Toward an Optimal Online Checkpoint Solution under a Two-Level HPC Checkpoint Model

Sheng Di,* *Member, IEEE*; Yves Robert,* *Fellow, IEEE;*
Frédéric Vivien,* *Member, IEEE;* and Franck Cappello, *Member, IEEE,*

Abstract—The traditional single-level checkpointing method suffers from significant overhead on large-scale platforms. Hence, multilevel checkpointing protocols have been studied extensively in recent years. The multilevel checkpoint approach allows different levels of checkpoints to be set (each with different checkpoint overheads and recovery abilities), in order to further improve the fault tolerance performance of extreme-scale HPC applications. How to optimize the checkpoint intervals for each level, however, is an extremely difficult problem. In this paper, we construct an easy-to-use two-level checkpoint model. Checkpoint level 1 deals with errors with low checkpoint/recovery overheads such as transient memory errors, while checkpoint level 2 deals with hardware crashes such as node failures. Compared with previous optimization work, our new optimal checkpoint solution offers two improvements: (1) it is an online solution without requiring knowledge of the job length in advance, and (2) it shows that periodic patterns are optimal and determines the best pattern. We evaluate the proposed solution and compare it with the most up-to-date related approaches on an extreme-scale simulation testbed constructed based on a real HPC application execution. Simulation results show that our proposed solution outperforms other optimized solutions and can improve the performance significantly in some cases. Specifically, with the new solution the wall-clock time can be reduced by up to 25.3% over that of other state-of-the-art approaches. Finally, a brute-force comparison with all possible patterns shows that our solution is always within 1% of the best pattern in the experiments.

Index Terms—High-Performance Computing, Fault Tolerance, Optimization, Multilevel Checkpoint

1 Introduction

The execution scale of the largest high-performance computing (HPC) applications steadily increases over time, driven by the demand of solving the problems that were so far out of reach. In the near future, extreme-scale HPC environments with several millions of cores will be the norm for scientific simulations. An 83,000-processor supercomputer, for example, can match only 1% of the human brain [1].

In an extreme-scale HPC environment, however, users often experience a number of hardware crashes or transient errors during execution. Indeed, the most powerful existing computers suffer on the order of one failure per day [2]. Without fault-tolerance mechanisms protecting their execution, HPC applications (such as MPI programs) would have to be restarted from scratch after any kind of failure or error, leading to huge resource waste and resulting in extremely low performance.

The main approaches used to protect HPC executions against unexpected interruptions or failures are based on replication, either space-based or time-based. With space-based replication, a number of replicated processes (i.e., replicas) are launched in order to process the same work as the original processes. This model, called the *replica execution* model [3], [4], [5], [6], leads to a huge waste of resources. The reason is that the total number of cores or processes used to process an application will double or triple, depending on the fault tolerance capability demanded by users. With time-based replication, the replication happens only in the case of failures and takes the form of a re-execution. This is also known as the *checkpoint/recovery* model. It protects the execution by periodically setting checkpoints (i.e., saving the runtime memory or the key variable states into stable storage devices such as disks). In the case of failures, the saved states are read back, and the execution restarts from the snapshot of the application that they provide. Such a model avoids some of the resource waste created by the replica execution model (e.g., a doubling of required computational resources). Nevertheless, it suffers from several overhead sources: time required to write checkpoints to stable storage, time lost in recovery (read-back of checkpoints), and time elapsed between the last checkpoint and the time the failure struck (the corresponding computations must be re-executed). Furthermore, the time required to store the runtime memory state into the parallel file system increases with the application's execution scale [7], [8], and it is expected to be prohibitive in extreme-scale HPC environments.

Multilevel checkpointing [9], [10], [11], [12], [13], [14] has recently been proposed to resolve the huge checkpoint/recovery overhead issue. Multilevel checkpointing allows the use of different levels/types[1] of checkpoints during the execution. Obviously, a multilevel checkpoint model is much more flexible than the traditional single-level checkpoint model that uses only one type of storage device or mechanism (typically stable but also slow) to perform checkpoints. However, these different checkpoint levels correspond to different resilience and recovery capabilities and thus suffer from different checkpoint/recovery overheads.

• *Yves Robert and Frédéric Vivien are with Laboratoire LIP, CNRS, ENS Lyon, INRIA, and UCB Lyon, Lyon, France. Yves Robert is with the University of Tennessee Knoxville. Sheng Di and Franck Cappello are with the Mathematics and Computer Science (MCS) Division at Argonne National Laboratory, USA. [*: equal contributions are made by the first three authors.]*

1. We will use *checkpoint levels* and *checkpoint types* interchangeably throughout this paper.

One of the critical issues with the multilevel checkpoint model, then, is how to optimize it: What should be the checkpointing period on each different checkpoint level for a given HPC application? This problem is challenging because of the following factors: (1) the different levels protect the execution against different types of errors or failures, and each type of failure follows its own distribution; and (2) the different levels cannot be optimized independently because their relative performance impacts each other, so that the whole multilevel checkpointing system must be globally optimized.

In this paper, we formulate the research as a two-level (or two-type) checkpoint model, in comparison with the traditional single-level checkpoint model. The two-type checkpoint model is able to cover most of the checkpoint demands and is easy to use in practice. Many of the existing multilevel checkpoint toolkits provide two levels or types of checkpoints, such as the double in-memory checkpoint scheme [15], two-level incremental checkpoint recovery scheme [11], and Fault Tolerance Interface (FTI) [13]. On the other hand, based on the in-depth analysis in our previous work [7], the two-level model (e.g., one level is storing checkpoints onto the local memory/disk and the other one is using partner-copy technology) is already able to deal with a vast majority of failure cases, since the probability with many components being crashed simultaneously is relatively low. Under the two-level checkpoint model, we derive an optimal solution (i.e., the optimal checkpoint intervals for every checkpoint level) based on the checkpoint/recovery overheads and the failure probabilities for each level. The key contributions are listed below.

- We derive an online optimal solution (where the fault-free job length[2] is unknown a priori) to determine the optimal checkpoint intervals for the two checkpoint levels/types. Compared with our previous work [7], we rebuild the checkpoint model and remove some assumptions or conditions that had to be provided in the previous optimization work. For example, our previous work is an offline solution because it requires knowing the fault-free job length before the optimization. It also assumed that the failures would not occur during the checkpoint and that the total wall-clock length must be close to the fault-free job length. Such assumptions simplified the problem but also introduced skewness in the optimization such that the derived solution deviates from the best experimental solution significantly in some cases, as confirmed by our experiments. On the contrary, we do not rely on any first-order approximation in this paper.
- We theoretically prove that the optimal solution is to set the checkpoints periodically with the same checkpoint intervals at each particular level, and we analytically determine the optimal intervals. To confirm the optimality of our online solution, we also derive an offline optimal solution by assuming that the job length is known already. We prove that the level-1 (i.e., type-1) checkpoint intervals are the same for the online solution and offline solution.
- We evaluate our solution by using an elaborate HPC simulation testbed, which is constructed according to the real execution progress of an MPI program. We compare our new solution with the solutions of the most up-to-date related research such as [7]. The accuracy of the simulator is also confirmed by running a MPI scientific benchmark (called Heat Distribution) on a real cluster environment with up to 1024 cores. We highlight three significant findings: (1) our online optimal solution leads to exactly the same results with the offline optimal solution, which confirms the optimality of our online solution; (2) our online optimal solution always outperforms other optimized solutions and improves the performance significantly in some cases. Specifically, with the new solution, the wall-clock time can be reduced by up to 25.3% compared with that of other state-of-the-art approaches; and (3) the optimal solution derived in this paper is close to the best experimental solution obtained via brute-force search, with the difference (in wall-clock time) less than 1% in most cases.

The rest of the paper is organized as follows. In Section 2, we formulate the two-level checkpoint problem and propose a flexible pattern-based checkpoint model, in which any HPC execution can be split into multiple consecutive patterns, each containing a number of low-level checkpoint intervals (also called *chunks* in the paper). In Section 3, we compute the expected execution time for a pattern and show that it is minimized when all the chunks in the pattern have the same size. This result is key to deriving the optimal solution, which is presented in Section 4 for online scheduling and in Section 5 for offline scheduling. Although the whole derivation is technically involved, the solution is still simple to implement. In Section 6, we provide an in-depth analysis of how to use our derived solution in practice. In Section 7, we evaluate the proposed solution on an extreme-scale simulation environment and compare it with other state-of-the-art approaches. In Section 8, we discuss related work and highlight the differences with our new solution. In Section 9, we provide concluding remarks and directions for future work.

2 Problem Formulation

In this section, we formally present the problem under study. We consider a failure-prone platform where two types of faults can strike. Type-1 faults are independent and identically distributed (IID) and follow an exponential distribution of failure rate λ_1. Type-2 faults are IID and follow an exponential distribution of failure rate λ_2. Faults of different types are independent. Two types of checkpoints are possible.

- A type-2 checkpoint takes time C_2. Recovery from a type-2 checkpoint takes time R_2. A type-2 checkpoint enables recovery from both type-1 and type-2 faults.
- A type-1 checkpoint takes time C_1. Recovery from a type-1 checkpoint takes time R_1. A type-1 checkpoint enables recovery only from type-1 faults. In other words, type-1 checkpoints are destroyed by type-2 faults.

After a fault or failure[3] occurs, there is a downtime of length D followed by a recovery of the same type (e.g., a recovery from a type-2 checkpoint after a type-2 fault). In our analysis, for simplicity we assume that faults can strike during checkpoints but not during recoveries. Consider the situation in which a type-2 fault strikes during a type-1 recovery following a type-1 fault. Then the actual recovery should be from a type-2 checkpoint, rather than from a type-1 checkpoint. On the one hand, taking into account all the possible scenarios of failures striking during

2. *Job length* here refers to the fault-free productive length. For simplicity of presentation, we will interchangeably use different terms (such as *work size* and *productive length*) in the following text. They all mean job length.

3. *Fault* often refers to the transient error, whereas *failure* means a fail-stop error. We will use the two terms interchangeably in the following text, in that our generic model is suitable for both of them.

recoveries would lead to a complex analysis. On the other hand, the probability of a failure striking the system during a recovery is low because recoveries are expected to be faster than checkpoints[4]. Therefore, in the analysis we assume that faults or failures do not strike during recoveries, since we expect that this approximation will have only a limited impact. In fact, this expectation will be corroborated by the performance evaluation (Section 7) because in our simulations the failures can strike at any time, including during recoveries.

Intuitively, type-2 faults are more dramatic than type-1 faults because they delete all type-1 data. As a motivating example, consider a two-level memory system, with main memory and hard disks. Type-1 checkpoints are kept in main memory and enable recovery form soft errors [15], while type-2 checkpoints are kept in stable storage and enable recovery from situations where the content of the main memory is lost. With existing multilevel checkpoint tools (such as FTI [13]), a type of checkpoint can be set based on two modes: according to either the workload processed (i.e., the productive time) or the number of other types of checkpoints taken during the execution. For example, the type-2 checkpoints can be taken every 100 seconds (*mode-A*) or every five type-1 checkpoints (*mode-B*) in the execution. Mode-A checkpointing is also called *interval*-based checkpointing, and mode-B is also known as *pattern*-based checkpointing. Our checkpoint solution is suitable for both modes. In what follows, we will formulate the problem based on mode-B and extend it to mode-A later. We define a checkpoint pattern as follows (see Figure 1).

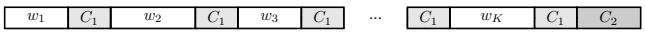


Fig. 1: Pattern including K chunks of respective sizes w_1, ..., w_K.

Definition 1. A pattern $\text{PATTERN}(K, W, w_1, ..., w_K)$ (or simply $\text{PATTERN}(K, W)$ when there is no ambiguity) is a sequence of K computational chunks. The ith chunk is of size w_i and $\sum_{i=1}^{K} w_i = W$. Each chunk is followed by a type-1 checkpoint. The last type-1 checkpoint is followed by a type-2 checkpoint.

Throughout the paper, we assume unit-speed execution, so we can speak of time or work size interchangeably. A key technical contribution of this paper (provided in Section 3) is to compute the expected time $\mathbb{E}(\text{PATTERN}(K, W, w_1, ..., w_K))$ to execute a given pattern $\text{PATTERN}(K, W, w_1, ..., w_K)$. Then we envision two scenarios, depending on whether the total fault-free job length is known a priori or not. Both scenarios use the value of $\mathbb{E}(\text{PATTERN}(K, W, w_1, ..., w_K))$ computed in Section 3.

