HAL
open science

Checkpointing Workflows à la Young/Daly Is Not Good Enough

Anne Benoit, Lucas Perotin, Yves Robert, Hongyang Sun

To cite this version:

Anne Benoit, Lucas Perotin, Yves Robert, Hongyang Sun. Checkpointing Workflows à la Young/Daly Is Not Good Enough. [Research Report] RR-9413, Inria - Research Centre Grenoble – Rhône-Alpes. 2021, pp.54. hal-03264047

HAL Id: hal-03264047

https://inria.hal.science/hal-03264047

Submitted on 17 Jun 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Checkpointing Workflows à la Young/Daly Is Not Good Enough

Anne Benoit, Lucas Perotin, Yves Robert Hongyang Sun

RESEARCH REPORT
N° 9413
June 2021
Project-Team ROMA

ISSN 0249-6399 ISRN INRIA/RR--9413--FR+ENG

Checkpointing Workflows à la Young/Daly Is Not Good Enough

Anne Benoit*, Lucas Perotin*, Yves Robert*† Hongyang Sun‡

Project-Team ROMA

Research Report n° 9413 — June 2021 — 54 pages

Abstract: This paper revisits checkpointing strategies when workflows composed of multiple tasks execute on a parallel platform. The objective is to minimize the expectation of the total execution time. For a single task, the Young/Daly formula provides the optimal checkpointing period. However, when many tasks execute simultaneously, the risk that one of them is severely delayed increases with the number of tasks. To mitigate this risk, a possibility is to checkpoint each task more often than with the Young/Daly strategy. But is it worth slowing each task down with extra checkpoints? Does the extra checkpointing make a difference globally? This paper answers these questions. On the theoretical side, we prove several negative results for keeping the Young/Daly period when many tasks execute concurrently, and we design novel checkpointing strategies that guarantee an efficient execution with high probability. On the practical side, we report comprehensive experiments that demonstrate the need to go beyond the Young/Daly period and to checkpoint more often, for a wide range of application/platform settings.

Key-words: Checkpoint, workflow, concurrent tasks, Young/Daly formula.

* Université de Lyon and Laboratoire LIP, École Normale Supérieure de Lyon, CNRS, Inria & UCB Lyon, France

† University of Tennessee Knoxville, USA

‡ Vanderbilt University, Nashville, TN, USA

RESEARCH CENTRE GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

La période de checkpoint de Young/Daly n'est pas optimale pour l'exécution de graphes de tâches

Résumé : Cet article étudie les stratégies de checkpoint pour l'exécution de graphes de tâches (applications de type *workflow*). La formule de Young/Daly est optimale pour minimiser l'espérance du temps d'exécution d'une seule tâche. Mais quand plusieurs tâches s'exécutent en parallèle, le risque est grand que l'une d'entre elles soit retardée significativement, et partant, que soit retardée l'exécution de ses successeurs dans le graphe de tâches. Nous étudions la meilleure stratégie de checkpoitnt dans ce contexte, et montrons qu'effectivement il faut prendre des checkpoints plus souvent pour obtenir une solution efficace avec très grande probabilité. Nous conduisons des simulations sur des graphes de tâches de référence, qui confirment les résultats théoriques.

Mots-clés : checkpoint, graphe de tâches, tâches concurrentes, formule de Young/Daly.

1 Introduction

Checkpointing is the standard technique to protect applications running on HPC (High Performance Computing) platforms. Every day, the platform will experience a few fail-stop errors (or failures, we use both terms indifferently). 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, after some downtime (enroll a spare to replace the faulty processor) and a recovery (read the checkpoint).

Consider an application, composed of a unique task, executing on a platform whose nodes are subject to fail-stop errors. How frequently should it be checkpointed so that its expected execution time is minimized? 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 optimal checkpointing period is given by the Young/Daly formula as $W_{YD} = \sqrt{2\mu C}$ [37, 11], where μ is the application MTBF (Mean Time Between Failures) and C the checkpoint duration. Recall that if the application executes on p processors, its MTBF is $\mu = \frac{\mu_{ind}}{p}$, where μ_{ind} is the individual processor's MTBF: in other words, the MTBF is inversely proportional to the number of processors enrolled, which is intuitive in terms of failure frequency (see [25] for a formal proof).

Now, assume that, say, 300 independent applications (each with a unique task) have been launched concurrently on the platform. For the sake of illustration, assume that these 300 applications are identical: same number of processors $p = 30$, same length $T_{base} = 10$ hours, and same checkpoint duration $C = 6$ minutes. Hence, the platform has at least $m = 9,000$ processors. Assume a short downtime $D = 1$ minute, and recovery time $R = C$. Finally, assume that each task has 0.5% chances to fail during execution; this setting corresponds to an individual MTBF μ_{ind} such that $1 - e^{-\frac{pT_{base}}{\mu_{ind}}} = 0.005$, i.e., $\mu_{ind} = 59,850$ hours (or 6.8 years). This is in accordance with MTBFs typically observed on large-scale platforms, which range from a few years to a few dozens of years [7]. For each task, the Young/Daly period is $W_{YD} = \sqrt{2\frac{\mu_{ind}}{p}C} \approx 20$ hours, and the expected execution time of a single task $\mathbb{E}(T_{1\text{-}task})$, is minimized either when no checkpoint is taken or if a single checkpoint is taken at the end of the execution (see Section 2.6). We assume that we always take a checkpoint at the end of the execution of a task, e.g., to save final results on stable storage[1]. Then, we derive that $\mathbb{E}(T_{1\text{-}task}) \approx 10.4$ (see Section 2.6).

Is it safe to checkpoint each task individually à la Young/Daly? The problem comes from the fact that the expectation $\mathbb{E}(T_{all\text{-}tasks})$ of the maximum execution time over all tasks, i.e., the expectation of the total time required to complete all tasks, is far larger than the maximum of the expectations (which in the example have all the same value $\mathbb{E}(T_{1\text{-}task})$). When a single checkpoint is taken at the end of each task, we compute that $\mathbb{E}(T_{all\text{-}tasks}) > 14$, while adding four intermediate checkpoints to each task reduces it down to $\mathbb{E}(T_{all\text{-}tasks}) < 12.75$

[1] We make this assumption throughout the paper for simplicity. Appendix A extends the analysis to the case where no checkpoint is taken at the end of the execution of a task. Changes are minimal and results are quite similar.

(see Section 2.6 for details of the computation of both numbers). Intuitively, this is because adding these intermediate checkpoints greatly reduces the chance of re-executing any single task from scratch when it is struck by a failure, and the probability of having at least one failed task increases with the number of tasks. Of course, there is a penalty from the user's point of view: Adding four checkpoints to each task augments their length by 24 minutes, while the majority of them will not be struck by a failure. In other words, users may feel that their response time has been unduly increased, and state it is not worth to add these extra checkpoints.

Going one step further, consider now a single application whose dependence graph is a simple fork-join graph, made of 302 tasks: an entry task, 300 parallel tasks identical to the tasks above (each task runs on $p = 30$ processors for $T_{base} = 10$ hours, and is checkpointed in $C = 6$ minutes) and an exit task. Such applications are typical of HPC applications that explore a wide range of parameters or launch subproblems in parallel. Now, the extra checkpoints make full sense, because the exit task cannot start before the last parallel task has completed. The expectation of the total execution time is $\mathbb{E}(T_{total}) = \mathbb{E}(T_{entry}) + \mathbb{E}(T_{all\text{-}tasks}) + \mathbb{E}(T_{exit})$, where $\mathbb{E}(T_{entry})$ and $\mathbb{E}(T_{exit})$ are the expected durations of the entry and exit tasks, and $\mathbb{E}(T_{total})$ is minimized when $\mathbb{E}(T_{all\text{-}tasks})$ is minimized. By diminishing $\mathbb{E}(T_{all\text{-}tasks})$, we save 1.25 hour, or 75 minutes (and in fact much more than that, because the lower and upper bounds for $\mathbb{E}(T_{all\text{-}tasks})$ are loosely computed).

This last example shows that the optimal execution of large workflows on failure-prone platforms requires to checkpoint each workflow task more frequently than prescribed by the Young/Daly formula. The main focus of this paper is to explore various checkpointing strategies, and our main contributions are the following:

- We provide approximation bounds for the performance of MinExp, a strategy à la Young/Daly that minimizes the expected execution time of each task, and for a novel strategy CheckMore that performs more checkpoints than MinExp.
- Both bounds apply to workflows of arbitrary shape, and whose tasks can be either rigid or moldable. In addition, we exhibit an example where the bounds are tight and where CheckMore can be an order of magnitude better than MinExp.
- The novel CheckMore strategy comes in two flavors, one that tunes the number of checkpoints as a function of the degree of parallelism in the failure-free schedule, and a simpler one that does not require any knowledge of the failure-free schedule, beyond a priority list to decide in which order to start executing the tasks.
- We report comprehensive simulations results based on WorkflowHub testbeds [18], which demonstrate the significant gain brought by CheckMore over MinExp for almost all testbeds.

The paper is organized as follows. We first describe the model in Section 2. We assess the performance of MinExp in Section 3; performance bounds are proven both for independent tasks and for general workflows. Section 4 presents the novel strategy CheckMore that checkpoints workflow tasks more often than MinExp, and analyzes its theoretical performance. The experimental evaluation in Section 5 presents extensive simulation results comparing both strategies. Finally, we discuss related work in Section 6, and conclude in Sec-

tion 7.

2 Model and Background

In this section, we first detail the platform and application models, and describe how to practically deploy a workflow with checkpointed jobs. Then, we discuss the objective function before providing background on the optimal checkpointing period for preemptible tasks, and getting back to the example of the introduction. Key notations are summarized in Table 1.

2.1 Platform

We consider a large parallel platform with m identical processors, or nodes. These nodes are subject to fail-stop errors, or failures. A failure interrupts the execution of the node and provokes the loss of its whole memory. Consider a parallel application running on several nodes: when one of these nodes is struck by a failure, the state of the application is lost, and execution must restart from scratch, unless a fault-tolerance mechanism has been deployed.

The classical technique to deal with failures makes use of a checkpoint-restart mechanism: the state of the application is periodically checkpointed, i.e., all participating nodes take a checkpoint simultaneously. This is the standard coordinated checkpointing protocol, which is routinely used on large-scale platforms [9], where each node writes its share of application data to stable storage (checkpoint of duration C). When a failure occurs, the platform is unavailable during a downtime D, which is the time to enroll a spare processor that will replace the faulty processor [11, 25]. Then, all application nodes (including the spare) recover from the last valid checkpoint in a coordinated manner, reading the checkpoint file from stable storage (recovery of duration R). Finally, the execution is resumed from that point on, rather than starting again from scratch. Note that failures can strike during checkpoint and recovery, but not during downtime (otherwise we can include the downtime in the recovery time).

Throughout the paper, we add a final checkpoint at the end of each application task, to write final outputs to stable storage. Symmetrically, we add an initial recovery when re-executing the first checkpointed segment of a task (to read inputs from stable storage) if it has been struck by a failure before completing the checkpoint. See Appendix A for an extension relaxing either or both assumptions.

m	Total number of processors
p	Number of processors per task
n	Number of tasks
$\mu_{ind} = \frac{1}{\lambda}$	Individual processor's MTBF
C	Checkpoint time
R	Recovery time
D	Downtime
T_{base}	Task duration without failures
N_{ME}	Number of segments with MinExp strategy
W_{ME}	Segment length with MinExp strategy

Table 1: Key notations.

We assume that each node experiences failures whose inter-arrival times follow an Exponential distribution $Exp(\lambda)$ of parameter $\lambda > 0$, whose PDF (Probability Density Function) is $f(x) = \lambda e^{-\lambda x}$ for $x \geq 0$. The individual MTBF of each node is $\mu_{ind} = \frac{1}{\lambda}$. Even if each node has an MTBF of several years, large-scale parallel platforms are composed of so many nodes that they will experience several failures per day [17, 7]. Hence, a parallel application using a significant fraction of the platform will typically experience a failure every few hours.

2.2 Application

We focus on HPC applications expressed as workflow graphs, such as those available in WorkflowHub [18] (formerly Pegasus [35]). The shape of the task graph is arbitrary, and the tasks can be parallel. We further assume that all tasks are preemptible, i.e., that we can take a checkpoint at any instant.

For the theoretical analysis, we use workflows whose tasks can be rigid or moldable parallel tasks. A moldable task can be executed on an arbitrary number of processors, and its execution time depends on the number of processors allotted to it. This corresponds to a variable static resource allocation, as opposed to a fixed static allocation (rigid tasks) and a variable dynamic allocation (malleable tasks) [15]. Scheduling rigid or moldable workflows is a difficult NP-hard problem (see the related work in Section 6) . We take as input a failure-free schedule for the workflow and transform it by adding checkpoints as follows. The failure-free schedule provides an ordered list of tasks, sorted by non-decreasing starting times. Our failure-aware algorithms are list schedules that greedily process the tasks (augmented with checkpoints) in this order: if task T is number i in the original failure-free schedule, then T is scheduled after the $i-1$ first tasks in the failure-aware schedule, and no other task can start before T does. Hence, the processors allocated to T in the failure-aware schedule may differ from those allocated in the failure-free schedule. Enforcing the same ordering of execution of the tasks may be sub-optimal, but it is the key to guarantee approximation ratios for the total execution time.

For the experiments, we restrict to workflows with uni-processor tasks, in accordance with the characteristics of the workflow benchmarks from WorkflowHub.

2.3 Implementation in a Cluster Environment

This section briefly describes two approaches to deploy a workflow with checkpointed jobs in a cluster environment.

The first approach is to use the job scheduler LSF [28] and to submit a set of jobs with their dependencies: there are as many jobs as tasks in the workflow, and these jobs are declared *checkpointable*. The system will relaunch a job after it is hit by a failure, from the last checkpoint on and until success (see the 'job failover' section in [28]). If the failed job was using j processors, then it releases $j-1$ surviving processors right after the failure; if there is at least one other processor available, the job can be rescheduled right away (jobs usually get high priority when they are rescheduled after a failure). Otherwise, the failed job will have to wait and this waiting time, a.k.a. the re-submission time, is dependent on the platform scheduling policy and on the availability of nodes.

A second approach is to submit a single job with $p + q$ processors, where p processors represent the allotment for the whole workflow and q processors are spare. The job uses a master process that spans the workflow tasks and controls how their execution progresses; the tasks are checkpointed using a standard software such as VeloC [8]. The spare nodes are mutualized across the tasks either by using a fault tolerant MPI library like ULFM [5, 14], or by having the master process launch each task as independent MPI applications spanning on subgroups of the reservation, and re-launching them from their last checkpoint on the surviving nodes and the spare nodes if some task is subject to failure.

In the first approach, the downtime would be non-constant, because it corresponds to the re-submission time, while in the second approach with spares, the downtime can be approximated as a constant. Regardless, all the results of this paper are taken in expectation, and they extend to using an average value of the downtime whenever a fixed value is not appropriate.

Finally, we stress that this work is agnostic of system management policies and does not modify any parameter specified by the user for the job allocations; we simply increase the checkpoint frequency when needed, which results in shorter execution time and better processor utilization for the workflow.

2.4 Objective Function

Given a workflow composed of a set of tasks, where each task executes on a given number of processors, the objective function is to minimize the expected makespan of the workflow, i.e., the expected total execution time to complete all tasks. We aim at determining the best checkpointing strategy for the tasks that compose the workflow. This is the only parameter that we modify in the execution: we keep the number of processors specified by the user, and we even keep the order of the tasks as given by the user schedule. The replacement of failed nodes, or the resubmission of failed tasks, is decided by the system and does not depend upon the checkpointing policy, either à la Young/Daly, or one of our new strategies.

As a result, minimizing the expected makespan of the workflow also maximizes processor utilization of the platform, because the processors reserved by the user will be released earlier on, and with no additional cost for the rest of the platform.

In the analysis of the checkpointing strategies, we focus on bounding the *ratio*, which is defined as the expected makespan of the workflow (i.e., the expected total execution time) divided by the makespan in the failure-free execution (no checkpoints nor failures), given a user-specified schedule. Hence, the ratio shows the overhead induced by failures and the checkpointing strategy: the closer to one, the better.

2.5 Checkpointing Period

Consider an application A composed of a single parallel task executing on p processors. Assume that the task is preemptible, which means that it can be checkpointed at any instant. The key for an efficient checkpointing policy is to decide when to checkpoint. If the application A runs for a duration T_{base} (base time without checkpoints nor failures), then the optimal checkpointing strategy,

i.e., the strategy minimizing the expected execution time of the application, can be derived as shown below.

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

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

Proof. This is the result of [6, Theorem 1]. Note that Lemma 1 also applies when the segment is not followed by a checkpoint (take $C=0$). □

The *slowdown function* is defined as $f(W, C, R) = \frac{\mathbb{E}(W,C,R)}{W}$.
We have the following properties:

Lemma 2. *The slowdown function $W \mapsto f(W, C, R)$ 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, this is the result of [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 $W_{YD} = \sqrt{\frac{2C}{p\lambda}}$. □

Lemma 2 shows that infinite tasks should be partitioned into segments of size W_{YD} followed by a checkpoint. What about finite tasks? Back to our application A of duration T_{base}, we partition it into N_c segments of length W_i, $1 \le i \le N_c$, each followed by a checkpoint C. By linearity of the expectation, the expected time to execute the application A is

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

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

$$\mathbb{E}(A) = N_c \mathbb{E}\left(\frac{T_{base}}{N_c}, C, R\right) = f\left(\frac{T_{base}}{N_c}, C, R\right) \times T_{base},$$

where f is the slowdown function. Define $K_{opt} = \frac{T_{base}}{W_{opt}}$, where W_{opt} achieves the minimum of the slowdown function. K_{opt} would be the optimal value if we could have a non-integer number of segments. Lemma 2 shows that the optimal value N_{ME} of N_c is either $N_{ME} = \max(1, \lfloor K_{opt} \rfloor)$ or $N_{ME} = \lceil K_{opt} \rceil$, whichever leads to the smallest value of $\mathbb{E}(A)$. In this paper, to avoid the numerical evaluation of the Lambert function for W_{opt}, we use the simplified expression $N_{ME} = \left\lceil \frac{T_{base}}{W_{YD}} \right\rceil$:

Definition 1. *The* MinExp *checkpointing strategy partitions a parallel task of length T_{base}, with p processors and checkpoint time C, into $N_{ME} = \left\lceil \frac{T_{base}}{W_{YD}} \right\rceil$ equal-length segments, each followed by a checkpoint, where $W_{YD} = \sqrt{\frac{2C}{p\lambda}} = \sqrt{\frac{2\mu_{ind}C}{p}}$. Each segment is of length $W_{ME} = \frac{T_{base}}{N_{ME}}$.*

2.6 Back to the Example

In the introduction, we used the example of 300 identical tasks, each with $T_{base} = 10$ hours, $p = 30$, and $C = 6$ minutes. We also had $D = 1$ minute and $R = C$. We assume that each task has 0.5% chances to fail during execution,

which corresponds to an individual MTBF μ_{ind} such that $1 - e^{-\frac{pT_{base}}{\mu_{ind}}} = 0.005$. This equality leads to $\mu_{ind} = 59,850$ hours. We derive $W_{YD} = \sqrt{2\mu_{ind}C/p} \approx 20$ hours, hence $N_{ME} = 1$. With a single segment, we then compute the optimal expected execution time $\mathbb{E}(T_{1\text{-}task})$ for each task as:

$$\mathbb{E}(T_{1\text{-}task}) = \left(\frac{\mu_{ind}}{p} + D\right) e^{\frac{pR}{\mu_{ind}}} \left(e^{\frac{p}{\mu_{ind}}(T_{base}+C)} - 1\right) \approx 10.4.$$

With 300 tasks executing concurrently, we compute that the expectation of the total time required to complete all tasks is at least $\mathbb{E}(T_{all\text{-}tasks}) > 14$, hence the ratio is $\frac{14}{10} = 1.4$. Indeed, there is no failure at all with probability $\left(e^{-\frac{p(T_{base}+C)}{\mu_{ind}}}\right)^{300} < 0.23$, and in this case the execution time is $T_{base} + C = 10.1$. The other case, happening with a probability larger than 0.77, is when at least one failure occurs in the process, and we will bound its expected execution time if exactly one failure occurs, which is clearly lower than the actual expected execution time. To that end, we compute the expected time lost before the failure occurs when attempting to successfully execute for $T = T_{base} + C$ hours: $\mathbb{E}(T_{lost}(T)) = \int_0^\infty x\mathbb{P}(X = x|X < T)dx = \frac{1}{\mathbb{P}(X<T)} \int_0^T xe^{-p\lambda x}dx$, with $\mathbb{P}(X < T) = 1 - e^{-p\lambda T}$. Integrating by parts, we derive that:

$$\mathbb{E}(T_{lost}(T)) = \frac{1}{p\lambda} - \frac{T}{e^{p\lambda T} - 1}. \tag{2}$$

In the example, we have $T = T_{base}+C = 10.1$, $p = 30$, and $\lambda = \frac{1}{\mu_{ind}} = \frac{-\ln(0.995)}{pT_{base}}$. Thus, if a failure strikes one of the tasks, the expected time lost is higher than 5.045 hours. After that, we also have to wait $D > 0.016$ hour of downtime and recover for a duration of $R = 0.1$ hour. Overall, the expected execution time satisfies $\mathbb{E}(T_{all\text{-}tasks}) \geq 10.1+0.77\times(\mathbb{E}(T_{lost}(T))+R+D) > 10.1+0.77\times 5.161 > 14$. Note that this lower bound is far from tight.

