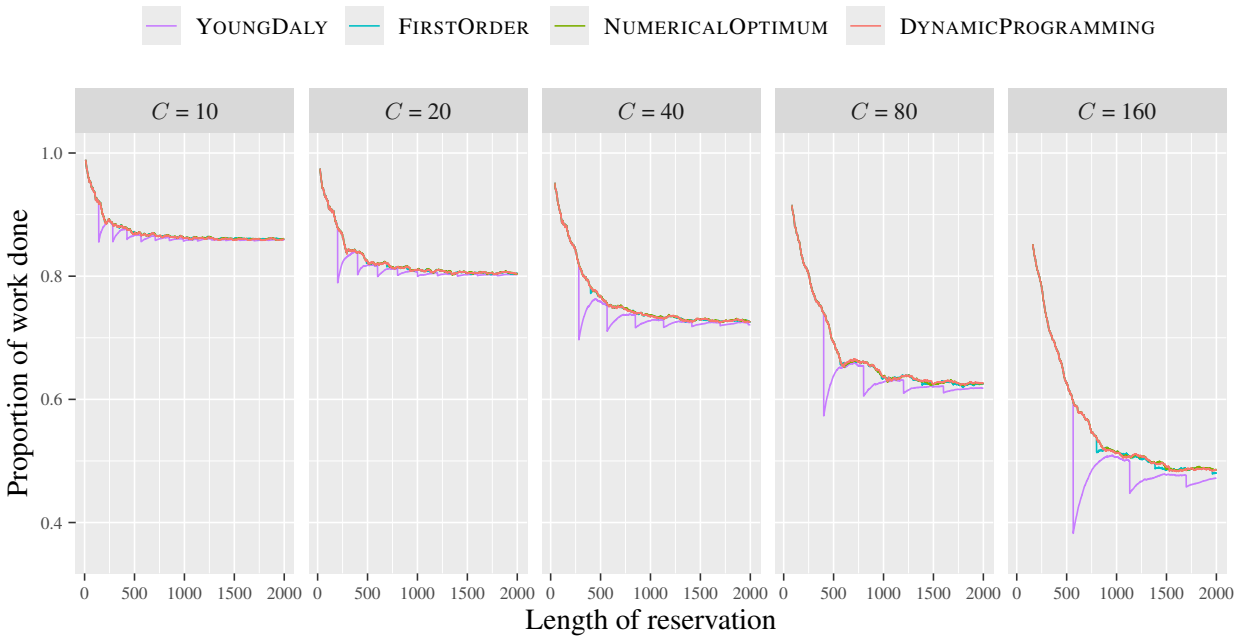


Figure 2. Proportion of work completed by the four checkpointing strategies when $\lambda = 0.001$ and $D = 0$.