2.1 Online scheduling

The first scenario, where the total job length is not known, applies to online scheduling problems. We can also think of jobs running a very long (even infinite) time on the platform. In this scenario, we seek the best pattern, namely, the one whose overhead is minimal. The *overhead* of a pattern is defined as follows.

4. Recovery/restart time is expected to be smaller than checkpoint time. Our experiments with the real-world simulation benchmark protected by FTI toolkits confirmed this point (as shown in Figure 2). Moreover, the difference between recovery time and checkpoint time would be even larger with data lossy compression techniques, because recent evaluation [16] indicates that the decompression time is generally only 1/5 of the compression time for an error-bounded lossy compressor.

Definition 2. The *overhead* of the pattern $\text{PATTERN}(K, W, w_1, ..., w_K)$ is the ratio

$$\text{OVERHEAD}(\text{PATTERN}(K, W, w_1, ..., w_K)) = \frac{\mathbb{E}(\text{PATTERN}(K, W, w_1, ..., w_K))}{W} - 1. \quad (1)$$

One main contribution of this paper is to provide an algorithm to compute the optimal pattern $\text{PATTERN}(K_{online}, \mathcal{W}_{online})$. First we compute the best pattern with a fixed number of chunks K, and then we show how to determine the optimal value K_{online} of K. While the derivation is technical, the approach is fully constructive and determines the total length of the pattern $\mathcal{W}_{online}$ and the size of each of the K_{online} chunks inside the pattern.

Determining the optimal pattern, and then repeatedly using it until job completion, is the optimal approach with exponential failure distributions and infinite jobs. Indeed, once a pattern is successfully executed, the optimal strategy is to re-execute the same pattern. This is because of the memoryless property of exponential distributions: the history of failures has no impact on the solution, so if a pattern is optimal at some point in time, it stays optimal later in the execution, because we have no further information about the amount of work still to be executed.

2.2 Offline scheduling

The second scenario assumes that the total work length $\mathcal{W}_{total}$ is known before execution, either exactly or through an accurate estimation based on benchmarking and sampling (this is the case for some scientific applications such as numerical linear algebra libraries [17]). The optimal strategy for this scenario is more complicated to derive. Indeed, after executing a first pattern $\text{PATTERN}(K_1, W_1)$, the remaining work to execute is $\mathcal{W}_{total} - W_1$, and the second pattern might have a different length W_2. We derive an optimal solution when we relax the problem and allow patterns to have a noninteger number of chunks. In this case, we show that all patterns have the same length, and we fully characterize the optimal solution.

For very long jobs (i.e., for large values of $\mathcal{W}_{total}$), the optimal online strategy (repeating the pattern $\text{PATTERN}(K_{online}, \mathcal{W}_{online})$) also can be used, at the price of a shorter (suboptimal) pattern at the end of the computation when $\mathcal{W}_{total}$ is not a multiple of $\mathcal{W}_{online}$. In Section 7, we experimentally compare both approaches.

3 Performance of Patterns

In this section, we analyze the expected execution time of a pattern and prove that the best patterns have same-size chunks.

3.1 Expected execution time of a pattern

We first determine the expectation $\mathbb{E}(\text{PATTERN}(K, W, w_1, ..., w_K))$ of the execution time for an arbitrary pattern composed of K chunks, where the ith chunk has size w_i, with $\sum_{i=1}^{K} w_i = W$. The derivation is lengthy and technical, and the reader may want to skip the proof. We need a few definitions before stating the main result. For convenience, main definitions are summarized in Table 1.

TABLE 1: Notations.

$\textsc{Pattern}(K, W)$	Pattern of size W, with K chunks
w_i	Size of i-th chunk in pattern
λ_1, C_1, R_1	Fault-rate, checkpoint time, recovery time for level-1 faults
λ_2, C_2, R_2	Fault-rate, checkpoint time, recovery time for level-2 faults
λ	Cumulated rate of faults: $\lambda = \lambda_1 + \lambda_2$
L	Fraction of level-2 faults: $L = \lambda_2/\lambda$
$\mathcal{R}$	Averaged overhead: $\mathcal{R} = D + (1 + \lambda_1 R_1 + \lambda_2 R_2)/\lambda$
$w_{opt}(K)$	Optimal chunk size for a pattern with K chunks

Definition 3. Let $\lambda = \lambda_1 + \lambda_2$, $L = \frac{\lambda_2}{\lambda}$ and $\mathcal{R} = \frac{1+\lambda_1 R_1+\lambda_2 R_2}{\lambda} + D$. For $1 \leq j \leq K$, let $e_j = e^{\lambda(w_j+C_1)} - 1$. Let $F_0 = 0$ and, for $1 \leq j \leq K$, let

$$F_j = \sum_{k=1}^{j} L^{k-1}\sigma_k(e_1, ..., e_j)$$

where σ_k is the symmetric function of order k.

In Defnition 3, λ is the cumulated failure rate, which includes both failure types, while L is the fraction of type-2 failures among all failures. Therefore, $0 \leq L \leq 1$, with $L = 0$ if there are no type-2 failures, and $L = 1$ if there are no type-1 failures. Finally, $\mathcal{R}$ is some averaged overhead, which simplifies to $\mathcal{R} = \frac{1}{\lambda_1} + D + R_1$ if there are no type-2 failures. Also, here is an example for the F_j's with $K = 3$: we have $\sigma_1(e_1, e_2, e_3) = e_1 + e_2 + e_3$, $\sigma_2(e_1, e_2, e_3) = e_1e_2 + e_1e_3 + e_2e_3$ and $\sigma_3(e_1, e_2, e_3) = e_1e_2e_3$. We obtain $F_1 = e_1$, $F_2 = e_1 + e_2 + L(e_1e_2) = F_1 + (1 + LF_1)e_2$ and $F_3 = e_1 + e_2 + e_3 + L(e_1e_2 + e_1e_3 + e_2e_3) + L^2(e_1e_2e_3) = F_2 + (1 + LF_2)e_3$.

Proposition 1.

$$\mathbb{E}(\textsc{Pattern}(K, W, w_1, ..., w_K)) = \mathcal{R}F_K + \mathcal{R}\left(1 + LF_K\right)\left(e^{\lambda C_2} - 1\right) \quad (2)$$

Proof. We introduce some simplified notation to ease the reading of the proof. Let $\mathbb{E}(\textsc{Pattern})$ denote $\mathbb{E}(\textsc{Pattern}(K, W, w_1, ..., w_K))$. Let $\mathbb{E}(w_1)$ be the expected time to successfully execute the first chunk w_1 and the following type-1 checkpoint C_1. For $i \in [2, K]$, let $\mathbb{E}(w_i)$ be the expected time to successfully execute chunk w_i and the following type-1 checkpoint, after the completion of the $(i-1)$th type-1 checkpoint. Let $\mathbb{E}(C_2)$ be the expected time to successfully execute the final type-2 checkpoint in the pattern, after the Kth (and last) type-1 checkpoint. By the linearity of expectations,

$$\mathbb{E}(\textsc{Pattern}) = \left(\sum_{i=1}^{K} \mathbb{E}(w_i)\right) + \mathbb{E}(C_2). \quad (3)$$

Computing $\mathbb{E}(w_i)$

Consider the beginning of the execution of the ith chunk w_i, right after the successful completion of the $(i-1)$th type-1 checkpoint. Let X_1 be the time at which the first type-1 fault strikes the system. X_1 is thus a random variable following an exponential distribution of failure rate parameter λ_1. Let X_2 be the time at which the first type-2 fault strikes the system. X_2 is thus a random variable following an exponential distribution of parameter λ_2.

Let X be the time at which the first fault (of any type) strikes the system. We consider three cases:

1) No fault strikes the system before time $w_i + C_1$ (expected completion time $\mathcal{E}_0$)
2) A fault strikes the system before time $w_i + C_1$, and the first fault to strike the system is a type-1 fault (expected completion time $\mathcal{E}_1$). In this case, we can recover from the last type-1 checkpoint and re-execute only the current chunk. This holds true even if $i = 1$: for the first chunk of the pattern the most recent checkpoint is a type-2 checkpoint, but it was preceded by a type-1 checkpoint, which is cheaper to recover from.
3) A fault strikes the system before time $w_i + C_1$, and the first fault to strike the system is a type-2 fault (expected completion time $\mathcal{E}_2$). In this case, we have to recover from the last type-2 checkpoint, and we have to re-execute all the chunks of the pattern, together with their type-1 checkpoints, starting again from w_1.

By the law of total probability, $\mathbb{E}(w_i) = \mathcal{E}_0 + \mathcal{E}_1 + \mathcal{E}_2$. We compute $\mathcal{E}_0$, $\mathcal{E}_1$, and $\mathcal{E}_2$ as follows:

Case 1. Without fault, we have $X = \min\{X_1, X_2\} \geq w_i + C_1$. Because the minimum of two exponential distributions is an exponential distribution whose parameter is the sum of the parameters, this happens with probability $e^{-(\lambda_1+\lambda_2)(w_i+C_1)}$. Hence, $\mathcal{E}_0 = e^{-(\lambda_1+\lambda_2)(w_i+C_1)}(w_i + C_1)$.

Case 2. When a type-1 fault strikes first, we have $X_1 < w_i + C_1$ and $X_1 \leq X_2$. Then the system performs a type-1 recovery and retries processing the same amount of work. The re-execution has expectation $\mathbb{E}(w_i)$ by the memoryless property. Hence, we have the following (see the extended version [18] for intermediate steps in the computation of $\mathcal{E}_1$):

$$\begin{aligned}\mathcal{E}_1 &= \int_{X_1=0}^{X_1=w_i+C_1} \int_{X_2=X_1}^{X_2=+\infty} (X_1 + D + R_1 + \mathbb{E}(w_i)) \\ &\qquad \times \lambda_1 e^{-\lambda_1 X_1} \lambda_2 e^{-\lambda_2 X_2} dX_2 dX_1 \\ &= \int_{X_1=0}^{X_1=w_i+C_1} (X_1 + D + R_1 + \mathbb{E}(w_i))\lambda_1 e^{-(\lambda_1+\lambda_2)X_1} dX_1 \\ &= (D + R_1 + \mathbb{E}(w_i)) \int_{X_1=0}^{X_1=w_i+C_1} \lambda_1 e^{-(\lambda_1+\lambda_2)X_1} dX_1 \\ &\quad + \int_{X_1=0}^{X_1=w_i+C_1} X_1 \lambda_1 e^{-(\lambda_1+\lambda_2)X_1} dX_1\end{aligned}$$

We derive that

$$\begin{aligned}\mathcal{E}_1 = (D + R_1 + \mathbb{E}(w_i))\tfrac{\lambda_1}{\lambda_1+\lambda_2}\left(1 - e^{-(\lambda_1+\lambda_2)(w_i+C_1)}\right) \\ - \tfrac{\lambda_1}{\lambda_1+\lambda_2}(w_i + C_1)e^{-(\lambda_1+\lambda_2)(w_i+C_1)} \\ + \tfrac{\lambda_1}{(\lambda_1+\lambda_2)^2}\left(1 - e^{-(\lambda_1+\lambda_2)(w_i+C_1)}\right).\end{aligned}$$

Case 3. When a type-2 fault strikes first, we have $X_2 < w_i + C_1$ and $X_2 \leq X_1$. Then we have to recover from the last type-2 checkpoint and re-execute all chunks from w_1 up to w_i. Hence, we have the following.

$$\begin{aligned}\mathcal{E}_2 &= \int_{X_2=0}^{X_2=w_i+C_1} \int_{X_1=X_2}^{X_1=+\infty} (X_2 + D + R_2 + \mathbb{E}(w_1) + ... + \mathbb{E}(w_i)) \\ &= (D + R_2 + \mathbb{E}(w_1) + ... + \mathbb{E}(w_i)) \int_{X_2=0}^{X_2=w_i+C_1} \lambda_2 e^{-(\lambda_1+\lambda_2)X_2} dX_2 \\ &\quad + \int_{X_2=0}^{X_2=w_i+C_1} X_2 \lambda_2 e^{-(\lambda_1+\lambda_2)X_2} dX_2\end{aligned}$$

