Brief Contributions

Complexity Analysis of Checkpoint Scheduling with Variable Costs

Mohamed-Slim Bouguerra,
Denis Trystram, and Frédéric Wagner

Abstract—The parallel computing platforms available today are increasingly larger and thus, more and more subject to failures. Consequently it is necessary to develop efficient strategies providing safe and reliable completion for HPC parallel applications. Checkpointing is one of the most popular and efficient technique for developing fault-tolerant applications on such context. However, checkpoint operations are costly in terms of time, computation, and network communication. This will certainly affect the global performance of the application. In this work, we propose a performance model that expresses formally the checkpoint scheduling problem. This model exhibits the tradeoff between the impact of the checkpoints operations and the lost computation due to failures. Based on this model, we study the computational complexity of the problem of scheduling checkpoints with variable costs for general failure distributions. More precisely, we provide a new computational complexity analysis that explicits in depth the relations between the probabilistic failure model, the checkpoint cost, and the computational model. In particular, we prove that the checkpoint scheduling problem is NP-hard even in the simple case of uniform failure distribution. We also present a dynamic programming scheme for determining the optimal checkpointing times in all the variants of the problem.

Index Terms—Fault tolerance, checkpoint scheduling, failure detection

1 Introduction

1.1 Context and Motivation

In the recent years, parallel computing systems have turned increasingly to extremely large-scale parallel platforms composed of thousands of processors. Many actual applications run on such systems for long durations, up to several days or weeks. Recently, statistical analysis about failures on high performance computing (HPC) platforms emphasize that the mean time between failures may not exceed few hours [7], [27]. In this context, it is necessary to develop efficient strategies that provide safe and reliable completion of applications. This may be achieved through redundancy [26] or by storing intermediate computation states on reliable external devices [12] (this technique is called *checkpointing*). The saved states are then used to restart computations from the last checkpoint in case of failure. This last approach is one of the most popular fault-tolerance technique in parallel and distributed systems and there exist various protocols able to capture a global valid state of a parallel application (see [2] for more details).

- *M.-S. Bouguerra is with INRIA Rhone-Alpes Grenoble, 51 avenue Kuntzmann, 38300 Montbonnot St. Martin, France. E-mail: slim.bouguerra@imag.fr.*
- *D. Trystram is with the Grenoble Institute of Technology - ENSIMAG, Institut Universitaire de France (IUF), 51 avenue Kuntzmann, 38300 Montbonnot St. Martin, France. E-mail: denis.trystram@imag.fr.*
- *F. Wagner is with the Grenoble Institute of Technology - ENSIMAG, 51 avenue Kuntzmann, 38300 Montbonnot St. Martin, France. E-mail: frederic.wagner@imag.fr.*

Manuscript received 22 July 2011; revised 3 Jan. 2012; accepted 15 Feb. 2012; published online 28 Feb. 2012.
Recommended for acceptance by D. Bader.
For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-2011-07-0500.
Digital Object Identifier no. 10.1109/TC.2012.57.

One of the main problems for this technique is to determine the right series of intervals for checkpointing. Indeed, too many checkpoints would increase the time overhead while too few checkpoints would lead to a large loss of computation time in case of failures. The time when to perform the checkpoints depends mainly on two parameters, namely, the volume of data to checkpoint (due to communication times) and the failure arrival time distribution.

1.2 Related Works

There exist various performance models for computing the optimal checkpointing strategies that minimize the delay due to failures during the execution of an application and the induced checkpoint overhead. Let us review below the most relevant results.

Young proposed in [33] a first order approximation of the optimal interval length between checkpoints that minimizes the expected lost time before failure. Considering that checkpoints are periodically scheduled, their cost is supposed to be constant during the entire execution and under the hypothesis that failures arrivals follow a Poisson's process. Daly extended Young's result in [9] and proposed a higher order approximation considering the same hypothesis. In [30], Vaidya studied the impact of the checkpoint latency time on the model proposed by Young. Trivedi et al. developed in [14] a model that expresses the expected completion time of a program when the failure distribution is general and considering a periodic checkpoint scheduling with rejuvenation capability on the platform. Then, they provided an optimal interval for checkpointing and rejuvenation when their cost is constant.

Since the periodic assumption on the checkpoint intervals does not lead to an optimal solution when the failure rate is not constant, some authors remove this hypothesis. Duda proposed in [11] a recursive objective function that expresses the expected completion time when the checkpoint cost is considered as constant. However, this formulation is mathematically intractable and determining optimal solutions is too time consuming. In [29], Toueg and Babaoğlu. provided a dynamic programming technique to minimize the expected completion time considering general failure distributions. Unfortunately, this model expresses only an upper bound for the expected completion time. Later, Ling et al. introduced a new variational technique in [32] that gives a first order approximation of the checkpoint frequency for minimizing the expected lost time before the system failure. This method is based on the first order of the Taylor expansion for the failure distribution under the hypothesis that the checkpoint cost is constant and failure-free. It does not affect the probability of failure of the application. Recently, Dohi et al. extended their previous works in [10], [22], and [21] and provided a numerical algorithm using the Brender's classical fixed-point theorem considering a failure distribution with increasing failure rate property (known as the IFR assumption [1]). Hence, the proposed algorithm converges to a near-optimal solution, considering that the checkpoint cost is constant and failure-free under the IFR assumption, while recent computing platforms rather exhibit a behavior corresponding to decreasing failure rates (DFR). For instance, the failure distribution of the computing system number 20 of the Top 500 from the LANL supercomputing center is well fitted by a Weibull distribution with a shape parameter 0.7 [27] [28]. Similarly, many failure traces of recent computing systems are close to a Weibull distribution with a shape parameter less than 1 [17]. Those results correspond to a DFR. In [24], Ren et al. were interested on the utilization of the large amount of idle computation recourses available on the volunteer computing