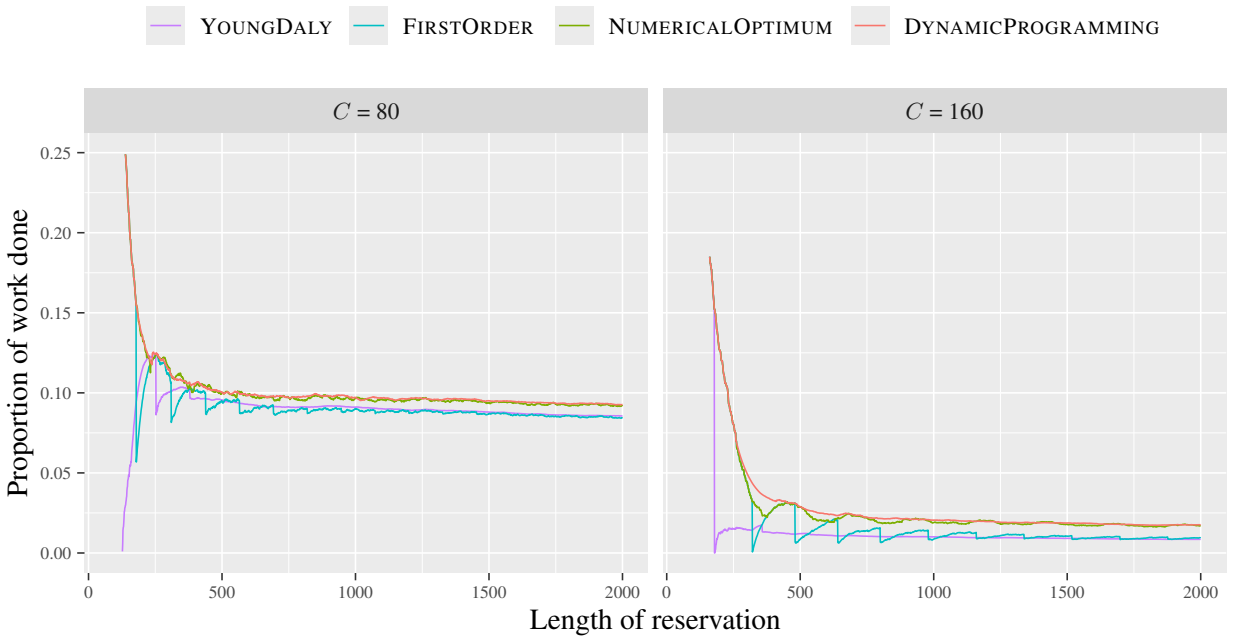


Figure 3. Proportion of work completed by the four checkpointing strategies when $\lambda = 0.01$, $D = 0$, and $C \in \{80, 160\}$.

- 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).

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.

6 Fixed-Time Checkpointing, Stochastic Checkpoint Time

This section extends the previous approach for the FTC problem to the case where the checkpoint time obeys some probability distribution. We start in Section 6.1 with the simple case without failures, which is worth investigating on its own: given a fixed-sized reservation, when should we take a single stochastic checkpoint at the end of the reservation? We proceed to the general case with failures in Section 6.2.

6.1 Without Failures

This section deals with a simple but interesting problem: an application executes for a fixed duration T and takes a checkpoint at the end. Here, we target a failure-free platform, so that the size of the application (sequential or parallel) is irrelevant; what matters is the volume of the data that needs to be saved before the end of the execution, or equivalently, the time needed to checkpoint that data. The (very natural) objective is to squeeze the most out of the reservation by executing as much work as possible before checkpointing. Obviously, in a perfect world, with a reservation of duration T and a checkpoint of duration C, one should checkpoint exactly C seconds before the end of the reservation, i.e., at time $T - C$ if the execution started at time 0.

While assuming a perfect knowledge of the value of T is quite reasonable (you know what you paid for), assuming a perfect knowledge of the value of C is more questionable. If the actual value of C exceeds the one planned by, say, a few seconds, all the work executed during the reservation will be lost. In fact, it is very likely that the value of C would vary from one execution to another; the range of variation of the possible values of C would typically depend upon the application. What is the best strategy then? If we know a worst-case value C_{max} for C, should we always use it and checkpoint at time $T - C_{max}$? Using C_{max} means taking no risk at all, but this pessimistic approach leads to wasting execution time whenever the actual value of C is significantly smaller than C_{max}.

A natural approach is to assume that a probability distribution law $\mathcal{D}_C$ for the values of C is known (instead of just an upper bound). The question becomes to determine the instant to checkpoint that maximizes the expected amount of work that will be saved before the end of the reservation. The probability distribution can be learned from traces of previous checkpoints. In this section, we show how to solve this problem for an arbitrary distribution $\mathcal{D}_C$, and we determine the solution for a variety of widely-used distributions whose support lies in an interval $[a, b]$, where $a = C_{min}$ and $b = C_{max}$ represent the extreme values that C can take. Such distributions include Uniform($[a, b]$), the uniform law in the interval $[a, b]$, and Exponential or Normal laws truncated to $[a, b]$.

6.1.1 General Approach. Starting the execution at time 0, we take a checkpoint at time $T - X$, where $0 \leq X \leq T$. The time to checkpoint C is a random variable that obeys a probability distribution law $\mathcal{D}_C$ with support $[a, b]$, where $0 < a < b \leq T$. In particular, we always have $a \leq C \leq b$. In fact, the lower bound a of C leads to refine the range of X as $a \leq X \leq T$: if $X < a$, there is simply not enough time left to checkpoint! We use this range $a \leq X \leq T$ below. How to choose X to maximize the expectation $\mathbb{E}(W(X))$ of the amount of work $W(X)$ that is saved when checkpointing at time $T - X$? The work saved by a checkpoint at time $T - X$ is

$$W(X) = \begin{cases} (T - X)\mathbb{1}_{[a,X]}(C) & \text{if } X \leq b \\ T - X & \text{if } X > b \end{cases}$$

Here, $\mathbb{1}_{[a,X]}(C)$ is the indicator function whose value is 1 is $C \in [a, X]$ and 0 otherwise. Indeed, we save $T - X$ if $C \leq X$ and nothing otherwise. This confirms that choosing $X > b$ will never be optimal, but it may well be the case that the optimal is reached for $X < b$.