Transposing the integral computations of Case 2, we obtain the

following.

$$\begin{aligned}\mathcal{E}_2 = \quad & (D + R_2 + \mathbb{E}(w_1) + ... + \mathbb{E}(w_i))\frac{\lambda_2}{\lambda_1+\lambda_2} \\ & \times \left(1 - e^{-(\lambda_1+\lambda_2)(w_i+C_1)}\right) \\ & -\frac{\lambda_2}{\lambda_1+\lambda_2}(w_i + C_1)e^{-(\lambda_1+\lambda_2)(w_i+C_1)} \\ & +\frac{\lambda_2}{(\lambda_1+\lambda_2)^2}\left(1 - e^{-(\lambda_1+\lambda_2)(w_i+C_1)}\right)\end{aligned}$$

Since $\mathbb{E}(w_i) = \mathcal{E}_0 + \mathcal{E}_1 + \mathcal{E}_2$, we derive:

$$\begin{aligned}\mathbb{E}(w_i) \quad & = e^{-(\lambda_1+\lambda_2)(w_i+C_1)}(w_i + C_1) \\ & +(D + R_1 + \mathbb{E}(w_i))\frac{\lambda_1}{\lambda_1+\lambda_2} \\ & \qquad \times \left(1 - e^{-(\lambda_1+\lambda_2)(w_i+C_1)}\right) \\ & -\frac{\lambda_1}{\lambda_1+\lambda_2}(w_i + C_1)e^{-(\lambda_1+\lambda_2)(w_i+C_1)} \\ & +\frac{\lambda_1}{(\lambda_1+\lambda_2)^2}\left(1 - e^{-(\lambda_1+\lambda_2)(w_i+C_1)}\right) \\ & +(D + R_2 + \mathbb{E}(w_1) + ... + \mathbb{E}(w_i))\frac{\lambda_2}{\lambda_1+\lambda_2} \\ & \qquad \times \left(1 - e^{-(\lambda_1+\lambda_2)(w_i+C_1)}\right) \\ & -\frac{\lambda_2}{\lambda_1+\lambda_2}(w_i + C_1)e^{-(\lambda_1+\lambda_2)(w_i+C_1)} \\ & +\frac{\lambda_2}{(\lambda_1+\lambda_2)^2}\left(1 - e^{-(\lambda_1+\lambda_2)(w_i+C_1)}\right) \\ & = \left(\frac{1+\lambda_1 R_1+\lambda_2(R_2+\mathbb{E}(w_1)+...+\mathbb{E}(w_{i-1}))}{\lambda_1+\lambda_2} + D + \mathbb{E}(w_i)\right) \\ & \qquad \times \left(1 - e^{-(\lambda_1+\lambda_2)(w_i+C_1)}\right)\end{aligned}$$

This leads to

$$\begin{aligned}\mathbb{E}(w_i) = & \left(\frac{1 + \lambda_1 R_1 + \lambda_2(R_2 + \sum_{j=1}^{i-1}\mathbb{E}(w_j))}{\lambda_1 + \lambda_2} + D\right) \\ & \times \left(e^{(\lambda_1+\lambda_2)(w_i+C_1)} - 1\right).\end{aligned}$$

With the notations of Definition 3, we obtain

$$\mathbb{E}(w_i) = \left(\mathcal{R} + L\sum_{j=1}^{i-1}\mathbb{E}(w_j)\right)e_i. \tag{4}$$

Simplifying $\mathbb{E}(w_i)$

Lemma 1.

$$\mathbb{E}(w_i) = \mathcal{R}(1 + LF_{i-1})e_i \quad \textit{for } 1 \le i \le K \tag{5}$$

Proof. Recall the notation given in Definition 3. We prove the result by induction. It holds for $i = 1$ because Equation (5) leads to $\mathbb{E}(w_1) = \mathcal{R}e_1$, since $F_0 = 0$. Assume that the result holds up to rank i, and consider $\mathbb{E}(w_{i+1})$. From Equation (4), we have

$$\begin{aligned}\mathbb{E}(w_{i+1}) &= \left(\mathcal{R} + L\sum_{j=1}^{i}\mathbb{E}(w_j)\right)e_{i+1} \\ &= \mathcal{R}\left(1 + L\sum_{j=1}^{i}(1 + LF_{j-1})e_j\right)e_{i+1},\end{aligned} \tag{6}$$

the last equality coming from the induction hypothesis. We further detail the term inside the summation:

$$\begin{aligned}&(1 + LF_{j-1})e_j \\ &= e_j + Le_j\textstyle\sum_{k=1}^{j-1}L^{k-1}\sigma_k(e_1, ..., e_{j-1}) \\ &= e_j + \textstyle\sum_{k=1}^{j-1}L^k\sigma_k(e_1, ..., e_{j-1})e_j \\ &= e_j + \textstyle\sum_{k=1}^{j-1}L^k\left(\sigma_{k+1}(e_1, ..., e_j) - \sigma_{k+1}(e_1, ..., e_{j-1})\right) \\ &= e_j + \textstyle\sum_{k=1}^{j-1}L^k\sigma_{k+1}(e_1, ..., e_j) \\ &\quad - \textstyle\sum_{k=1}^{j-2}L^k\sigma_{k+1}(e_1, ..., e_{j-1})\end{aligned}$$

Plugging back into Equation (6), we obtain the following.

$$\begin{aligned}&\mathbb{E}(w_{i+1}) \\ &= \mathcal{R}\Big(1+L\textstyle\sum_{j=1}^{i}(1+LF_{j-1})e_j\Big)\,e_{i+1} \\ &= \mathcal{R}\Big(1+L\textstyle\sum_{j=1}^{i}\Big(e_j + \Big(\sum_{k=1}^{j-1}L^k\sigma_{k+1}(e_1, ..., e_j)\Big) \\ &\qquad - \Big(\sum_{k=1}^{j-2}L^k\sigma_{k+1}(e_1, ..., e_{j-1})\Big)\Big)\Big)\,e_{i+1} \\ &= \mathcal{R}\Big(1+L\sigma_1(e_1, ..., e_i)+\textstyle\sum_{j=1}^{i}\sum_{k=1}^{j-1}L^{k+1}\sigma_{k+1}(e_1, ..., e_j) \\ &\qquad - \textstyle\sum_{j=1}^{i}\sum_{k=1}^{j-2}L^{k+1}\sigma_{k+1}(e_1, ..., e_{j-1})\Big)\,e_{i+1} \\ &= \mathcal{R}\Big(1+L\sigma_1(e_1, ..., e_i)+\textstyle\sum_{j=2}^{i}\sum_{k=1}^{j-1}L^{k+1}\sigma_{k+1}(e_1, ..., e_j) \\ &\qquad - \textstyle\sum_{j=3}^{i}\sum_{k=1}^{j-2}L^{k+1}\sigma_{k+1}(e_1, ..., e_{j-1})\Big)\,e_{i+1} \\ &= \mathcal{R}\Big(1+L\sigma_1(e_1, ..., e_i)+\textstyle\sum_{j=2}^{i}\sum_{k=1}^{j-1}L^{k+1}\sigma_{k+1}(e_1, ..., e_j) \\ &\qquad - \textstyle\sum_{j=2}^{i-1}\sum_{k=1}^{j-1}L^{k+1}\sigma_{k+1}(e_1, ..., e_j)\Big)\,e_{i+1} \\ &= \mathcal{R}\Big(1+L\sigma_1(e_1, ..., e_i)+\textstyle\sum_{k=1}^{i-1}L^{k+1}\sigma_{k+1}(e_1, ..., e_i)\Big)\,e_{i+1} \\ &= \mathcal{R}\Big(1 + \textstyle\sum_{k=0}^{i-1}L^{k+1}\sigma_{k+1}(e_1, ..., e_i)\Big)\,e_{i+1} \\ &= \mathcal{R}\Big(1+L\textstyle\sum_{k=1}^{i}L^{k-1}\sigma_k(e_1, ..., e_i)\Big)\,e_{i+1} \\ &= \mathcal{R}(1+LF_i)\,e_{i+1}\end{aligned}$$

□

Using Equation (5), we simplify the expression $\sum_{i=1}^{j}\mathbb{E}(w_i)$. Indeed, comparing with Equation (4), we obtain

$$\sum_{j=1}^{i}\mathbb{E}(w_j) = \mathcal{R}F_i \quad \text{for } 0 \le i \le K. \tag{7}$$

Computation of $\mathbb{E}(C_2)$

We compute $\mathbb{E}(C_2)$, the expected time to complete the final type-2 checkpoint of the pattern, similar to the way we computed $\mathbb{E}(w_i)$ before. We consider three cases:

1) No fault strikes the system before time C_2. Then the expectation is $\mathcal{E}_0 = e^{-(\lambda C_2)}C_2$.
2) A fault strikes the system before time C_2, and the first fault to strike the system is a type-1 fault. Then we perform a type-1 recovery and re-execute C_2. The expectation is
$$\begin{aligned}\mathcal{E}_1 = \int_{X_1=0}^{X_1=C_2}\int_{X_2=X_1}^{X_2=+\infty}&(X_1 + D + R_1 + \mathbb{E}(C_2)) \\ &\times\ \lambda_1 e^{-\lambda_1 X_1}\lambda_2 e^{-\lambda_2 X_2}dX_2dX_1.\end{aligned}$$
3) A fault strikes the system before time C_2, and the first fault to strike the system is a type-2 fault: $X_2 < C_2$ and $X_2 \le X_1$. Then we recover from the last type-2 checkpoint and re-execute the whole pattern. The expectation is
$$\begin{aligned}\mathcal{E}_2 = \int_{X_2=0}^{X_2=C_2}\int_{X_1=X_2}^{X_1=+\infty}&(X_2+D+R_2+\sum_{i=1}^{K}\mathbb{E}(w_i)+\mathbb{E}(C_2)) \\ &\times\ \lambda_2 e^{-\lambda_2 X_2}\lambda_1 e^{-\lambda_1 X_1}dX_1dX_2.\end{aligned}$$

We compute the integrals as before and use $\sum_{i=1}^{K}\mathbb{E}(w_i) = \mathcal{R}F_K$ to derive the final result.

$$\mathbb{E}(C_2) = \mathcal{R}\left(1 + LF_K\right)\left(e^{\lambda C_2} - 1\right) \tag{8}$$

We have from Equations (7) and (8) that

$$\sum_{i=1}^{K}\mathbb{E}(w_i) + \mathbb{E}(C_2) = \mathcal{R}F_K + \mathcal{R}\left(1 + LF_K\right)\left(e^{\lambda C_2} - 1\right),$$

which concludes the proof of Proposition 1. □

3.2 Best patterns have equal-size chunks

Proposition 1 enables us to prove a key result: when both the total size W and the number of chunks K of a pattern are known, then the expectation of the execution time $\mathbb{E}(\text{PATTERN}(K, W, w_1, ..., w_K))$ is minimized when all chunks have same size W/K. Note that the overhead of the pattern is also minimized in that case, because W is given.

Proposition 2. *The expectation* $\mathbb{E}(\text{PATTERN}(K, W, w_1, ..., w_K))$ *of a pattern with* K *chunks and total size* $W = \sum_{i=1}^{K} w_i$ *is minimized when* $w_i = \frac{W}{K}$ *(same-size chunks). Letting* $\mathbb{E}(\text{PATTERN}(K, W))$ *denote this minimal value, we have*

$$\mathbb{E}(\text{PATTERN}(K, W)) = \alpha + \frac{\beta}{L}(1 + L(e^{\lambda(\frac{W}{K}+C_1)} - 1))^K, \quad (9)$$

where $\alpha = \mathcal{R}(e^{\lambda C_2} - 1) - \frac{\beta}{L}$ *and* $\beta = \mathcal{R}(1 + L(e^{\lambda C_2} - 1))$.

Equation (9) simplifies when $\lambda_2 = C_2 = 0$, which corresponds to type-1 failures only. In that case, $\mathbb{E}(\text{PATTERN}(K, W)) = K(\frac{1}{\lambda_1} + D + R_1)(e^{\lambda \frac{W}{K}+C_1} - 1)$, in accordance with [19]. Similarly when $\lambda_1 = C_1 = 0$ (only type-2 failures), the pattern reduces to a single chunk of length W and $\mathbb{E}(\text{PATTERN}(K, W)) = (\frac{1}{\lambda_2} + D + R_2)(e^{\lambda(W+C_2)} - 1)$.