When adding four intermediate checkpoints to each task, we reduce $\mathbb{E}(T_{all\text{-}tasks})$ down to $\mathbb{E}(T_{all\text{-}tasks}) < 12.75$. Indeed, the tasks are now slightly longer (10.5 hours without failure), and they fail with probability $1 - e^{-\frac{30\times 10.5}{59850}} < 0.006$. Let M_f denote the maximum number of failures of any tasks. Clearly, we have $\mathbb{P}\{M_f \geq k\} \leq 300 \times 0.006^k$. The worst-case scenario for each failure is when it happens just before the end of a checkpoint, and in that case we loose at most $2 + 0.1 + 0.1 + 0.017 < 2.22$ for each failure (the length of a segment, the checkpoint time, the recovery time and the downtime). Thus, $\mathbb{E}(T_{all\text{-}tasks}) < 10.5 + 2.22 \sum_{k\geq 1} \mathbb{P}\{M_f \geq k\} < 10.5 + 2.22 + 2.22 \times 300 \times \sum_{k\geq 2} 0.006^k < 12.75$, hence a ratio lower than 1.275, to compare with 1.4 with the MinExp strategy. Note that this upper bound is far from tight. This example shows that the optimal checkpointing strategy should not only be based upon the task profiles, but also upon the number of other tasks that are executing concurrently.

3 Young/Daly for Workflows: the MinExp Strategy

In this section, we prove performance bounds for the MinExp checkpointing strategy, which adds N_{ME} checkpoints to each task, thereby minimizing the expected execution time for each task. We start in Section 3.1 with independent

tasks, first identical and then arbitrary, that can be executed concurrently (think of a shelf of tasks). Next, we move to general workflows in Section 3.2.

3.1 MinExp for Independent Tasks

We start with a word of caution: throughout this section, the proofs of the theorems and the analysis of the examples are long and technically involved. We state the results and provide proof sketches in the text below; all details are available in the Appendices.

3.1.1 Identical Independent Tasks

First we consider identical independent tasks that can be executed concurrently. Recall that m is the total number of processors. We identify a task $\mathcal{T}$ with its type (i.e., set of parameters) $\mathcal{T}= (T_{base}, p, C, R)$: length T_{base}, number of processors p, checkpoint time C, recovery time R.

Theorem 1. *Consider n identical tasks of same type $\mathcal{T}$=(T_{base},p,C,R) to be executed concurrently on $n \times p \leq m$ processors with individual failure rate $\lambda = \frac{1}{\mu_{ind}}$. The downtime is D. For the* MinExp *strategy, N_{ME} is the number of checkpoints, and W_{ME} is the length of each segment, as given by Definition 1. Let $P_{suc}(\tilde{R}) = e^{-p\lambda(W_{ME}+C+\tilde{R})}$ be the probability of success of a segment with re-execution cost $\tilde{R}$ ($\tilde{R} = 0$ if no re-execution, or $\tilde{R} = R$ otherwise), and $Q^* = \frac{1}{1-P_{suc}(R)}$. Let the ratio be $r_{id}^{ME}(n, \mathcal{T}) = \frac{\mathbb{E}(T_{tot})}{T_{base}}$, where $\mathbb{E}(T_{tot})$ is the expectation of the total time T_{tot} of the* MinExp *strategy. We have:*

$$r_{id}^{ME}(n,\mathcal{T}) \leq \left(\frac{\log_{Q^*}(n)}{N_{ME}} + \log_{Q^*}(\log_{Q^*}(n)) + 1 + \frac{\ln(Q^*)}{12N_{ME}} + \frac{1}{\ln(Q^*)N_{ME}}\right) \times \left(1 + \frac{C+R+D}{W_{ME}}\right) + \frac{C}{W_{ME}} + 1 + o(1). \tag{3}$$

Note that if n is small, the ratio holds by replacing all negative or undefined terms by 0.

Proof. First, a segment consists of the re-execution cost $\tilde{R}$, the work W_{ME} and the checkpoint cost C. Since failures may occur during recovery or checkpoint, the total processing time is $W_{ME} + \tilde{R} + C$. Thus, given the exponential failure probability, we have $P_{suc}(\tilde{R}) = e^{-p\lambda(W_{ME}+C+\tilde{R})}$. The MinExp strategy is a $r_{id}^{ME}(n, \mathcal{T})$-approximation of the base time $T_{base} = N_{ME}W_{ME}$, hence also of the optimal expected execution time. Let M_f be the maximum number of failures over all tasks. We process N_{ME} segments of length $W_{ME}+C$, and each failure in a segment incurs an additional time upper bounded by $D+R+W_{ME}+C$. The expectation $\mathbb{E}(T_{tot})$ of the total time T_{tot} of the MinExp strategy is at most:

$$\mathbb{E}(T_{tot}) \leq T_{base} + N_{ME}C + \mathbb{E}(M_f)(W_{ME} + C + R + D), \quad \text{hence}$$

$$r_{id}^{ME}(n, \mathcal{T}) = \frac{\mathbb{E}(T_{tot})}{T_{base}} \leq 1 + \frac{C}{W_{ME}} + \frac{\mathbb{E}(M_f)}{N_{ME}}\left(1 + \frac{C+R+D}{W_{ME}}\right). \tag{4}$$

We continue with the computation of $\mathbb{E}(M_f)$. We first study the random variable (RV) N_f of the number of failures before completing a given task. We have identical segments $(s_1, s_2, ...)$ to process, each of them having a probability of success $p_{s_i} \in \{P_{suc}(R), P_{suc}(0)\}$, and we stop upon reaching the N_{ME} successes. Hence, s_1 is the first trial of the first segment; if s_1 succeeds, which

happens with probability $P_{suc}(0)$, s_2 corresponds to the first trial of the second segment, and succeeds with probability $P_{suc}(0)$; otherwise, s_2 corresponds to the second trial of the first segment, and succeeds with probability $P_{suc}(R)$. We are interested in the number of failures N_f before having N_{ME} successes. Clearly, if N'_f represents the RV for the same problem except that all segments have the same probability of success $P_{suc}(R)$, all segments are less likely or equally likely to succeed, and

$$\forall x, \mathbb{P}\{N'_f \leq x\} \leq \mathbb{P}\{N_f \leq x\}. \tag{5}$$

Now, let M'_f be the RV equal to the maximum of n IID (Independent and identically Distributed) RVs following N'_f. Equation (5) leads to $\mathbb{E}(M'_f) \geq \mathbb{E}(M_f)$. Each N'_f is a negative binomial RV with parameters $(N_{ME}, P_{suc}(R))$. We refine the analysis from [20] by bounding the sum of some Fourier coefficients (see Appendix B for details) to show that

$$\begin{aligned}\mathbb{E}(M'_f) \leq \quad & \log_{Q^*}(n) + (N_{ME}-1)\log_{Q^*}(\log_{Q^*}(n)) \\ & + N_{ME} + \left(\tfrac{\ln(Q^*)}{12} + \tfrac{1}{\ln(Q^*)}\right) + o(1).\end{aligned} \tag{6}$$

Recall that $Q^* = \frac{1}{1-P_{suc}(R)}$. Here, we assume for convenience that $\log_{Q^*}(\log_{Q^*}(n)) \geq 0$, but otherwise we can replace it by 0 and the ratio holds. Plugging the bound of Equation (6) back into Equation (4) leads to Equation (3). □

We provide an informal simplification of the bound in Equation (3). Under reasonable settings, we have $C, D, R \ll \mu_{ind}$, and the probability of success P_{suc} of each segment is pretty high, hence $Q^* > e$. For this reason, we have (i) $\forall x, \log_{Q^*}(x) < \ln(x)$; (ii) $\frac{C+R+D}{W_{ME}} \approx 0$; (iii) $\frac{\ln(Q^*)}{12N_{ME}} \leq 1$; and (iv) $\frac{1}{\ln(Q^*)N_{ME}} \approx 0$. Altogether, the bound simplifies to:

$$r_{id}^{ME}(n, \mathcal{T}) \leq \frac{\ln(n)}{N_{ME}} + \ln(\ln(n)) + 3 + o(1). \tag{7}$$

Here is a more precise statement (proof in Appendix C):

Proposition 1. *We have* $r_{id}^{ME}(n, \mathcal{T}) \leq \frac{4}{5}\left(\frac{\ln(n)}{N_{ME}} + \ln(\ln(n))\right) + 3 + \frac{3}{N_{ME}} + o(1)$ *under the following assumptions:*

- *A checkpoint of length C succeeds with probability at least* 0.99*;*
- $D \leq R \leq C$*;*
- *A segment of length W_{ME} fails with probability at least* 10^{-10}*;*
- $T_{base} > 2(C+R+D)$ *(otherwise the tasks are so small that no checkpoints are needed).*

3.1.2 Tightness of the bound $r_{id}^{ME}(n, \mathcal{T})$ of Theorem 1

Consider a set $\mathcal{T}$ of n identical uni-processor tasks with $T_{base} = 2K-1$, $C = 1$, $D = R = 0$ and $\lambda = \frac{\ln(1+\frac{1}{2K})}{2K}$ so that $e^{-\lambda(T_{base}+C)} = \frac{2K}{2K+1}$. Here, $K \geq 2$ is fixed, and n is the variable. We assume that all tasks execute in parallel, i.e., $m \geq n$. Under these settings, we show in Appendix G.1 that $r_{id}^{ME}(n, \mathcal{T}) = \Theta(\ln(n))$, thereby showing the tightness of the bound given in Theorem 1.

3.1.3 Arbitrary independent tasks

We now proceed with different independent tasks that can be executed concurrently:

Theorem 2. *Consider a set $\mathcal{T}$ of n tasks. The i-th task has profile $\mathcal{T}_i = (T^i_{base}, p_i, C_i, R_i)$. These tasks execute concurrently, hence $\sum_{i=1}^n p_i \leq m$. The individual fault rate on each processor is λ. The downtime is D. For the* MinExp *strategy, N^i_{ME} is the number of checkpoints, and W^i_{ME} is the length of each segment, for task i. Let $P^i_{suc}(R_i) = e^{-p_i\lambda(W^i_{ME}+C_i+R_i)}$ be the probability of success of a segment of task i with re-execution cost R_i, and $Q^*_i = \frac{1}{1-P^i_{suc}(R_i)}$. Then, the* MinExp *strategy is a $r^{ME}(n, \mathcal{T})$-approximation of the failure-free execution time, hence also of the optimal expected execution time, where:*

$$r^{ME}(n, \mathcal{T}) \leq 2 \max_{1\leq i\leq n} \left(r^{ME}_{id}(n, \mathcal{T}_i)\right). \tag{8}$$

The key element of the (very long) proof of Theorem 2 is an important new result (to the best of our knowledge) on expectations of RVs. Please refer to Appendix D. Similarly to identical tasks, under reasonable assumptions, we derive a simplified bound:
$r^{ME}(n, \mathcal{T}) \leq 2\frac{\ln(n)}{\min_{1\leq i\leq n}(N^i_{ME})} + 2\ln(\ln(n)) + 6 + o(1)$.

3.2 MinExp for Workflows

We proceed to the study of MinExp for a workflow of tasks, with task dependencies. We build upon the results for identical tasks (see Equation (3)), that can be reused for each task of the workflow.

Theorem 3. *Let $\mathbb{S}$ be a failure-free schedule of a workflow $\mathcal{W}$ of n tasks. The i-th task has profile $\mathcal{T}_i = (T^i_{base}, p_i, C_i, R_i)$. The individual fault rate on each processor is λ. The downtime is D. Let Δ be the maximum number of tasks processed concurrently by the failure-free schedule $\mathbb{S}$ at any instant. Then, the* MinExp *strategy is a $r^{ME}(\Delta, \mathcal{W})$-approximation of the failure-free execution time, where*

$$r^{ME}(\Delta, \mathcal{W}) \leq 2 \max_{1\leq i\leq n} r^{ME}_{id}(\Delta, \mathcal{T}_i). \tag{9}$$

In other words, the degree of parallelism Δ of the schedule becomes the key parameter to bound the performance of the MinExp strategy, rather than the total number n of tasks in the workflow. Similarly to independent tasks, under reasonable assumptions, we derive a simplified bound:
$r^{ME}(\Delta, \mathcal{W}) \leq 2\frac{\ln(\Delta)}{\min_{1\leq i\leq n}(N^i_{ME})} + 2\ln(\ln(\Delta)) + 6 + o(1)$.

Proof. As stated in Section 2.2, we enforce the same ordering of starting times in the initial schedule $\mathbb{S}$ and in the failure-aware schedule $\mathbb{S}$' returned by MinExp: if task i starts after task j in $\mathbb{S}$, the same will hold in $\mathbb{S}$'. However, we greedily start a task as soon as enough processors are available, which may result in using different processors for a given task in $\mathbb{S}$ and $\mathbb{S}$'. Consider an arbitrary failure scenario, and let T_i be the execution time of task i in $\mathbb{S}$'. Let $T(\mathbb{S}')$ be the total execution time of $\mathbb{S}$'. We want to prove that:

$$\mathbb{E}(T(\mathbb{S}')) \leq 2 \max_{1\leq i\leq n} r^{ME}_{id}(\Delta, \mathcal{T}_i)T(\mathbb{S}), \tag{10}$$

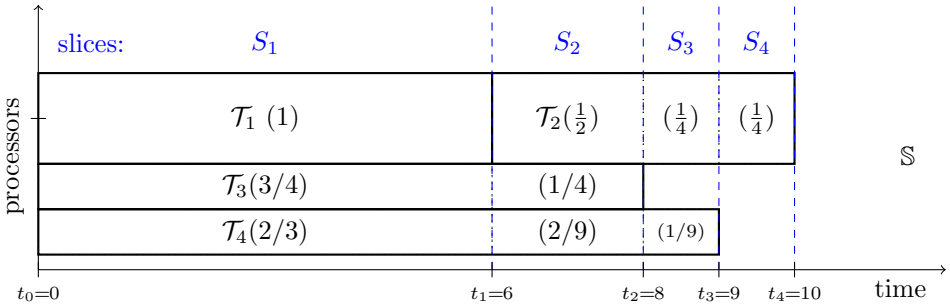

Figure 1: Example for the proof of Theorem 3: Schedule $\mathbb{S}$.

where $T(\mathbb{S})$ is the (deterministic) total execution time of $\mathbb{S}$.

To analyze $\mathbb{S}$', we partition $\mathbb{S}$ into a series of execution slices, where a slice is determined by two consecutive events. An event is either the starting time or the ending time of a task. Formally, let s_i be the starting time of task i in $\mathbb{S}$, and e_i be its ending time. We let $\{t_j\}_{0\le j\le K} = \cup_{i=1}^{n}\{s_i, e_i\}$ denote the set of events, labeled such that $\forall j \in [0, K-1], t_j < t_{j+1}$. Note that we may have $K+1 < 2n$ if two events coincide. We partition $\mathbb{S}$ into K slices S_j, $1 \le j \le K$, which are processed sequentially. Slice S_j spans the interval $[t_{j-1}, t_j]$. In other words, the length of S_j is $t_j - t_{j-1}$. Let $B_j \subset \mathcal{W}$ denote the subset of tasks that are (partially or totally) processed during slice S_j; note that $\Delta = \max_{j\in[1,K]} |B_j|$. Finally, for a task i in B_j, let $a_{i,j}$ be the fraction of the task that is processed during S_j (and let $a_{i,j} = 0$ if $i \notin B_j$).

As an example, we consider a workflow $\mathcal{W}$ consisting of $n = 4$ independent tasks, with $T^1_{base} = 6$, $T^2_{base} = 4$, $T^3_{base} = 8$ and $T^4_{base} = 9$. We have $m = 4$, $p_1 = p_2 = 2$ and $p_3 = p_4 = 1$. The optimal failure-free schedule $\mathbb{S}$ is shown in Figure 1, and has length 10. Note that task i is represented by its profile $\mathcal{T}_i$. There are five time-steps where an event occurs, thus $K = 4$ and $\{t_j\}_{0\le j\le K} = \{0, 6, 8, 9, 10\}$. Therefore, $\mathbb{S}$ is decomposed into four slices, S_1 running in $[0, 6]$, S_2 in $[6, 8]$, S_3 in $[8, 9]$, and S_4 in $[9, 10]$. The $(a_{i,j})_{i\in[1,n],j\in[1,K]}$ are represented in brackets. Finally, $B_1 = \{1, 3, 4\}$, $B_2 = \{2, 3, 4\}$, $B_3 = \{2, 4\}$, $B_4 = \{4\}$, and $\Delta = 3$. We use the decomposition into slices to define a virtual schedule $\mathbb{S}^{virt}$, which consists of scaling the slices S_j to account for failures in $\mathbb{S}$'. For each slice S_j, the scaling is the largest ratio $\frac{T_i}{T^i_{base}}$ over all tasks $i \in B_j$. Hence, $\mathbb{S}^{virt}$ is composed of K slices S^{virt}_j whose length is $T(S^{virt}_j) = \left(\max_{i\in B_j} \frac{T_i}{T^i_{base}}\right) T(S_j)$. Within each slice S^{virt}_j, for each task $i \in B_j$, we execute the same fraction $a_{i,j}$ of task i as in the original schedule $\mathbb{S}$, for a duration $a_{i,j}T_i$, so that some tasks in B_j may not execute during the whole length of S^{virt}_j, contrarily to during the initial schedule $\mathbb{S}$.

We can then bound the total execution time of $\mathbb{S}^{virt}$ by using the result on independent tasks on each slice, each with a maximum degree of parallelism of Δ. Finally, to obtain Equation (10), there remains to show that $\mathbb{E}(T(\mathbb{S}')) \le \mathbb{E}(T(\mathbb{S}^{virt}))$. In fact, we show that under any failure scenario, $T(\mathbb{S}') \le T(\mathbb{S}^{virt})$, and the result follows. We relabel the tasks by non-decreasing starting time in $\mathbb{S}$ and prove by induction that no task starts nor ends later in $\mathbb{S}$' than in $\mathbb{S}^{virt}$. The key element is that the ordering of starting times from $\mathbb{S}$ is preserved in both $\mathbb{S}^{virt}$ and $\mathbb{S}'$, see Appendix E for details.

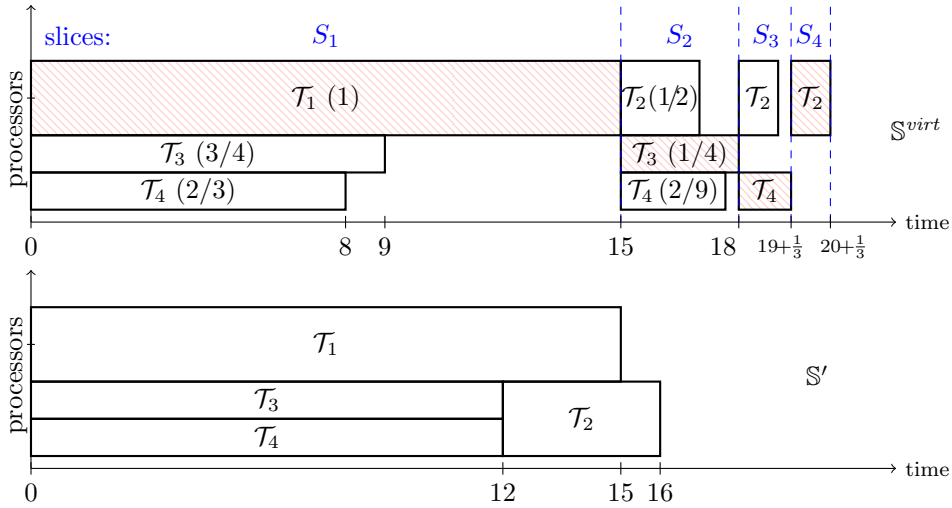

Figure 2: Schedules $\mathbb{S}^{virt}$ (top) and $\mathbb{S}'$ (bot.) for the example.

Going back to the example, assume that $T_1 = 15$, $T_2 = 4$, and $T_3 = T_4 = 12$ in $\mathbb{S}$'. Then, we obtain the task with the largest ratio $\frac{T_i}{T_{base}^i}$ for each slice: task 1 for S_1, task 3 for S_2, task 4 for S_3, and task 2 for S_4. The schedule $\mathbb{S}^{virt}$ is shown at the top of Figure 2 and has length $T(\mathbb{S}^{virt}) = 20 + 1/3$ (and the tasks with largest ratio in each slice are hatched). Finally, the schedule $\mathbb{S}'$ is shown at the bottom of Figure 2, and $T(\mathbb{S}') = 16$. □

We point out that Theorem 3 applies to workflows with arbitrary dependences, and with rigid or moldable tasks. The bound given for $r^{ME}(\Delta, \mathcal{W})$ is relative to the execution time of the failure-free schedule. If this failure-free schedule is itself a ρ-approximation of the optimal solution, then we have derived a $r^{ME}(\Delta, \mathcal{W}) \times \rho$ approximation of the optimal solution.

4 The CheckMore Strategies

The previous section has shown that, in the presence of failures, the ratio of the actual execution time of a workflow over its failure-free execution time, critically depends upon the maximum degree of parallelism Δ achieved by the initial schedule.