Let Z be a random variable with cumulative distribution function (CDF) F and probability density function (PDF) f with possibly an infinite support. The law $\mathcal{D}_C$ of C is defined as the law of Z truncated within $[a, b]$. Then we have:

$$\begin{aligned} \mathbb{P}(C \leq x) &= \mathbb{P}(Z \leq x | a \leq Z \leq b) \\ &= \frac{\mathbb{P}((Z \leq x) \cap (a \leq Z \leq b))}{\mathbb{P}(a \leq Z \leq b)} \\ &= \begin{cases} 0 & \text{if } x < a \\ \frac{\int_a^{\min\{x,b\}} f(t)dt}{\int_a^b f(t)dt} = \frac{F(x)-F(a)}{F(b)-F(a)} & \text{otherwise} \end{cases} \end{aligned}$$

This gives the CDF F_C of C. Rewriting it as

$$F_C(X) = \mathbb{P}(C \leq x) = \frac{\int_a^x f(t)dt}{\int_a^b f(t)dt} = \int_a^x \frac{f(t)}{\int_a^b f(u)du}dt,$$

we obtain that the PDF f_C of C is $t \longmapsto \frac{f(t)}{F(b)-F(a)}$.

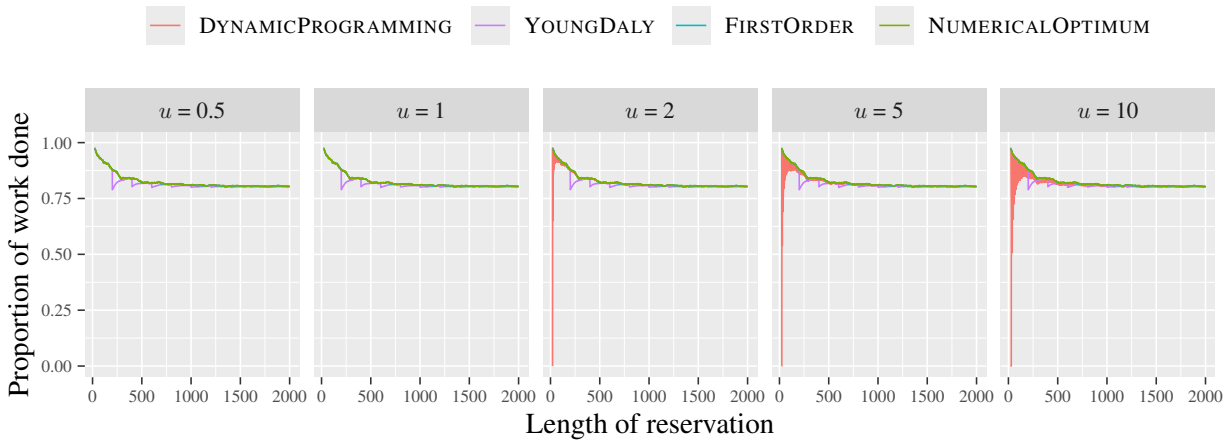


Figure 4. Impact of quantum size on the performance of DynamicProgramming when $\lambda = 0.001$, $D = 0$, and $C = 20$.

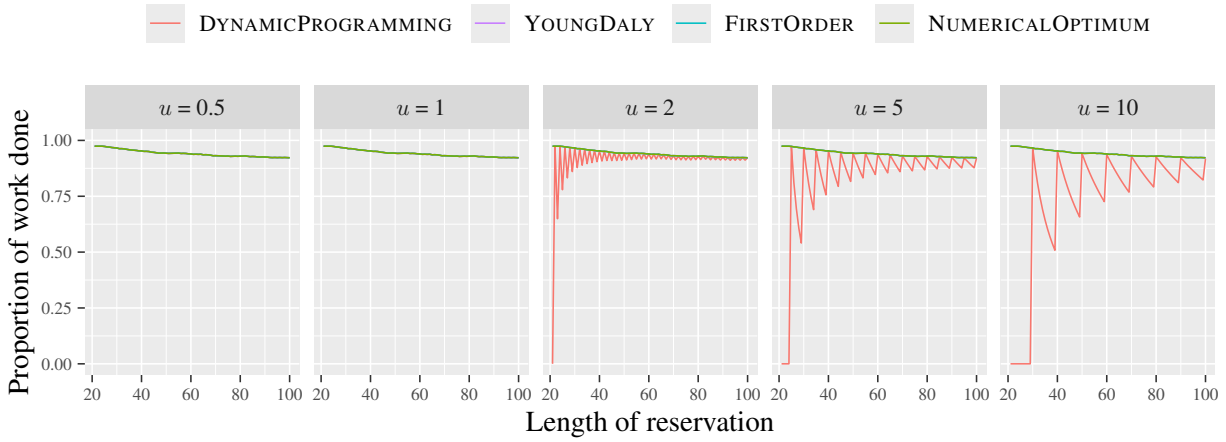


Figure 5. Impact of quantum size on the performance of DynamicProgramming when $\lambda = 0.001$, $D = 0$, and $C = 20$: detail from Figure 4.

