Assessing general-purpose algorithms to cope with fail-stop and silent errors

Anne Benoit, Aurélien Cavelan, Yves Robert, Hongyang Sun

Assessing general-purpose algorithms to cope with fail-stop and silent errors

Anne Benoit, Aurélien Cavelan, Yves Robert, Hongyang Sun

RESEARCH REPORT

N° 8599

19 September 2014

Project-Teams ROMA

ISSN 0249-6399 ISRN INRIA/RR--8599--FR+ENG

Assessing general-purpose algorithms to cope with fail-stop and silent errors

Anne Benoit*, Aurélien Cavelan*, Yves Robert*†,
Hongyang Sun*

Project-Teams ROMA

Research Report n° 8599 — 19 September 2014 — 42 pages

* Ecole Normale Superieure de Lyon, CNRS & INRIA, France
† University of Tennessee Knoxville, USA

Abstract: In this paper, we combine the traditional checkpointing and rollback recovery strategies with verification mechanisms to cope with both fail-stop and silent errors. The objective is to minimize makespan and/or energy consumption. For divisible load applications, we use first-order approximations to find the optimal checkpointing period to minimize execution time, with an additional verification mechanism to detect silent errors before each checkpoint, hence extending the classical formula by Young and Daly for fail-stop errors only. We further extend the approach to include intermediate verifications, and to consider a bi-criteria problem involving both time and energy (linear combination of execution time and energy consumption). Then, we focus on application workflows whose dependence graph is a linear chain of tasks. Here, we determine the optimal checkpointing and verification locations, with or without intermediate verifications, for the bi-criteria problem. Rather than using a single speed during the whole execution, we further introduce a new execution scenario, which allows for changing the execution speed via dynamic voltage and frequency scaling (DVFS). In this latter scenario, we determine the optimal checkpointing and verification locations, as well as the optimal speed pairs for each task segment between any two consecutive checkpoints. Finally, we conduct an extensive set of simulations to support the theoretical study, and to assess the performance of each algorithm, showing that the best overall performance is achieved under the most flexible scenario using intermediate verifications and different speeds.

Key-words: HPC, resilience, checkpoint, verification, failures, fail-stop error, silent data corruption, silent error

Evaluation d'algorithmes génériques tolérant pannes et erreurs silencieuses

Résumé : Dans cet article, nous combinons les techniques traditionnelles de prise de points de sauvegarde (checkpoint) avec des mécanismes de vérification afin de prendre en compte à la fois les pannes et les corruptions mémoire silencieuses. L'objectif est soit de minimiser le temps d'exécution, soit de minimiser la consommation d'énergie. DVFS est une approche populaire pour réduire la consommation d'énergie, mais utiliser des vitesses ou des fréquences trop basses peut accroître le nombre d'erreurs, et ainsi compliquer le problème. Nous considérons des applications dont le graphe de dépendance est une chaîne de tâches, et nous étudions trois scénarios: (i) une seule vitesse est utilisée pendant toute la durée de l'exécution; (ii) une seconde vitesse, pouvant être plus élevée, est utilisée pour toutes les ré-exécutions potentielles; (iii) différentes paires de vitesses peuvent être utilisées entre deux checkpoints pendant l'exécution. Pour chaque scénario, nous déterminons le placement optimal des checkpoints et des vérifications (et les vitesses optimales pour le troisième scénario) afin de minimiser l'un ou l'autre des objectifs. Les différents scénarios d'exécution sont ensuite testés et comparés à l'aide de simulations.

Mots-clés : HPC, tolérance aux erreurs, point de sauvegarde, checkpoint, vérification, pannes, erreur fail-stop, corruption silencieuse de données, erreur silencieuse

1 Introduction

For HPC applications, scale is a major opportunity. Massive parallelism with 100,000+ nodes is the most viable path to achieving sustained petascale performance. Future platforms will enrol even more computing resources to enter the exascale era.

Unfortunately, scale is also a major threat. Resilience is the first challenge. Even if each node provides an individual MTBF (Mean Time Between Failures) of, say, one century, a machine with 100,000 such nodes will encounter a failure every 9 hours on average, which is smaller than the execution time of many HPC applications. Furthermore, a one-century MTBF per node is an optimistic figure, given that each node is composed of several hundreds of cores. Worse, several types of errors need to be considered when computing at scale. In addition to the classical fail-stop errors (such as hardware failures), silent errors (a.k.a. silent data corruptions) constitute another threat that cannot be ignored any longer [32, 46, 44, 45, 30].

Another challenge is energy consumption. The power requirement of current petascale platforms is that of a small town, hence measures must be taken to reduce the energy consumption of future platforms. A popular technique is dynamic voltage and frequency scaling (DVFS): modern processors can run at different speeds, and lower speeds induce bigger savings in energy consumption. In a nutshell, this is because the dynamic power consumed when computing at speed s is proportional to s^3, while execution time is proportional to $1/s$. As a result, computing energy (which is the product of time and power) is proportional to s^2. However, static power must also be accounted for, and it is paid throughout the duration of the execution, which calls for a shorter execution (at higher speeds). Overall, there are trade-offs to be found, but in most practical settings, using lower speeds reduces the global energy consumption.