Proof. Consider a pattern whose number of chunks K and total size $W = \sum_{i=1}^{K} w_i$ are given. From Equation (2), the expected execution time is minimal when F_K is minimal. Recall the definition of F_j: $F_j = \sum_{k=1}^{j} L^{k-1}\sigma_k(e_1, ..., e_j)$, where $e_i = e^{\lambda(w_j+C_1)} - 1$. Note that F_j is a symmetric function of e_1, ..., e_j. Consider any two indices i and j between 1 and K, and fix the sum $w_i + w_j = \omega$ as well as all the w_k's except w_i and w_j. Then we can rewrite F_K as a sole function of w_i. Then there exist two constants γ_1 and γ_2, with $\gamma_1 > 0$, such that

$$F_K(w_i) = \gamma_1 e^{\lambda(w_i+C_1)} + \gamma_1 e^{\lambda(\omega-w_i+C_1)} + \gamma_2.$$

By convexity, $F_K(w_i)$ is minimal when $w_i = w_j = \frac{\omega}{2}$. By induction, F_K is minimized when all the w_j's are equal.

We now compute the value of F_K when all chunks have same size $w = \frac{W}{K}$. We show by induction on j that

$$F_j = \frac{1}{L}\left(\left(1 + L\left(e^{\lambda(w+C_1)} - 1\right)\right)^j - 1\right), \quad (10)$$

when $w_1 = ... = w_K = w$. For $j = 1$, Equation (10) is $F_1 = e^{\lambda(w+C_1)} - 1$, which is the definition of F_1 when $w_1 = w$. Assume that Equation (10) holds up to some index i included. From Equations (5) and (7) we have $F_i = \sum_{j=1}^{i} e_j(1 + LF_{j-1})$. Then

$$\begin{aligned}
F_{i+1} &= \textstyle\sum_{j=1}^{i+1} e_j(1 + LF_{j-1}) \\
&= \textstyle\sum_{j=1}^{i+1} \left(e^{\lambda(w+C_1)}-1\right)\left(1+L\frac{1}{L}\left(1+L\left(e^{\lambda(w+C_1)}-1\right)\right)^{j-1}-1\right) \\
&= \textstyle\sum_{j=1}^{i+1} \left(e^{\lambda(w+C_1)}-1\right)\left(1+L\left(e^{\lambda(w+C_1)}-1\right)\right)^{j-1} \\
&= \textstyle\sum_{j=1}^{i+1} \frac{\left(-1+1+L\left(e^{\lambda(w+C_1)}-1\right)\right)}{L}\left(1+L\left(e^{\lambda(w+C_1)}-1\right)\right)^{j-1} \\
&= -\frac{1}{L}\textstyle\sum_{j=1}^{i+1}\left(1+L\left(e^{\lambda(w+C_1)}-1\right)\right)^{j-1} \\
&\quad +\frac{1}{L}\textstyle\sum_{j=1}^{i+1}\left(1+L\left(e^{\lambda(w+C_1)}-1\right)\right)^{j} \\
&= \frac{1}{L}\left(\left(1+L\left(e^{\lambda(w+C_1)}-1\right)\right)^{i+1}-1\right),
\end{aligned}$$

which proves the result for F_K. Plugging this value into Equation (2) leads to the result. □

We give an example to describe how to compute the expected wall-clock time by Proposition 2 (or Equation (9)). Suppose the checkpoint/recovery overheads on the two levels are C_1=R_1=20 seconds and C_2=R_2=50 seconds respectively. The failure rates on the two levels are 24/86,400=2.78×10^{-4} and 4/86,400=4.63×10^{-5} respectively, which means that there are 24 (4) failures occurring at level 1 (level 2) per day. The downtime is set to 0 for simplicity. Consider a pattern with K=4 same-size chunks and the workload size W is 1,472 seconds, then the chunk size is equal to 368 seconds in length, and the expected wall-clock length can be computed as 1773.2 seconds based on Equation (9).

4 ONLINE SCHEDULING

In this section, we characterize the optimal pattern for jobs whose length is not known in advance. We first determine the optimal pattern when the number K of chunks is given, and we then derive the best value of K.

4.1 Optimal pattern with given number K of chunks

Proposition 3. *When the number of chunks,* K*, is given, the optimal pattern has same-size chunks* $w_i = w_{opt}(K)$*, where* $w_{opt}(K)$ *is the unique solution of the equation*

$$\begin{aligned}
&\beta\lambda K w e^{\lambda(w+C_1)}(1 + L(e^{\lambda(w+C_1)} - 1))^{K-1} \\
&\quad = \alpha + \tfrac{\beta}{L}(1 + L(e^{\lambda(w+C_1)} - 1))^K
\end{aligned} \quad (11)$$

The pattern overhead is then

$$\text{OVERHEAD}(K) = \frac{\alpha + \frac{\beta}{L}(1+L(e^{\lambda(w_{opt}(K)+C_1)} - 1))^K}{K w_{opt}(K)} - 1.$$

Proof. Consider a pattern whose number of chunks K is given. If the total amount of work W is given, Proposition 2 shows that the overhead of the pattern is minimal when chunks have same-size W/K. But what is the optimal pattern size $W_{opt}(K)$ for a pattern with K chunks? Or equivalently, what is the optimal size $w_{opt}(K)$ of the chunks for a pattern with K chunks (and then the total size of the pattern will be $W_{opt}(K) = Kw_{opt}(K)$)?

Consider a pattern with K chunks of same size w. We express its overhead as a function $f(w)$ of the chunk size w. Using Proposition 2 (with the constants α and β defined there), we have

$$f(w) = \frac{\alpha + \frac{\beta}{L}\left(1 + L\left(e^{\lambda(w+C_1)} - 1\right)\right)^K}{Kw} - 1. \quad (12)$$

To simplify the derivation, let $E(w) = e^{\lambda(w+C_1)}$ and $N(w) = 1 + L(E(w) - 1)$. Note that $E(w) \geq 1$ and $N(w) \geq 1$ for all $w \geq 0$. We have $f(w) = \frac{\alpha+\frac{\beta}{L}(N(w))^K}{Kw} - 1$. Differentiating, we obtain that $f'(w) = 0$ if and only if w is a solution of the equation

$$\beta\lambda K w E(w)(N(w))^{K-1} = \alpha + \frac{\beta}{L}(N(w))^K, \quad (13)$$

which is identical to Equation (11).

To conclude the proof, we show that Equation (13) has a unique positive solution $w_{opt}(K)$ for any given value of K. Let $g(w) = \beta\lambda K w E(w)(N(w))^{K-1} - \alpha - \frac{\beta}{L}(N(w))^K$. Differentiating again, we get $g'(w) = \beta\lambda^2 K w E(w)(N(w))^{K-2}(N(w) + (K-1)LE(w))$, which is always positive. Hence $g(w)$ is increasing. We have $g(0) = -\alpha - \frac{\beta}{L}(N(0))^K$ and $N(0) \geq 1$; hence $g(0) \leq -(\alpha + \frac{\beta}{L}) = -\mathcal{R}(e^{\lambda C_2} - 1) < 0$. Furthermore,

we easily see that $\lim_{w\to+\infty} g(w) = +\infty$. Hence $g(w)$ has a unique positive zero, which shows that Equation (11) has a unique minimum. □

Remark 1. Equation (11) simplifies when $K = 1$ and $\lambda_2 = C_2 = 0$, which corresponds to the optimal single-chunk pattern for type-1 failures of rate $\lambda = \lambda_1$. In that case, $L = 1$, $\alpha = -\beta$. After some easy transformations, we obtain

$$(\lambda w_{opt}(1) - 1)e^{\lambda w_{opt}(1)-1} = \mathbb{L}(-\lambda C_1 - 1). \qquad (14)$$

Here $\mathbb{L}$ is the Lambert function, defined as $\mathbb{L}(z) = x$ if $xe^x = z$. We retrieve a result of [19] that, when approximated to the first order, is similar to a result of [20].

4.2 Optimal pattern

In this section we show how to determine the number K_{online} of chunks in the optimal pattern. From Proposition 3 we know that each chunk will have length $w_{opt}(K_{online})$ and that the total length of the optimal pattern will be $\mathcal{W}_{online} = K_{online} \times w_{opt}(K_{online})$.

Theorem 1. *The number of chunks K_{online} in the optimal pattern is either $\max\{1, \lfloor K^* \rfloor\}$ or $\lceil K^* \rceil$ (whichever leads to the pattern with smallest overhead), where K^* is the unique solution of the equation*

$$N(w^*)\ln(N(w^*)) = \lambda L w^* E(w^*) \qquad (15)$$

Here $w^* = w_{opt}(K^*)$ is the unique solution of Equation (11) *when $K = K^*$.*

Proof. From Proposition 3, the overhead of the optimal pattern with K chunks is

$$\textsc{Overhead}(K) = \frac{\alpha+\frac{\beta}{L}(1+L(e^{\lambda w_{opt}(K)+C_1}-1))^K}{K w_{opt}(K)} - 1,$$

where $w_{opt}(K)$ is the unique solution of Equation (11). We note that we can still define $w_{opt}(K)$ as the unique solution of Equation (11) for arbitrary values of K, not just integer ones. This allows us to define $\textsc{Overhead}(K)$ for (positive) real values of K. We study this function $\textsc{Overhead}(K)$ and will show that it has a unique minimum K^*, which is the unique solution of Equation (15).

To simplify notations, let $f(K) = \textsc{Overhead}(K)$ for $K > 0$. To differentiate $f(K)$, we need to differentiate $w_{opt}(K)$ as a function of K. We know $w_{opt}(K)$ only implicitly, through Equation (11), but we can differentiate both sides of this equation to determine $w'_{opt}(K)$. We will compute $w'_{opt}(K)$ later when proving existence and unicity of the solution of Equation (15). We first differentiate $f(K)$. We have

$$\begin{cases} (E(w_{opt}(K)))' = \lambda E(w_{opt}(K))w'_{opt}(K) \\ (N(w_{opt}(K)))' = L(E(w_{opt}(K)))' \\ (N^K(w_{opt}(K)))' = (e^{K\ln(N(w_{opt}(K)))})' \\ \quad = (\ln(N(w_{opt}(K))) + K\frac{(N(w_{opt}(K)))'}{N(w_{opt}(K))})N^K(w_{opt}(K)) \end{cases}$$

from which we derive

$$f'(K) = \frac{1}{K^2(w_{opt}(K))^2}\Big(Kw_{opt}(K)\tfrac{\beta}{L}(N^K(w_{opt}(K)))' - (\alpha+\tfrac{\beta}{L}N^K(w_{opt}(K)))(Kw'_{opt}(K)+w_{opt}(K))\Big).$$

The numerator of $f'(K)$ is of the form $Aw'_{opt}(K) + Bw_{opt}(K)$, where

$$\begin{aligned} A = \;& Kw_{opt}(K)\tfrac{\beta}{L}N^K(w_{opt}(K))\tfrac{KL\lambda E(w_{opt}(K))}{N(w_{opt}(K))} \\ & -K(\alpha+\tfrac{\beta}{L}N^K(w_{opt}(K))) \\ B = \;& K\tfrac{\beta}{L}N^K(w_{opt}(K))\ln(N(w_{opt}(K))) \\ & -(\alpha+\tfrac{\beta}{L}N^K(w_{opt}(K))). \end{aligned}$$

From Equation (13), we obtain $A = 0$, which means that the term in $w'_{opt}(K)$ cancels out, and

$$\begin{aligned} B = K\tfrac{\beta}{L}N^{K-1}(w_{opt}(K))\,[&N(w_{opt}(K))\ln(N(w_{opt}(K))) \\ &-L\lambda w_{opt}(K)E(w_{opt}(K))] \end{aligned}$$

which shows that the sign of $f'(K)$ is that of

$$\begin{aligned} g(K) = \;& N(w_{opt}(K))\ln(N(w_{opt}(K))) \\ & -\lambda L w_{opt}(K)E(w_{opt}(K)). \end{aligned}$$

In particular, $f'(K) = 0$ if and only if Equation (15) is satisfied. Here, $g(K)$ uses only the value $w_{opt}(K)$, but we can study the function

$$h(w) = N(w)\ln(N(w)) - \lambda L w E(w)$$

for arbitrary positive values of w. We have $g(K) = h(w_{opt}(K))$. Thus, if we show that $h(w) > 0$ for all w, we will derive that $g(K) > 0$ for all K. We differentiate $h(w)$ as follows.

$$h'(w) = \lambda L E(w)[\ln(N(w)) - \lambda w]$$

We need to study yet another auxiliary function to determine the sign of $h'(w)$. Let $h_1(w) = \ln(N(w)) - \lambda w$ for $w > 0$. Recall that $N(w) = 1 + L(E(w) - 1)$ and $E(w) = e^{\lambda(w+C_1)}$. We get $h'_1(w) = \frac{\lambda(L-1)}{N(w)} < 0$ because $L < 1$; hence, $h_1(w)$ is decreasing. We have $h_1(0) > 0$ and $\lim_{w\to+\infty} h_1(w) = \ln(L) + \lambda C_1$. We study two cases: (1) when the latter limit is positive, then $h_1(w)$ is always positive; and (2) when it is negative, then $h_1(w)$ is positive for $w > w_0$ and negative for $w < w_0$, for some w_0 whose exact value does not matter for this proof.