We can now derive the expectation of the work saved when checkpointing at time X:

$$\mathbb{E}(W(X)) = \int_a^b W(X)\frac{f(c)}{F(b)-F(a)}dc$$
$$= \begin{cases} \int_a^b (T-X)\mathbb{1}_{[a,X]}(c)\frac{f(c)}{F(b)-F(a)}dc \\ \quad = \frac{F(X)-F(a)}{F(b)-F(a)}(T-X) & \text{if } X \leq b \\ T-X & \text{otherwise} \end{cases} \quad (16)$$

We use Equation (16) to find the optimal value of X for a uniform probability distribution in Section 6.1.2, for a (truncated) Exponential, distribution in Section 6.1.3, and for a (truncated) Normal distribution in Section 6.1.4, see [1] for LogNormal distribution laws.

6.1.2 Uniform Probability Distribution. For a uniform law in $[a, b]$, there is no need for truncating, and we directly have the PDF and CDF as

$$f_C(t) = \frac{1}{b-a} \quad \text{and} \quad F_C(X) = \int_a^X f(t)dt = \frac{X-a}{b-a}.$$

The expectation of the work saved when checkpointing at time X is:

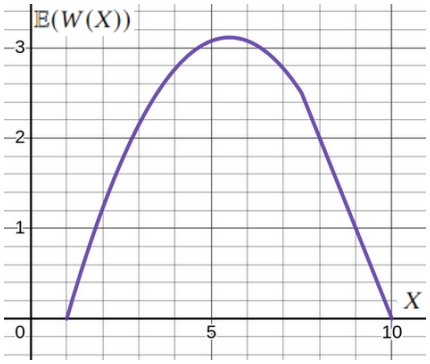
$$\mathbb{E}(W(X)) = \begin{cases} \frac{X-a}{b-a}(T-X) & \text{if } X \leq b \\ T-X & \text{otherwise} \end{cases} \quad (17)$$

The trinomial $x \longmapsto (x-a)(T-x)$ is maximum for $x = \frac{T+a}{2}$. This is the optimal value X_{opt} of X if $\frac{T+a}{2} < b$, otherwise the maximum is obtained for some x larger than b, and then b is optimal in the interval $[a, b]$. Altogether,

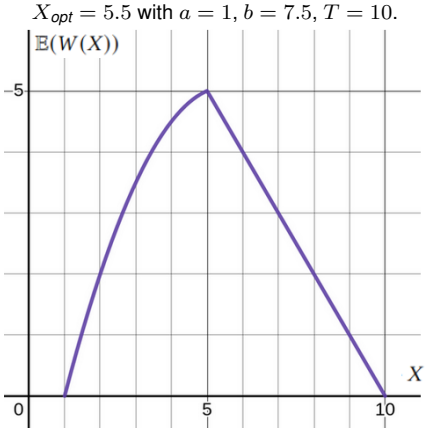
$$X_{opt} = \begin{cases} \frac{T+a}{2} & \text{if } T \leq 2b-a \\ b & \text{otherwise} \end{cases} = min\left(\frac{T+a}{2}, b\right).$$

Figure 6 provides an example of each case (optimal reached before b or at b). Recall that the range of X is $[a, T]$. When there remains $X = a$ seconds before the end of the reservation, the checkpoint will fail almost surely, and the expectation of the work saved is $\mathbb{E}(W(a)) = 0$. Similarly, if we checkpoint at the very beginning of the reservation, i.e., $X = T$, no work is executed and $\mathbb{E}(W(T)) = 0$. In between, the expectation $\mathbb{E}(W(X))$ of the work done obeys Equation (17). In particular, it decreases linearly from $X = b$ to $X = T$. In Figure 6(a), the maximum of $\mathbb{E}(W(X))$ is reached for $X_{opt} = \frac{T+a}{2} = 5.5$, with $\mathbb{E}(W(X_{opt})) \approx 3.1$; the pessimistic approach would use $X = C_{max} = b$ and get $\mathbb{E}(W(b)) = 2.5$, reaching only 80% of the optimal work amount in average. On the contrary, in Figure 6(b), the pessimistic approach is optimal since $X_{opt} = b$. The main take-away is that deciding to checkpoint with $X = b$, hence

preparing for the worst-case of checkpoint duration, is not always a good strategy.



(a) Graph of $\mathbb{E}(W(X))$. The maximum is $X_{opt} = 5.5$ with $a = 1, b = 7.5, T = 10$.



(b) Graph of $\mathbb{E}(W(X))$. The maximum is $X_{opt} = b$ with $a = 1, b = 5, T = 10$.

Figure 6. Both cases for X_{opt} with a Uniform law.