platforms. To handle the unavailability of the volunteer computing nodes a checkpoint-based approach to tolerate failures were proposed. First of all they formulated the problem of selecting the volunteer nodes to store checkpoints fragment then a greedy algorithm is provided to solve it. Second, by considering an arbitrary failure distribution to model failures, the optimal checkpoint interval selecting problem is considered to handle the tradeoff between the lost work due to failure and the checkpoint overhead cost. To compute the optimal checkpointing schedule, one step look ahead heuristic were provided. In this heuristic, the checkpointing decision is taken based on the comparison between the cost of checkpointing immediately with the cost of delaying the checkpoint later. Also in the context of volunteer computing Nurmi et al. [20] extended the work of Vaidya [30] by proposing a numerical-based approach to compute an optimal checkpointing schedule. In this work, authors built a system that records the traces of the availability and unavailability period from each resource and a statistical model is fitted to the collected traces using the well-known estimation techniques, namely, Maximum Likelihood Estimation or Expectation Maximization. Then, using the induced failure distribution model they provided a numerical method based on the Golden Section optimization algorithm to compute the optimal scheduling. It is well worth mentioning the fact that the authors did not provide any technical proof about the existence or the uniqueness of the solution or the computational complexity of the numerical optimization with regards to the failure distribution. Bougeret et al. presented in a recent paper [4] an optimal solution for the problem on a single machine assuming an exponential failure distribution. They computed the optimal intervals and showed that they are periodic. This confirms the result provided in 2009 by Bouguerra et al. [5] for a slightly different objective function. They also proposed a dynamic programming scheme for the restricted case of constant checkpoint costs and constant block lengths in a parallel platform.

After this overview of sequential checkpointing models, let us focus on parallel and distributed checkpointing models designed for parallel application. All the contributions described below share the following assumptions. The application is modeled by a perfect linear model such that the speedup is equal to the number of processors and the checkpoint cost is constant. The parallel platform is composed from Q identical processors in terms of computing capability and failure distribution. Finally, failures and unavailability times are exponentially distributed. Considering those assumptions Plank et al. considered the problem of choosing k computing processors among Q processors to replace the faulty nodes as soon as possible by safe processors [23]. To manage the trade off between the number of nodes to be involved in the computation, the number of spares that should be on standby and the checkpoint period to maximize the expected fraction of time in which the system is executing useful work, the stationary distribution of the markov chain is computed with respect to computing pool cardinality, spare pool cardinality and the checkpoint interval. However, it worth noting that such result will be impossible to use in the context of HPC due the huge number of processors involved in the computation given that the computational complexity of the algorithm provided by Plank et al. is exponential with respect to the number of processors. In [19], Jin et al. provided an analytical model to express the expected completion time in presence of failures with respect to the number of the involved processors in the computation and the amount of workload to execute. Using the queuing theory and by applying Taylor expansion to the expression of the expected completion time of the application with respect to the involved processors in the computation they provided a first order approximation to the optimal checkpoint interval considering that the checkpoint cost is constant.

1.3 Contributions

The past literature provides different approximation solutions for determining the checkpoint intervals for a single processor under some simplified assumptions: unbounded execution time, preemptive execution process, failure-free checkpoints, constant checkpoint costs, and failure distributions with increasing failure rates. As we have previously seen that most of the proposed algorithms for the arbitrary failure distribution case are based on a numerical optimization approach. It is well worth mentioning the fact that such technique of optimization requires some mathematical properties that the objective function should satisfy like the continuity or the existence of the first derivative. Which is not always true for example if we consider an arbitrary checkpoint cost function. Moreover, most of the proposed algorithms are without any guarantee on the time needed to compute the optimal checkpointing solution.

In this paper, we fill this gap by studying the general problem, considering an arbitrary failures distribution function, checkpoints costs are not constant and parallel blocks with different durations. We provide a new computational complexity analysis that explicits in depth the relation between the probabilistic fault model, the checkpoint cost and the computational model. Finally, we present a dynamic programming scheme for all the variants of the problem that provides an optimal checkpointing schedule.

In Section 2, we propose a new discrete mathematical formulation for the checkpoint scheduling problem for parallel application in HPC platforms. This formulation aims at minimizing the expected wasted time considering the variability on checkpoint costs and the time to detect failures. We show in Section 3 that the described problem of checkpoint scheduling is NP-complete even for the case of simple failure distributions like the uniform distribution. However, we exhibit some special cases that are polynomially solvable. Finally, we propose in Section 4 an optimal scheduling strategy for the different variants of the problem with a low computational cost. It is interesting to notice that according to our knowledge there exist no related works taking into account general failure distribution, variable checkpoint costs and latency on the time needed to detect failures.

2 Performance Models

In this section, we present the performance model that will be used to express the problem of optimizing the tradeoff between the checkpoints overhead and the lost computation time due to failures. In what follows we present the main components of this model, namely, the execution model and the application failure model. Then, we exhibit formally the optimization problem.

2.1 Execution Model

In this work, we are interested in efficient implementations of parallel applications on HPC platforms. The target parallel platform is a cluster composed of identical processors. Many HPC scientific programs are implemented in a series of synchronized computing blocks (iterative numerical methods [8], *super steps* of programs written in the BSP paradigm [31], and so on). The basic computations within each block are evenly distributed over all the processors. At the end of the execution of each block, the processors exchange their results and a new block can start its computations. Fig. 1 represents the basic computation scheme for the execution of a series of n parallel blocks on Q processors where block i has a processing time p_i. Its execution can only start after the completion of block $(i-1)$.