In this section, we introduce CheckMore strategies, which checkpoint workflow tasks more often than MinExp, with the objective to decrease the ratio above. The number of checkpoints for each task becomes a function of the degree of parallelism in the execution. We define SafeCheck(δ), the number of checkpoints for a task, given a parameter δ (typically the degree of parallelism):

Definition 2. SafeCheck(δ) *partitions a parallel task of length T_{base}, with p processors and checkpoint time C, into $N_{SC}(\delta) = \left\lceil \frac{(\ln(\delta)+1)T_{base}}{W_{YD}} \right\rceil$ equal-length segments, each followed by a checkpoint, where $W_{YD} = \sqrt{\frac{2C}{p\lambda}} = \sqrt{\frac{2\mu_{ind}C}{p}}$. Each segment is of length $W_{SC}(\delta) = \frac{T_{base}}{N_{SC}(\delta)}$.*

Note that MinExp corresponds to applying SafeCheck (1) to all tasks, since $N_{SC}(1) = N_{ME}$. The key building block of the analysis of MinExp is

Theorem 1 for identical independent tasks. The good news is that Theorem 1 holds for any checkpointing strategy, not just for the Young/Daly approach, and can easily be extended if each task is checkpointed following SafeCheck(δ):

Theorem 4. *Consider n identical tasks of same type $\mathcal{T}$=(T_{base},p,C,R) to be executed concurrently on $n \times p \leq m$ processors with individual failure rate $\lambda = \frac{1}{\mu_{ind}}$. The downtime is D. For the* SafeCheck(δ) *strategy, N_{SC} (δ) is the number of checkpoints, and W_{SC} (δ) is the length of each segment, as given by Definition 1. Let $P_{suc}(\tilde{R}) = e^{-p\lambda(W_{SC}+C+\tilde{R})}$ be the probability of success of a segment with re-execution cost $\tilde{R}$ ($\tilde{R} = 0$ if no re-execution, or $\tilde{R} = R$ otherwise), and $Q^* = \frac{1}{1-P_{suc}(R)}$. Let $r_{id}^{SC}(\delta, n, \mathcal{T}) = \frac{\mathbb{E}(T_{tot})}{T_{base}}$, where $\mathbb{E}(T_{tot})$ is the expectation of the total time T_{tot} of the* SafeCheck(δ) *strategy. Then:*

$$\begin{aligned} r_{id}^{SC}(\delta, n, \mathcal{T}) \leq & \Big(\tfrac{\log_{Q^*}(n)}{N_{SC}(\delta)} + \log_{Q^*}(\log_{Q^*}(n)) + 1 + \tfrac{\ln(Q^*)}{12 N_{SC}(\delta)} \\ & + \tfrac{1}{\ln(Q^*) N_{SC}(\delta)} \Big) \times \Big(1 + \tfrac{C+R+D}{W_{SC}(\delta)} \Big) + \tfrac{C}{W_{SC}(\delta)} + 1 + o(1). \end{aligned} \tag{11}$$

Note that if n is small, the ratio holds by replacing all negative or undefined terms by 0.

To prove this theorem, we reuse the proof of Theorem 1: we just need to replace N_{ME} by $N_{SC}(\delta)$, and W_{ME} by $W_{SC}(\delta)$.

The idea behind SafeCheck(δ) is the following: when processing δ jobs in parallel, the expected maximum number of failures given by Equation (6) eventually grows proportionally to its first term, $\log_{Q^*}(\delta)$, which is $\Theta(\ln(\delta))$. To accommodate this growth, we reduce the segment length by a factor $\ln(\delta)$, so that the total failure-induced overhead does not increase much. This is exactly what SafeCheck(δ) does, when δ tasks are processed in parallel. Similarly, the first term $\frac{\log_{Q^*}(n)}{N_{ME}(n)}$ of the ratio in Equation (3) was dominant for MinExp, while it becomes almost constant in Equation (11). To that extent, CheckMore generalizes this idea to general workflows using SafeCheck(δ) as a subroutine. We provide two variants of CheckMore:

Definition 3. *Consider a failure-free schedule $\mathbb{S}$ for a workflow $\mathcal{W}$ of n tasks:*
- *The* CheckMore *algorithm applies* SafeCheck(Δ_i) *to each task i, where Δ_i is the largest number of tasks that are executed concurrently at some point during the processing of task i.*
- *The* BasicCheckMore *algorithm applies* SafeCheck($\min(n, m)$) *to all tasks, where m is the number of processors.*

The main reason for introducing BasicCheckMore is that we do not need to know the maximum degree Δ of parallelism in $\mathbb{S}$ to execute BasicCheckMore (because we always have $\Delta \leq \min(n, m)$). In fact, we do not even need to know the failure-free schedule for BasicCheckMore (contrarily to CheckMore), we just need an ordered list of tasks and to greedily start them in this order.

Theorem 5. *Let $\mathbb{S}$ be a failure-free schedule of a workflow $\mathcal{W}$ of n tasks. The i-th task has profile $\mathcal{T}_i = (T_{base}^i, p_i, C_i, R_i)$. Let Δ_i be the maximum number of tasks processed concurrently to task i by $\mathbb{S}$ at any instant, and let $\Delta = \max_{1 \leq i \leq n} \Delta_i$. Then* CheckMore *is a $r^{CM}((\Delta_i)_{i \leq n}, \mathcal{W})$-approximation of the failure-free execution time in expectation:*

$$r^{CM}((\Delta_i)_{i\leq n}, \mathcal{W}) \leq 2 \max_{1\leq i\leq n} r_{id}^{SC}(\Delta_i, \Delta_i, \mathcal{T}_i). \tag{12}$$

And BasicCheckMore *is a* $r^{BCM}(\min(n,m), \mathcal{W})$*-approximation of the failure-free execution time in expectation:*

$$r^{BCM}(\min(n,m), \mathcal{W}) \leq 2 \max_{1\leq i\leq n} r_{id}^{SC}(\min(n,m), \Delta, \mathcal{T}_i). \tag{13}$$

See Appendix F for the proof. Note that $\forall i, \Delta_i \leq \Delta$ and $\Delta_i \leq min(n,m)$, so it is extremely likely that the bound obtained for r^{CM} is smaller than the one obtained for r^{BCM}. To illustrate the difference between the bounds of CheckMore and MinExp, we show in Appendix H that for a shelf of n identical uni-processor tasks running in parallel, r_{id}^{CM} is an order of magnitude lower than r_{id}^{ME} under reasonable assumptions and when n is large enough.
We conclude this section by returning to the example of Section 3.1.2, and showing that CheckMore (equivalent to BasicCheckMore in this case) can be arbitrarily better than MinExp. The proof is given in Appendix G.2.

Proposition 2. *Consider a set* $\mathcal{T}$ *of* $n(K)$ *identical uni-processor tasks with type* $\mathcal{T} = (2K-1, 1, 10)$*,* $D = 0$ *and* $\lambda(K) = \frac{\ln(1+\frac{1}{2K})}{2K}$*. We assume that all tasks execute in parallel, i.e.,* $m \geq n(K)$*. When letting* $n(K) = \lfloor e^{\sqrt{2/\lambda(K)-1}} \rfloor$ *(hence* $\ln(n(K)) = \Theta(K)$*), and* K *tending to infinity, we have* $r^{ME}(n(K), \mathcal{T}) = \Theta\left(\frac{K}{\ln(K)}\right)$ *and* $r^{BCM}(n(K), \mathcal{T}) = \Theta(1)$*.*

5 Experimental Evaluation

In this section, we evaluate the performance of the different checkpointing strategies through simulations. We describe the simulation setup in Section 5.1, present the main performance comparison results in Section 5.2, and assess the impact of different parameters on the performance in Section 5.3. Our in-house simulator is written in C++ and is publicly available for reproducibility purpose.

5.1 Simulation Setup

We evaluate and compare the performance of the three checkpointing strategies MinExp, CheckMore and BasicCheckMore. All strategies are coupled with a failure-free schedule computed by a list scheduling algorithm (see below). The workflows used for evaluation are generated from WorkflowHub [18] (formerly Pegasus [35]), which offers realistic synthetic workflow traces with a variety of characteristics. They have been shown to accurately resemble the ones from real-world workflow executions [2, 18]. Specifically, we generate the following nine types of workflows offered by WorkflowHub that model applications in various scientific domains: Blast, Bwa, Epigenomics, Genome, SoyKB and Sras are bioinformatics workflows; Cycles is an agroecosystem workflow; Montage is an astronomy workflow; and Seismology is a seismology workflow; see Appendix I.1 for more details.

Each trace defines the general structure of the workflow, whose number of tasks and total execution time can be specified by the user[2]. All tasks generated

[2]Note that the workflow generator may offer a different number of tasks so as to guarantee the structure of the workflow. The difference, however, is usually small.

in WorkflowHub are uni-processor tasks.

In the experiments, we evaluate the checkpointing strategies under the following parameter settings:

- Number of processors: $m = 2^{14} = 16384$;
- Checkpoint/recovery/down time: $C = R = 1$ min, $D = 0$;
- MTBF of individual processor: $\mu_{ind} = 10$ years;
- Number of tasks of each workflow: $n \approx 50000$.

Furthermore, the total failure-free execution times of all workflows are generated such that they complete in 3-5 days. This is typical of the large scientific workflows that often take days to complete as observed in some production log traces [1, 33] (Appendix I.2 also presents similar experimental results, which utilize small workflow traces that take less than a day to complete). Section 5.2 will present the comparison results of different checkpointing strategies under the above parameter settings. In Section 5.3, we will further evaluate the impacts of different parameters (i.e., m, C, μ_{ind} and n) on the performance.

The evaluation methodology is as follows: for each set of parameters and each type of workflow trace, we generate 30 different workflow instances and compute their failure-free schedules. We use the list scheduling algorithm that orders the tasks using the Longest Processing Time (LPT) first policy: if several tasks are ready and there is at least one processor available, the longest ready task is assigned to the available processor to execute. Since all tasks are uni-processor tasks, LPT is known to be a 2-approximation algorithm [21]; also, LPT is known to be a good heuristic for ordering the tasks [30]. This order of execution will be enforced by all the checkpointing strategies. For each workflow instance, we further generate 50 different failure scenarios. Here, a failure scenario consists of injecting random failures to the tasks by following the Exponential distribution as described in Section 2.1. The same failure scenario will then be applied to each checkpointing strategy to evaluate its execution time for the workflow. We finally compute the *ratio* of a checkpointing strategy under a particular failure scenario as $\frac{T}{T_{base}}$, where T_{base} is the failure-free execution time of the workflow, and T is the execution time under the failure scenario. The statistics of these $30 \times 50 = 1500$ experiments are then compared using boxplots (that show the mean, median, and various percentiles of the ratio) for each checkpointing strategy. The boxes bound the first to the third quantiles (i.e., 25th and 75th percentiles), the whiskers show the 10th percentile to the 90th percentile, the black lines show the median, and the stars show the mean.

5.2 Performance Comparison Results

Figure 3 shows the boxplots of the three checkpointing strategies in terms of their ratios for the nine different workflows.

First, we observe that CheckMore and BasicCheckMore have very similar performance, which in most cases are indistinguishable. This shows that BasicCheckMore offers a simple yet effective solution without the need to inspect the failure-free schedule, thus making it an attractive checkpointing strategy in practice. Also, both versions of CheckMore perform significantly better and with less variation than MinExp, except for the few workflows where the ratios of all strategies are very close to 1 (e.g., Bwa, SoyKB, Sras). Overall, the 90th percentile ratio of CheckMore never exceeds 1.08, whereas that of MinExp is much higher for most workflows and reaches almost 1.5 for Montage. Similarly, the average ratio of CheckMore never exceeds 1.03, while

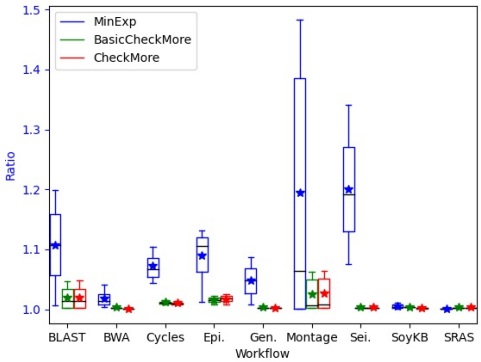

Figure 3: Performance (ratio) comparison of the three checkpointing strategies for the nine different workflows.

that of MinExp is again significantly higher and reaches more than 1.2 for Seismology and Montage.

We now examine a few workflows more closely to better understand the performance. For Sras, MinExp is slightly better than CheckMore, but the ratios of all strategies are near optimal (i.e., ¡1.003). In this workflow, very few tasks are extremely long while many others are very short, and there are very few dependencies among them. Thus, failures hardly ever hit the long tasks due to their few number, while failures that hit short tasks have little impact on the overall execution time. This is why the ratio is so small for all strategies. It also explains why MinExp outperforms CheckMore: although the maximum degree of parallelism is important, only a few tasks matter and they should be checkpointed à la Young/Daly to minimize their own expected execution time, and thereby that of the entire workflow. SoyKB and Bwa also have very low ratios. In the case of SoyKB, there is just not enough parallelism during the majority of the execution time, so all strategies are making reasonable checkpointing decisions, with CheckMore performing slightly better for taking into account this small parallelism. Bwa, on the other hand, has two source tasks that must be executed first and two sink tasks that must be executed last. Among them, one source task and one sink task are extremely long, so failures in other tasks have little impact (as in the case of Sras). Yet the small tasks are not totally negligible here, because the dominant sink task must be processed after all of them, so it is still worth to optimize these tasks with CheckMore, which explains why it is slightly better than MinExp.

For all the other workflows, CheckMore performs better than MinExp by a significant margin. This is due to CheckMore's more effective checkpointing strategies given the specific structure of these workflows. For instance, Montage has some key tasks that are dominant, so a failure that strikes most of the other tasks does not impact the overall execution time. This is similar to the case of Sras and explains why, for all strategies, the first quantile of the ratio is very low (i.e., around 1). However, when a failure does strike one of the key tasks, the execution time will be heavily impacted. The difference with Sras is that Montage contains more key tasks that can run in parallel, so it is much more likely that one of them will fail, which is why checkpointing them with CheckMore is better. Next, Blast and Seismology have some source and sink tasks (as Bwa), which, however, are not so dominant in length, making the difference between CheckMore and MinExp higher even from the first quantile. Other workflows also have similar structures, which eventually

contribute to the better performance of CheckMore over MinExp.

5.3 Impact of Different Parameters

We now study the impact of different parameters on the performance of the checkpointing strategies. In each set of experiments below, we vary a single parameter while keeping the others fixed at their base values. All figures and comments are available in Appendix I.3; due to lack of space, we focus here on Blast, Seismology, Genome and Sras. The results are shown in Figure 4, where the scale of the y-axis is kept the same for ease of comparison. For some figures with really small values, zoomed-in plots are also provided on the original figure for better viewing.

Impact of Number of Processors (m). We first assess the impact of the number of processors, which is varied between 4096 and 50000. In general, increasing the number of processors increases the ratio. This corroborates our theoretical analysis, because for most types of workflows, having more processors means having a larger Δ and thus a larger potential ratio, until m surpasses the width of the dependence graph. However, CheckMore and BasicCheckMore appear less impacted than MinExp.

For Blast, Seismology, Genome, the ratio is very close to 1 when m is small for all checkpointing strategies. In fact, for these workflows, most tasks are quite independent. Thus, when n is large compared to m, even if a failure strikes a task, it will have little impact on the starting times of the other tasks. This is because we only maintain the order of execution but do not stick to the same mapping as in the failure-free schedule. For this reason, it is better to minimize each task's own execution time by using MinExp (i.e., CheckMore checkpoints a bit too much). However, when m becomes large, the performance of MinExp degrades significantly, with an average ratio even reaching 1.7 for Blast at $m = 50000$, whereas it stays below 1.1 for CheckMore.

Finally, for Sras, as the number of dominating tasks that could be run in parallel is way less than 4096, the ratio of MinExp does not vary much with m, while that of CheckMore increases with m as it tends to checkpoint more with an increasing number of processors. Also, in more than 90% of the cases, the failures have strictly no impact on the overall execution time, since they do not hit the dominating tasks. This is why the average ratio is above the 90th percentile for all checkpointing strategies.

Impact of Checkpoint Time (C). We now evaluate the impact of the checkpoint time by varying it between 15 and 240 seconds. The ratio generally increases with C; this is consistent with Equation (3). when $R = C$ and $D = 0$, the approximation ratio satisfies $r \leq \left(\frac{X}{N_c} + Y\right)\left(\frac{2C}{W} + 1\right) + \frac{C}{W} + Z$, where X, Y and Z barely depend on C, N_c decreases with C, and $\frac{C}{W} \approx \sqrt{\frac{C}{2\mu}}$ increases with C. Intuitively, the checkpoint time impacts the ratio in two ways. First, as C increases, we pay more for each checkpoint, which could lead to an increased ratio. Second, as we use $W_{YD} = \sqrt{\frac{2C}{p\lambda}}$ to determine the checkpointing period and hence the number of checkpoints, a task will become less safe when C increases, because it will be checkpointed less, and this could also increase the ratio.

Looking at Genome under MinExp, we can see a clear increase in the ratio when C increases from 15 to 21. This is because the typical number

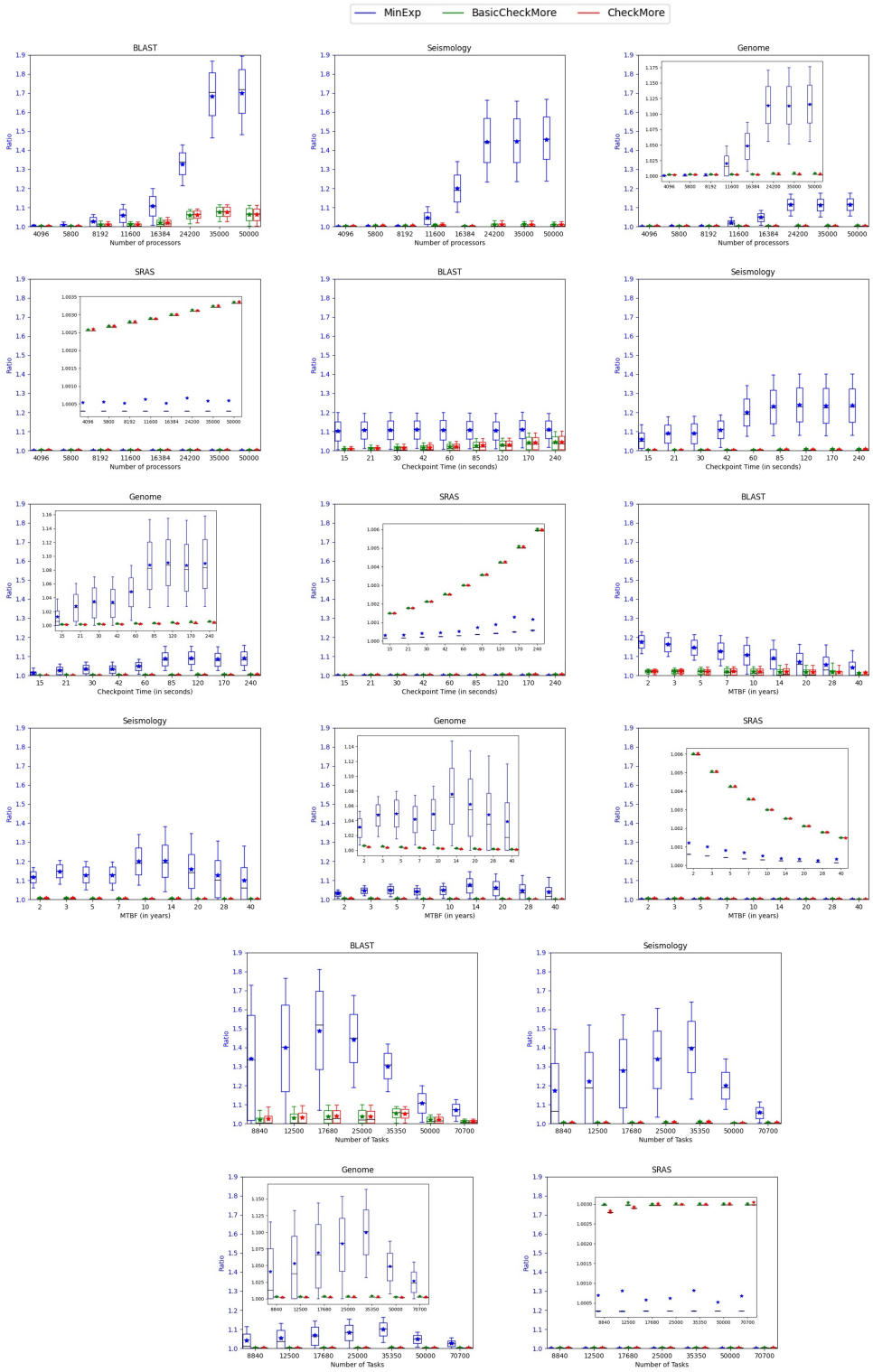

Figure 4: Impact of parameters on the performance of the checkpointing strategies for different workflows.

of checkpoints for the critical tasks (that affect the overall execution time the most) drops from 3 to 2, thus the time wasted due to a failure increases from 33% to 50%. As C increases from 60 to 85, the typical number of checkpoints of these tasks further drops from 2 to 1, making the waste per failure increase to 100%, and so the ratio also greatly increases. For values of C between 21 and 42, even if the number of checkpoints does not change, the ratio increases smoothly due to the increase in checkpoint time. The ratio of CheckMore, on the other hand, only increases slightly with the checkpoint time, which is, however, not visible in the figure due to the small values. Seismology also clearly illustrates these phenomena. For Sras, since most failures do not affect the overall execution time, the ratio of all strategies is only impacted by the checkpoint time. For Blast under MinExp, because most tasks are short and we have a single checkpoint to start with, the increase in checkpoint time is negligible compared to the waste induced by failures.