6.1.3 Exponential Probability Distribution. Let F and f be the CDF and PDF of an Exponential distribution law of parameter $\lambda = \frac{1}{\mu}$ with $\mu \in [a, b]$. We have $f(t) = \lambda e^{-\lambda t}$ and $F(x) = 1 - e^{-\lambda x}$. The distribution law of C is this Exponential law truncated to $[a, b]$. From Section 6.1.1, we have

$$\mathbb{E}(W(X)) = \begin{cases} \frac{e^{-\lambda a} - e^{-\lambda X}}{e^{-\lambda a} - e^{-\lambda b}}(R - X) & \text{if } X \leq b \\ R - X & \text{otherwise} \end{cases}$$

Differentiating for $X \leq b$, we obtain

$$\frac{d\mathbb{E}(W(X))}{dX} = \frac{-e^{-\lambda a} + (\lambda R + 1)e^{-\lambda X} - \lambda X e^{-\lambda X}}{e^{-\lambda a} - e^{-\lambda b}}.$$

Using Wolfram Alpha[19], this derivative has a unique zero for

$$X = \frac{-\mathcal{W}(e^{-\lambda a + \lambda R + 1}) + \lambda R + 1}{\lambda},$$

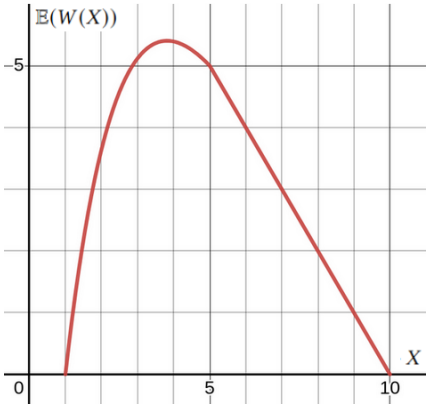
where $\mathcal{W}$ is the main branch of Lambert's $\mathcal{W}$ function (defined as $\mathcal{W}(z) = x \Leftrightarrow xe^x = z$). Differentiating again, we get

$$\frac{d^2\mathbb{E}(W(X))}{dX^2} = \frac{-\lambda(2 + \lambda R - \lambda X)e^{-\lambda X}}{e^{-\lambda a} - e^{-\lambda b}}.$$

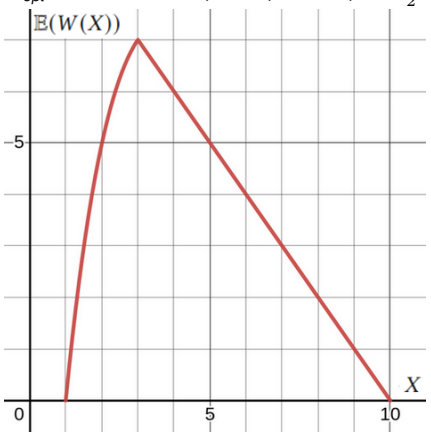
Since $X < R + \frac{2}{\lambda}$, we have $2 + \lambda R - \lambda X > 0$. Moreover, $e^{-\lambda a} - e^{-\lambda b} > 0$ (since $t \longmapsto e^{-\lambda t}$ is decreasing) and $-\lambda e^{-\lambda X} < 0$. Therefore, the second derivative is strictly negative on $[a, b]$. The expectation $\mathbb{E}(W(X))$ of the work saved when checkpointing at time X is a concave function on $[a, b]$. The zero of the first derivative is thus a maximum. Altogether, the optimal value X_{opt} is given by

$$X_{opt} = \min\left(\frac{-\mathcal{W}(e^{-\lambda a + \lambda R + 1}) + \lambda R + 1}{\lambda}, b\right).$$

Figure 7 provides an example of each case (optimal reached before b or at b). Again, the main take-away is that preparing for the worst-case of checkpoint duration and choosing $X = b$ is not always a good strategy. Contrarily to the Uniform law, the Exponential law requires to compute a complicated value for X_{opt}, but this can be done easily with available tools like[19].



(a) Graph of $\mathbb{E}(W(X))$. The maximum is $X_{opt} \approx 3.9$ with $a = 1, b = 5, R = 10, \lambda = \frac{1}{2}$.



(b) Graph of $\mathbb{E}(W(X))$. The maximum is $X_{opt} = b$ with $a = 1, b = 3, R = 10, \lambda = \frac{1}{2}$.

Figure 7. Both cases for X_{opt} with an Exponential law.