We assume that the blocks are not preemptive. Thus, the checkpoints can only be performed between blocks. The ith checkpoint concerns the intermediary results between the ith and $(i+1)$th blocks. According to most actual applications [16],

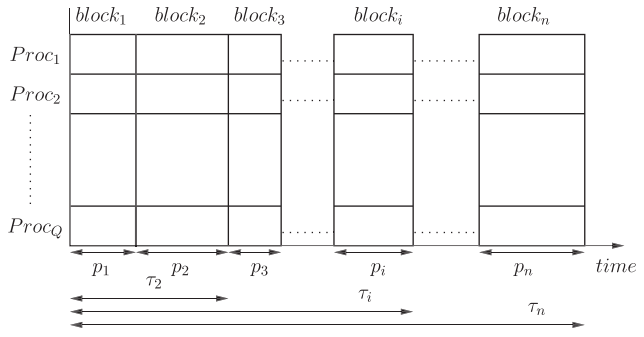

Fig. 1. Execution scheme.

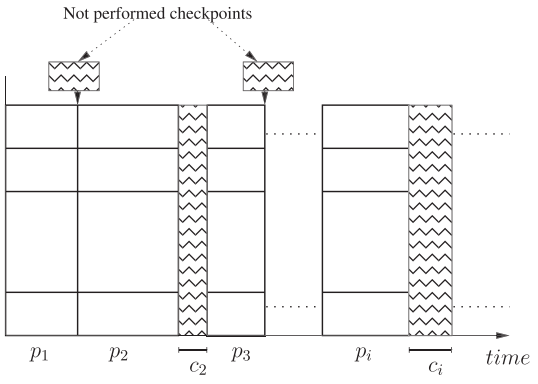

Fig. 2. Execution with checkpoints.

the amount of data to save between the ith and $(i+1)$th blocks depends on the ith block. Therefore, checkpointing is a global operation that takes c_i units of time if it is performed after the ith block. For the checkpoint failure model, we suppose that the checkpoint operations are not failure-free as it is usually considered in the literature [33], [9], [14] and that the time spent in this phase increases the failure probability of the application.

Furthermore, we assume that checkpoints can also be used for the purpose of failure detection if there is no dedicated failure detector mechanism. Technically, checkpointing mechanisms in parallel middleware [6], [25], [18], [3] rely on a global synchronization phase before starting the creation of the checkpoint files. If there exists no dedicated detection mechanisms into the middleware, it is possible to detect failures using the coordination phase into the global synchronization in the checkpoint barrier. Therefore, if failures are not detected immediately, the time needed to detect a failure depends on the next scheduled checkpoint after the actual failure. However, remark that if failures are detected immediately, the proposed model remains valid as will be discussed in the next section. Fig. 2 depicts the execution scheme in the presence of checkpoints.

2.2 Failure Model

We present now the application failure distribution that will be used to build the performance model. In this section, we describe the different methods that can be used to express the failure distribution of the application. More precisely, the application failure distribution $F(x)(resp.\ \overline{F}(x))$ expresses the probability that the application will fail (*resp.* will not fail) in the next x units of time ($\overline{F}(x) = 1 - F(x)$). Most HPC applications executed on parallel platforms are highly synchronized and they immediately fail if only one processor failure occurs. Then, all surviving application processes are aborted and the whole application is restarted from the last checkpoint. Moreover, upon a given failure, the faulty processor is either replaced by a new one or the failure root-cause is fixed immediately and this can be neglected. This is a reasonable assumption since an application usually uses only a part of the available processors on the large machine. Considering all these hypotheses, it is clear that the overall failure pattern of the platform is the superposition of the failure patterns of the different processors where each processor has its own failure distribution and failure/repair history. Failure/repair history represents the duration since the processor has been put into operation after the last repair or replacement (i.e., processor's age).

We recall that $\overline{F}(x)$ is the probability that the execution will not fail in the next x units of time. This probability is equivalent to the probability that all the processors survive and not fail in the next x units of time. In addition, each processor has its own age (defined as the time since the last failure) and failure distribution function. Various methods can be used to express this probability function. We present the two most common methods. In the first method, the application failure distribution is expressed considering the exact processor ages. In the second method, the processor ages are not provided and the application failure distribution is expressed using the asymptotic remaining life distribution. In the remaining of this section, we present these two methods in more detail. Let us emphasize that while the results presented in this paper require a failure distribution, end users are free to use the distribution function of their choice.

Formally, suppose that the execution of the user application starts at time t. Each processor has its own failure/repair history denoted by $(h_1, h_2, \ldots, h_q)$, where h_q is the time since the last failure on processor q, failure distribution $G_q(x)$, and mean time to failure denoted by μ_q.

In the first method, the failure distribution of the application is given by the product over all the conditional failure distributions of the processors used by the application. Let $G_q(x|h_q)$ denote the probability that processor q fails in the next x units of time knowing that h_q units of time are elapsed since the last failure. Therefore, the conditional failure distribution of processor q is given by

$$G_q(x|h_q) = \frac{G_q(x+h_q) - G_q(h_q)}{\overline{G}_q(h_q)}.$$

The probability distribution $\overline{F}(x)$ that the application will not fail in the next x units of time is given by

$$\overline{F}(x) = \prod_{q=1}^{Q} \overline{G}_q(x|h_q). \tag{1}$$

For instance, suppose that the failures are exponentially distributed with $G_q(x) = 1 - exp^{-x/\mu_q}$. In this case due to the memoryless property of the exponential distribution, the application failure distribution is not history dependent and is given by

$$\overline{F}(x) = exp^{-(x/\sum_{q=1}^{Q}\mu_q)}. \tag{2}$$

In the second method, instead of using the failure/repair history of each processor to express the failure distribution of the application, the asymptotic remaining life distribution of each processor is used. We notice that in most HPC platforms the exact failure/repair history of each processor is not provided to the users. Therefore, it will be more convenient to express the failure distribution using this method. Various approaches were proposed to derive the distribution of the remaining life of a single processor and a survey may be found in [1]. We present below the asymptotic distribution when the processors are considered in the steady state regime (i.e., when $t \to \infty$). Let $R_q(t)$ denote the remaining life of the processor in use at time t. Assume that the next failure on the qth processor will occur at time T_q with $(T_q > t)$ we have $R_q(t) = T_q - t$. Suppose now that all the processors are in the steady state regime. The asymptotic probability distribution of the remaining life of the qth processor is given by

$$\lim_{t\to\infty} \mathbb{P}[R_q(t) > x] = \frac{1}{\mu_q}\int_x^{\infty} \overline{G}_q(u)du.$$

Thus, the probability that the application will not fail in the next x units of time, is given by the probability that the remaining life of each processor is greater than as given in x.

$$\overline{F}(x) = \prod_{q=1}^{Q}\left[\frac{1}{\mu_q}\int_x^{\infty} \overline{G}_q(u)du\right]. \quad (3)$$

Notice that using this method, if failures are exponentially distributed, the global application failure distribution will be the same as the distribution obtained with the first method in (2).