Impact of Individual MTBF (μ_{ind}). We evaluate the impact of individual processor's MTBF by varying it between 30 months and 40 years. Intuitively, when μ_{ind} increases (or equivalently, the failure rate λ decreases), we would have fewer failures and expect the ratio to decrease. This is generally true for CheckMore but not always for MinExp. To understand why, we refer again to the simplified approximation ratio $r \leq \left(\frac{X}{N_c} + Y\right)\left(\frac{2C}{W} + 1\right) + \frac{C}{W} + Z$, where X, Y and Z are barely affected by μ_{ind}. Here, when the number of failures decreases, $W_{YD} = \sqrt{\frac{2C}{p\lambda}}$ increases, so the number of checkpoints decreases and the time wasted for each failure increases. This could potentially lead to an increase in the ratio. To illustrate this compound effect, we again look at Genome under MinExp. When μ_{ind} goes from 2.5 to 3.5 years, the typical number of checkpoints for the critical tasks (that affect the overall execution time the most) drops from 3 to 2, which increases the waste per failure by around 50%. This together with the fact that MinExp does not take into account the parallelism results in an increase in the ratio. When μ_{ind} goes from 3.5 to 7 years, the ratio decreases simply because we have fewer failures. As μ_{ind} continues to increase to 14 years, the number of checkpoints for the critical tasks further drops from 2 to 1. This increases the waste per failure to 100%, which again leads to an increase in the ratio. From this point on, the ratio will just decrease with μ_{ind}, again due to fewer failures. The same can be observed for Seismology. In Blast and Sras, the ratio simply decreases with μ_{ind}. For Blast, even when μ_{ind} is small, we only checkpoint once, so the ratio decreases due to fewer failures. For Sras, failures usually do not impact the overall execution time, so the decrease in ratio is mainly due to the decrease in the number of checkpoints.

Finally, it is worth noting that the ratio variance increases as μ_{ind} increases. This is because when there are only a few failures and the length of the segments is large, the failure location (inside the segments) will matter significantly, especially for MinExp.

Impact of Number of Tasks (n). Finally, we study the impact of the number of tasks in the workflow, which is varied between 8800 and 70000. Again, the ratio is impacted by the number of tasks in two different ways. First, when n increases, the width of the graph increases and so does Δ, and this would increase the ratio according to our analysis. Second, when n increases and m is fixed, the average number of tasks executed by each processor increases. This means that, if a failure occurs early in the execution, it is less likely to have

a significant impact on the ratio, since multiple other tasks will be processed afterwards to balance the load, especially if the tasks are relatively independent.

These two phenomena are clearly observed in BLAST under MINEXP. This workflow mainly consists of a large batch of independent tasks. When n increases to 17680, which is approximately the number of processors ($m = 16384$), the ratio increases because Δ increases. After that, the ratio starts to decrease because $n > m$. In this case, when a failure strikes an early task, the subsequent tasks could be assigned to other processors to reduce the impact of the failure. Ultimately, if $n \gg m$, MINEXP would become more efficient. Indeed, since the tasks are almost independent and uni-processor tasks, list scheduling is able to dynamically balance the loads of different processors. Thus, minimizing the expected execution time of each individual task using MINEXP would be a good strategy for the overall execution time of the workflow.

For SEISMOLOGY and GENOME, we observe the same up-and-down effect as a result of these two phenomena, but nor for SRAS, which is not impacted by the number of tasks. For this workflow, only a few key dominating tasks matter and their width remains well below the number of processors. Since these tasks form a small proportion of the total number of tasks, varying n does not significantly alter their chance of being hit by a failure, so the ratio remains close to 1.

Summary. Our experimental evaluation demonstrates that MINEXP is not resilient enough for checkpointing workflows, although it provides an optimal strategy for each individual task. On the other hand, CHECKMORE proves to be a very useful strategy, except for SRAS whose ratios are extremely low. When varying the key parameters, the simulation results nicely corroborate our theoretical analysis. Furthermore, the easy-to-implement BASICCHECKMORE strategy always leads to ratios that are close to those of CHECKMORE, regardless of the parameters.

6 Related work

Scheduling Workflows. Scheduling a computational workflow consisting of a set of tasks in a dependency graph to minimize the overall execution time (or makespan) is a well-known NP-complete problem [19]. Only a few special cases are known to be solvable in polynomial time, such as when all tasks are of the same length and the dependency graph is a tree [27] or when there are only two processors [10]. For the general case, some branch-and-bound algorithms [34, 26] have been proposed to compute the optimal solution, but the problem remains tractable only for small instances. In the seminal work, Graham [21] showed that the list scheduling strategy, which organizes all tasks in a list and schedules the first ready task at the earliest time possible, achieves an execution time that is no worse than $2 - \frac{1}{m}$ times the optimum, where m denotes the total number of processors, i.e., the algorithm is a $(2 - \frac{1}{m})$-approximation. This performance guarantee holds regardless of the order of the tasks in the list. Some heuristics further explore the impact of different task orderings on the overall execution time, with typical examples including task execution times, bottom-levels and critical paths (see [30] for a comprehensive survey of the various heuristic strategies).

While the results above are for workflows with uni-processor tasks (or tasks that share the same degree of parallelism), scheduling workflows with parallel

tasks has also been considered. Li [32] proved that, for precedence constrained tasks with fixed parallelism of different degrees (i.e., rigid tasks), the worst-case approximation ratio for list scheduling under a variety of task ordering rules is m. However, if all tasks require no more than qm processors, where $0 < q < 1$, the approximation ratio becomes $\frac{(2-q)m}{(1-q)m+1}$. Demirci et al. [12] proved an $O(\log n)$-approximation algorithm for this problem using divide-and-conquer, where n is the number of tasks in the workflow. Furthermore, for parallel tasks that can be executed using a variable number of processors at launch time (i.e., moldable tasks), list scheduling is shown to be an $O(1)$-approximation when coupled with a good processor allocation strategy under reasonable assumptions on the tasks' speedup profiles [16, 31, 29].

In this paper, we augment the workflow scheduling problem with the checkpointing problem for its constituent tasks. We analyze the approximation ratios of some checkpointing strategies while relying on the ratios of existing scheduling algorithms to provide an overall performance guarantee for the combined problem.

Checkpointing Workflows. Checkpoint-restart is one of the most widely used strategy to deal with fail-stop errors. Several variants of this policy have been studied; see [25] for an overview. The natural strategy is to checkpoint periodically, and 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 an preemptible application, given the checkpointing cost C and platform MTBF μ, the classical formula due to Young [37] and Daly [11] states that the optimal checkpointing period is $W_{YD} = \sqrt{2\mu C}$.

Going beyond preemptible applications, some works have studied task-based applications, using a model where checkpointing is only possible right after the completion of a task. The problem is then to determine which tasks should be checkpointed. This problem has been solved for linear workflows (where the task graph is a simple linear chain) by Toueg and Babaoglu [36], using a dynamic programming algorithm. This algorithm was later extended in [4] to cope with both fail-stop and silent errors simultaneously. Another special case is that of a workflow whose dependence graph is arbitrary but whose tasks are parallel tasks that each executes on the whole platform. In other words, the tasks have to be serialized. The problem of ordering the tasks and placing checkpoints is proven NP-complete for simple join graphs in [3], which also introduces several heuristics. Finally, for general workflows, deciding which tasks to checkpoint has been shown #P-complete [22], but several heuristics are proposed in [23].

In this paper, we depart from the above model and assume that each workflow task is a preemptible task that can be checkpointed at any instant. This assumption is quite natural for many applications, such as those involving dense linear algebra kernels or tensor operations. It is even mandatory for coarse-grain workflows: unless the failure rate can be decreased below the current standard, the successful completion of any large task, say executing a few hours with 1K nodes, is very unlikely.

7 Conclusion

In this paper, we have investigated checkpointing strategies for parallel workflows, whose tasks are either sequential or parallel, and in the latter case either rigid or moldable. Because HPC tasks may have a large granularity, we assume that they can be checkpointed at any instant. Starting from a failure-free schedule, the natural MinExp strategy consists in checkpointing each task so as to minimize its expected execution time; hence MinExp builds upon the classical results of Young/Daly, and uses the optimal checkpointing period for each task. We derive a performance bound for MinExp, and exhibit an example where this bound is tight.

Intuitively, MinExp may perform badly in some cases, because there is an important risk that the delay of one single task will slow down the whole workflow. To mitigate this risk, we introduce CheckMore strategies that may checkpoint some tasks more often than other tasks, and more often than in the MinExp strategy. This comes in two flavors. CheckMore decides, for each task, how many checkpoints to take, building upon its degree of parallelism in the corresponding failure-free schedule. BasicCheckMore is just using, as degree of parallelism for each task, the maximum possible value $\min(n, m)$ (hence it is equivalent to CheckMore for independent tasks all running in parallel). The theoretical bounds for BasicCheckMore are not as good as those of CheckMore, but its performance in practice is very close, thus BasicCheckMore proves to be very efficient despite its simplicity.

An extensive set of simulations is conducted at large scale, using realistic synthetic workflows from WorkflowHub with between 8k and 70k tasks, and running on a platform with up to 50k processors. The results are impressive, with ratios very close to 1 on all workflows for both CheckMore strategies, while MinExp has much higher ratios, for instance 1.7 on average for Blast and 1.46 for Seismology. Hence, the simulations confirm that it is indeed necessary in practice to checkpoint workflow tasks more often than the classical Young/Daly strategy.

As future work, we plan to extend the simulation campaign to parallel tasks (rigid or moldable), as soon as workflow benchmarks with parallel tasks are available to the community. We will also investigate the impact of the failure-free list schedule on the final performance in a failure-prone execution, both theoretically and experimentally. Indeed, list schedules that control the degree of parallelism in the execution may provide a good trade-off between efficiency (in a failure-free framework) and robustness (when many failures strike during execution).

Acknowledgements. We thank the reviewers for all their comments and suggestions.

References

[1] Argonne Leadership Computing Facility (ALCF). Mira log traces. https://reports.alcf.anl.gov/data/mira.html.

[2] M. Atkinson, S. Gesing, J. Montagnat, and I. Taylor. Scientific workflows: Past, present and future. *Future Generation Computer Systems*, 75:216–227, 2017.

[3] G. Aupy, A. Benoit, H. Casanova, and Y. Robert. Scheduling computational workflows on failure-prone platforms. *Int. J. of Networking and Computing*, 6(1):2–26, 2016.

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

[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. Research Report 7520, INRIA, France, Jan. 2011. Available at `http://graal.ens-lyon.fr/~fvivien/`.

[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] F. Cappello, K. Mohror, et al. VeloC: very low overhead checkpointing system. `https://veloc.readthedocs.io/en/latest/`, march 2019.

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

[10] E. G. Coffman and R. L. Graham. Optimal scheduling for two-processor systems. *Acta Inf.*, 1(3):200–213, 1972.

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

[12] G. Demirci, H. Hoffmann, and D. H. K. Kim. Approximation algorithms for scheduling with resource and precedence constraints. In *STACS*, pages 25:1–25:14, 2018.

[13] B. Eisenberg. On the expectation of the maximum of iid geometric random variables. *Statistics & Probability Letters*, 78(2):135–143, 2008.

[14] Fault-Tolerance Research Hub. User level failure mitigation, 2021. `https://fault-tolerance.org`.

[15] D. G. Feitelson and L. Rudolph. Toward convergence in job schedulers for parallel supercomputers. In *Job Scheduling Strategies for Parallel Processing*, pages 1–26. Springer, 1996.

[16] A. Feldmann, M.-Y. Kao, J. Sgall, and S.-H. Teng. Optimal on-line scheduling of parallel jobs with dependencies. *Journal of Combinatorial Optimization*, 1(4):393–411, 1998.

[17] 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.

[18] R. Ferreira da Silva, L. Pottier, T. a. Coleman, E. Deelman, and H. Casanova. Workflowhub: Community framework for enabling scientific workflow research and development. In *2020 IEEE/ACM Workflows in Support of Large-Scale Science (WORKS)*, pages 49–56, 2020.

[19] M. R. Garey and D. S. Johnson. *Computers and Intractability, a Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.

[20] P. J. Grabner and H. Prodinger. Maximum statistics of n random variables distributed by the negative binomial distribution. *Combinatorics, Probability and Computing*, 6(2):179–183, 1997.

[21] R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17(2):416–429, 1969.

[22] L. Han, L.-C. Canon, H. Casanova, Y. Robert, and F. Vivien. Checkpointing workflows for fail-stop errors. *IEEE Trans. Computers*, 67(8):1105–1120, 2018.

[23] L. Han, V. L. Fèvre, L.-C. Canon, Y. Robert, and F. Vivien. A generic approach to scheduling and checkpointing workflows. In *ICPP'2018, the 47th Int. Conf. on Parallel Processing*. IEEE Computer Society Press, 2018.

[24] G. H. Hardy and E. M. Wright. *An Introduction to the Theory of Numbers*. Oxford, fourth edition, 1975.

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

[26] U. Hönig and W. Schiffmann. A parallel branch-and-bound algorithm for computing optimal task graph schedules. In *Second International Workshop on Grid and Cooperative Computing GCC*, pages 18–25, 2003.

[27] T. C. Hu. Parallel sequencing and assembly line problems. *Oper. Res.*, 9(6):841–848, 1961.

[28] IBM Spectrum LSF Job Scheduler. Fault tolerance and automatic management host failover, 2021. `https://www.ibm.com/docs/en/spectrum-lsf/10.1.0?topic=cluster-fault-tolerance`.

[29] K. Jansen and H. Zhang. An approximation algorithm for scheduling malleable tasks under general precedence constraints. *ACM Trans. Algorithms*, 2(3):416–434, 2006.

[30] Y.-K. Kwok and I. Ahmad. Static scheduling algorithms for allocating directed task graphs to multiprocessors. *ACM Comput. Surv.*, 31(4):406–471, 1999.

[31] R. Lepère, D. Trystram, and G. J. Woeginger. Approximation algorithms for scheduling malleable tasks under precedence constraints. In *ESA*, pages 146–157, 2001.

[32] K. Li. Analysis of the list scheduling algorithm for precedence constrained parallel tasks. *Journal of Combinatorial Optimization*, 3(1):73–88, 1999.

[33] National Energy Research Scientific Computing Center (NERSC). Cori log traces. https://docs.nersc.gov/systems/cori/.

[34] A. Z. S. Shahul and O. Sinnen. Scheduling task graphs optimally with A*. *The Journal of Supercomputing*, 51:310–332, 2010.

[35] P. Team. Pegasus workflow generator. `https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator`, 2014.

[36] S. Toueg and O. Babaoğlu. On the optimum checkpoint selection problem. *SIAM J. Comput.*, 13(3), 1984.

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

A Extension without final checkpoint nor initial recovery

Consider a task of type $\mathcal{T} = (T_{base}, p, C, R)$, which is partitioned into segments followed by a checkpoint. This section deals with the case where no checkpoint is enforced at the end of the last segment. By symmetry, we also deal with the case where no recovery is paid when re-executing the first segment after a failure.

The task is partitioned into N segments of length W_i, with checkpoint cost C_i and recovery cost R_i. Let $C_{tot} = \sum_{i=1}^{N} C_i$ and $R_{tot} = \sum_{i=1}^{N} C_i$. In the model of Section 2, we had $C_i = C$, $R_i = R$ for $1 \leq i \leq N$, $C_{tot} = NC$, and $R_{tot} = NR$. If no checkpoint is taken after the last segment, $C_N = 0$ and $C_{tot} = (N-1)C$. If no recovery is paid when re-executing the first segment, $R_1 = 0$ and $R_{tot} = (N-1)R$.

What is the optimal strategy to minimize the expected execution time $\mathbb{E}$ of the task? From Lemma 1, we have:

$$\mathbb{E} = \sum_{i=1}^{N} \mathbb{E}(W_i, C_i, R_i) = \left(\frac{1}{p\lambda} + D\right) \sum_{i=1}^{N} e^{p\lambda R_i} \left(e^{p\lambda(W_i + C_i)} - 1\right), \tag{14}$$

where $\sum_{i=1}^{N} W_i = T_{base}$. Given N, $\mathbb{E}$ is minimized when the sum $\sum_{i=1}^{N} e^{p\lambda(W_i+C_i+R_i)}$ is minimized. By convexity of the Exponential function, or by using Lagrange multipliers, we see that $\mathbb{E}$ is minimized when the $W_i + C_i + R_i$ take a constant value W_{seg}. This value is given by

$$NW_{seg} = T_{base} + C_{tot} + R_{tot}, \tag{15}$$

and the length W_i of each segment is then computed as $W_i = W_{seg} - C_i - R_i$. If $C_N = 0$, the last segment involves an additional amount C of work duration. Similarly, if $R_1 = 0$, the first segment involves an additional amount R of work duration.

Then, we can derive the optimal value of N and W_{seg} as follows: Equation (15) gives $N = \frac{T_{base} - R - C + R_1 + C_N}{W_{seg}}$. Plugging this value into

$$\mathbb{E} = \left(\frac{1}{p\lambda} + D\right) \left[(N-1)e^{p\lambda R} + e^{p\lambda R_1} + Ne^{p\lambda W_{seg}}\right]$$

enables to solve for W_{seg}, using the Lambert function in a similar way as in [6].

While the derivation is painful, the conclusion is comforting: in the optimal solution, all segments have same length of work, up to an additional recovery for the first segment and an additional checkpoint for the last one. The Young/Daly approximation still holds, as well as all the results of the paper (whose presentation is much simpler with equal-size work segments).

B Maximum of n IID negative binomial variables of parameters $(N_{\boldsymbol{c}}, P_{suc})$

Consider n tasks, that each need to process N_c segments with probability of success P_{suc}. We want to bound the expectation $\mathbb{E}(M_f)$ of the maximal number

of failures. Let $Q = \frac{1}{1-P_{suc}}$. From [20], we obtain:

$$\begin{aligned}\mathbb{E}(M_f) &= \log_Q(n) + (N_c - 1)(\log_Q(\log_Q(n)) + \log_Q(P_{suc}) + 1) \\ &- \log_Q(N_c - 1)! + \frac{1}{2} + \frac{\gamma}{\ln(Q)} \\ &+F[\log_Q(n) + N_c - 1(\log_Q(\log_Q(n)) + \log_Q(P_{suc}) + 1) - \log_Q(N_c - 1)!] \\ &+o(1),\end{aligned}$$

where F is a C^∞ periodic function of period 1 and mean value 0, and whose Fourier-coefficients, for $k \in \mathbb{Z}\backslash\{0\}$ are given by

$$\hat{F}(k) = -\frac{1}{\ln(Q)}\Gamma\left(-\frac{2k\pi i}{\ln(Q)}\right),$$

and $\hat{F}(0) = 0$. Our first result is the following bound on F:

Lemma 3. $\forall x \in \mathbb{R}, F(x) \leq \frac{\ln(Q)}{12}$.