For case (1): $h'(w) > 0$ for all $w > 0$; hence, $h(w)$ is increasing. Recall that $N(0) > 1$; hence $h(0) = N(0)\ln(N(0)) > 0$, and $h(w)$ is positive for all w, which implies that $g(K)$ is positive for all K. We conclude that $f'(K)$ is positive for all K; hence $f(K)$ is increasing, and the optimal pattern is obtained for $K = 1$, with a single chunk.

For case (2): $h'(w) > 0$ for $w < w_0$, and $h'(w) < 0$ for $w > w_0$; hence $h(w)$ is increasing up to $w = w_0$ and then decreasing after that, down to some limit ℓ. We have $h(0) > 0$ as before; so if $\ell \geq 0$, then $h(w)$ is positive for all w, and we get the same conclusion as for case (1). On the contrary, if $\ell < 0$, then $h(w)$ has a unique zero $w_1 > w_0$, and $h(w)$ is positive for $w < w_1$ and negative for $w > w_1$. This implies that $g(K)$ is positive if $w_{opt}(K) < w_1$ and negative if $w_{opt}(K) > w_1$. In the following we prove that $w_{opt}(K)$ is a decreasing function of K and that $\lim_{K\to\infty} w_{opt}(K) = 0$. This result was expected: the more chunks in a pattern, the shorter these chunks to trade off failure-free overhead and re-execution time. Assuming these results hold true, we again have two cases. If $w_{opt}(1) < w_1$, we have $w_{opt}(K) < w_1$ for all K; hence $g(K)$ and $f'(K)$ are always positive. Then the best pattern is obtained with $K = 1$, as before. But if $w_{opt}(1) > w_1$, then $g(K)$ remains negative while $w_{opt}(K) > w_1$ and becomes positive when $w_{opt}(K) < w_1$ (which will eventually happen, because $w_{opt}(K)$ is decreasing

down to its limit 0). Then $f(K)$ is decreasing to a unique minimum K^* obtained when $g(K^*) = 0$, which is the solution of Equation (15). The best pattern requires an integer number of chunks; hence it is obtained with either $\max\{1, \lfloor K^* \rfloor\}$ or $\lceil K^* \rceil$ chunks.

To conclude the proof, we need to show that $w_{opt}(K)$ is a decreasing function of K and that $\lim_{K\to\infty} w_{opt}(K) = 0$. We first show that $w'_{opt}(K) < 0$ for all $K \geq 1$. To do so, we differentiate Equation (13) and derive that

$$Pw'_{opt}(K) = Q,$$

where (after some simplification)

$$P = \lambda^2\beta KE(w_{opt}(K))N^{K-2}(w_{opt}(K)N(w_{opt}(K)) + L(K-1))$$
$$Q = -\alpha\ln(N(w_{opt}(K))) - \tfrac{1}{K}(\alpha + \tfrac{\beta}{L}N^K(w_{opt}(K))).$$

We have $P > 0$ (obvious) and will show that $Q < 0$, so that $w'_{opt}(K) < 0$ for all K. To obtain the latter inequality $Q < 0$, we rewrite Q as

$$Q = -(\alpha + \tfrac{\beta}{L})(\ln(N(w_{opt}(K))) + \tfrac{1}{K}) + \tfrac{\beta}{L}(\ln(N(w_{opt}(K))) + \tfrac{1-N^K(w_{opt}(K))}{K}).$$

We have $\alpha + \frac{\beta}{L} = \mathcal{R}(e^{\lambda C_2} - 1) > 0$. We will show that $\ln(N(w_{opt}(K))) + \frac{1-N^K(w_{opt}(K)))}{K}) < 0$, which leads us to conclude that $Q < 0$. Consider the function $h_2(x) = \ln(x) + \frac{1-x^K}{K}$ for $x \geq 1$ (remember that $N(w_{opt}(K)) > 1$). We have $h'_2(x) = \frac{1-x^K}{x} < 0$ and $h_2(1) = 0$, hence $h_2(x) < 0$ for $x > 1$.

To determine $\lim_{K\to\infty} w_{opt}(K)$, we consider Equation (11) and divide both sides by $N(w_{opt}(K))^K$. We obtain

$$\beta\lambda K w_{opt}(K)\frac{E(w_{opt}(K))}{N(w_{opt}(K))} = \frac{\alpha}{N(w_{opt}(K))^K} + \frac{\beta}{L}.$$

Since we know that $E(w)$ and $N(w)$ are increasing functions of w, for all $w > 0$ we have

$$\frac{1}{L + \frac{1-L}{E(0)}} \leq \frac{E(w)}{N(w)} = \frac{1}{L + \frac{1-L}{E(w)}} \leq \frac{1}{L}$$

$$\frac{\beta}{L} \leq \frac{\alpha}{N(w)^K} + \frac{\beta}{L} \leq \frac{\alpha}{N(0)^K} + \frac{\beta}{L}.$$

Hence $Kw_{opt}(K)$ is bounded above and below for all K by two constants. This shows that $\lim_{K\to\infty} w_{opt}(K) = 0$, and concludes the proof. □

We here use the example presented in the end of Section 3.2 to describe how to obtain the optimal number of chunks K^* and the optimal chuck size w^* by combining Equation (11) and Equation (15). First, w^* can be computed by conducting Newton's method on Equation (15) because w is the unique variable in this equation. For the example presented in Section 3.2, w^* is computed as 368 seconds. Then, we can obtain K^* by running Newton's method for Equation (11) with the computed w^*. In that example, the optimal number of chunks K^* would be computed as 3.51 by Equation (11). Note that the value of K^* computed here is not an integer, so we need to tune the solution for fitting the practice, either by selecting a rounded integer number to stick with the pattern-based checkpoint model or by extending the solution to the interval-based checkpoint model with rational number of K^*, which will be discussed in more details later.

5 Offline Scheduling

In this section we discuss the best strategy for offline scheduling, that is, when the total amount of work $\mathcal{W}_{total}$ (i.e., fault-free job length) is known in advance. In fact, one can always use the online approach and repeat the optimal pattern (of length $\mathcal{W}_{online}$) until there remains a final piece of work of size smaller than $\mathcal{W}_{online}$ to execute. When $\mathcal{W}_{total}$ is not an exact multiple of $\mathcal{W}_{online}$, however, the last piece of work might introduce some extra overhead. For small values of $\mathcal{W}_{total}$, say requiring only a few patterns $\mathcal{W}_{online}$, this extra overhead may be significant. To this end, if the job length is already known, one may prefer to further improve the performance by adopting an offline solution instead of always using the online approach.

The question then is how to optimize the offline solution by determining the optimal number and size of the patterns. We will show that the optimal solution is to use the same-size patterns, and we will compute their optimal number (or equivalently, their optimal size). First, however, we address the following question: Given a pattern size W, what is the optimal number of chunks to execute it? This is the dual question of the one addressed in Section 4.1, where the number of chunks was given and we were searching for the optimal work size. Due to space limitations, we do not provide proofs in this section, but instead refer to the extended version [18].

5.1 Optimal pattern with given size W

In this section we derive the best pattern when the work size W is given.

Proposition 4. *When W is given, the optimal pattern of size W has $K_{opt} = \max\{1, \lfloor K^* \rfloor\}$ or $K_{opt} = \lceil K^* \rceil\}$ same-size chunks, where $K^* = \frac{W}{w_{opt}}$ and w_{opt} is the unique solution (if it exists) of the following equation:*

$$\ln\left(1 + L\left(e^{\lambda(w_{opt}+C_1)} - 1\right)\right) = \lambda w_{opt}\left(1 + \frac{L-1}{1 + L\left(e^{\lambda(w_{opt}+C_1)} - 1\right)}\right) \quad (16)$$

We observe that Equation (16) can be rewritten as Equation (15). More precisely, if we let $w^* = w_{opt}$ in Equation (15), we retrieve Equation (16). Hence the optimal size w_{opt} of the chunks in a pattern is always obtained in the same way, by solving Equation (16), and this result is true for any value of the pattern length W, not only when $W = \mathcal{W}_{online}$ (as was shown by Equation (15)).

5.2 Solution with rational number of chunks

We are ready to determine the optimal offline solution when allowing rational numbers of chunks. Given a job whose total size is $\mathcal{W}_{total}$, we divide it into p patterns of size $W_1, W_2, \ldots, W_p$, where $\mathcal{W}_{total} = \sum_{i=1}^{p} W_i$. From Proposition 4, once we know the pattern sizes, we know how to subdivide them into chunks. We will accept a fractional number of patterns and a fractional number of chunks for each pattern in our solution. In Section 6, we explain how to round up the solution. The following result gives the optimal number of patterns and their sizes:

Proposition 5. *The optimal rational offline solution for a job of total size $\mathcal{W}_{total}$ is to divide it into p^* same-size patterns, where*

$$p^* = \frac{\mathcal{W}_{total}\ln(N(w_{opt}))}{\left(\mathbb{L}\left(\frac{\alpha L}{\beta e}\right) + 1\right)w_{opt}}, \quad (17)$$

where w_{opt} is given by Equation (16)*, and letting $N(w) = 1 + L(e^{\lambda(w+C_1)} - 1)$ as before. Here $\mathbb{L}$ is the Lambert function (see Equation* (14)*).*

6 Discussion

In this section, we discuss how to implement the above-mentioned optimal solutions. In practice, we recommend using the interval-based optimal online checkpoint solution, because it exhibits the best performance in our simulations (as shown in the next section). As mentioned in Section 2, our derivation is based on a pattern-based checkpoint mode (the *mode-B* of Section 2), which can be extended easily to an interval-based checkpoint mode (the *mode-A*). For this purpose, one simply needs to further compute the optimal level-2 checkpoint interval (denoted by w^*_{opt2}) based on the optimal chunk size w^*_{opt} and on the optimal number (denoted by K^*) of chunks in each pattern, as follows:

$$w^*_{opt2} = K^* \cdot w^*_{opt} \tag{18}$$

Under a pattern-based mode, type-2 checkpoints are taken every K^* type-1 checkpoints, and K^* must be an integer. Under an interval-based mode, a new type-2 checkpoint is taken when the amount of workload *successfully* processed since the last type-2 checkpoint was taken is equal to w^*_{opt2}. In the latter case, K^* can take non-integer values in Equation (18).

We have a total of four solutions based on either interval- or pattern-based checkpoint modes, as listed below.