Therefore, with regard to the availability of the information about processors failure/repair history, the user can decide which failure distribution to use for computing the checkpoint scheduling strategy. Let us finally remark that $dF(t)$ exists if all the $G_q(x)$ distribution functions are derivable that implies the existence of a general application failure density.

2.3 Description of the Optimization Problem

With the failure distributions and the execution model defined in previous sections, we are now able to formally describe the checkpoint scheduling problem.

Usually, the most common objective in HPC is to minimize the completion time of the application. To achieve this objective, we focus on the minimization of the wasted time as a pertinent dependability metric. It corresponds to the overhead time that will be added to the completion time of the application without failures. More precisely, the wasted time is composed of three terms: First, the overhead induced by checkpoints themselves which will increase the completion time. Second, the amount of work which is lost after a failure between the last checkpoint and the actual failure time. Finally, as the failures are not necessarily detected immediately, there is another delay to consider corresponding to the amount of time since the actual failure until the moment when it is detected.

To be more precise, an instance of the optimization checkpoint scheduling problem denoted in short by CS_F is defined as follows: The inputs of the problem are $2n$ positive rational numbers p_i and c_i for $1 \leq i \leq n$, where p_i is the processing time of the ith block and c_i is the overhead that will delay the completion of the application if a checkpoint is performed after block i. The objective is to provide a schedule that minimizes the mathematical expectation of the wasted time. The schedule vector is defined by $A = \{a_1, \ldots, a_n\} \in \{0,1\}^n$, where a_i is a decision variable ($a_i = 1$ if a checkpoint is scheduled after the ith block and $a_i = 0$ otherwise). We remark that the failure probability distribution $F(x)$ which is a mapping from $\mathbb{R}^+$ to $[0,1]$ is also considered as an input of the problem.

Let us detail below the mathematical expression of the expected wasted time denoted by $W_{(A)}$ and composed of the three following terms:

- The first term is the expected checkpoint overhead. Let σ_i denote the cumulative checkpoint overhead from the beginning until the completion of the ith block. Based on this definition, σ_i is given by $\sigma_i = \sum_{j=1}^{i} a_j c_j$ with $\sigma_0 = 0$. Let τ_i denote the useful computing time from the beginning until the ith block such that $\tau_i = \sum_{j=1}^{i} p_j$ with $\tau_0 = 0$ and $dF(x)$ denotes the derivative of the application failure distribution. Therefore, by conditioning that the failure occurs during the block $i+1$ at time x (i.e., $\tau_i + \sigma_i \leq x < \tau_{i+1} + \sigma_{i+1}$). The expected checkpoint overhead is given by the following expression:

$$C_{(A)} = \sum_{i=0}^{n-1}\int_{\tau_i+\sigma_i}^{\tau_{i+1}+\sigma_{i+1}} \sigma_i dF(x). \quad (4)$$

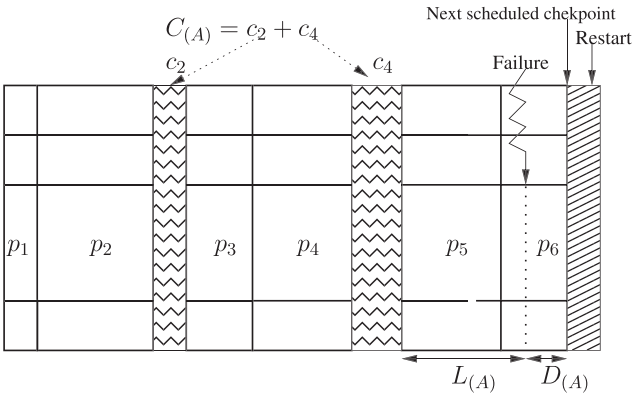

Fig. 3. Typical execution.

- The second term represents the expected lost computation due to failure. We recall briefly that after a failure, all the computation results since the last checkpoint are lost. Thus, the lost computation is the amount of work since the last checkpoint until the actual failure time (see Fig. 3). In the same way, we condition that the failure occurs after the execution of the ith block and before the completion of the $i+1$th one at time x. Also the last performed checkpoint before the block $i+1$ is executed at time $\max_{j\leq i}(a_j\tau_j + \sigma_j)$. Hence, the amount of lost work is $x - \max_{j\leq i}(a_j\tau_j + \sigma_j)$. The expected lost work is given by

$$L_{(A)} = \sum_{i=0}^{n-1}\int_{\tau_i+\sigma_i}^{\tau_{i+1}+\sigma_{i+1}} (x - \max_{j\leq i}(a_j\tau_j + \sigma_j))dF(x). \quad (5)$$

- The last term is the failure detection latency which is the elapsed time from the actual failure time until the time of the failure detection. As pointed out in section a detection mechanism is included into each checkpoint. Let $next(i)$ denote the time of the next checkpoint after the ith block. $next(i)$ is given by

$$next(i) = \max\left(a_{i+1}\tau_{i+1}, \max_{i+1<j\leq n}\left(a_j\tau_j \prod_{k=i+1}^{j-1}\overline{a}_k\right)\right).$$