Proof. We first show that $(\hat{F}(n))_{n\in\mathbb{Z}}$ is summable. Indeed, for all $z \in \mathbb{C}\backslash\mathbb{Z}^-$,

$$\Gamma(z) = \frac{e^{-\gamma z}}{z}\prod_{k\geq 1}\frac{e^{\frac{z}{k}}}{1+\frac{z}{k}},$$

where γ is the Euler constant [24]. We know that for $n \in \mathbb{Z}\backslash\{0\}$,

$$\Gamma\left(\frac{-2n\pi i}{\ln(Q)}\right) = \frac{\ln(Q)}{-2n\pi i}exp\left(\frac{2\gamma n\pi i}{\ln(Q)}\right)\prod_{k\geq 1}\frac{exp\left(\frac{-2n\pi i}{k\ln(Q)}\right)}{1-\frac{2n\pi i}{k\ln(Q)}}.$$

We have $|e^{ix}| = 1$ for $x \in \mathbb{R}$, hence

$$\left|\Gamma\left(\frac{-2n\pi i}{\ln(Q)}\right)\right| = \frac{\ln(Q)}{2|n|\pi}\prod_{k\geq 1}\frac{k\ln(Q)}{|k\ln(Q) - 2n\pi i|}.$$

Since $\left|\frac{k\ln(Q)}{|k\ln(Q)-2n\pi i|}\right|^2 = \frac{k^2\ln(Q)^2}{k^2\ln(Q)^2+4n^2\pi^2} \leq 1$, we can omit all the terms of the product except the first where $k = 1$, and we obtain:

$$\left|\Gamma\left(\frac{-2n\pi i}{\ln(Q)}\right)\right| \leq \frac{\ln(Q)^2}{2|n|\pi\sqrt{\ln(Q)^2 + 4n^2\pi^2}} \leq \frac{\ln(Q)^2}{4\pi^2 n^2}.$$

The dependence in $\frac{1}{n^2}$ shows us that $(\hat{F}(n))_{n\in\mathbb{Z}}$ is summable, thus we can bound directly F using the result on Fourier series:

$$\begin{aligned}\forall x \in \mathbb{R}, F(x) \leq |F(x)| &= \left|\sum_{n\in\mathbb{Z}}\hat{F}(n)e^{2x\pi ni}\right| \leq \sum_{n\in\mathbb{Z}}|\hat{F}(n)| \\ &\leq \sum_{n\in\mathbb{Z}}\frac{1}{\ln(Q)}\left|\Gamma\left(\frac{-2n\pi i}{\ln(Q)}\right)\right| \\ &\leq \frac{\ln(Q)}{4\pi^2}\times 2\sum_{n\in\mathbb{N}^+}\frac{1}{n^2} \leq \frac{\ln(Q)}{2\pi^2}\frac{\pi^2}{6} \leq \frac{\ln(Q)}{12}.\end{aligned}$$

□

Using Lemma 3, we finally obtain:

$$\mathbb{E}(M_f) \leq \log_Q(n) + (N_c - 1)(\log_Q(\log_Q(n)) + 1) + \frac{1}{2} + \frac{1}{\ln(Q)} + \frac{\ln(Q)}{12} + o(1).$$

C Proof of Proposition 1

Proof. First, straightforwardly, because checkpoint and recovery succeed with probability $e^{-p\lambda C} > 0.99$ (resp. $e^{-p\lambda R} > 0.99$),

$$\begin{aligned} W_{ME} = \frac{T_{base}}{\left\lceil \frac{T_{base}}{W_{YD}} \right\rceil} &\leq W_{YD} = \sqrt{\frac{2\mu C}{p}}; \\ p\lambda C = -\ln(e^{-p\lambda C}) &\leq -\ln(0.99) < \frac{1}{99}; \\ p\lambda R = -\ln(e^{-p\lambda R}) &\leq -\ln(0.99) < \frac{1}{99}. \end{aligned}$$

Thus,

$$\begin{aligned} \ln(Q^*) = \ln\left(\frac{1}{1 - P_{suc}(R)}\right) &= \ln\left(\frac{1}{1 - e^{-p\lambda(W_{ME}+C+\tilde{R})}}\right) \\ &> \ln\left(\frac{1}{1 - e^{-\sqrt{2p\lambda C} - p\lambda C - p\lambda R}}\right) \\ &> \ln\left(\frac{1}{1 - e^{-\sqrt{\frac{2}{99}} - \frac{2}{99}}}\right) > \frac{15}{8}. \end{aligned}$$

Also, because a segment fails with probability at least 10^{-10}, we have $\frac{15}{8} < \ln(Q^*) < \ln(10^{10}) < 23.1$. Since $f(x) = \frac{x}{12} + \frac{1}{x} < 2$ for $x \in [\frac{15}{8}, 23.1]$, we obtain:

$$\frac{\ln(Q^*)}{12} + \frac{1}{\ln(Q^*)} < 2. \tag{16}$$

We now consider two cases:
Case 1: If $N_{ME} = 1$, then $W_{ME} = T_{base}$. In that case, as we supposed $2(C + R + D) \leq T_{base}$, we conclude:

$$\frac{C + R + D}{W_{ME}} \leq \frac{1}{2} \text{ and } \frac{C}{W_{ME}} \leq \frac{1}{2}.$$

Plugging this altogether with Equation (16) in the ratio given in Equation (3), we obtain:

$$\begin{aligned} r_{id}^{ME}(n, \mathcal{T}) &\leq \frac{3}{2}\left(\frac{\log_{Q^*}(n)}{N_{ME}} + \log_{Q^*}(\log_{Q^*}(n)) + \frac{2}{N_{ME}}\right) + 3 + o(1) \\ &\leq \frac{3}{2\ln(Q^*)}\left(\frac{\ln(n)}{N_{ME}} + \ln(\ln((n))\right) + 3 + \frac{3}{N_{ME}} + o(1) \\ &\leq \frac{4}{5}\left(\frac{\ln(n)}{N_{ME}} + \ln(\ln(n))\right) + 3 + \frac{3}{N_{ME}} + o(1). \end{aligned}$$

Case 2: Suppose now that $N_{ME} > 1$.
First, we should note that as $e^{-p\lambda C} < 0.99$, $72p\lambda C < 1$. Moreover $N_{ME} = \left\lceil \frac{T_{base}}{W_{YD}} \right\rceil \geq 2$, thus $\left\lceil \frac{T_{base}}{W_{YD}} \right\rceil \leq \frac{T_{base}}{W_{YD}} + 1 \leq 2\frac{T_{base}}{W_{YD}}$ and finally

$$W_{ME} = \frac{T_{base}}{\left\lceil \frac{T_{base}}{W_{YD}} \right\rceil} \geq \frac{W_{YD}}{2} \geq \sqrt{\frac{\mu C}{2p}} \geq \sqrt{72p\lambda C}\sqrt{\frac{\mu C}{2p}} \geq 6C \geq 6R \geq 6D.$$

This also shows that

$$\frac{C+R+D}{W_{ME}} \leq \frac{1}{2} \text{ and } \frac{C}{W_{ME}} \leq \frac{1}{2},$$

and we obtain the same result in that case, i.e.

$$r_{id}^{ME}(n, \mathcal{T}) \leq \frac{4}{5}\left(\frac{\ln(n)}{N_{ME}} + \ln(\ln(n))\right) + 3 + \frac{3}{N_{ME}} + o(1),$$

which concludes the proof. □

D Proof of Theorem 2

The key element of this proof is a new result (to the best of our knowledge) on expectations of RVs:

Theorem 6. *Let $(X_1, \ldots, X_n)$ be n independent positive RVs with finite expectation, and let $Y = \max(X_1, \ldots, X_n)$. Let Z_i be the maximum of n IID RVs $X_{i,j}$ with the same law as X_i (thus, $Z_i = \max(X_{i,1}, X_{i,2}, \ldots, X_{i,n})$). Then, $\mathbb{E}(Y) \leq 2\max_i(\mathbb{E}(Z_i))$.*

We start with two lemmas before proving Theorem 6:

Lemma 4. *Let $z, a, b \in \mathbb{R}^+$ and $n \in \mathbb{N}^*$ defined in the following domain:*

$$\begin{aligned} 1 &\leq z \\ 1 &\leq n \\ \left(\frac{z-1}{z}\right)^{\frac{1}{n}} &\leq p_0 < 1 \\ 0 &< p_z < 1 - p_0 \end{aligned}$$

Then the following equation holds

$$(1 - (p_0 + p_z))\frac{z[(p_0+p_z)^n - p_0^n]}{1-(p_0+p_z)^n} - p_z(z-1) \geq 0 \tag{17}$$

Proof. If we multiply by $1 - (p_0 + p_z)^n$ and develop, we see that this equation is equivalent to proving that the polynomial $P(z, p_0, p_z)$ is nonnegative over its domain, where

$$\begin{aligned} P(z, p_0, p_z) = {} & z(p_0+p_z)^n + z(p_0+p_z)p_0^n \\ & + (z-1)p_z(p_0+p_z)^n - zp_0^n - z(p_0+p_z)^{n+1} - (z-1)p_z \end{aligned}$$

Consider fixed values of of p_0 and p_z with $0 < p_z < 1 - p_0$. The polynomial $Q(z) = P(z, p_0, p_z)$ s affine in z. The range of z is $1 \leq z \leq \frac{1}{1-p_0^n}$, so it is sufficient to prove that $Q(1) \geq 0$ and $Q(\frac{1}{1-p_0^n}) \geq 0$ to get the result.

We compute easily that $Q(1) = (1 - p_0 - p_z)((p_0 + p_z)^n - p_0^n) \geq 0$. Now, $1 - \frac{1}{1-p_0^n} = \frac{p_0^n}{1-p_0^n}$ and

$$Q(\frac{1}{1-p_0^n}) = \frac{1}{1-p_0^n}\left[((p_0+p_z)^n - p_0^n)(1-p_0) - (p_0+p_z)^n(1-p_0^n)p_z\right]$$

Letting $R(p_z) = ((p_0 + p_z)^n - p_0^n)(1 - p_0) - (p_0 + p_z)^n(1 - p_0^n)p_z$, we need to show that $R(p_z) \geq 0$ for $0 \leq p_z \leq 1 - p_0$. But we have $R(0) = R(1 - p_0) = 0$, and differentiating, $R'(p_z) = (p_0 + p_z)^{n-1}S(p_z)$ where

$$S(p_z) = n(1 - p_0) - p_0(1 - p_0^n) - p_z(1 - p_0^n)(1 + p_0)$$

We see that $S(p_z)$ is affine, positive then negative over $\mathbb{R}$, hence $R(p_z)$ is strictly increasing and then strictly decreasing over $\mathbb{R}$. Given that $R(0) = R(1-p_0) = 0$, $R(p_z)$ is nonnegative for $0 \leq p_z \leq 1 - p_0$, which concludes the proof. □

Lemma 5. *Let $\epsilon \in (0, 1)$ and $(X_1, X_2, \ldots, X_n)$ be n independent random variables such that X_i can take only two values : 0 and $M_i > 0$. We assume that for all i, $\mathbb{E}(max(X_{i,1}, X_{i,2}, \ldots, X_{i,n})) = 1$, where all the $X_{i,j}$ follow the same law as X_i. We finally define X_0, a constant random variable always equal to 1. Then if for all $i > 0$, $M_i \geq \frac{2}{\epsilon}$, we have $\mathbb{E}(Y) = \mathbb{E}(max(X_0, X_1, X_2, \ldots, X_n)) \leq 2 + \epsilon$.*

Proof. We define for all $i > 0$, $p_i = \mathbb{P}\{X_i = 0\} = 1 - \mathbb{P}\{X_i = X_i\}$. From the condition $\forall i > 0, \mathbb{E}(max(X_{i,1}, X_{i,2}, \ldots, X_{i,n})) = 1$, we obtain the following relation between M_i and p_i:

$$\forall i > 0, \mathbb{E}(\max(X_{i,1}, X_{i,2}, \ldots, X_{i,n})) = M_i(1 - p_i^n) = 1$$

From which we derive :

$$\forall i > 0, M_i = \frac{1}{1-p_i^n}$$
$$\forall i > 0, p_i^n = 1 - \frac{1}{X_i} \geq 1 - \frac{\epsilon}{2}$$

We can then upper bound our $\mathbb{E}(Y)$ to obtain :

$$\begin{aligned}
\mathbb{E}(Y) = \mathbb{E}(\max(X_0, X_1, X_2, \ldots, X_n)) &\leq \sum_{i=0}^{n} \mathbb{E}(X_i) \\
&\leq 1 + \sum_{i=1}^{n}(1 - p_i)M_i \leq 1 + \sum_{i=1}^{n} \frac{1-p_i}{1-p_i^n} = 1 + \sum_{i=1}^{n} \frac{1}{\sum_{j=0}^{n-1} p_i^j} \\
\mathbb{E}(Y) \leq 1 + \sum_{i=1}^{n} \frac{1}{np_i^n} &\leq 1 + \frac{n}{n(1-\frac{\epsilon}{2})} = 1 + 1 + \frac{\epsilon}{2-\epsilon} \leq 2 + \epsilon
\end{aligned}$$

□

We are now ready to prove Theorem 6.

Proof of Theorem 6. The sketch of the proof is as follows. Let $R_0 = \frac{\mathbb{E}(Y)}{\max_i(\mathbb{E}(Z_i))}$. We fix $\epsilon \in (0,1)$ arbitrarily small. We apply a set of transformations to the x_i so that they eventually satisfy the conditions described in Lemma 5, while decreasing the ratio by a factor at most $(1+\epsilon)^3$. To show this, we will use the equation of Lemma 4. This will prove that the ratio is less than $(1+\epsilon)^3(2+\epsilon)$ for all ϵ, thus not greater than 2.

Transformation 1: Bound the X_i

We prove that we can bound the X_i in such a way that the ratio is increased by a factor at most $(1+\epsilon)$. Let $i > 0$ and f be the probability density function of Z_i, thus $\mathbb{E}(Z_i) = \int_0^\infty x f(x)\,dx$. First consider $g(z) = \int_0^z x f(x)\,dx$. We know that $g(z)$ increases monotonically towards $\mathbb{E}(Z_i)$ thus there exists M_i such that $g(M_i) \geq \mathbb{E}(Z_i)\left(1 - \frac{\epsilon}{2}\right)$. We define Z_i'' as follows:

$$Z_i'' = Z_i \text{ if } Z_i \leq M_i$$
$$Z_i'' = 0 \text{ otherwise}$$

Then $\mathbb{E}(Z_i'') = g(M_i) \geq \mathbb{E}(Z_i)\left(1 - \frac{\epsilon}{2}\right)$. We now define X_i' in a similar manner :

$$X_i' = X_i \text{ if } X_i \leq M_i$$
$$X_i' = 0 \text{ otherwise}$$

Clearly, if we let $Z_i' = \max(X_{i,1}', X_{i,2}', \ldots, X_{i,n}')$, with all the $X_{i,j}'$ corresponding to the bounded $X_{i,j}$, we obtain that

$$Z_i' = Z_i \text{ if } Z_i \leq M_i$$
$$Z_i' \geq 0 \text{ otherwise}$$

Thus $\mathbb{E}(Z_i') \geq \mathbb{E}(Z_i'') \geq \mathbb{E}(Z_i)\left(1 - \frac{\epsilon}{2}\right)$.
We can apply this for all i, and replace the X_i by the X_i'. Clearly $\mathbb{E}(Y)$ will decrease and $\max_i(\mathbb{E}(Z_i))$ decreases by a factor at most $\left(1 - \frac{\epsilon}{2}\right)$. Thus after the first transformation, the new ratio R_1 will verify :

$$R_0 \leq \frac{R_1}{1 - \frac{\epsilon}{2}} = R_1\left(1 + \frac{\epsilon}{2-\epsilon}\right) \leq R_1(1+\epsilon)$$

From now on, we assume that all the X_i are bounded by M_i.

Transformation 2: Discretize the X_i

We prove that we can transform the X_i so that they take only a finite number of values and so that the ratio is increased by a factor at most $(1+\epsilon)$.

For all $i > 0$, we split the domain of X_i, $[0, M_i]$ into $N_i = \lceil \frac{M_i}{\epsilon\mathbb{E}(Y)} \rceil$ segments so that each segment is smaller than $\epsilon\mathbb{E}(Y)$, and if X_i is in a segment we replace it by the largest value of the segment. This will naturally increase $\mathbb{E}(Y)$ by a factor at most $(1+\epsilon)$ and it will also increase $\max_i(\mathbb{E}(Z_i))$. Thus the ratio R_2 will verify $R_1 \leq (1+\epsilon)R_2$. More formally, we define X_i' as the following:

$$X_i' = \left\lceil \frac{X_i N_i}{M_i} \right\rceil \frac{M_i}{N_i}$$

We apply this change for all i and define $Y' = \max(X_1', \ldots, X_n')$ as well as the

$Z'_i = \max(X'_{i,1}, X'_{i,2}, \ldots, X'_{i,n})$ and we have:

$$\begin{aligned}
&Y' - Y \leq \max_i(X'_i - X_i) \leq \left(\left\lceil \frac{X_i N_i}{M_i} \right\rceil - \frac{X_i N_i}{M_i}\right) \frac{M_i}{N_i} \leq \frac{M_i}{N_i} \leq \epsilon \mathbb{E}(Y) \\
&\mathbb{E}(Y') \leq (1+\epsilon)\mathbb{E}(Y) \\
&\mathbb{E}(Z'_i) \geq \mathbb{E}(Z_i)
\end{aligned}$$

We can replace the X_i by the X'_i, and after transformation 2 we have

$$R_0 \leq R_1(1+\epsilon) \leq R_2(1+\epsilon)^2$$

From now on, we assume that all the X_i can take a finite number of values bounded by M_i.

Transformation 3: Add a high value in the domain of the X_i

In addition to needing that all the X_i take a finite number of values, we also want that X_i may be extremely large, in order to end up with the conditions described in Lemma 5. More precisely, for any i we define $M'_i = \max\left(\frac{2(1+\epsilon)\mathbb{E}(Z_i)}{\epsilon}, M_i\right)$, $p_i = \frac{\epsilon^2}{2(1+\epsilon)n^2}$ and X'_i as follows:

$$\begin{aligned}
&X'_i = X_i \text{ with probability } 1 - p_i \\
&X'_i = M'_i \text{ otherwise (for exemple if } X_i \\
&\quad \text{is within the } p_i \text{ proportion of its highest values)}
\end{aligned}$$

We define Z'_i and Y' accordingly. Then:

$$\begin{aligned}
\forall i, \mathbb{E}(Z'_i) - \mathbb{E}(Z_i) &\leq n p_i M'_i \leq \frac{\epsilon \mathbb{E}(Z_i)}{n} \\
\mathbb{E}(Y') - \mathbb{E}(Y) &\leq \sum_{i=1}^{n} p_i M'_i \leq \sum_{i=1}^{n} \frac{\epsilon \mathbb{E}(Z_i)}{n^2} \\
&\leq \sum_{i=1}^{n} \frac{\epsilon \mathbb{E}(X_i)}{n} \leq \sum_{i=1}^{n} \frac{\epsilon \mathbb{E}(Y)}{n} \leq \epsilon \mathbb{E}(Y) \\
\forall i, \mathbb{E}(Z_i) &\leq \mathbb{E}(Z'_i) \leq (1+\epsilon)\mathbb{E}(Z_i) \\
\mathbb{E}(Y) &\leq \mathbb{E}(Y') \leq (1+\epsilon)\mathbb{E}(Y) \\
\forall i, M'_i &\geq \frac{2\mathbb{E}(Z'_i)}{\epsilon}
\end{aligned}$$

We replace the X_i by the X'_i (with maximum value $M_i := M'_i$ and the new ratio R_3 verifies :

$$R_0 \leq R_2(1+\epsilon)^2 \leq R_3(1+\epsilon)^3$$

From now on, we assume that all the x_i can take a finite number of values, with a maximal value M_i larger than $\frac{2E(Z_i)}{\epsilon}$.

Transformation 4: Normalize the $\mathbb{E}(Z_i)$

For all i, we alter the X_i such that all the $\mathbb{E}(Z_i)$ becomes equal to one. In practice we do the following:

$$X'_i = \frac{X_i}{\mathbb{E}(Z_i)}$$

As usual, if we define Z'_i accordingly to Z_i under a draw $(X_{i,1}, \ldots, X_{i,n})$, as well as Y' accordingly to Y from a draw of $(X_1, \ldots, X_n)$, we straightforwardly have:

$$\begin{aligned}
\forall i, \mathbb{E}(Z'_i) &= 1 \\
\frac{\mathbb{E}(Y')}{\mathbb{E}(Y)} &\geq \frac{1}{\max_i(\mathbb{E}(Z_i))} \\
\frac{\max_i(\mathbb{E}(Z'_i))}{\max_i(\mathbb{E}(Z_i))} &= \frac{1}{\max_i(\mathbb{E}(Z_i))} \\
M'_i &\geq \frac{M_i}{\mathbb{E}(Z_i)} \geq \frac{2}{\epsilon}
\end{aligned}$$

As before we can replace the X_i by the X'_i, and the new ratio increases, so we have:

$$R_0 \leq R_3(1+\epsilon)^3 \leq R_4(1+\epsilon)^3 = \mathbb{E}(Y)(1+\epsilon)^3$$

From now on, we assume that the X_i verify: $\mathbb{E}(Z_i) = 1$, X_i can only take a finite number of different values, whose highest is at least $\frac{2}{\epsilon}$. We are getting closer to the conditions of Lemma 5, we just need to add X_0 (easy) and to transform the X_i so that they can take only two values.

Transformation 5: Add $X_0 = 1$

We add the random variable X_0 which is always equal to one. We adapt $Y := \max(X_0, X_1, \ldots, X_n)$. Clearly Y can only increase while $\max_i(\mathbb{E}(Z_i))$ is still equal to one. Thus

$$R_0 \leq R_4(1+\epsilon)^3 \leq R_5(1+\epsilon)^3 = \mathbb{E}(Y)(1+\epsilon)^3$$

From this point, we apply a transformation that zero out the smaller possible value for X_i, reducing the number of possible values while $E(Z_i)$ remains unchanged even though M_i and $E(Y)$ increases. The first step will be zeroing out all the positive values smaller than one.

Transformation 6: Removing the minimal strictly positive possible value for a $X_i \neq X_0$, if this value is at most 1 (to be processed iteratively until all the X_i can only be equal to 0 or larger than 1)

This is easy to understand: if z is the minimal strictly positive value that X_i can reach with probability p_z, with $z \leq 1$, and if we want to transform X_i so that $p_z = 0$ while keeping $\mathbb{E}(Z_i) = 1$, we will need to increase the other values (or increase their probability). Either way, if $X_i = z \leq 1$ or if $X_i = 0$ it is strictly the same for Y because $Y \geq X_0 \geq 1$. Thus this transformation can only increase $\mathbb{E}(Y)$.