- *Online-Pattern-OPT*: The first one is the optimal pattern solution without knowing the productive job length before execution. With this solution, the system will set the level-1 checkpoints periodically, based on the optimal level-1 checkpoint interval (denoted by w^*_{opt}), and set level-2 checkpoints periodically too, based on the number of level-1 checkpoints taken (denoted by K^*) in the pattern. In our implementation, the values of w^*_{opt} and K^* are obtained by solving Equation (15) and Equation (11) using Newton's method. Note that the optimized checkpoint frequency at level 2 is supposed to be an integer, so K^* will be rounded to the nearest nonzero integer number in this solution. This solution is represented by $\{w^*_{opt}, K^*\}$, where K^* is a positive integer.
- *Online-Interval-OPT*: This solution is the optimal online checkpoint solution, in which the checkpoint positions (for both level 1 and level 2) during the execution are always set based on checkpoint intervals. Specifically, the level-1 checkpoint interval is the same as that of *Online-Pattern-OPT*, while the level-2 checkpoints will not be set based on the number of level-1 checkpoints taken but will be set periodically based on a level-2 checkpoint interval (denoted by w^*_{opt2}). In our implementation, we first compute w^*_{opt} and K^* by solving Equation (15) and Equation (11), respectively, and then obtain w^*_{opt2} based on Equation (18). This optimal solution is represented by $\{w^*_{opt}, w^*_{opt2}\}$.
- *Offline-Pattern-OPT*: This solution is an offline solution, which requires to know the productive job length in advance. Similarly to the *Online-Pattern-OPT* solution, *Offline-Pattern-OPT* sets the level-1 checkpoints based on checkpoint intervals (i.e., the productive time used for processing the workload) and sets the level-2 checkpoints every K^* level-1 checkpoints, where K^* is a positive integer number. Implementing this method involves three steps: (1) given a known job length $\mathcal{W}_{total}$, use Proposition 5 to compute p^* (i.e., the optimal number of patterns in the execution); (2) compute the optimal level-1 checkpoint interval w^*_{opt} (i.e., chunk size) by solving Proposition 4 using Newton's method; and (3) compute the optimal number K^* of chunks in each pattern by $\frac{\mathcal{W}_{total}}{p^* \cdot w^*_{opt}}$. Similar to the *Online-Pattern-OPT* solution, the value of K^* computed will likely not be an integer number, so we will use its nearest nonzero integer number instead in practice.
- *Offline-Interval-OPT*: Similarly to *Offline-Pattern-OPT*, *Offline-Interval-OPT* requires knowing the job length beforehand. The difference is that level-1 checkpoints and level-2 checkpoints both will be set based on checkpoint intervals under this solution. In particular, w^*_{opt} is the same as the one in *Offline-Pattern-OPT*, and w^*_{opt2} is equal to $w^*_{opt} \cdot K^*$.

7 Performance Evaluation

In this section, we present the evaluation results, by comparing our proposed optimal solution with other optimized multilevel checkpoint methods. We first describe the experimental setting and then present the evaluation results.

7.1 Experimental Setup

For our experimental evaluation, we have access to the Argonne FUSION cluster [21]. However, there are at most 128 physical nodes (for a grand total of 1,024 cores) available in that cluster with a limited resource usage quota. This is far too limited a scale to validate our results, since our fault-tolerance research is designed for exascale applications. Therefore, we have to use simulations to evaluate our multilevel checkpoint solutions. In addition, we validate the accuracy of our simulator, by performing practical experiments deployed with FTI [13] and scientific simulation benchmark based on a real-world diffusion problem (2D heat distribution) over the Fusion cluster.

The application used in our benchmark is a well-known MPI program, called *Heat Distribution*, whose MPI communication algorithms (such as the ghost array design between adjacent blocks) are commonly adopted in real scientific projects such as parallel ocean simulation [22], [23]. The key reason we adopted this application in our evaluation is that it involves many MPI function calls, including MPI_Bcast, MPI_Barrier, MPI_Recv, MPI_Send, MPI_Irecv, MPI_Isend, MPI_Waitall, and MPI_Allreduce. The checkpoint and recovery overheads are both dependent on two factors: the program memory size determined by the problem size, and the execution scale (i.e., the number of processes).

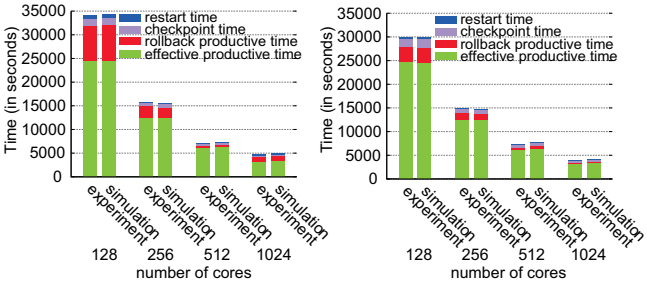
Our simulations are performed as follows. The simulator emulates the running of some MPI program by processing a fixed amount of parallel workload. The parallel productive time is fixed for a particular test case. Each simulation is driven by ticks (one tick is set to one second in the simulation), simulating the whole procedure of running the MPI program. Our simulator takes into account any possible unexpected events during the execution, such as the occurrence of failures during the recovery.

The simulator emulates the checkpoint/recovery procedure based on the real-world multi-level checkpoint toolkit FTI [13], so there are a total of four checkpoint levels implemented in our simulator, including local disk storage, partner-copy technique, RS-encoding technique, and PFS file system. Local disk storage level allows the checkpoints to be stored on local disks of the execution nodes. Partner-copy technology makes each checkpoint

file have two copies stored in the local storage device and another partner-node respectively. Thus, upon a failure event with multiple simultaneous hardware crashes, the whole execution can still be recovered via partner-copy as long as there are no adjacent/partner nodes crashed. Reed-solomon encoding is a more advanced technology, which allows the application to be recovered with M missing checkpoints. PFS is the top level, which means the checkpoints are to be stored in the parallel file system. Obviously, local storage corresponds to *no hardware failure*, partner-copy corresponds to *nonadjacent node failure*, Reed-solomon encoding corresponds to *adjacent node failures with at most M failed nodes*, and PFS corresponds to *any other failure cases*. In the evaluation of our propsoed solution, we take two out of the four optional checkpoint levels with different overheads.

We verify the accuracy of our simulation by comparing its performance results to that of the real runs on the Argonne FUSION cluster [21] with Heat Distribution benchmark using up to 1024 cores. All of the failures were injected by using the command "kill -9" to terminate some of the processes running by the MPI program. As suggested by the developer of FTI, we just need to remove some checkpoint files to emulate the missing nodes. For example, in order to emulate the single node failure, we select one node and remove its local checkpoint file and then restart the application using FTI. FTI automatically recovers the missing checkpoint file before moving on. In order to simulate the multiple node crashes simultaneously striking the application, we just need to delete multiple checkpoint files. The FTI would also automatically recover the multiple missing checkpoint files, which may suffer from longer recovery overhead.

The verification results are shown in Figure 2, which confirms very similar results between simulation and the real experiments. The failure rates injected follow a Poisson distribution, with the mean failure intervals being set to 861 seconds, 1722 seconds, 4305 seconds and 8610 seconds respectively. In absolute terms, with the same setting on the productive length and checkpoint/restart overheads, the difference between the simulation results and experimental results is less than 4%. Note that the model proposed in this work involves two levels instead of four, so the users can select any two of the checkpoint levels based on the characterization or analysis of the failure rates in practice, as we did in our evaluation.



(a) ckpt intvl:1,3,5,7 min (b) ckpt intvl:1,2,3,4 min

Fig. 2: Confirming the Effectiveness of the Simulator

Our evaluation involves nine test cases, regarding various checkpoint/restart overheads, different failure rates, and different application execution lengths. All of the test cases respect the observations based on our experiments with the benchmark running on the Argonne FUSION cluster under the protection of FTI. Specifically, we observe that the higher the checkpoint level is, the higher checkpoint/recovery overhead is, while the checkpoint overhead does not change a lot with different scales on the same level. Moreover, the higher-level failure rate should be lower than lower-level failure rate, as analyzed in our previous work [7].

The settings of the nine test cases are presented in Table 2, where C_1 and C_2 refer to checkpoint overheads (in seconds), R_1 and R_2 refer to restart overheads (in seconds), and λ_1 and λ_2 refer to the failure rates (number of failures per day) at level 1 and level 2, respectively. We conservatively overestimate recovery times and set them equal to checkpoint times, because this represents the worst-case scenario for the simulations. The failures are injected randomly in terms of the Poisson process, which is a standard approach [24], [20]. That is, the failure intervals follow exponential distribution with the failure rates λ_1 and λ_2 at the two levels.

TABLE 2: Simulation setting of 9 test cases.

	C_1	C_2	R_1	R_2	λ_1	λ_2	**Length**
Case 1	20	50	20	50	24	4	86400
Case 2	20	50	20	50	50	10	86400
Case 3	20	100	20	100	100	20	86400
Case 4	10	40	10	40	100	20	86400
Case 5	10	40	10	40	200	40	86400
Case 6	10	100	10	100	200	40	43200
Case 7	40	200	40	200	300	60	21600
Case 8	50	300	50	300	400	60	21600
Case 9	50	300	50	300	400	60	10800

We compare five methods. The first two methods (namely, *Online-Pattern-OPT* and *Online-Interval-OPT*) are online solutions without knowing fault-free job length. The third and fourth solutions are the offline solutions that require to know the job length beforehand. These two offline solutions are derived in Section 5. The fifth method (namely, *Approximate-OPT*) is the approach proposed in our previous work [7], and it is also an offline solution because it requires knowing the job length beforehand. In our previous work, *Approximate-OPT* was deemed the best solution compared with other state-of-the-art approaches such as Young's formula [25]. In that work [7], an approximate optimal algorithm was proposed, by leveraging the fixed-point iteration method. Such a solution is subject to a few assumptions; for example, the failures are assumed not to occur during the checkpoint/restart period, and the final wall-clock length is not extended significantly compared with the fault-free productive length. As such, the output of the algorithm may deviate from the practical optimal solution, especially when the checkpoint/restart overhead is relatively large compared with the length (as is confirmed in our evaluation, shown later).

7.2 Evaluation Results

First of all, we compare the outputs (including interval-based model and pattern-based model) of the optimal online solution versus the optimal offline solution. We also compare these solutions to our previous work [7], as shown in Table 3.

From the table we observe two significant findings. We can see that our online optimal solution leads to the same results as the offline optimal solution does. In fact, the difference between the online optimal solution and offline optimal solution is tiny. For case 1, for instance, the offline optimal solution is derived as $\{w^*_{opt}$=368.64474109273493, K^*=3.5134717932162443$\}$, while the online optimal solution is $\{w^*_{opt}$=368.64474109270884, K^*=3.513471793216452$\}$. Hence, the optimality of our online solution (optimization without knowing the job length) is equal