For instance suppose that the next checkpoint scheduled after the ith block is performed after the block $i+d$ such that $i+1 < i+d \leq n$. Therefore, all the a_k are equal to 0 for $j = i+1$ to $j = i+d-1$. Hence, $a_j\tau_j\prod_{k=i+1}^{j-1}\overline{a}_k > 0$ only for $j = i+d$. Suppose that a failure occurs at time x such that $\tau_i + \sigma_i \leq x < \tau_{i+1} + \sigma_{i+1}$. The detection latency is given by $next(i) + \sigma_i - x$. By conditioning on failure times we obtain the following expression for the expected time to detect failures:

$$D_{(A)} = \sum_{i=0}^{n-1}\int_{\tau_i+\sigma_i}^{\tau_{i+1}+\sigma_{i+1}} (next(i) + \sigma_i - x)dF(x). \quad (6)$$

As a summary, the expected amount of wasted time is the summation of (4), (5), and (6).

$$W_{(A)} = C_{(A)} + L_{(A)} + D_{(A)}. \quad (7)$$

3 Complexity Analysis

With the checkpoint scheduling problem now formally defined, we propose to start its study with a computational complexity analysis. Formally, the decision version CS_F of the *checkpoint scheduling* problem is expressed as follows:

INPUT: $2n+1$ positive rational numbers $\{p_1, \ldots, p_n\}$ $\{c_1, \ldots, c_n\}$ and W that corresponds to a target to achieve.

QUESTION: Does it exist a schedule $A = \{a_1, \ldots a_n\}$ such that $W_{(A)} \leq W$?

Clearly, the general objective function $W_{(A)}$ is a complicated nonlinear function due to the general failure distribution function and the max operator. Therefore, without loss of generality to prove the NP-completeness of the problem we consider a particular failure distribution and a subfamily of instances as described in the next section.

3.1 Subfamily of Instances

We describe in what follows the particular failure distribution and the considered subfamily of instances that will be used to prove the NP-completeness of the problem.

- *Detection latency.* First of all, we start by assuming that the detection latency is null (i.e., $D_{(A)} = 0$). We notice that this hypothesis leads to the classical problem formulation in the literature where failures are supposed to be detected immediately. Therefore, the objective function is given by

$$W_{(A)} = \sum_{i=0}^{n-1} \int_{\tau_i+\sigma_i}^{\tau_{i+1}+\sigma_{i+1}} \left(x - \left(\max_{j \leq i} a_j \tau_j\right)\right) dF(x). \quad (8)$$

- *Uniform distribution.* Usually, it is well established that the most convenient failure distribution in terms of mathematical manipulation is the uniform distribution. Thus, we consider the particular problem such that $F(x) = x$ for $x \in [0, 1[$. This enables to simplify (8) of the expected wasted time and $W_{(A)}$ is given by

$$\begin{aligned} W_{(A)} = & \frac{1}{2}(\tau_n + \sigma_n)^2 \\ & - \sum_{i=0}^{n-1} (p_{i+1} + a_{i+1} c_{i+1}) \left(\max_{j \leq i} a_j \tau_j\right). \end{aligned} \quad (9)$$

  This variant of the problem with the uniform failure distribution is denoted by CS_U instead of CS_F.
- *Checkpoint cost.* Finally, the last hypothesis concerns the checkpoint costs. Intuitively, the idea is to replace the max operator by a linear operator. To simplify the objective function, we consider a subfamily of instance where the checkpoint cost is null for the blocks with even index. Therefore, it is clear that such a checkpoint with null cost does not have any impact on the accumulated checkpoint overhead when it is performed while it reduces the lost work in case of failure. Thus, we are able to prove the following lemma.

Lemma 1. *In any optimal schedule that minimizes $W_{(A)}$ we have: $c_i = 0 \Rightarrow a_i = 1$.*

Proof. Considering a particular instance where the ith checkpoint has a null cost $c_i = 0$: Let $\tilde{A} = <\tilde{a}_1, \ldots, \tilde{a}_i, \ldots \tilde{a}_n>$ and $\tilde{a}_i = 0$. Suppose by contradiction that $\tilde{A}$ is an optimal schedule for this instance, such that, $\forall A$, $W_{(\tilde{A})} \leq W_{(A)}$. Let $A' = \tilde{A}$ except for the ith index such that, $(a'_i = 1 - \tilde{a}_i = 1)$. Therefore, based on the (9), we have

$$W_{(A')} = W_{(\tilde{A})} + \frac{c_i^2}{2} + c_i \sum_{j \neq i} (a_j c_j) - \tau_i (p_{i+1} + a_{i+1} c_{i+1}).$$

Recall that $c_i = 0$. Thus,

$$\frac{c_i^2}{2} + c_i \sum_{j \neq i} (a_j c_j) - \tau_i (p_{i+1} + a_{i+1} c_{i+1}) < 0.$$

Which leads to the contradiction $W_{(A')} < W_{(\tilde{A})}$. □

Thus, without loss of generality, according to Lemma 1 we have for all blocks with even index $\forall i, c_{2i} = 0$ and $a_{2i} = 1$. This transformation leads to the simplification of the max operator as follows:

$$\max_{j \leq i} a_j \tau_j = \begin{cases} \tau_i, & if \quad i \text{ is even}, \\ a_i \tau_i + \overline{a}_i \tau_{i-1}, & if \quad i \text{ is odd}. \end{cases}$$

Let I denote the set of odd indices up to n. It is straightforward to verify that (9) becomes

$$W_{(A)} = \frac{(\tau_n + \sigma_n)^2}{2} - \sum_{i \in I} (\tau_{i-1}(p_i + p_{i+1}) + a_i(\tau_{i-1} c_i + p_i p_{i+1})). \quad (10)$$

Then, let $K = \tau_n^2/2 - \sum_{i \in I} \tau_{i-1}(p_i + p_{i+1})$, which is a constant with respect to any scheduling strategy. The objective function of the considered subfamily can be expressed as follows:

$$W_{(A)} = K + \sigma_n^2/2 - \sum_{i \in I} a_i c_i \left(\tau_{i-1} - \tau_n + \frac{p_i p_{i+1}}{c_i}\right). \quad (11)$$

3.2 NP-Completeness

We exhibit in this section a polynomial reduction for the presented subfamily of instances from the well-known Subset Sum problem which is NP-complete [13]. Formally, an instance of Subset Sum is given by

INPUT: m positive numbers $s_1, s_2, \ldots, s_m$, and an integer S.