Let us choose i such that X_i can be in $(0, 1]$ and fix z as the minimal strictly positive value that X_i can reach. (if such i does not exist, we move on to the next step). We let $p_0 \triangleq \mathbb{P}\{X_i = 0\}$, $p_z \triangleq \mathbb{P}\{X_i = z\}$ and $p_{z^+} \triangleq \mathbb{P}\{X_i > z\}$. Similarly, we define $P_0 \triangleq \mathbb{P}\{Z_i = 0\}$, $P_z \triangleq \mathbb{P}\{Z_i = z\}$ and $P_{z^+} \triangleq \mathbb{P}\{Z_i > z\}$. We first compute all these values using only p_0 and p_z:

$$\begin{aligned}
p_{z^+} &= 1 - p_0 - p_z \\
P_0 &= p_0^n \\
P_z &= (p_0 + p_z)^n - P_0 = (p_0 + p_z)^n - p_0^n \\
P_{z^+} &= 1 - (P_0 + P_z) = 1 - (p_0 + p_z)^n
\end{aligned}$$

The idea of the transformation is the following : we zero out p_z ($p_0 := p_0 + p_z$ and $p_z := 0$) which makes $\mathbb{E}(Z_i)$ decrease by zP_z. To balance that, we increase all the other possible values (and not their probability) by $X = \frac{zP_z}{P_{z^+}}$ which will increase $\mathbb{E}(Z_i)$ by zP_z. Formally, we define our new variable X'_i as follows:

$$X'_i = 0 \text{ if } X_i \leq z$$
$$X'_i = X_i + \frac{zP_z}{P_{z^+}} \text{ otherwise}$$

Consider a draw $(x_1, \ldots, x_n)$ following $(X_1, \ldots, X_n)$, and compare $Y = \max(x_0, \ldots, x_n)$ and $Y' = \max(x_0, x_1, \ldots, x_{i-1}, x'_i, x_{i+1}, \ldots, x_n)$. If $x_i \leq z < 1$ or if $x'_i \leq Y$, $Y = Y'$; otherwise $Y' > Y$. Thus $\mathbb{E}(Y') > \mathbb{E}(Y)$, $E(Z'_i) = 1$, and the ratio increases as well as M_i.

When we may not apply transformation 6 again, each X_i can only be equal to 0 or larger than 1, with a maximum possible value greater than $\frac{2}{\epsilon}$, and we can move on to the last transformation before concluding the proof.

Transformation 7: If some X_i can take more than two different values, remove its smallest strictly positive value (to be processed iteratively until meeting the conditions of Lemma 5)

We take an X_i and its minimum strictly positive value $z \geq 1$; we apply the same transformation as in transformation 6 although the analysis differs, i.e.

$$X'_i = 0 \text{ if } X_i \leq z$$
$$X'_i = X_i + \frac{zP_z}{P_{z^+}} \text{ otherwise}$$

Consider a draw $(x_1, \ldots, x_n)$ following $(X_1, \ldots, X_n)$, and compare $Y = \max(x_0, \ldots, x_n)$ and $Y' = \max(x_0, x_1, \ldots, x_{i-1}, x'_i, x_{i+1}, \ldots, x_n)$. If $x_i \leq z < 1$ or if $x'_i \leq Y$, $Y = Y'$; otherwise $Y' > Y$. Thus $\mathbb{E}(Y') > \mathbb{E}(Y)$, $E(Z'_i) = 1$, and the ratio increases as well as M_i. Straightforwardly if $x_i \leq Y$ then $Y' \geq Y$; otherwise $x_i = \max_{j=0}^{n}(x_j)$. There are two cases:

- **Case 1:** $x_i = z$. This happens with a probability $p_z \prod_{i=0, i\neq j}^{n} \mathbb{P}\{x_j \leq z\}$ and in this case $Y' - Y = \max_{i=0,i\neq j}(x_j) - z \geq 1 - z$.
- **Case 2:** $x_i > z$. This happens with a probability $p_{z^+} \prod_{i=0,i\neq j}^{n} \mathbb{P}\{x_j \leq x_i | x_i > z\} \geq p_{z^+} \prod_{i=0,i\neq j}^{n} \mathbb{P}\{x_j \leq z\}$ and in this case $Y' - Y = \frac{zP_z}{P_{z^+}}$.

We are now able to bound $\mathbb{E}(Y' - Y)$ and show that it is nonnegative.

$$\mathbb{E}(Y' - Y) \geq \prod_{i=0,i\neq j}^{n} \mathbb{P}\{x_j \leq z\} \left(p_{z^+} \frac{zP_z}{P_{z^+}} - p_z(z-1) \right)$$

If $\prod_{i=0,i\neq j}^{n} \mathbb{P}\{x_j \leq z\} = 0$, we are done; otherwise:

$$\begin{aligned}\mathbb{E}(Y' - Y) \geq 0 &\Leftrightarrow p_{z^+} \frac{zP_z}{P_{z^+}} - p_z(z-1) \geq 0 \\ &\Leftrightarrow (1 - (p_0 + p_z)) \frac{z[(p_0 + p_z)^n - p_0^n]}{1 - (p_0 + p_z)^n} - p_z(z-1) \geq 0\end{aligned}$$

We are now under the conditions described in Lemma 4 and can conclude. Indeed, the following conditions are obvious:

$$\begin{aligned} n &\in \mathbb{N}^* \\ 1 &\leq z \\ 1 &\leq n \\ p_0 &< 1 \\ 0 &< p_z < 1 - p_0 \end{aligned}$$

So we only need to show that $p_0 \geq (\frac{z-1}{z})^{\frac{1}{n}}$, i.e. $P_0 \geq \frac{z-1}{z}$. A quick study of $\mathbb{E}(Z_i)$ shows us what we need:

$$\begin{aligned} 1 = \mathbb{E}(Z_i) = (1 - P_0)\mathbb{E}(Z_i | Z_i > 0) \geq (1 - P_0)z \text{ thus} \\ P_0 \geq 1 - \frac{1}{z} \geq \frac{z-1}{z} \end{aligned}$$

Applying Lemma 4, we finally show that $\mathbb{E}(Y') \geq \mathbb{E}(Y)$. We fix $x_i := x'_i$ and we have increased the ratio while decreasing the number of possible values for x_i, increasing M_i and keeping $\mathbb{E}(Z_i) = 1$. Once all the x_i can only take two possible values, 0 and $X_i \geq \frac{2}{\epsilon}$, we cannot apply transformation 7 any more, and are ready to conclude.

Conclusion of the proof of Theorem 6:
We are now exactly under the conditions of Lemma 5, Furthermore, transformations 6 and 7 increased the ratio, thus:

$$R_0 \leq (1 + \epsilon)^3 \mathbb{E}(Y) \leq (1 + \epsilon)^3 (2 + \epsilon)$$

Now suppose there exists a case such that $R_0 = 2 + \mu$ with $\mu > 0$. Applying the transformations with ϵ small enough (for example $\epsilon = min(\frac{1}{2}, \frac{\mu}{22})$) we reach a contradiction. We can finally conclude the proof and claim:

$$R_0 \leq 2$$

Tightness:
For any $\epsilon > 0$, it is possible to build an example such that $R_0 \geq 2 - \frac{1}{n} - \epsilon$. We provide a brief argument as follows. Consider n independent positive random variables, $X_1 = 1$ and for $i > 1$, $X_i = 0$ with probability p and x otherwise, such that $\mathbb{E}(Z_i) = 1$. As in Lemma 5 we have $x = \frac{1}{1-p^n}$. Then

$$\begin{aligned} \mathbb{E}(Y) &= \mathbb{P}\{Y = 1\} + X_i \mathbb{P}\{Y = x\} \\ &= p^{n-1} + \frac{1 - p^{n-1}}{1 - p^n} \\ &= 1 + p^{n-1} - \frac{1}{\sum_{i=0}^{n-1} p^i} \xrightarrow[p \to 1]{} 2 - \frac{1}{n} \end{aligned}$$

This shows that we can build an example with a ratio arbitrarily close to $2 - \frac{1}{n}$. □

Back to the proof of Theorem 2.

In fact, Theorem 6 allows us to directly extend Theorem 1 to Theorem 2. For all i, let T_i be the Random Variable representing the execution time of the task with profile $\mathcal{T}_i$ and let $X_i = \frac{T_i}{T^i_{base}}$. Clearly, the X_i's are independent and positive, so $Y = \max(X_1, \ldots, X_n)$ matches the condition of Theorem 2. Furthermore, for any given i, we suppose that we were to schedule a shelf of n identical tasks of type $\mathcal{T}_i$ in which $T_{i,j}$ is the Random Variable representing their execution time and $X_{i,j} = \frac{T_{i,j}}{T^i_{base}}$. Then, clearly, the $X_{i,j}$'s are IID and follow the same law as X_i, thus we can define $Z_i = \max_j(X_{i,j})$ to reach the conditions of Theorem 6. Finally, the two following equations hold:

$$r^{ME}(n, \mathcal{T}) = \mathbb{E}\left(\max_i\left(\frac{T_i}{\max_i(T^i_{base})}\right)\right) \leq \mathbb{E}\left(\max_i\left(\frac{T_i}{T^i_{base}}\right)\right) = \mathbb{E}(Y)$$

$$\forall i, r^{ME}_{id}(n, \mathcal{T}_i) = \mathbb{E}\left(\max_j\left(\frac{T_{i,j}}{T^i_{base}}\right)\right) = \mathbb{E}\left(\max_j(X_{i,j})\right) = \mathbb{E}(Z_i)$$

We can then apply Theorem 6 and obtain the result:

$$r^{ME}(n, \mathcal{T}) \leq \mathbb{E}(Y) \leq 2\max_i(\mathbb{E}(Z_i)) = 2\max_{1\leq i\leq n}\left(r^{ME}_{id}(n, \mathcal{T}_i)\right). \tag{18}$$

E Proof of Theorem 3

The $(a_{i,j})_{i\in[1,n],j\in[1,K]}$ for the example in the proof of Theorem 3 are given by

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1/2 & 1/4 & 1/4 \\ 3/4 & 1/4 & 0 & 0 \\ 2/3 & 2/9 & 1/9 & 0 \end{bmatrix}$$

The total execution time of $\mathbb{S}^{virt}$ is $\sum_{j=1}^{K} T(S^{virt}_j)$. From Theorem 2, we directly have that

$$\mathbb{E}(T(S^{virt}_j)) \leq 2\max_{i\in B_j}(r^{ME}_{id}(|B_j|, \mathcal{T}_i))T(S_j) \leq 2\max_{1\leq i\leq n}(r^{ME}_{id}(\Delta, \mathcal{T}_i))T(S_j).$$

The second inequality holds because $|B_j| \leq \Delta$ and because r^{ME}_{id} increases when the number of tasks increases. Finally,

$$\mathbb{E}(T(\mathbb{S}^{virt})) = \sum_{j=1}^{K}\mathbb{E}(T(S^{virt}_j)) \leq 2\max_{1\leq i\leq n}(r^{ME}_{id}(\Delta, \mathcal{T}_i))\sum_{j=1}^{K}T(S_j),$$

where $\sum_{j=1}^{K} T(S_j) = T(\mathbb{S})$.

Finally, here is the proof by induction to prove that no task starts nor ends later in $\mathbb{S}$' than in $\mathbb{S}^{virt}$. Recall that the key element is that the ordering of starting times from $\mathbb{S}$ is preserved in both $\mathbb{S}^{virt}$ and $\mathbb{S}'$.

- The first task starts at time 0 for both $\mathbb{S}^{virt}$ and $\mathbb{S}'$, and may be suspended in $\mathbb{S}^{virt}$, hence its ending time cannot be larger in $\mathbb{S}'$;

- Let $i > 1$, and suppose the induction hypothesis holds for the first $i-1$ tasks. Let x be the starting time of task i in schedule $\mathbb{S}^{virt}$. By definition of the slices in $\mathbb{S}^{virt}$, when a task starts, all the unfinished tasks are running concurrently; thus there are enough processors to process task i and all the unfinished tasks among the first $i-1$ ones. Since the ending time of all these unfinished tasks may not be larger in $\mathbb{S}'$, there are enough processors to start task i in $\mathbb{S}$' at time x if it has not started yet. Because we try to start the i-th task in $\mathbb{S}$' before the ones that are started later in $\mathbb{S}$, we will indeed not start task i later in $\mathbb{S}'$ than in $\mathbb{S}^{virt}$. And because task i will not be suspended in $\mathbb{S}'$, it will not end later either.

This concludes the induction and the proof of Theorem 3.

F Proof of Theorem 5

Proof. To prove this theorem, we just need to adapt the proof of Theorem 3. In fact, the analysis in Theorem 3 did not depend upon the checkpoint strategy, thus using the same slices $(S_j)_{j\in[1,K]}$ and virtual schedule $\mathbb{S}^{virt}$, we have for both CheckMore and BasicCheckMore:

$$\mathbb{E}(T(\mathbb{S}')) \leq \mathbb{E}(T(\mathbb{S}^{virt})) = \sum_{j=1}^{K} \mathbb{E}(T(S_j^{virt})). \tag{19}$$

Again, for all slices S_j and all tasks $i \in B_j$ (recall that B_j is the set of tasks in slice S_j), we have $X_i = \frac{T_i}{T_{base}^i}$. Then, the scaling of S_j, $\frac{T(S_j^{virt})}{T(S_j)}$, corresponds to $\max_{i\in B_j} X_i$. We then can safely use Theorem 6; assuming that each task i is checkpointed according to SafeCheck(δ_i), we obtain:

$$\frac{\mathbb{E}(T(S_j^{virt}))}{T(S_j)} \leq 2 \max_{i\in B_j} \left(r_{id}^{SC}(\delta_i, |B_j|, \mathcal{T}_i)\right) \leq 2 \max_{1\leq i\leq n} \left(r_{id}^{SC}(\delta_i, \Delta_i, \mathcal{T}_i)\right) \tag{20}$$

Finally, using Equation (20), Equation (19), and $\forall i, \Delta_i \leq \Delta$, we obtain for CheckMore and BasicCheckMore respectively:

$$\mathbb{E}(T(\mathbb{S}')) \leq 2 \max_{1\leq i\leq n} \left(r_{id}^{SC}(\Delta_i, \Delta_i, \mathcal{T}_i)\right) \sum_{j=1}^{K} T(S_j);$$

$$\mathbb{E}(T(\mathbb{S}')) \leq 2 \max_{1\leq i\leq n} \left(r_{id}^{SC}(min(n,m), \Delta, \mathcal{T}_i)\right) \sum_{j=1}^{K} T(S_j).$$

□

G Example for the tightness of the bound of Theorem 1

In the example, we have n identical tasks of type $\tau = (2K-1, 1, 1, 0)$, with $K \geq 2$. We have $n \leq m$, $D = 0$ and $e^{-\lambda(T_{base}+C)} = \frac{2K}{2K+1}$.

G.1 MinExp

In this section, we study the MinExp strategy for the example, when K is fixed and n is the variable.

Lemma 6.

$$\frac{3}{8} \leq \frac{T_{base}}{W_{YD}} \leq \frac{1}{\sqrt{2}} \text{ and } N_{ME} = 1$$

Proof. We have $T_{base} = 2K-1$, $W_{YD} = \sqrt{\frac{4K}{\ln(1+\frac{1}{2K})}}$, and$(\frac{T_{base}}{W_{YD}})^2 = \frac{(2K-1)^2}{4K}\ln(1+\frac{1}{2K})$. Hence, using $\frac{x}{2} \leq \ln(1+x) \leq x$ for $0 \leq x \leq 1$ and $K \geq 2$, we have

$$(\frac{3}{8})^2 \leq (\frac{2K-1}{4K})^2 \leq (\frac{T_{base}}{W_{YD}})^2 \leq \frac{(2K-1)^2}{8K^2} \leq \frac{1}{2}$$

hence

$$\frac{3}{8} \leq \frac{T_{base}}{W_{YD}} \leq \frac{1}{\sqrt{2}} \tag{21}$$

Finally, $N_{ME} = \lceil \frac{T_{base}}{W_{YD}} \rceil = 1$. □

Let $M_{f,1}$ denote the maximal number of failures over all tasks when using the MinExp strategy.

Lemma 7. *If K is fixed, $r_{id}^{ME}(n, \mathcal{T}) = \Theta(\ln(n))$, showing the tightness of the bound given in Theorem 1*

Proof. Let $\mathbb{E}(T_{lost}(T_{base}+C))$ be the expected time lost due to one failure if we take only one checkpoint (strategy MinExp). We get $E_1 \geq \mathbb{E}(M_{f,1})\mathbb{E}(T_{lost}(T_{base}+C)) + T_{base} + C$, where E_1 is the expected total time for MinExp. Indeed E_1 is at least equal to the expectation of a task with the maximal number of failures. The expectation of such a task is the expected number of failures time the expected waste per failures (independent RVs) plus the time $T_{base} + C$ of the first execution.

Now the expected time lost before a failure when attempting to successfully execute for T seconds (either work or checkpoint) is computed as

$$\mathbb{E}(T_{lost}(T)) = \int_0^\infty x\mathbb{P}(X = x|X < T)dx = \frac{1}{\mathbb{P}(X < T)}\int_0^T xe^{-\lambda x}dx$$

and $\mathbb{P}(X < T) = 1 - e^{-\lambda T}$. Integrating by parts, we derive that $\mathbb{E}(T_{lost}(T)) = \frac{1}{\lambda} - \frac{T}{e^{\lambda T}-1}$.
We can check that:

$$\frac{T}{2} \geq \mathbb{E}(T_{lost}(T)) \geq \frac{e^{-\lambda T}T}{2}. \tag{22}$$

The first inequality is easy, the second is obtained by differentiating the function $g(T) = \mathbb{E}(T_{lost}(T)) - \frac{e^{-\lambda T}T}{2}$.

Using Equation (22), we have

$$\mathbb{E}(T_{lost}(T_{base}+C)) \geq \frac{e^{-\lambda(T_{base}+C)}(T_{base}+C)}{2}$$

hence $\mathbb{E}(T_{lost}(T_{base}+C)) \geq \frac{2K^2}{2K+1} \geq \frac{4K}{5}$ for $K \geq 2$. Thus $E_1 \geq \frac{4K}{5}\mathbb{E}(M_{f,1}) + 2K$.

Furthermore, we have that

$$E_1 \leq T_{base} + C + \mathbb{E}(M_{f,1})(T_{base} + C) = 2K(1 + \mathbb{E}(M_{f,1})) \tag{23}$$

because the re-execution time after each failure is bounded by $T_{base} + C = 2K$. Altogether we have shown that:

$$2K\mathbb{E}(M_{f,1})) + 2K \geq E_1 \geq \frac{4K}{5}\mathbb{E}(M_{f1}) + 2K \tag{24}$$

Since $\mathbb{E}(M_{f1})$ corresponds to the maximum of n IID geometric laws of parameter $P_{suc} = \frac{2K}{2K+1}$, we know from [13] that

$$\frac{H_n}{-\ln(1 - P_{suc})} - 1 \leq \mathbb{E}(M_{f1}) \leq \frac{H_n}{-\ln(1 - P_{suc})} \tag{25}$$

where $H_n = \sum_{k=1}^{n} \frac{1}{k}$ is the n-th Harmonic number and verifies:

$$\ln(n) \leq H_n \leq \ln(n) + 1 \tag{26}$$

From Equations (24), (25) and (26), and using $r_{id}^{ME}(n, \mathcal{T}) = \frac{E_1}{T_{base}}$, we derive

$$\frac{2\ln(n)}{5\ln(2K+1)} + \frac{3}{5} \leq r_{id}^{ME}(n, \mathcal{T}) \leq \frac{4(\ln(n)+1)}{3\ln(2K+1)} \tag{27}$$

For the last inequality, we have used that $\frac{2K}{2K-1} \leq \frac{4}{3}$ for $K \geq 2$. We have derived that $r_{id}^{ME}(n, \mathcal{T}) = \Theta(\ln(n))$. □

G.2 CheckMore

In this section, we compare the MinExp and CheckMore strategies for the example. We want to take $2K-1$ checkpoints for each task, which will partition them into $2K-1$ segments of total length 2. According to Definition 2, this coincides to SafeCheck(δ), where $2K-1 = N_{SC}(\delta) = \left\lceil \frac{(\ln(\delta)+1)T_{base}}{W_{YD}} \right\rceil$. Finally, we choose the number n of tasks as $n = \delta$. Note that BasicCheckMore coincides with CheckMore in this example.

In Lemma 8, we show that the corresponding value of n is

$$n = \left\lfloor e^{W_{YD}-1} \right\rfloor = \left\lfloor e^{\sqrt{\frac{4K}{\ln(1+\frac{1}{2K})}}-1} \right\rfloor. \tag{28}$$

Then, we derive a relation between the performance of MinExp and CheckMore when K is the variable, while n obeys Equation (28) as a function of K.