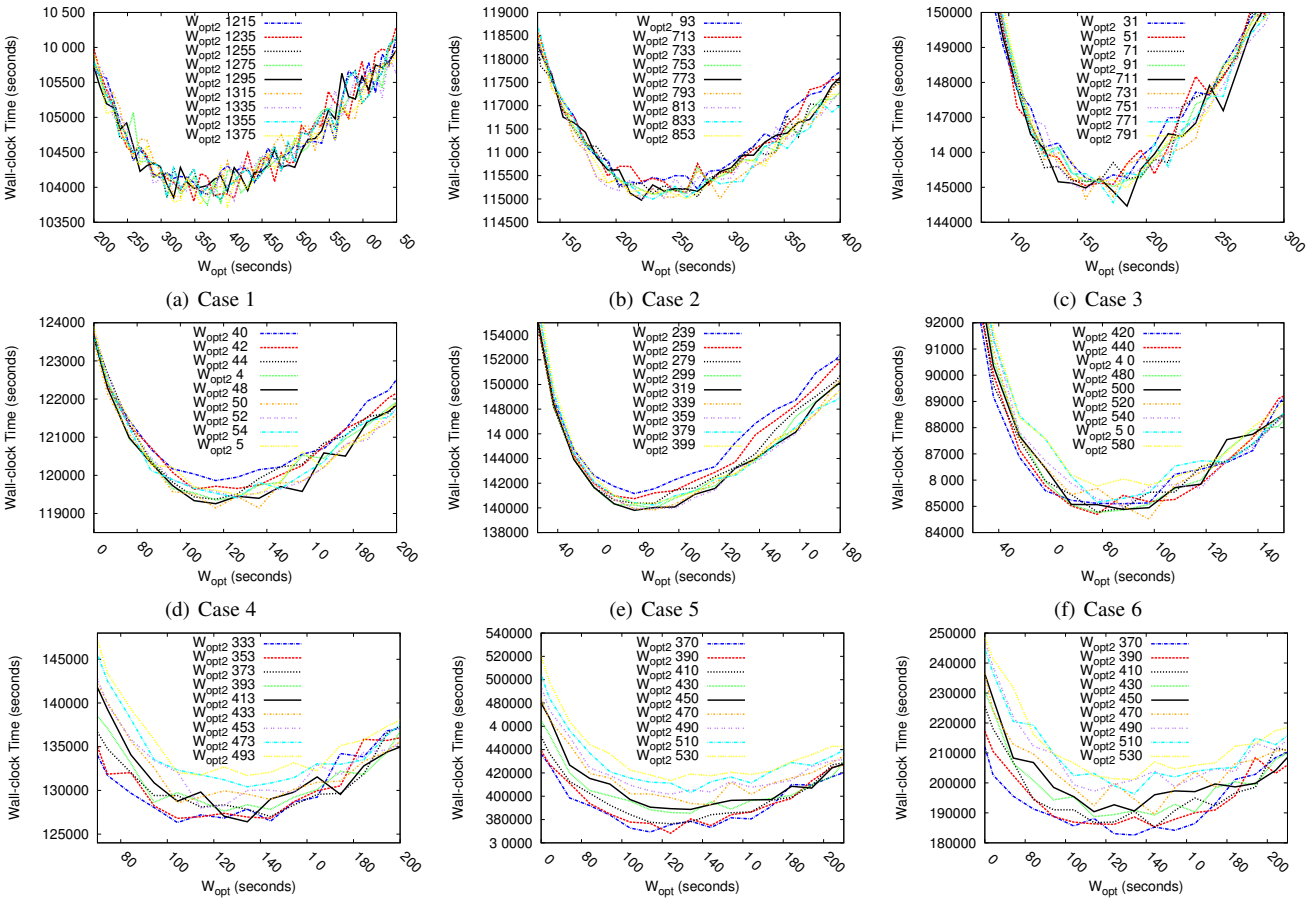


Fig. 3: Verification of the optimality of our online interval-OPT solution.

TABLE 3: The optimized solutions under different approaches.

	Online Solution				**Offline Solution**					
	Pattern-OPT		**Interval-OPT**		**Pattern-OPT**		**Interval-OPT**		**Approximate-OPT**	
	w^*_{opt}	K^*	w^*_{opt}	w^*_{opt2}	w^*_{opt}	K^*	w^*_{opt}	w^*_{opt2}	w^*_{opt}	w^*_{opt2}
Case 1	368.6	4 (3.51)	368.6	1295.2	368.6	4 (3.51)	368.6	1295.2	385.7	1433
Case 2	252.7	3 (3.06)	252.7	773	252.7	3 (3.06)	252.7	773	269.6	896.9
Case 3	175.9	4 (4.04)	175.9	711.3	175.9	4 (4.04)	175.9	711.3	195.2	885
Case 4	126.4	4 (3.85)	126.4	486.1	126.4	4 (3.85)	126.4	486.1	135.7	567.3
Case 5	88.0	4 (3.63)	88.0	319	88.0	4 (3.63)	88.0	319	97.1	395.8
Case 6	88.0	6 (5.68)	88.0	499.9	88.0	6 (5.68)	88.0	499.9	99.5	626.5
Case 7	134.4	3 (3.07)	134.4	412.7	134.4	3 (3.07)	134.4	412.7	168.8	682.4
Case 8	124.1	4 (3.62)	124.1	449.5	124.1	4 (3.62)	124.1	449.5	166.5	815.1
Case 9	124.1	4 (3.62)	124.1	449.5	124.1	4 (3.62)	124.1	449.5	166.5	815.1

to that of the offline optimal solution (the optimization with knowledge of job length). However, we can also see that the three approaches (*Online-Interval-OPT*, *Online-Pattern-OPT*, and *Approximate-OPT*) output different checkpoint solutions. As for the *Online-Pattern-OPT* (i.e., $\{W^*_{opt}, K^*\}$), the level-2 checkpoint frequency has to be an integer number in practice, so its final value will be rounded to its nearest integer for checkpoint level 2. In the table, the K^* column shows the rounded integer value (outside parentheses) and the original real number (inside parentheses) computed by our solution. Such a rounding operation will degrade the optimality of checkpoint solution, as will be shown later.

We validate the optimality of our proposed solution (*Online-Interval-OPT*) to see how close it is to the best experimental solution[5]. In this evaluation, different solutions are generated by combining different level-1 checkpoint intervals and level-2 checkpoint intervals, to evaluate whether our solution indeed leads to the best performance (with minimum wall-clock length). For each test case and each checkpoint solution, we run the simulation 1,000 times each with different random arrival time points of failure events but with the same settings on job length (i.e., productive time), failure rate, checkpoint overhead and recovery overhead. We then compute the mean values for every portion of times consumed in the execution, including productive time, checkpoint overhead, restart overhead, and rollback time.

5. The best experimental solution is obtained by brute-force searching/traversing different combinations of checkpoint intervals (regarding the two levels/types of failures), which are increased gradually by a tiny increment (5 seconds in experiments) from a small checkpoint interval (20 seconds).

Figure 3 presents the wall-clock times consumed with different checkpoint intervals set in various failure cases, where the wall-clock time is the sum of the above four time portions. To make Figure 3 easy to read, we highlight our solutions by solid curves. We present the results only for the solutions with a particular checkpoint interval range, because the solutions with other checkpoint intervals lead to much higher wall-clock lengths (worse results). In Figure 3 we see that the wall-clock time with our optimal checkpoint solution is really (or fairly close to) the minimum wall-clock length among all other solutions. In Table 4, we present the wall-clock time of our optimal checkpoint solution versus that of the best experimental solution selected by traversing all the checkpoint intervals. It shows that the wall-clock time difference between our solution and the best experimental solution is always within 0.7% for the first seven test cases, and it can be limited within 7.7% for the other two test cases in which the checkpoint/recovery overheads are large (up to 5 minutes for each level-2 checkpoint) and failure rates are also high (about one failure every three minutes on average). The relatively high difference between our solution and the best experimental solution in the two test cases is due to the assumption that no faults occur during the recovery period, whereas the failure-over-recovery would occur often in the last two test cases. For these two test cases, we later show that our optimal solution achieves significant performance gains compared with the results of the previous optimized solution proposed in [7].

TABLE 4: Our derived optimal solution vs. best experimental solution (in seconds).

	$\{w^*_{opt}, w^*_{opt2}\}$	Best Experimental Solution	Difference
Case 1	104024	103788	0.23%
Case 2	115220	114890	0.28%
Case 3	144883	144461	0.29%
Case 4	119451	119144	0.26%
Case 5	140029	139801	0.16%
Case 6	84884	84517	0.43%
Case 7	126407	125523	0.7%
Case 8	389354	362493	6.9%
Case 9	190764	175701	7.7%

In Figure 4, we compare the total wall-clock time and different portions of time consumed by our proposed online checkpoint solution (*Interval-OPT*) versus that of the other two optimized approaches (*Pattern-OPT* and *Approximate-OPT*). For the first six test cases, we observe that the three solutions lead to very close wall-clock times: the difference is smaller than 2% in wall-clock length. For the last three test cases, our optimal online solution (*Interval-OPT*) outperforms the other two solutions significantly. In test case 8, the wall-clock time of our solution is reduced by about 11% and 25.3%, compared with *Pattern-OPT* solution and *Approximate-OPT* [7], respectively. In test case 9, the wall-clock time can be reduced by 12.5% and 23.6%, respectively. The key reason for the poor performance of *Pattern-OPT* is that the level-2 checkpoint frequencies adopted are always conducted strictly based on the number (i.e., rounded integer) of level-1 checkpoints, definitely deviating from the best experimental solution. The key reason for the degraded performance of the *Approximate-OPT* algorithm proposed in [7] is that it is subject to a few significant assumptions such as the relatively low failure rates and not-much-extended wall-clock length in comparison to the fault-free productive time. In particular, when the failure rate is not high or the checkpoint/restart overhead is relatively low (i.e., the first 6 test cases), the performance of our previous *Approximate-OPT* algorithm is close to that of the optimal solution proposed in this paper. However, with high failure rates and heavy checkpoint/restart overhead, the performance degradation of *Approximate-OPT* is clear, as confirmed by the last three test cases in Figure 4.

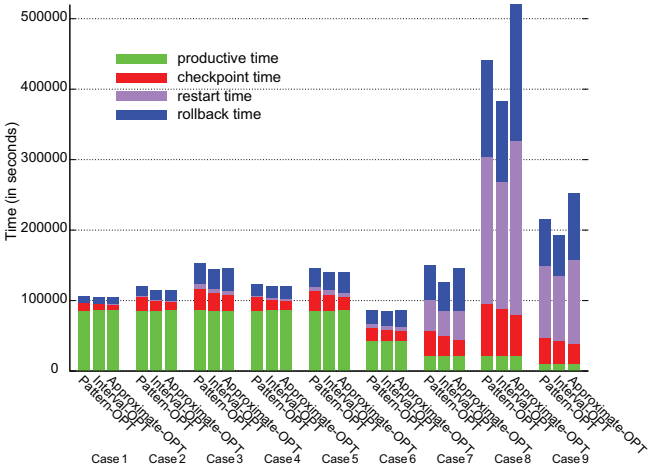


Fig. 4: Performance comparison of our solution with others.

8 Related Work

The optimization issue of finding the optimal checkpoint intervals for HPC applications has been studied for decades. In 1973, Young [25] proposed a formula to optimize the checkpoint interval with the first-order approximation. In 2006, Daly [20] extended his solution to a higher-order approximation and took recovery overhead into account. These two works consist of the fundamental and classic optimization strategies. Thereafter, more checkpoint solutions were proposed to resolve more complicated cases. Chen and Ren [26], for example, proposed an optimal checkpoint solution from the perspective of the overall system performance. Liu et al. [27] proposed a reliability-aware optimal checkpoint/restart strategy for a large-scale HPC system. Compared with other checkpoint/restart optimization work, their model can deal with a varying checkpoint interval with different failure distributions. Jin et al. [24] optimized the checkpoint intervals and execution scales simultaneously. The optimal checkpoint interval is derived by assuming exponential failure distribution and using first-order approximation. They derived the expected wall-clock length regarding various overheads and rollback loss, but they did not prove that it is a convex function with respect to the variables before using Newton's method to approximate the optimal solution. In that situation, the converged result may not be globally optimized; and even worse, the algorithm may not converge given inappropriate initial values. Most important, all these works are restricted to the single-level checkpoint model, which cannot work efficiently in an extreme-scale HPC environment because of inevitable huge checkpoint/recovery overheads.

Multi-level checkpoint model was introduced in order to address the limitation of single-level checkpoint strategies. In 1995, Vaidya et al. proposed a two-level checkpoint model [28], which is similar to the pattern-based checkpoint model in our paper. By numerical analysis, they confirmed that multi-level checkpoint model is able to improve the performance compared to single-level checkpoint model. To this end, some toolkits (such as [12], [13]) already support multilevel checkpoint protocols. That is, the users are allowed to set various types of checkpoints with

different overheads for protecting have been developed to support applications against different types of failures. However, all of these works did not investigate what the checkpoint intervals are supposed to be for a particular HPC application.

How to optimize the checkpoint intervals for multilevel checkpointing has been studied in the recent years. Hakkarinen and Chen [29] proposed a multilevel diskless checkpoint model. This model is similar to our proposed multi-level checkpoint model, although it focuses on a diskless checkpoint environment. Because of the complexity of the problem, the authors mainly derived the representation formula for the expected wall-clock time regarding different checkpoint overheads and failure rates and narrowed the value range for the optimal checkpoint intervals but did not provide exact optimal checkpoint intervals for the multilevel checkpoint model. Moreover, they evaluated their solutions by a hypothetical simulation based on the estimated wall-clock length with numerically summed overheads and rollback loss. In contrast, we carefully implement a fundamental simulation testbed, and we confirm the accuracy of our simulation using a real-world HPC application (called Heat Distribution). In our previous work, we proposed an approximate solution to optimize the checkpoint intervals for different checkpoint levels [7], [8]. In [7], we formulated the multilevel checkpoint model based on the existing multilevel checkpoint toolkits [12], [13]. In that work, to make the problem tractable, we made two important assumptions: (1) failures do not occur during the checkpoint period, and (2) the total wall-clock time is not expanded significantly compared with the fault-free execution length. With such two assumptions, an iterative algorithm was proposed to get an approximate optimal solution based on convex-optimization theory. An extended work was proposed in [8] by taking into account the environment with uncertain execution scales. In comparison to such previous works, the new solution proposed in this paper completely removes the two assumptions mentioned above and provides exact solutions instead of first-order approximations or fixed-point iteration approximations. Specifically, we reconstruct the multilevel checkpoint model and derive a new optimal checkpoint solution that is much closer to the best experimental solution. Our elaborate simulation shows that the new optimal solution outperforms the previous solutions by up to 25.3% in some cases.