QUESTION: Does there exist a subset denoted by E of $\{1, 2, \ldots, m\}$ such that $\sum_{i \in E} s_i = S$?

Theorem 1. *The checkpoint scheduling CS_U problem is NP-Complete.*

Proof. It is straightforward to verify that the decision version of the checkpoint scheduling problem belongs to NP. Since a deterministic algorithm in polynomial time bounded by $O(n)$ can be used to check if $W_{(A)} \leq W$ for any schedule A.

The reduction is based on a transformation of Subset Sum to CS_U.

- *Checkpoint cost mapping.* For each item $s_i (1 \leq i \leq m)$ create two checkpoints such that: $c_{2i-1} = s_i$ and $c_{2i} = 0$. Therefore, we remark that $n = 2m$.
- *Processing time mapping.* The idea is to set p_i values in such a way that

$$\begin{aligned} & \sum_{i=1}^{m} \left[a_{2i-1} c_{2i-1} \left(\tau_{2i-1} - \tau_n + \frac{p_{2i} p_{2i+1}}{c_{2i-1}} \right) \right] \\ & \quad = \sum_{i=1}^{m} [a_{2i-1} c_{2i-1} S]. \end{aligned} \quad (12)$$

  Therefore, $\forall i, 1 \leq i \leq m$ create two blocks with $p_{2i} = 2nS$ and $p_{2i-1} = c_{2i-1}(2n - \sum_{j=1}^{i-1}(1 + \frac{p_{2j-1}}{2nS}))$.
- *Target mapping.* Finally, the target is defined by $W = K - S^2/2$.

Suppose now that Subset Sum has a positive instance denoted by E where $\sum_{i \in E} s_i = S$.

We consider the following schedule A given by: $\forall i, 1 \leq i \leq m$ $a_{2i} = 1$ (such that all the checkpoints with even indices are performed). $\forall i, 1 \leq i \leq m$ if $i \in E$ then $a_{2i-1} = 1$ else $a_{2i-1} = 0$.

We mention that, by construction $\forall i \in I$ we have $\tau_{2(i-1)} - \tau_{2m} + \frac{p_{2i}p_{2i-1}}{c_{2i-1}} = S$ and $\sigma_{2m} = S$ therefore $W_{(A)} = K + \sigma_{2m}^2/2 - \sigma_{2m}S = K - S^2/2 = W$ which is a positive instance for CS_U.

Conversely, suppose now that CS_U has a positive instance A such that $W_{(A)} \leq W$. Therefore, $K + \sigma_n^2/2 - S\sigma_n \leq K - S^2/2$. As the inequality $1/2(\sigma_n - S)^2 \leq 0$ has one solution where $\sigma_n = S$. Therefore, Subset Sum has a positive solution. □

This completes the proof that CS_F is weakly NP-complete and thus, we cannot hope for better than pseudopolynomial algorithms unless $P = NP$.

4 Algorithm Description

We present in this section a pseudopolynomial algorithm for scheduling the checkpoints with variable costs for any failure distribution and prove its optimality. We show also that the problem is fully polynomial in the restricted case of constant checkpoint costs.

4.1 General Case

We present in this section a scheduling algorithm for solving the general problem. This algorithm is based on a dynamic programming approach that provides an optimal schedule minimizing the objective function given in (7). We start by introducing some notations.

Let Π_i^θ be the set of schedules A_{i,σ_i} verifying the following constraints.:

$$A_{i,\sigma_i} = \{a_1, \ldots, a_i\} \in \Pi_i^\theta \text{ iff } \begin{cases} \sum_{j=1}^{i} a_j c_j = \theta \\ a_i = 1. \end{cases}$$

More precisely, A_{i,σ_i} is a checkpoint schedule for the i first blocks, such that the cumulative checkpoint overhead until the i first blocks is exactly equal to $\sigma_i = \theta$ and a checkpoint is necessarily scheduled after the ith block that is $a_i = 1$.

It is clear that Π_i^θ is a finite set which implies the existence of an optimal schedule A_{i,σ_i}^* with the minimal expected waste time value over this set. Let $W_{i,\theta}^*$ denoted this optimal value such that $W_{(A_{i,\sigma_i}^*)} = W_{i,\theta}^*$.

We also introduce r that represents the block index of the penultimate checkpoint in A_{i,σ_i} (i.e., the last checkpoint before the ith one) such that r is given by $r = \arg\max_{j<i} a_j\tau_j$. Based on this definition, we introduce the schedule A'_{r,σ_r} such that A'_{r,σ_r} is subvector of A_{i,σ_i} for values with indices between 1 and r. More accurately, A'_{r,σ_r} is exactly the same scheduling solution as A_{i,σ_i} for checkpoints up to r ($\forall j, 1 \leq j \leq r$ we have $a_j = a'_j$). This relation may be interpreted in the following way: The scheduling solution for checkpoints considering only the i first blocks with cumulative checkpoint overhead $\sigma_i = \theta$ can be obtained from a scheduling solution for checkpoints with cumulative checkpoint overhead $\sigma_r = \theta - c_i$ for blocks up to r. Considering this construction the expected wasted time in (7) can be expressed as follows:

Let $A_{i,\sigma_i} \in \Pi_i^\theta$ and $\Delta(r,i) = \int_{\tau_{r+\sigma_r}}^{\tau_i+\sigma_i} (\tau_i - \tau_r + \sigma_i) dF(t)$. We obtain

$$W_{(A_{i,\sigma_i})} = W_{(A'_{r,\theta-c_i})} + \Delta(r,i). \tag{13}$$

Based on this construction, we are able to establish the following proposition.