6.1.4 Normal Probability Distribution. Let Φ and φ the CDF and PDF of the standard Normal law: $\varphi(t) = \frac{e^{-\frac{t^2}{2}}}{\sqrt{2\pi}}$ and $\Phi(x) = \int_{-\infty}^{x} \varphi(t)dt$. The Normal law with mean μ and standard deviation σ has CDF $\Phi(\frac{x-\mu}{\sigma})$. We assume that C obeys the Normal law with mean $\mu \in [a, b]$ and standard deviation $\sigma > 0$ truncated to $[a, b]$. From Section 6.1.1, we have

$$\mathbb{E}(W(X)) = \begin{cases} \frac{\Phi(\frac{X-\mu}{\sigma}) - \Phi(\frac{a-\mu}{\sigma})}{\Phi(\frac{b-\mu}{\sigma}) - \Phi(\frac{a-\mu}{\sigma})}(R - X) & \text{if } X \leq b \\ R - X & \text{otherwise} \end{cases}$$

We get rid of the positive constant $K = \frac{1}{\Phi(\frac{b-\mu}{\sigma}) - \Phi(\frac{a-\mu}{\sigma})}$ by letting $g(X) = K\mathbb{E}(W(X))$. Differentiating for $X < b$, we get

$$\begin{aligned} g'(X) &= \frac{d\mathbb{E}(W(X))}{dX} = \frac{1}{\sigma}\varphi\left(\frac{X-\mu}{\sigma}\right)(R-X) \\ &\quad - \left[\Phi\left(\frac{X-\mu}{\sigma}\right) - \Phi\left(\frac{a-\mu}{\sigma}\right)\right] \end{aligned}$$

We have $g'(a) = \frac{1}{\sigma}\varphi(\frac{a-\mu}{\sigma})(R-a) > 0$ since $a < R$. We also have $g'(R) = -[\Phi(\frac{R-\mu}{\sigma}) - \Phi(\frac{a-\mu}{\sigma})]$. We see that $g'(R) < 0$ since Φ is an increasing function and $a < R$. Since g' is continuous, the intermediate value theorem shows that there exists $c \in [a, R]$ such that $g'(c) = 0$. Differentiating again:

$$\begin{aligned} g''(X) &= \frac{d^2\mathbb{E}(W(X))}{dX^2} \\ &= -\frac{\varphi(\frac{X-\mu}{\sigma})}{\sigma}\left[2 + \left(\frac{X-\mu}{\sigma}\right)(R-X)\right], \end{aligned}$$

and g'' has two zeros:

$$\begin{aligned} X_1 &= \frac{R+\mu-\sqrt{(R+\mu)^2+8\sigma^2-4\mu R\sigma}}{2}, \\ X_2 &= \frac{R+\mu+\sqrt{(R+\mu)^2+8\sigma^2-4\mu R\sigma}}{2}. \end{aligned}$$

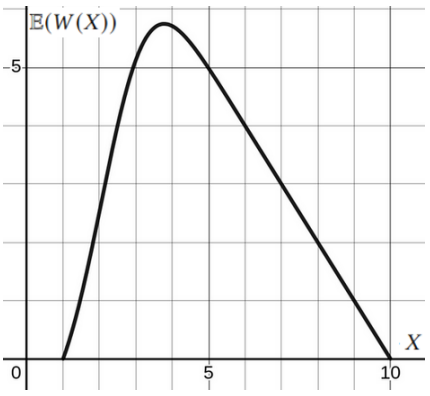
We see that $X_1 < \mu < R < X_2$. Furthermore, we have

$$\begin{cases} g''(X) > 0 \text{ if } X < X_1 \text{ or } X > X_2, \\ g''(X) < 0 \text{ if } X_1 < X < X_2. \end{cases}$$

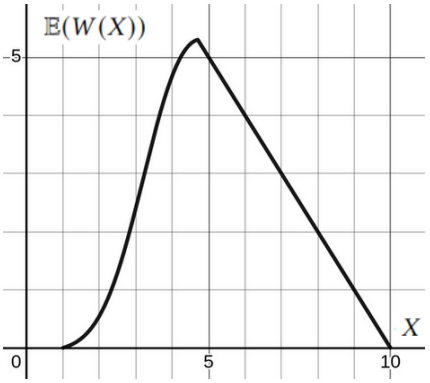
Hence, g is a concave function on $[X_1, X_2]$, and a convex function elsewhere. There are two possible cases:

1. Either $a \geq X_1$, and then g is concave on $[a, R]$; hence the zero c of g' is a maximum of g.
2. Or $X_1 > a$, and then g' is increasing $[a, X_1]$; since $g'(a) > 0$, g' is positive on $[a, X_1]$. Hence $c \in]X_1, R[$ and g is concave in a neighborhood of c, and c is again a maximum of g.

Altogether, we have shown the existence of a maximum $c \in]a, R]$ for the expectation $\mathbb{E}(W(X))$, namely $X_{opt} = \min(c, b)$. We do not have an explicit formula for X_{opt}, but we can evaluate it numerically. Figure 8 provides an example of each case (optimal reached before b or at b). The main take-away for the Normal law is the same as for the Exponential law.