Lemma 8. *Let $n = \delta = \left\lfloor e^{W_{YD}} - 1 \right\rfloor$. Then, $N_{SC}(n) = 2K - 1$, and we have:*

$$\frac{3\ln(n+1) + 8}{16} \leq K \leq \frac{\ln(n+1)}{2\sqrt{2}} + 1.$$

Proof. We observe that

$$\left\lceil \frac{(\ln(n)+1)T_{base}}{W_{YD}} \right\rceil = 2K-1 \Leftrightarrow 2K-2 < \frac{(\ln(n)+1)T_{base}}{W_{YD}} \leq 2K-1$$
$$\Leftrightarrow e^{\frac{(2K-2)W_{YD}}{T_{base}}-1} < n \leq e^{\frac{(2K-1)W_{YD}}{T_{base}}-1}$$

We can safely let $n = \left\lfloor e^{\frac{(2K-1)W_{YD}}{T_{base}}-1} \right\rfloor = \left\lfloor e^{W_{YD}-1} \right\rfloor$ as soon as

$$e^{\frac{(2K-1)W_{YD}}{T_{base}}} - e^{\frac{(2K-2)W_{YD}}{T_{base}}} \geq e.$$

This difference is clearly increasing when $\frac{W_{YD}}{T_{base}}$ increases (for fixed K) and when K increases (for fixed $\frac{W_{YD}}{T_{base}}$). Using $K \geq 2$ and $\frac{W_{YD}}{T_{base}} \geq \sqrt{2}$ (from Equation (21)), we obtain the minimum value

$$e^{3\sqrt{2}} - e^{2\sqrt{2}} \geq e,$$

hence the result.

Finally, from $\left\lceil \frac{(\ln(n)+1)T_{base}}{W_{YD}} \right\rceil = 2K-1$, we derive:

$$2K-2 < \frac{(\ln(n)+1)T_{base}}{W_{YD}} \leq 2K-1.$$

And using Equation (21),

$$\frac{3\ln(n)+11}{16} \leq K \leq \frac{\ln(n)+1}{2\sqrt{2}} + 1. \tag{29}$$

□

Let $M_{f,2}$ denote the maximal number of failures over all tasks when using the SafeCheck(n) strategy (i.e., CheckMore or BasicCheckMore), and recall that $M_{f,1}$ denote the maximal number of failures when using the MinExp strategy.

Lemma 9. $\mathbb{E}(M_{f,2}) \leq 2\mathbb{E}(M_{f,1})$.

Proof. For CheckMore, we have n tasks with each $N_{ME} = 2K-1$ segments of length 2, one unit of work and one unit of checkpoint. Consider a virtual task set with $2n$ tasks with a single segment of length $2K$, $2K-1$ units of work and one unit of checkpoint. We further assume that there are only n processors, and that each processor executes two tasks consecutively. Intuitively, the virtual set of tasks corresponds to executing twice the tasks of MinExp. Let $M_{f,3}$ denote the maximal number of failures for the virtual set.

Now, for a given processor, and for any failure scenario, we compare the execution of a real task in CheckMore and of two virtual tasks. Let t be the time-step of completion of the first virtual task, which corresponds to the end of the first interval of $2K$ units of time without failing. Then, during these $2K$ units of time, we could process the first K segments of the real task. Afterwards, we need an additional time-interval of length $2K$ without failures to process the second virtual task, which is enough to finish the real task. We

conclude that $M_{f,2} \leq M_{f,3}$. Because this is true for any failure scenario, we have $\mathbb{E}(M_{f,2}) \leq \mathbb{E}(M_{f,3})$.

Then, let X_i denote the RV for the number of failures of the first virtual task of processor i, and Y_i denote the RV for the number of failures of the second virtual task of processor i, where $1 \leq i \leq n$. We have $\mathbb{E}(M_{f,3}) = \mathbb{E}(\max_i(X_i + Y_i)) \leq \mathbb{E}(\max_i(X_i)) + \mathbb{E}(\max_i(Y_i)) = 2\mathbb{E}(M_{f,1})$. □

We are ready to compare the MinExp and CheckMore strategies. We show that the latter is an order of magnitude better than the former.

Proof of Proposition 2

Proof. Let E_2 be the expected total time for BasicCheckMore (equivalent to CheckMore in this case). Similarly to Equation (23), since each segment has length 2, we get:

$$E_2 \leq T_{base} + (2K-1)C + \mathbb{E}(M_{f,2}) \times 2 \leq 4K + 4\mathbb{E}(M_{f,1}). \tag{30}$$

The last inequality comes from Lemma 9.

From Equations (30), (25) and (26), and since $r^{BCM}(n, \mathcal{T}) = \frac{E_2}{T_{base}}$, we have:

$$r^{BCM}(n, \mathcal{T}) \leq 2\frac{\ln(n)+1}{K\ln(2K+1)} + 2.$$

Using Equation (29), we obtain:

$$r^{BCM}(n, \mathcal{T}) \leq \frac{32}{3\ln(2K+1)} + 2 = \Theta(1). \tag{31}$$

Finally, from Equations (29) and (27), as $\ln(n) = \Theta(K)$, we obtain:

$$r^{ME}(n, \mathcal{T}) = \Theta\left(\frac{K}{\ln(K)}\right). \tag{32}$$

□

H Asymptotic analysis

For a shelf of n identical uni-processor tasks executing in parallel on n processors, we have:

$$\begin{aligned} r_{id}^{ME}(n, \mathcal{T}) \leq & \left(\frac{\log_{Q^*_{ME}}(n)}{N_{ME}} + \log_{Q^*_{ME}}(\log_{Q^*_{ME}}(n)) + 1 + \frac{\ln(Q^*_{ME})}{12N_{ME}} \right. \\ & \left. + \frac{1}{\ln(Q^*_{ME})N_{ME}}\right) \times \left(1 + \frac{C+R+D}{W_{ME}}\right) + \frac{C}{W_{ME}} + 1 + o(1). \end{aligned}$$

and

$$\begin{aligned} r_{id}^{CM}(n, \mathcal{T}) \leq & \left(\frac{\log_{Q^*_{CM}}(n)}{N_{SC}(n)} + \log_{Q^*_{CM}}(\log_{Q^*_{CM}}(n)) + 1 + \frac{\ln(Q^*_{CM})}{12N_{SC}(n)} \right. \\ & \left. + \frac{1}{\ln(Q^*_{CM})N_{SC}(n)}\right) \times \left(1 + \frac{C+R+D}{W_{SC}(n)}\right) + \frac{C}{W_{SC}(n)} + 1 + o(1). \end{aligned}$$

In practice, increasing the number of checkpoints increases $\frac{C+R+D}{W}$ and $\frac{C}{W}$ and decreases the other terms. We show that if n is large enough, these two

terms do not become larger than 1 even for CheckMore, and thus if n is large enough, the dominating term of r_{id}^{ME} is $\left(\frac{\log_{Q^*_{ME}}(n)}{N_{ME}} + \log_{Q^*_{ME}}(\log_{Q^*_{ME}}(n))\right)$ while the dominating term of r_{id}^{CM} is $\left(\frac{\log_{Q^*_{CM}}(n)}{N_{SC}(n)} + \log_{Q^*_{CM}}(\log_{Q^*_{CM}}(n))\right)$.

We make the following reasonable assumptions:

1. Processing a single checkpoint with one processor can fail with probability at most 0.05%;
2. $n \leq 10^9$;
3. $2D \leq R \leq C \leq \frac{T_{base}}{25}$.

Because of Assumption (3), we have $C + R + D \leq 2.5C$, so

$$\frac{C+R+D}{W_{SC}(n)} = \frac{C+R+D}{T_{base}} \left\lceil \frac{(\ln(n)+1)T_{base}}{W_{YD}} \right\rceil \leq \frac{2.5C}{W_{YD}}(\ln(n)+1) + \frac{2.5C}{T_{base}}.$$

As $n \leq 10^9$ from Assumption (2), we have $\ln(n) \leq 21$, and because of $\frac{2.5C}{T_{base}} \leq 0.1$ from Assumption (3), we have:

$$\frac{C+R+D}{W_{SC}(n)} \leq 22\frac{2.5C}{\sqrt{2\mu C}} + 0.1 \leq \sqrt{\frac{22^2 \times 2.5^2}{2}\lambda C} + 0.1.$$

From Assumption (1), we directly derive that $e^{-\lambda C} \geq 0.9995$, which gives $\lambda C \leq -\ln(0.9995)$. By doing the numerical computation, we finally show that $\frac{C+R+D}{W_{SC}(n)} < 1$, and by extension $\frac{C}{W_{SC}(n)} < 1$, $\frac{C+R+D}{W_{ME}} < 1$ and $\frac{C}{W_{ME}} < 1$.

This shows that if n is large enough, the dominating terms of both checkpoint strategies are similar, i.e., $\left(\frac{\log_{Q^*_{ME}}(n)}{N_{ME}} + \log_{Q^*_{ME}}(\log_{Q^*_{ME}}(n))\right)$ for r_{id}^{ME}, and $\left(\frac{\log_{Q^*_{CM}}(n)}{N_{SC}(n)} + \log_{Q^*_{CM}}(\log_{Q^*_{CM}}(n))\right)$ for r_{id}^{CM}. Indeed, the other terms were already shown to be small for MinExp in Appendix C and they are even smaller for CheckMore.

Clearly, the number of checkpoints increases with CheckMore and both terms get lower when n increases. In particular, if $N_{ME} > 1$, $\frac{T_{base}}{W_{YD}} > 1$ and $N_{SC}(n) > \ln(n)$, the dominating term of r_{id}^{CM} becomes $\frac{\ln(\ln(n))}{\ln(Q^*_{CM})}$, which might be an order of magnitude lower than $\frac{\ln(n)}{\ln(Q_{ME^*})N_{ME}}$. If this is not the case, meaning that N_{ME} is large, then both dominating terms are $\log_{Q^*}(\log_{Q^*}(n))$ and we still have a significant gain because $Q^*_{CM} > Q^*_{ME}$, and this difference increases with n.

I Complete Experimental Evaluation

This appendix is a self-contained extended version of Section 5. For reader's convenience, it duplicates text and figures from Section 5, but also provides several additional figures and comments.

We evaluate the performance of the different checkpointing strategies through simulations. We describe the simulation setup in Section I.1, present the main

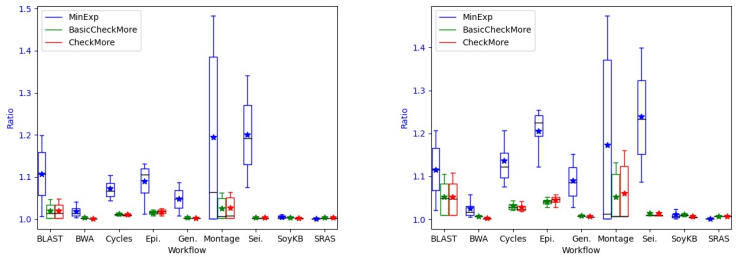

Figure 5: Performance (ratio) comparison of the three checkpointing strategies for the nine different workflows, with large workflows (i.e., failure-free execution time of 3-5 days) shown on the left, and small workflows (i.e., failure-free execution time of 15-24 hours) on the right.

performance comparison results in Section I.2, and assess the impact of different parameters on the performance in Section I.3. Our in-house simulator is written in C++ and is publicly available for reproducibility purpose.

I.1 Simulation Setup

We evaluate and compare the performance of the three checkpointing strategies MinExp, CheckMore and BasicCheckMore. All strategies are coupled with a failure-free schedule computed by a list scheduling algorithm (see below). The workflows used for evaluation are generated from WorkflowHub [18] (formerly Pegasus [35]), which offers realistic synthetic workflow traces with a variety of characteristics and they have been shown to accurately resemble the ones from real-world workflow executions [2, 18]. Specifically, we generate the following nine different types of workflows offered by WorkflowHub that model applications in various scientific domains:

- Blast: a bioinformatics workflow for searching biological sequence databases and identifying amino-acid or DNA sequences that resemble query sequences;
- Bwa: a bioinformatics workflow for performing DNA sequence alignment using the "Burrows-Wheeler Aligner";
- Cycles: an agroecosystem workflow for conducting simulations of crop production and water, carbon and nitrogen cycles in the soil-plant-atmosphere continuum;
- Epigenomics: a bioinformatics workflow for automating various operations in genome sequence processing;
- Genome: a bioinformatics workflow for identifying mutational overlaps to provide statistical evaluation of potential disease-related mutations;
- Montage: an astronomy workflow for analyzing multiple input images to create custom mosaics of the sky;
- Seismology: a seismology workflow for performing seismogram deconvolutions to estimate earthquake source time functions;
- SoyKB: a bioinformatics workflow for performing large-scale next-generation sequencing of soybean lines within the Soybean Knowledge Base (SoyKB);
- Sras: a bioinformatics workflow for downloading and aligning data in the Sequence Read Archive (SRA).

Each trace defines the general structure of the workflow, whose number of tasks and total execution time can be specified by the user[3]. All tasks generated in WorkflowHub are uni-processor tasks.

In the experiments, we evaluate the checkpointing strategies under the following parameter settings:

- Number of processors: $m = 2^{14} = 16384$;
- Checkpoint/recovery/down time: $C = R = 1$ min, $D = 0$;
- MTBF of individual processor: $\mu_{ind} = 10$ years;
- Number of tasks of each workflow: $n \approx 50000$.

Furthermore, the total failure-free execution times of all workflows are generated such that they complete in 3-5 days. This is typical of the large scientific workflows that often take days to complete as observed in some production log traces [1, 33]. To demonstrate the robustness of our evaluation, we also generate small workflow traces that take less than a day (i.e., 15 - 24 hours) to complete. This is roughly one fifth of the size of large workflows. As a result, to keep the average number of failures per task the same, we also scale the individual MTBF from 10 years to 2 years, while all the other parameters are kept the same. Section I.2 will present the comparison results of different checkpointing strategies under the above parameter settings. In Section I.3, we will further evaluate the impacts of different parameters (i.e., m, C, μ_{ind} and n) on the performance.

The evaluation methodology is as follows: for each set of parameters and each type of workflow trace, we generate 30 different workflow instances and compute their failure-free schedules. We use the list scheduling algorithm that orders the tasks using the Longest Processing Time (LPT) first policy: if several tasks are ready and there is at least one processor available, the longest ready task is assigned to the available processor to execute. Since all tasks are uni-processor tasks, LPT is known to be a 2-approximation algorithm [21]; also, LPT is known to be a good heuristic for ordering the tasks [30]. This order of execution will be enforced by all the checkpointing strategies. For each workflow instance, we further generate 50 different failure scenarios. Here, a failure scenario consists of injecting random failures to the tasks by following the Exponential distribution as described in Section 2.1. The same failure scenario will then be applied to each checkpointing strategy to evaluate its execution time for the workflow. We finally compute the *ratio* of a checkpointing strategy under a particular failure scenario as $\frac{T}{T_{base}}$, where T_{base} is the failure-free execution time of the workflow, and T is the execution time under the failure scenario. The statistics of these $30 \times 50 = 1500$ experiments are then compared using boxplots (that show the mean, median, and various percentiles of the ratio) for each checkpointing strategy. The boxes bound the first to the third quantiles (i.e., 25th and 75th percentiles), the whiskers show the 10th percentile to the 90th percentile, the black lines show the median, and the stars show the mean.

I.2 Performance Comparison Results

Figure 5 (left) shows the boxplots of the three checkpointing strategies in terms of their ratios for the nine different workflows, when their failure-free execution time is 3-5 days (i.e., large workflows).

[3]Note that the workflow generator may offer a different number of tasks so as to guarantee the structure of the workflow. The difference, however, is usually small.

First, we observe that CheckMore and BasicCheckMore have very similar performance, which in most cases are indistinguishable. This shows that BasicCheckMore offers a simple yet effective solution without the need to inspect the failure-free schedule, thus making it an attractive checkpointing strategy in practice. Also, both versions of CheckMore perform significantly better and with less variation than MinExp, except for the few workflows where the ratios of all strategies are very close to 1 (e.g., Bwa, SoyKB, Sras). Overall, the 90th percentile ratio of CheckMore never exceeds 1.08, whereas that of MinExp is much higher for most workflows and reaches almost 1.5 for Montage. Similarly, the average ratio of CheckMore never exceeds 1.03, while that of MinExp is again significantly higher and reaches more than 1.2 for Seismology and Montage.

We now examine a few workflows more closely to better understand the performance. For Sras, MinExp is slightly better than CheckMore, but the ratios of all strategies are near optimal (i.e., ¡1.003). In this workflow, very few tasks are extremely long while many others are very short, and there are very few dependencies among them. Thus, failures hardly ever hit the long tasks due to their few number, while failures that hit short tasks have little impact on the overall execution time. This is why the ratio is so small for all strategies. It also explains why MinExp outperforms CheckMore: although the maximum degree of parallelism is important, only a few tasks matter and they should be checkpointed à la Young/Daly to minimize their own expected execution time, and thereby that of the entire workflow. SoyKB and Bwa also have very low ratios. In the case of SoyKB, there is just not enough parallelism during the majority of the execution time, so all strategies are making reasonable checkpointing decisions, with CheckMore performing slightly better for taking into account this small parallelism. Bwa, on the other hand, has two source tasks that must be executed first and two sink tasks that must be executed last. Among them, one source task and one sink task are extremely long, so failures in other tasks have little impact (as in the case of Sras). Yet the small tasks are not totally negligible here, because the dominant sink task must be processed after all of them, so it is still worth to optimize these tasks with CheckMore, which explains why it is slightly better than MinExp.

For all the other workflows, CheckMore performs better than MinExp by a significant margin. This is due to CheckMore's more effective checkpointing strategies given the specific structure of these workflows. For instance, Montage has some key tasks that are dominant, so a failure that strikes most of the other tasks does not impact the overall execution time. This is similar to the case of Sras and explains why, for all strategies, the first quantile of the ratio is very low (i.e., around 1). However, when a failure does strike one of the key tasks, the execution time will be heavily impacted. The difference with Sras is that Montage contains more key tasks that can run in parallel, so it is much more likely that one of them will fail, which is why checkpointing them with CheckMore is better. Next, Blast and Seismology have some source and sink tasks (as Bwa), which, however, are not so dominant in length, making the difference between CheckMore and MinExp higher even from the first quantile. Other workflows also have similar structures, which eventually contribute to the better performance of CheckMore over MinExp.

Figure 5 (right) further shows the comparison results for the nine workflows when their failure-free execution time is 15-24 hours (i.e., small workflows). We

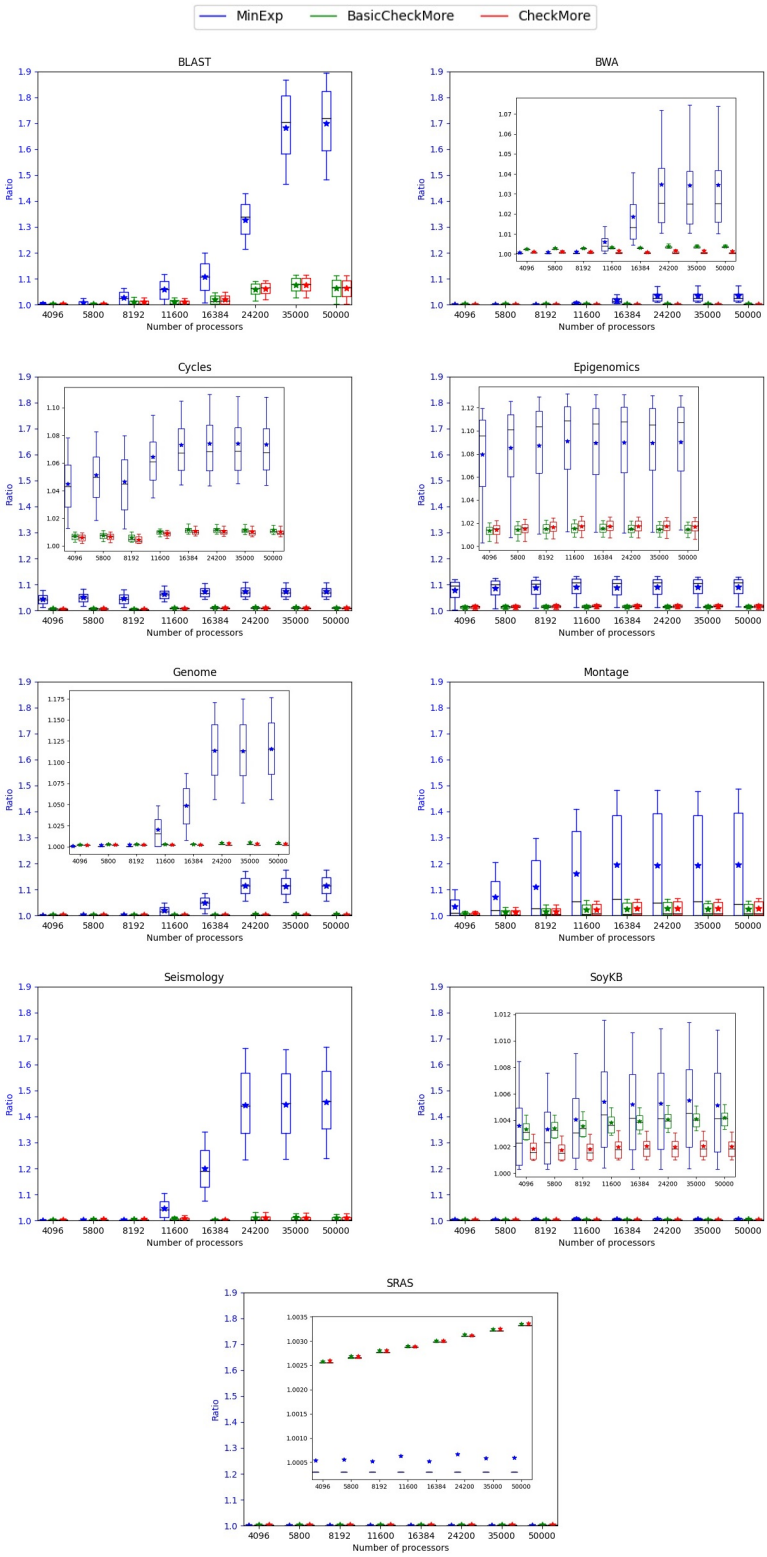

Figure 6: Impact of the number of processors (m) on the performance of the checkpointing strategies for different workflows.

can observe that the results are very similar to those for large workflows, which demonstrates that the relative performance of the three checkpointing strategies is not affected by the size of the workflows, provided that the average number of failures per task remains the same. Thus, in the subsequent experiments, we will only report results for the large workflows.