Proposition 1. *If A_{i,σ_i} is the optimal schedule in Π_i^θ then A'_{r,σ_r} is the optimal schedule in $\Pi_r^{\theta-c_i}$.*

Proof. Let $A_{i,\sigma_i} \in \Pi_i^\theta$ and suppose that $W(A_{i,\sigma_i}) = W_{i,\theta}^*$. First, it is straightforward to see that by construction we have $A'_{r,\sigma_r} \in \Pi_r^{\theta-c_i}$. Suppose now by contradiction that A'_{r,σ_r} is not optimal over the set $\Pi_r^{\theta-c_i}$. Thus, it exists a valid schedule $\hat{A}_{r,\sigma_r} \in \Pi_r^{\theta-c_i}$ such that $W_{(\hat{A}_{r,\sigma_r})} < W_{(A'_{r,\sigma_r})}$. Therefore, $W_{(\hat{A}_{r,\sigma_r})} + \Delta(r,i) < W_{(A_{i,\sigma_i})} < W_{i,\theta}^*$. This leads to a contradiction. □

Let W^* denote the global minimal wasted time for the general problem. It is clear that $W^* = \min_{\theta\in\Theta} W_{n,\theta}^*$ corresponds to the minimum taken over all possible integer values of θ between the minimum and the maximum checkpoint overhead for instance $\Theta = [0, \sum_{i=1}^n c_i]$. Using a recursion on the last checkpoint, we can directly derive from the proposed Proposition 1 the following dynamic programming scheme:

$$\begin{aligned} &\text{Initialization} : \forall i \leq n, \\ &W_{i,c_i}^* = (\tau_i) F(\tau_i + c_i), \\ &\forall \theta > c_i \ : \ W_{i,\theta}^* = \infty, \\ &\text{Recurrence} : \forall i \leq n \text{ and } \theta < \sum_{j=1}^{i} c_j, \\ &W_{i,\theta}^* = \min_{1\leq r<i} \left(W_{r,\theta-c_i}^* + \Delta(r,i) \right). \end{aligned} \tag{14}$$

This algorithm leads to an optimal schedule. It is pseudopolynomial time in $O(n^3 \times \max_i(c_i))$.

4.2 Constant Checkpoint Costs

Using a similar analysis, we show that when the checkpoint cost is constant, the previous algorithm can be simplified to an optimal polynomial-time algorithm. It is clear that when checkpoint cost is constant and equal to c. It is straightforward to remark that the cumulative checkpoint overhead is necessarily a multiple of c ($\sigma_i = lc$). Thus, considering a given checkpoint scheduling policy, the cumulative checkpoint overhead is fully characterized by the number of checkpoints l in this schedule. Thus, in this case the range of l is reduced to integer interval $[1, n]$. Using Proposition 1, the optimal scheduling is obtained after computing all the values $W_{i,l}^*$ for $1 \leq i \leq n$ and $1 \leq l \leq i$ using the following recurrence equations:

$\forall i \leq n$ and $l \leq i$,
$W_{i,l}^* = \min_{l-1\leq r<i}(W_{r,l-1}^* + \Delta(r,i))$,
where $W_{i,1}^* = (\tau_i + c)F(\tau_i + c) \forall i \leq n$.

This algorithm is polynomial and its computational complexity is in $O(n^3)$.

5 Concluding Remarks

We have presented in this paper a new combinatorial approach for scheduling checkpoints for HPC applications in parallel platforms. We have proven that the general problem with any failure distribution and variable checkpoint costs is NP-complete even in the case of uniform failure distribution. We have also proposed a dynamic programming algorithm for solving this problem. This algorithm leads to a low-cost polynomial time algorithm when the checkpoints costs are constant.

We believe that this work could serve as a solid basis for many further studies. For instance, it is clear that for nonpreallocated blocks, checkpoint scheduling, and block scheduling are strongly interleaved. Thus, the dynamic programming scheme can be used as a basic step for designing a mixed solution. Another extension of this work is to include advanced failure detection mechanisms into the proposed scheme. Finally, it may be interesting to investigate the effects of the steady state regime hypothesis on the failure distribution. In particular, it would be interesting to study eventual performance gaps between schedules obtained with the different failure distribution models.

REFERENCES

[1] R.E. Barlow, F. Proschan, and L.C. Hunter, *Mathematical Theory of Reliability.* SIAM, 1996.

[2] X. Besseron, M.S. Bouguerra, T. Gautier, É. Saule, and D. Trystram, "Numerical Analysis and Scientific Computing," *Fault Tolerance and Availability Awareness in Computational Grids,* ch. 5, Chapman and Hall/CRC Press, 2009.

[3] X. Besseron and T. Gautier, "Optimised Recovery with a Coordinated Checkpoint/Rollback Protocol for Domain Decomposition Applications," *Proc. Second Int'l Conf. Modelling, Computation and Optimization in Information Systems and Management Sciences (MCO),* pp. 497-506, 2008.

[4] M. Bougeret, H. Casanova, M. Rabie, Y. Robert, and F. Vivien, "Checkpointing Strategies for Parallel Jobs," Research Report RR-7520, INRIA, 04, 2011.

[5] M.S. Bouguerra, T. Gautier, D. Trystram, and J.M. Vincent, "A Flexible Checkpoint/Restart Model in Distributed Systems," *Proc. Eighth Int'l Conf. Parallel Processing and Applied Math.,* pp. 206-215, 2010.

[6] A. Bouteiller, T. Hérault, G. Krawezik, P. Lemarinier, and F. Cappello, "MPICH-V Project: A Multiprotocol Automatic Fault Tolerant MPI," *The Int'l J. High Performance Computing Applications,* vol. 20, pp. 319-333, 2006.

[7] F. Cappello, "Fault Tolerance in Petascale/Exascale Systems: Current Knowledge, Challenges and Research Opportunities," *Int'l J. High Performance Computing Applications,* vol. 23, no. 3, pp. 212-226, 2009.