(a) Graph of $\mathbb{E}(W(X))$. The maximum is $X_{opt} \approx 4.2$ with $a = 1$, $b = 5.5$, $R = 10$, $\mu = 2.3$, $\sigma = 1$.



(b) Graph of $\mathbb{E}(W(X))$. The maximum is $X_{opt} = b$ with $a = 1$, $b = 4.7$, $R = 10$, $\mu = 3.5$, $\sigma = 1$.

Figure 8. Both cases for X_{opt} with a Normal law.

6.2 With Failures

This section deals with the most difficult problem, namely FTC with a stochastic checkpoint time. Given the difficulty of the problem with a constant checkpoint time (see Section 5), we have to resort to time discretization and, once again, to a dynamic programming approach.

We re-use all the notations with discrete time quanta from Section 5.3. In addition, the cost of a checkpoint C^* can take any value v (in quanta) in the interval $[C^*_{\min}, C^*_{\max}]$, with probability p_v. Here, $C^*_{\min}$ and $C^*_{\max}$ are the minimum and maximum possible number of quanta for the checkpoint, and $\sum_{v=C^*_{\min}}^{C^*_{\max}} p_v = 1$. We assume that a recovery is proportional to the previous checkpoint: after a checkpoint of duration C^*, a recovery takes a time $R^* = \beta C^*$.

Adapting the notations of Section 5.3, let $\mathbb{E}(n, k, R^*)$ be the optimal work that can be achieved (in expectation) during n quanta, when taking exactly k checkpoints initially, and when we must start by making a recovery of duration R (if there is no recovery to be paid, then $R = 0$). We introduce an auxiliary function $\mathcal{E}(n, k, R^*, i, C^*)$, which is the optimal work that can be achieved (in expectation) during n quanta, when taking exactly k checkpoints initially, when the first checkpoint lasts a duration C^* and starts right after quantum i, and when we must start by making a recovery of duration R^*. When $k = 1$, the expression for $\mathcal{E}$ is easily derived from Equation (14) written for the case with constant time checkpoints: the only difference is the meaning of variable i, which is no longer the quantum at which the checkpoint completes, but the quantum after which the checkpoint starts (since we no longer know in advance how long a checkpoint will last):

$$\begin{aligned} \mathcal{E}(n, 1, R^*, i, C^*) &= \mathbb{P}^*_{succ}(i + C^*)(i - R^*) \\ &\quad + \sum_{f=1}^{i+C^*} p^*_f \mathbb{E}(n - f - D^*, 1, R^*). \end{aligned}$$

To derive the expression for $\mathbb{E}(n, 1, R^*)$ we only need to scan all the possible start times for the checkpoint and consider, with their probabilities, all potential checkpoint durations.

Hence:

$$\mathbb{E}(n,1,R^*) = \max_{i \geq 1+R^*} \sum_{C^*=C^*_{\min}}^{\min\{C^*_{\max}, n-i\}} p_{C^*} \mathcal{E}(n,1,R^*,i,C^*).$$

The sum is truncated at $n-i$ because if $C^* > n-i$, there is not enough time to complete the checkpoint and, therefore, no work is saved.

The derivation for the general case ($k > 1$) is then straightforward:

$$\begin{aligned}\mathcal{E}(n,k,R^*,i,C^*) = \\ \mathbb{P}^*_{succ}(i+C^*)(i-R^*+\mathbb{E}(n-i-C^*,k-1,\beta C^*)) \\ + \sum_{f=1}^{i+C^*} p^*_f \max_{1\leq k' \leq k} \mathbb{E}(n-f-D^*,k',R^*),\end{aligned}$$

and

$$\mathbb{E}(n,k,R^*) = \max_{i \geq 1+R^*} \sum_{C^*=C^*_{\min}}^{\min\{C^*_{\max}, n-i\}} p_{C^*} \mathcal{E}(n,k,R^*,i,C^*).$$

The initialization is similar to what we had previously:

$$\mathbb{E}(n,k,R^*) = 0 \quad \text{and} \quad \mathcal{E}(n,k,R^*,i,C^*) = 0$$

for $n \leq R^* + kC^*$, where possibly $n \leq 0$. The complexity of this dynamic programming algorithm is $O((T^*)^2 (k_{max})^2 (C^*_{\max} - C^*_{\min} + 1)^2)$.

7 Conclusion

This work has focused on the FTC problem, where the objective is to maximize the expected work achieved during a fixed-length reservation. The FTC problem is the dual of the classical FWC problem, where the objective is to minimize the expected time to execute a fixed amount of work. To the best of our knowledge, the *fixed-time checkpointing* problem has only been studied in our previous work [2], 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. Finally, we have extended the dynamic programming algorithm to deal with stochastic checkpoint values for the FTC problem.