9 Conclusion and Future Work

In this paper, we optimize the checkpoint solution based on a two-level checkpoint model, in which the system is allowed to set two different types of checkpoints for protecting the HPC applications against two types of failures. Optimizing the two-level checkpoint solution is difficult in that each type of failure follows its own distribution and the overall performance is synthetically determined by the two types of checkpoints and failures. To address the issue, we first formulate the research to be a pattern-based checkpoint model and then extend it to the interval-based checkpoint model. We derive the optimal two-level solutions based on both online scheduling and offline scheduling and prove that the optimal solution must adopt the equal-size checkpoint intervals on particular checkpoint levels. The key theoretical conclusion about how to conduct the optimal checkpoint strategies in practice is also discussed in Section 6. To evaluate the proposed solutions, we build a simulation testbed and confirm its accuracy by running a real-world MPI program on a real cluster with 1,024 cores. We highlight three significant findings from our evaluation.

- Our online optimal solution leads to the same results as the offline optimal solution, thus confirming the optimality of our online solution.
- Our online optimal solution always outperforms other optimized solutions, and improves the performance significantly in some cases. Specifically, the wall-clock time under the new solution can be reduced by up to 25.3% compared with the results of other state-of-the-art methods.
- The optimal solution derived in this paper is close to the best experimental solution (obtained via brute-force searching), with the difference (in the wall-clock time) less than 1% in most cases.

A possible future project could involve investigating how to optimize the two-level checkpoint solution by removing the assumption that the failures may not occur during the recovery period. In addition, we plan to evaluate the optimal solutions proposed using additional real-world applications.

Acknowledgments

We would like to thank the reviewers for their comments and suggestions, which greatly helped improve the final version of the paper. This research was funded in part by the European project SCoRPiO, and by the PIA ELCI project. Yves Robert is with Institut Universitaire de France. This research was also based upon work supported by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research Program, under Contract DE-AC02-06CH11357; and by the INRIA-Illinois-ANL-BSC-JSC-Riken Joint Laboratory on Extreme Scale Computing, and Center for Exascale Simulation of Advanced Reactors (CESAR) at Argonne.

References

[1] A. Feinberg, "An 83,000-processor supercomputer can only match 1% of your brain," 2013, available at http://gizmodo.com/an-83-000-processor-supercomputer-only-matched-one-perc-1045026757.

[2] H. Casanova, Y. Robert, F. Vivien, and D. Zaidouni, "On the impact of process replication on executions of large-scale parallel applications with coordinated checkpointing," *Future Generation Computer Systems*, vol. 51, pp. 7 – 19, 2015.

[3] D. Fiala, F. Mueller, C. Engelmann, R. Riesen, K. Ferreira, and R. Brightwell, "Detection and correction of silent data corruption for large-scale high-performance computing," in *Proc. SC '12*. IEEE Computer Society Press, 2012.

[4] A. R. Benson, S. Schmit, and R. Schreiber, "Silent error detection in numerical time-stepping schemes," *Int. J. High Perform. Comput. Appl.*, vol. 29, no. 4, pp. 403–421, 2015.

[5] J. P. Walters and V. Chaudhary, "A Scalable Asynchronous Replication-based Strategy for Fault Tolerant MPI Applications," in *Proc. HiPC'07*. Springer-Verlag, 2007, pp. 257–268.

[6] J. Walters and V. Chaudhary, "Replication-Based Fault Tolerance for MPI Applications," *IEEE Trans. Parallel Distributed Systems*, vol. 20, no. 7, pp. 997–1010, 2009.

[7] S. Di, M. Bouguerra, L. Bautista-Gomez, and F. Cappello, "Optimization of multi-level checkpoint model for large scale HPC applications," in *Proc. IPDPS*. IEEE, 2014.

[8] S. Di, L. Bautista-Gomez, and F. Cappello, "Optimization of a multilevel checkpoint model with uncertain execution scales," in *Proc SC '14*. IEEE, 2014.

[9] L. Gomez, A. Nukada, N. Maruyama, F. Cappello, and S. Matsuoka, "Low-overhead diskless checkpoint for hybrid computing systems," in *Proc. HiPC*, 2010.

[10] L. A. B. Gomez, N. Maruyama, F. Cappello, and S. Matsuoka, "Distributed diskless checkpoint for large scale systems," in *Proc CCGRID '10*. IEEE, 2010.

[11] H. Li, L. Pang, and Z. Wang, "Two-level incremental checkpoint recovery scheme for reducing system total overheads," *PLoS ONE*, vol. 9, no. 8, 2014.

[12] A. Moody, G. Bronevetsky, K. Mohror, and B. de Supinski, "Design, modeling, and evaluation of a scalable multi-level checkpointing system," in *Proc. SC'10*. IEEE, 2010.

[13] L. Bautista-Gomez, S. Tsuboi, D. Komatitsch, F. Cappello, N. Maruyama, and S. Matsuoka, "FTI: High performance fault tolerance interface for hybrid systems," in *Proceedings of Supercomputing (SC'11)*. ACM, 2011.

[14] A. Kulkarni, A. Manzanares, L. Ionkov, M. Lang, and A. Lumsdaine, "The design and implementation of a multi-level content-addressable checkpoint file system," in *Proc. HiPC*. IEEE, 2012.

[15] G. Zheng, X. Ni, and L. Kale, "A scalable double in-memory checkpoint and restart scheme towards exascale," in *Proc. DSN-W*. IEEE, 2012.

[16] S. Di and F. Cappello, "Fast error-bounded lossy hpc data compression with sz," in *Proc. IPDPS*. IEEE, 2016.

[17] J. Choi, J. Dongarra, S. Ostrouchov, A. Petitet, D. Walker, and R. C. Whaley, "The design and implementation of the ScaLAPACK LU, QR, and Cholesky factorization routines," *Scientific Programming*, vol. 5, pp. 173–184, 1996.

[18] S. Di, Y. Robert, F. Vivien, and F. Cappello, "Toward an Optimal Online Checkpoint Solution under a Two-Level HPC Checkpoint Model," INRIA, Research report RR-8851, 2016, available at graal.ens-lyon.fr/~yrobert/rr8851.pdf.

[19] M. Bougeret, H. Casanova, M. Rabie, Y. Robert, and F. Vivien, "Checkpointing strategies for parallel jobs," in *Proc. SC'11*. IEEE, 2011.

[20] J. T. Daly, "A higher order estimate of the optimum checkpoint interval for restart dumps," *Future Generation Computer Systems*, vol. 22, no. 3, pp. 303–312, 2006.

[21] "Fusion cluster," Available at http://www.lcrc.anl.gov/.

[22] R. Smith, P. Jones, B. Briegleb, F. Bryan, G. Danabasoglu, J. Dennis, J. Dukowicz, C. Eden, B. Fox-Kemper, P. Gent, M. Hecht, S. Jayne5, M. Jochum, W. Large, K. Lindsay, M. Maltrud, N. Norton, S. Peacock, M. Vertenstein, and S. Yeager, "The Parallel Ocean Program (POP) reference manual: Ocean component of the Community Climate System Model (CCSM)," Los Alamos National Laboratory, Tech. Rep. LAUR-10-01853, 2010.

[23] "Community Earth System Model (CESM)," Available at http://www2.cesm.ucar.edu.

[24] H. Jin, Y. Chen, H. Zhu, and X.-H. Sun, "Optimizing HPC Fault-Tolerant Environment: An Analytical Approach," in *Proc. ICPP*. IEEE, 2010.

[25] J. W. Young, "A first order approximation to the optimum checkpoint interval," *Comm. ACM*, vol. 17, no. 9, pp. 530–531, 1974.

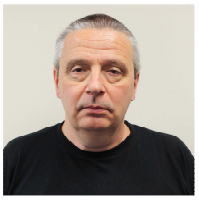
[26] N. Chen and S. Ren, "Adaptive optimal checkpoint interval and its impact on system's overall quality in soft real-time applications," in *Proc. SAC '09*. ACM, 2009.

[27] Y. Liu, R. Nassar, C. Leangsuksun, N. Naksinehaboon, M. Paun, and S. Scott, "An optimal checkpoint/restart model for a large scale high performance computing system," in *Proc. IPDPS*. IEEE, 2008.

[28] N. H. Vaidya, "A case for two-level distributed recovery schemes," in *Proceedings of the 1995 ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*, ser. SIGMETRICS '95/PERFORMANCE '95. New York, NY, USA: ACM, 1995, pp. 64–73.

[29] D. Hakkarinen and Z. Chen, "Multilevel diskless checkpointing," *IEEE Trans. Computers*, vol. 62, no. 4, pp. 772–783, 2013.

Sheng Di Sheng Di received his Ph.D. degree from the University of Hong Kong in 2011. He is currently a postdoctoral researcher at Argonne National Laboratory. Dr. Di's research interest involves resilience on high-performance computing (such as silent data corruption, optimization checkpoint model, and in-situ data compression) and broad research topics on cloud computing (including optimization of resource allocation, cloud network topology, and prediction of cloud workload/hostload). He is working on multiple HPC projects, such as detection of silent data corruption, characterization of failures and faults for HPC systems, and optimization of mutlilevel checkpoint models. He is the author of 13 papers published in international journals and 33 papers published at international conferences. He served as programming committee member 14 times for different conferences, and served as an external conference/journal reviewer over 50 times. Contact him at sdi1@anl.gov.



Yves Robert Yves Robert received his Ph.D. degree from the Institut National Polytechnique de Grenoble. He is currently a full professor in the Computer Science Laboratory LIP at ENS Lyon. He is the author of 7 books, 135 papers published in international journals, and 205 papers published in international conferences. He is the editor of 11 book proceedings and 13 journal special issues. He is the advisor of 29 Ph.D. candidates. His main research interests are scheduling techniques and resilient algorithms for large-scale platforms. He is a Fellow of the IEEE. He was elected a Senior Member of Institut Universitaire de France in 2007 and renewed in 2012. He has been awarded the 2014 IEEE TCSC Award for Excellence in Scalable Computing. He has held a visiting scientist position at the University of Tennessee Knoxville since 2011. Contact him at Yves.Robert@inria.fr.

Frédéric Vivien Frédéric Vivien received his Ph.D. degree from the École Normale Supérieure de Lyon in 1997. From 1998 to 2002, he was an associate professor at the Louis Pasteur University in Strasbourg, France. He spent the year 2000 working with the Computer Architecture Group of the MIT Laboratory for Computer Science. He is currently a senior researcher from INRIA, working at ENS Lyon, France. He leads the INRIA project-team Roma, which focuses on designing models, algorithms, and scheduling strategies to optimize the execution of scientific applications.He is the author of two books, more than 35 papers published in international journals, and more than 50 papers published in international conferences. His main research interests are scheduling techniques and parallel algorithms for distributed and/or heterogeneous systems. Contact him at Frederic.Vivien@inria.fr.

Franck Cappello Franck Cappello is program manager and senior computer scientist at Argonne National Laboratory. Previously, he held a joint position at INRIA and University of Illinois at Urbana-Champaign, where he initiated and codirected from 2009 the INRIA–Illinois Joint Laboratory on Petascale Computing. Until 2008, he led a team at INRIA where he initiated the XtremWeb (Desktop Grid) and MPICH-V (fault-tolerant MPI) projects. From 2003 to 2008, he initiated and directed the Grid5000 project, a nationwide computer science platform for research in large-scale distributed systems. He has authored papers in the domains of fault tolerance, high-performance computing, desktop Grids and Grids and contributed to more than 70 program committees. He is an editorial board member of the international Journal on Grid Computing, Journal of Grid and Utility Computing, and Journal of Cluster Computing. Contact him at cappello@mcs.anl.gov.