I.3 Impact of Different Parameters

We now study the impact of different parameters on the performance of the checkpointing strategies. In each set of experiments below, we vary a single parameter while keeping the others fixed at their base values. The results are shown in Figures 6-9, where the scale of the y-axis is kept the same for ease of comparison. For some figures with really small values, zoomed-in plots are also provided on the original figure for better viewing.

Impact of Number of Processors (m). We first assess the impact of the number of processors, which is varied between 4096 and 50000, and the results are shown in Figure 6. In general, increasing the number of processors increases the ratio. This corroborates our theoretical analysis, because for most types of workflows, having more processors means having a larger Δ and thus a larger potential ratio, until m surpasses the width of the dependence graph. However, CheckMore and BasicCheckMore appear less impacted than MinExp.

For Blast, Bwa, Genome, Seismology, the ratio is very low when m is small for all checkpointing strategies. In fact, for these workflows, most tasks are quite independent. Thus, when n is large compared to m, even if a failure strikes a task, it will have little impact on the starting times of the other tasks. This is because we only maintain the order of execution but do not stick to the same mapping as in the failure-free schedule. For this reason, it is better to minimize each task's own execution time by using MinExp (i.e., CheckMore checkpoints a bit too much). However, when m becomes large, the performance of MinExp degrades significantly, with an average ratio even reaching 1.7 for Blast at $m = 50000$, whereas it stays below 1.1 for CheckMore.

For Epigenomics, Cycles and Montage, the ratio does not vary significantly with the number of processors, but is not negligible for MinExp even when m is small (between 1.05 and 1.2 depending on the workflow). For these workflows, the ratio of MinExp is 4 to 10 times higher than that of CheckMore, demonstrating the advantage of the latter checkpointing strategy.

Finally, for Sras, as the number of dominating tasks that could be run in parallel is way less than 4096, the ratio of MinExp does not vary much with m, while that of CheckMore increases with m as it tends to checkpoint more with an increasing number of processors. Also, in more than 90% of the cases, the failures have strictly no impact on the overall execution time, since they do not hit the dominating tasks. This is why the average ratio is above the 90th percentile for all checkpointing strategies. Similarly, for SoyKB, the ratio is not impacted much for MinExp and CheckMore, especially for $m \geq 11000$.

Impact of Checkpoint Time (C). We now evaluate the impact of the checkpoint time by varying it between 15 and 240 seconds, and the results are shown in Figure 7. The ratio generally increases with C; this is consistent with Equation (3). when $R = C$ and $D = 0$, the approximation ratio satisfies $r \leq \left(\frac{X}{N_c} + Y\right)\left(\frac{2C}{W} + 1\right) + \frac{C}{W} + Z$, where X, Y and Z barely depend on C, N_c

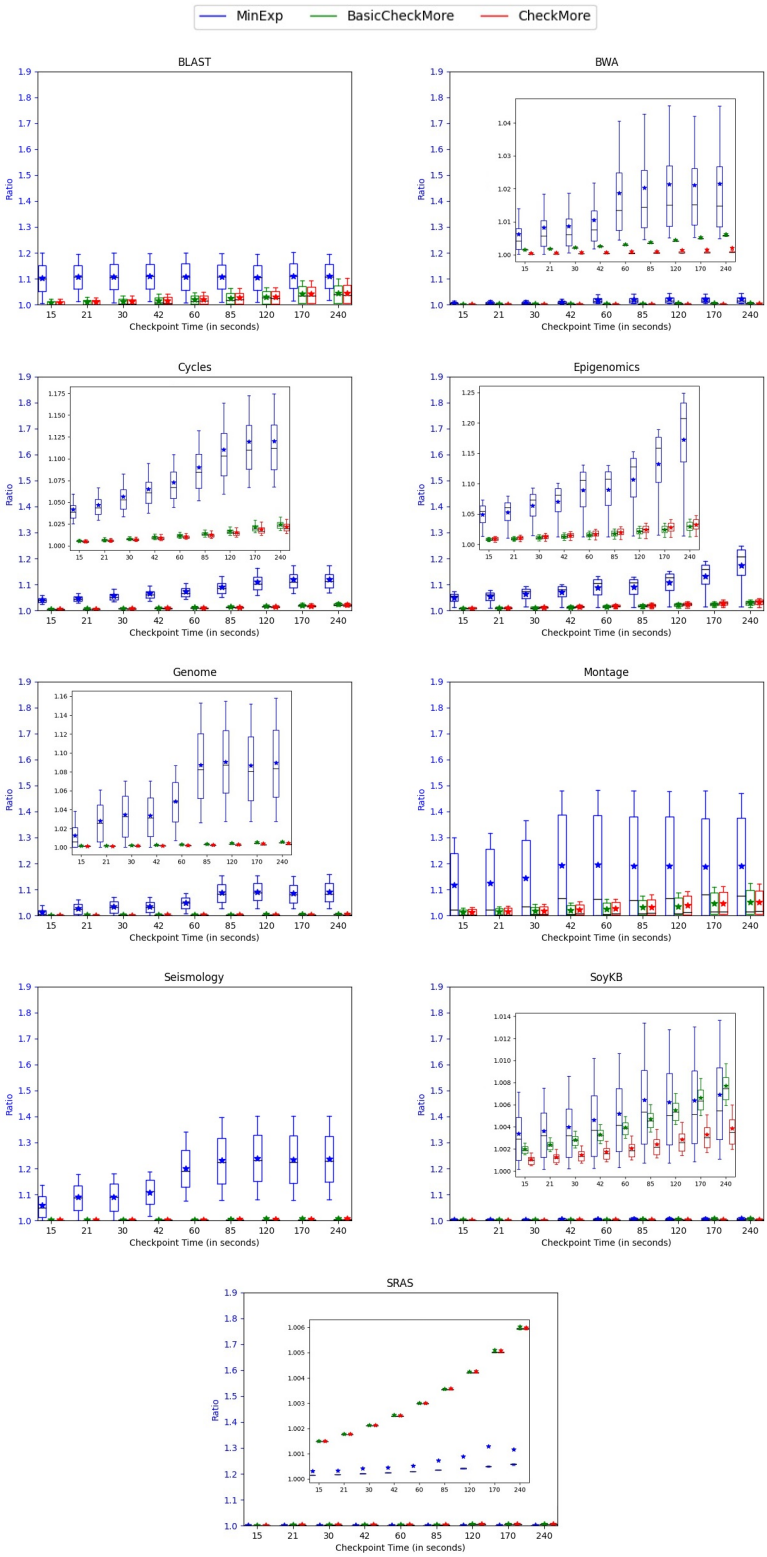

Figure 7: Impact of the checkpoint time (C) on the performance of the checkpointing strategies for different workflows.

decreases with C, and $\frac{C}{W} \approx \sqrt{\frac{C}{2\mu}}$ increases with C. Intuitively, the checkpoint time impacts the ratio in two ways. First, as C increases, we pay more for each checkpoint, which could lead to an increased ratio. Second, as we use $W_{YD} = \sqrt{\frac{2C}{p\lambda}}$ to determine the checkpointing period and hence the number of checkpoints, a task will become less safe when C increases, because it will be checkpointed less, and this could also increase the ratio.

For example, looking at GENOME under MINEXP, we can see a clear increase in the ratio when C increases from 15 to 21. This is because the typical number of checkpoints for the critical tasks (that affect the overall execution time the most) drops from 3 to 2, thus the time wasted due to a failure increases from 33% to 50%. As C increases from 60 to 85, the typical number of checkpoints of these tasks further drops from 2 to 1, making the waste per failure increase to 100%, and so the ratio also greatly increases. For values of C between 21 and 42, even if the number of checkpoints does not change, the ratio increases smoothly due to the increase in checkpoint time. The ratio of CHECKMORE, on the other hand, only increases slightly with the checkpoint time, which is, however, not visible in the figure due to the small values. Some other workflows, such as BWA, MONTAGE and SEISMOLOGY, also clearly illustrate these phenomena.

For the remaining workflows, we can again see the impact of these two factors or their combination on the ratio. For instance, as most failures in SRAS does not affect the overall execution time, the ratio of all strategies is only impacted by the checkpoint time. For BLAST under MINEXP, because most tasks are short and we have a single checkpoint to start with, the increase in checkpoint time is negligible compared to the waste induced by failures.

Impact of Individual MTBF (μ_{ind}). We evaluate the impact of individual processor's MTBF by varying it between 30 months and 40 years, and the results are shown in Figure 8. Intuitively, when μ_{ind} increases (or equivalently, the failure rate λ decreases), we would have fewer failures and expect the ratio to decrease. This is generally true for CHECKMORE but not always for MINEXP. To understand why, we refer again to the simplified approximation ratio $r \leq \left(\frac{X}{N_c} + Y\right)\left(\frac{2C}{W} + 1\right) + \frac{C}{W} + Z$, where X, Y and Z are barely affected by μ_{ind}. Here, when the number of failures decreases, $W_{YD} = \sqrt{\frac{2C}{p\lambda}}$ increases, so the number of checkpoints decreases and the time wasted for each failure increases. This could potentially lead to an increase in the ratio. To illustrate this compound effect, we again look at GENOME under MINEXP. When μ_{ind} goes from 2.5 to 3.5 years, the typical number of checkpoints for the critical tasks (that affect the overall execution time the most) drops from 3 to 2, which increases the waste per failure by around 50%. This together with the fact that MINEXP does not take into account the parallelism results in an increase in the ratio. When μ_{ind} goes from 3.5 to 7 years, the ratio decreases simply because we have fewer failures. As μ_{ind} continues to increase to 14 years, the number of checkpoints for the critical tasks further drops from 2 to 1. This increases the waste per failure to 100%, which again leads to an increase in the ratio. From this point on, the ratio will just decrease with μ_{ind}, again due to fewer failures. The same phenomenon can be observed for some other workflows, such as BWA, MONTAGE and SEISMOLOGY.

In yet some other workflows, the ratio simply decreases with μ_{ind}, such as

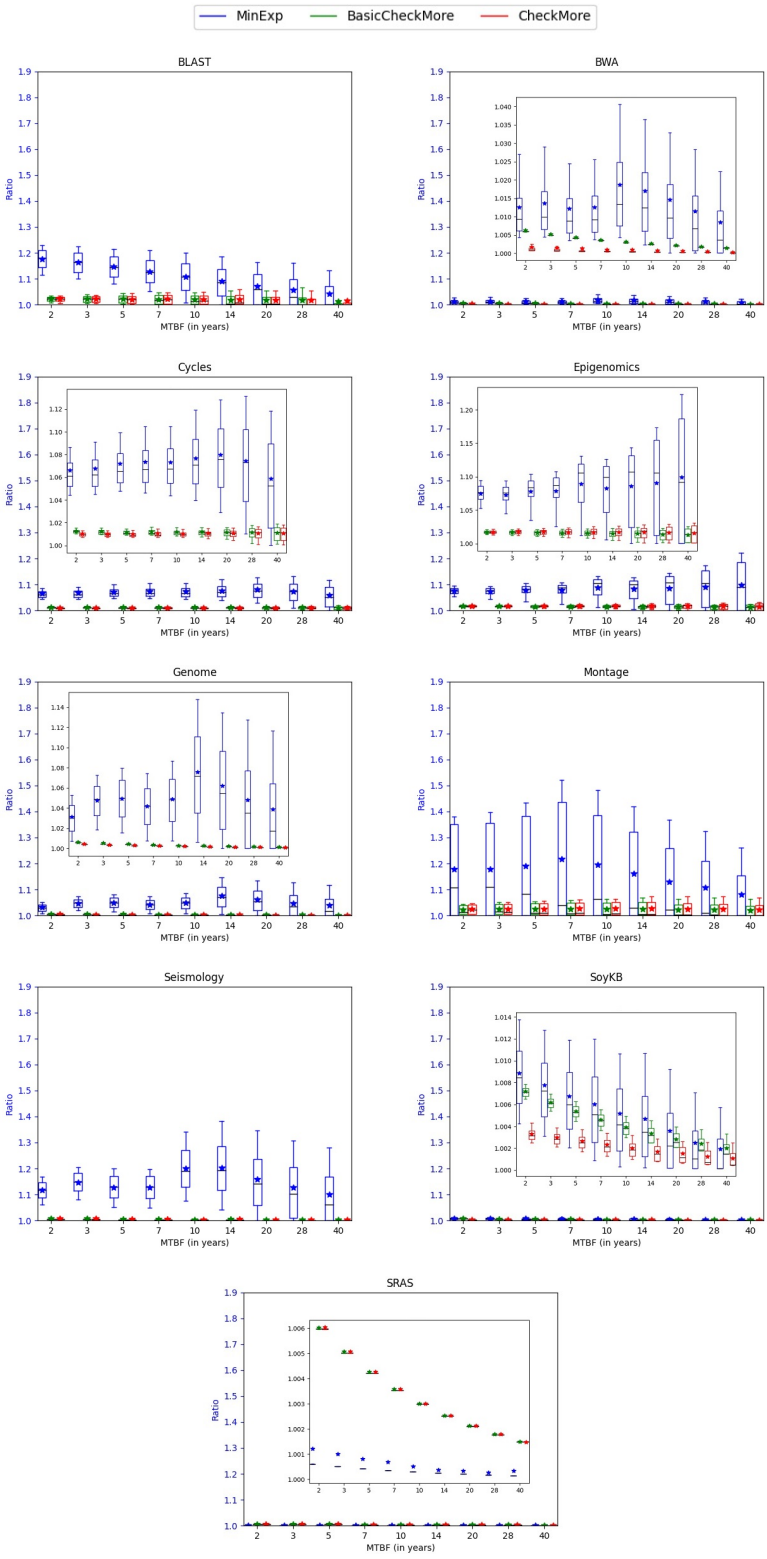

Figure 8: Impact of the individual MTBF (μ_{ind}) on the performance of the checkpointing strategies for different workflows.

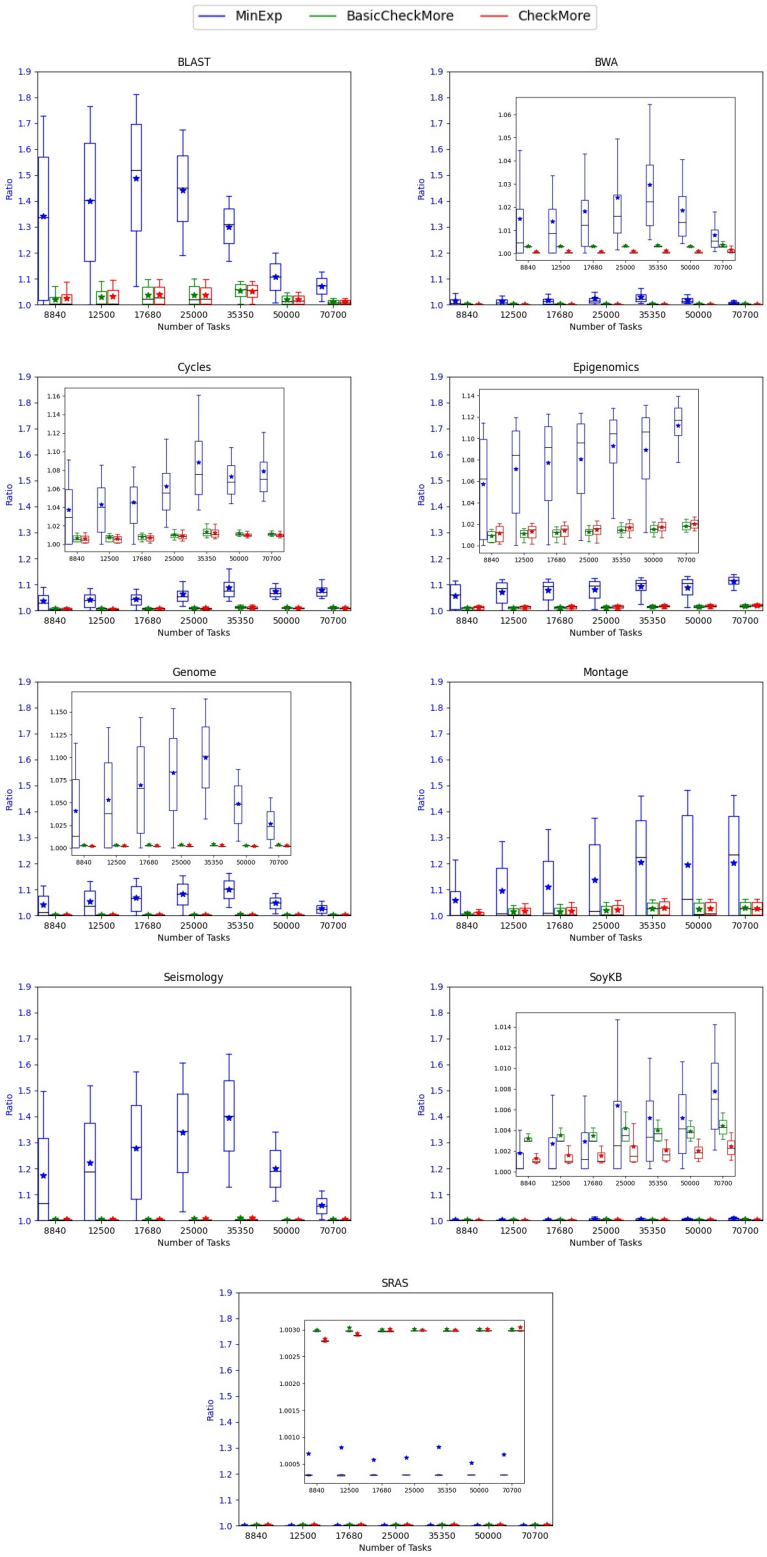

Figure 9: Impact of the number of tasks (n) on the performance of the checkpointing strategies for different workflows.

for BLAST and SRAS. For BLAST, even when μ_{ind} is small, we only checkpoint once, so the ratio decreases due to fewer failures. For SRAS, failures usually do not impact the overall execution time, so the decrease in ratio is mainly due to the decrease in the number of checkpoints.

Finally, it is worth noting that the ratio variance increases as μ_{ind} increases. This is because when there are only a few failures and the length of the segments is large, the failure location (inside the segments) will matter significantly, especially for MINEXP.

Impact of Number of Tasks (n). Finally, we study the impact of the number of tasks in the workflow, which is varied between 8800 and 70000, and the results are shown in Figure 9. Again, the ratio is impacted by the number of tasks in two different ways. First, when n increases, the width of the graph increases and so does Δ, and this would increase the ratio according to our analysis. Second, when n increases and m is fixed, the average number of tasks executed by each processor increases. This means that, if a failure occurs early in the execution, it is less likely to have a significant impact on the ratio, since multiple other tasks will be processed afterwards to balance the load, especially if the tasks are relatively independent.

These two phenomena are clearly observed in BLAST under MINEXP. This workflow mainly consists of a large batch of independent tasks. When n increases to 17680, which is approximately the number of processors ($m = 16384$), the ratio increases because Δ increases. After that, the ratio starts to decrease because $n > m$. In this case, when a failure strikes an early task, the subsequent tasks could be assigned to other processors to reduce the impact of the failure. Ultimately, if $n \gg m$, MINEXP would become more efficient. Indeed, since the tasks are almost independent and uni-processor tasks, list scheduling is able to dynamically balance the loads of different processors. Thus, minimizing the expected execution time of each individual task using MINEXP would be a good strategy for the overall execution time of the workflow.

For most of the other workflows, we can similarly observe the same up-and-down effect as a result of these two phenomena, except for SRAS, which is not impacted by the number of tasks. For this workflow, only a few key dominating tasks matter and their width remains well below the number of processors. Since these tasks form a small proportion of the total number of tasks, varying n does not significantly alter their chance of being hit by a failure, so the ratio remains close to 1.

Summary. Our experimental evaluation in this section has demonstrated that MINEXP is not resilient enough for checkpointing workflows, although it provides an optimal strategy for each individual task. On the other hand, CHECKMORE proves to be a very useful strategy, except for SRAS and SOYKB whose ratios are extremely close to 1. When varying the key parameters, the simulation results nicely corroborate our theoretical analysis. Furthermore, the easy-to-implement BASICCHECKMORE strategy always leads to ratios that are close to those of CHECKMORE, regardless of the parameters.

RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée
655 avenue de l'Europe Montbonnot
38334 Saint Ismier Cedex

Publisher
Inria
Domaine de Voluceau - Rocquencourt
BP 105 - 78153 Le Chesnay Cedex
inria.fr

ISSN 0249-6399