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. For the discretization, an experimental assessment of the value to choose for the quanta would be compelling, since it would reveal the best trade-off between accuracy of the result and speed of the approach. Also, a limitation of this work is to assume full knowledge of the platform MTBF, while one could envision dynamic scenarios where we adapt the heuristics to the on-the-fly discovery of the frequency at which failures are striking. In addition, it would be compelling to extend the study of FTC 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 thank the reviewers for their comments and suggestions, which helped improve the final version of the paper. A short preliminary version of this work has appeared in FTXS'2024 [2]. Section 6.1 is excerpted from our FTXS'2023 paper [1]. We would like to thank the anonymous reviewer of this latter paper, whose question motivated our work on the fixed-time checkpointing problem.

References

1. Barbut Q, Benoit A, Herault T, Robert Y and Vivien F (2023) 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.
2. Benoit A, Perotin L, Robert Y and Vivien F (2024) Checkpointing strategies for a fixed-length execution. In: *FTXS'2024, the 14th Workshop on Fault-Tolerance for HPC at Extreme Scale, in conjunction with SC'2024*. IEEE Press.
3. Benoit A, Perotin L, Robert Y and Vivien F (2024) Checkpointing strategies for a fixed-length execution. Research report RR-9552, Inria. Available at `https://inria.hal.science/hal-04668191`.
4. Benoit A, Perotin L, Robert Y and Vivien F (2024) Checkpointing strategies to tolerate non-memoryless failures on HPC platforms. *ACM Trans. Parallel Computing* 11(1).
5. Bland W, Bouteiller A, Herault T, Hursey J, Bosilca G and Dongarra JJ (2013) An evaluation of User-Level Failure Mitigation support in MPI. *Computing* 95(12): 1171–1184.
6. Bougeret M, Casanova H, Rabie M, Robert Y and Vivien F (2011) Checkpointing strategies for parallel jobs. In: *Proceedings of SC'11*.
7. Cappello F, Geist A, Gropp W, Kale S, Kramer B and Snir M (2014) Toward exascale resilience: 2014 update. *Supercomputing frontiers and innovations* 1(1).

8. Daly JT (2006) A higher order estimate of the optimum checkpoint interval for restart dumps. *Future Generation Comp. Syst.* 22(3): 303–312.
9. Dongarra J, Hérault T and Robert Y (2014) Performance and reliability trade-offs for the double checkpointing algorithm. *Int. J. of Networking and Computing* 4(1): 23–41.
10. Ferreira K, Stearley J, Laros JHI, Oldfield R, Pedretti K, Brightwell R, Riesen R, Bridges PG and Arnold D (2011) Evaluating the Viability of Process Replication Reliability for Exascale Systems. In: *SC'11*. ACM.
11. Gamell M, der Wijngaart RFV, Teranishi K and Parashar M (2016) Specification of fenix MPI fault tolerance library, version 1.0.1. Technical Report SAND2016-10522, Sandia National Laboratory. `https://www.osti.gov/servlets/purl/1330192`.
12. Herault T and Robert Y (eds.) (2015) *Fault-Tolerance Techniques for High-Performance Computing*, Computer Communications and Networks. Springer Verlag.
13. Levy S, Hemmert J, Ferreira KB and Pedretti K (2024) Characterizing the Impact of Job Execution on the Occurrence of Memory Failures on a Petascale HPC System. In: *IEEE 36th Int. Symp. Computer Architecture and High Performance Computing (SBAC-PAD)*. pp. 116–126.
14. Plank JS, Li K and Puening MA (1998) Diskless checkpointing. *IEEE Trans. Parallel and Distributed Systems* 9(10): 972–986.
15. Schroeder B and Gibson GA (2007) Understanding Failures in Petascale Computers. *Journal of Physics: Conference Series* 78(1).
16. Sigdel P, Yuan X and Tzeng N (2021) Realizing best checkpointing control in computing systems. *IEEE TPDS* 32(2): 315–329.
17. Wei XH, Tong SY, Sun ZA, Li X and Yue HS (2025) ResCheckpointer: Building Program Error Resilience-Aware Checkpointing Mechanism for HPC Systems. *Journal of Computer Science and Technology* : 1–15.
18. Wikipedia (2023) Application checkpointing. `https://en.wikipedia.org/wiki/Application_checkpointing`.
19. Wolfram Alpha (2023) Mathematics. `https://www.wolframalpha.com`.
20. Young JW (1974) A first order approximation to the optimum checkpoint interval. *Comm. of the ACM* 17(9): 530–531.
21. Zheng G, Ni X and Kalé LV (2012) A scalable double in-memory checkpoint and restart scheme towards exascale. In: *IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN 2012)*. IEEE, pp. 1–6.