[8] Z. Chen and J. Dongarra, "Highly Scalable Self-Healing Algorithms for High Performance Scientific Computing," *IEEE Trans. Computers,* vol. 58, no. 11, pp. 1512-1524, Nov. 2009.

[9] 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.

[10] T. Dohi, T. Ozaki, and N. Kaio, "Optimal Checkpoint Placement with Equality Constraints," *Proc. IEEE Second Int'l Symp. Dependable, Autonomic and Secure Computing,* pp. 77-84, 2006.

[11] A. Duda, "Effects of Checkpointing on Program Execution Time," *Information Processing Letters,* vol. 16, no. 5, pp. 221-229, 1983.

[12] E.N. Elnozahy and J.S. Plank, "Checkpointing for Peta-Scale Systems: A Look into the Future of Practical Rollback-Recovery," *IEEE Trans. Dependable Secure Computing,* vol. 1, no. 2, pp. 97-108, Apr.-June 2004.

[13] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* WH Freeman & Co., 1979.

[14] S. Garg, Y. Huang, C. Kintala, and K.S. Trivedi, "Minimizing Completion Time of a Program by Checkpointing and Rejuvenation," *Proc. ACM SIGMETRICS Int'l Conf. Measurement and Modeling of Computer Systems,* pp. 252-261, 1996.

[15] L.A.B. Gomez, N. Maruyama, F. Cappello, and S. Matsuoka, "Distributed Diskless Checkpoint for Large Scale Systems," *Proc. IEEE/ACM 10th Int'l Conf. Cluster, Cloud and Grid Computing,* pp. 63-72, 2010.

[16] R. Gupta, H. Naik, and P. Beckman, "Understanding Checkpointing Overheads on Massive-Scale Systems: Analysis of the IBMS Blue Gene/P System," *Int'l J. High Performance Computing Applications,* vol. 25, no. 2, pp. 180-192, 2011.

[17] E. Heien, D. Kondo, A. Gainaru, D. LaPine, B. Kramer, and F. Cappello, "Modeling and Tolerating Heterogeneous Failures in Large Parallel Systems," *Proc. Int'l Conf. for High Performance Computing, Networking, Storage and Analysis,* p. 45, 2011.

[18] J. Hursey, J.M. Squyres, T.I. Mattox, and A. Lumsdaine, "The Design and Implementation of Checkpoint/Restart Process Fault Tolerance for Open MPI," *Proc. IEEE Int'l Parallel and Distributed Processing Symp. (IPDPS),* Mar. 2007.

[19] H. Jin, Y. Chen, H. Zhu, and X.H. Sun, "Optimizing HPC Fault-Tolerant Environment: An Analytical Approach," *Proc. 39th Int'l Conf. Parallel Processing,* pp. 525-534, 2010.

[20] D. Nurmi, R. Wolski, and J. Brevik, "Model-Based Checkpoint Scheduling for Volatile Resource Environments," *Proc. Cluster,* 2004.

[21] T. Ozaki, T. Dohi, and N. Kaio, "Numerical Computation Algorithms for Sequential Checkpoint Placement," *Performance Evaluation,* vol. 66, no. 6, pp. 311-326, 2009.

[22] T. Ozaki, T. Dohi, H. Okamura, and N. Kaio, "Distribution-Free Checkpoint Placement Algorithms Based on Min-Max Principle," *IEEE Trans. Dependable Secure Computing,* vol. 3, no. 2, pp. 130-140, Apr.-June 2006.

[23] J.S. Plank and M.G. Thomason, "Processor Allocation and Checkpoint Interval Selection in Cluster Computing Systems," *J. Parallel and Distributed Computing,* vol. 61, no. 11, pp. 1570-1590, 2001.

[24] X. Ren, R. Eigenmann, and S. Bagchi, "Failure-Aware Checkpointing in Fine-Grained Cycle Sharing Systems," *Proc. 16th Int'l Symp. High Performance Distributed Computing,* pp. 33-42, 2007.

[25] S. Sankaran, J.M. Squyres, B. Barrett, A. Lumsdaine, J. Duell, P. Hargrove, and E. Roman, "The LAM/MPI Checkpoint/Restart Framework: System-Initiated Checkpointing," *Int'l J. High Performance Computing Applications,* vol. 19, no. 4, pp. 479-493, 2005.

[26] E. Saule and D. Trystram, "Analyzing Scheduling with Transient Failures," *Information Processing Letters,* vol. 109, no. 11, pp. 539-542, 2009.

[27] B. Schroeder and G.A. Gibson, "Understanding Failures in Petascale Computers," *J. Physics: Conf. Series,* vol. 78, pp. 012-022, 2007.

[28] B. Schroeder and G.A. Gibson, "A Large-Scale Study of Failures in High-Performance Computing Systems," *IEEE Trans. Dependable and Secure Computing,* vol. 7, no. 4, pp. 337-351, Oct. 2010.

[29] S. Toueg and Ö. Babaoğlu, "On the Optimum Checkpoint Selection Problem," *SIAM J. Computing,* vol. 13, no. 3, pp. 630-649, 1984.

[30] N.H. Vaidya, "Impact of Checkpoint Latency on Overhead Ratio of a Checkpointing Scheme," *IEEE Trans. Computers,* vol. 46, no. 8, pp. 942-947, Aug. 1997.

[31] L.G. Valiant, "A Bridging Model for Parallel Computation," *Comm. ACM,* vol. 33, no. 8, pp. 103-111, 1990.

[32] X. Lin, Y. Ling, and J. Mi, "A Variational Calculus Approach to Optimal Checkpoint Placement," *IEEE Trans. Computers,* vol. 50, no. 7, pp. 699-708, July 2001.

[33] J.W. Young, "A First Order Approximation to the Optimum Checkpoint Interval," *Comm. ACM,* vol. 17, no. 9, pp. 530-531, 1974.