To further complicate the picture, energy savings have an impact on resilience. Obviously, the longer the execution time, the higher the expected number of errors, hence using a lower speed to save energy may well induce extra time and overhead to cope with more errors throughout execution. Even worse (again!), lower speeds are usually obtained via lower voltages, which themselves induce higher error rates and further increase the latter overhead.

In this paper, we introduce a model that addresses both challenges: resilience and energy consumption. In addition, we address both fail-stop and silent errors, which, to the best of our knowledge, has only been achieved before through costly replication techniques [31]. While checkpoint/restart [10, 18] is the de-facto recovery technique for dealing with fail-stop errors, there is no widely adopted general-purpose technique to cope with silent errors. The problem with silent errors is *detection latency*: contrarily to a fail-stop error whose detection is immediate, a silent error is identified only when the corrupted data is activated and/or leads to an unusual application behavior. However, checkpoint and rollback recovery assumes instantaneous error detection, and this raises a new difficulty: if the error stroke before the last checkpoint, and is detected after that checkpoint, then the checkpoint is corrupted and cannot be used to restore the application. To solve this problem, one may envision to keep several checkpoints in memory, and to restore the application from the last *valid* checkpoint, thereby rolling back to the last *correct* state of the application [28]. This multiple-checkpoint approach has three major drawbacks. First, it is very demanding in terms of stable storage: each checkpoint typically represents a copy of the entire memory footprint of the application, which may well correspond to several terabytes. The second drawback is the possibility of fatal failures. Indeed, if we keep k checkpoints in memory, the approach assumes that the error that is currently detected did not strike before all the checkpoints still kept in memory, which would be fatal: in that latter case, all live checkpoints are corrupted, and one would have to re-execute the entire application from scratch. The probability of a fatal failure is evaluated in [3] for various error distribution laws and values of k. The third drawback of the approach is the most serious, and applies even without memory constraints, i.e., if we could store an infinite number of checkpoints in storage. The critical question is to determine which checkpoint is the last valid one. We need this information to safely recover from that point on. However, because of the detection latency, we do not know when the silent error has indeed occurred, hence we cannot identify the last valid checkpoint, unless some verification mechanism is enforced.

We consider such a verification mechanism in this paper. This approach is agnostic of the nature of the verification mechanism (checksum, error correcting code, coherence tests, etc.). It is also fully general-purpose, although application-specific information, if available, can always be used to decrease the cost of verification (see the overview of related work in Section 2 for examples). In this context, the simplest protocol is to take only verified checkpoints (VC), which corresponds to performing a verification just before taking each checkpoint. If the verification succeeds, then one can safely store the checkpoint. If the verification fails, it means that a silent error has struck since the last checkpoint, which was duly verified, and one can safely recover from that checkpoint to resume the execution of the application. Of course, if a fail-stop error strikes, we can also safely recover from the last checkpoint, just as in the classical checkpoint and rollback recovery method. We refer to this protocol as the VC-ONLY protocol, and it basically amounts to replacing the cost C of a checkpoint by the cost $V + C$ of a verification followed by a checkpoint. However, because we deal with two sources of errors, one detected immediately and the other only when we reach the verification, the analysis of the optimal strategy is more involved.

While taking checkpoints without verifications seems a bad idea (because of the memory cost, and of the risk of saving corrupted data), taking a verification without checkpointing may be interesting. Indeed, if silent errors are frequent enough, it is worth verifying the data in between two (verified) checkpoints, so as to detect a possible silent error earlier in the execution, and thereby re-executing less work. We refer to this protocol as the VC+V protocol, which allows for both verified checkpoints and intermediate verifications.

One major objective of this paper is to study both VC-ONLY and VC+V protocols, and to analytically determine the best balance of verifications between checkpoints so as to minimize makespan (total execution time) and/or energy consumption. To achieve this ambitious goal, we restrict to two simplified, yet realistic, application frameworks. First, we consider divisible load applications, which represent the standard framework to analyze resilience protocols, because checkpoints and verifications can be taken at any time during the execution. In this case, we focus on periodic computing patterns, and use first-order approximations to find the optimal checkpointing and verification periods. Our results extend the classical formula by Young [41] and Daly [12] to deal with both fail-stop and silent errors. Then, we consider application workflows consisting of a number of parallel tasks that execute on the platform, and that exchange data at the end of their execution. In other words, the task graph is a linear chain, and each task (except maybe the first one and the last one) reads data from its predecessor and produces data for its successor. This scenario corresponds to a high-performance computing application whose workflow is partitioned into a succession of (typically large) tightly-coupled computational kernels, each of which is identified as a task by the model. At the end of each task, we have the opportunity either to perform a verification of the task output or to perform a verification followed by a checkpoint. For such applications, we improve and extend the result by Toueg and Babaoglu [39] and derive the exact optimal solutions using dynamic programming algorithms, while accounting for both fail-stop and silent errors.

For both application frameworks, we show where to place checkpoints and verifications in order to minimize makespan, or a linear combination of makespan and energy, hence tackling the bi-criteria problem, when the whole application is executed at a single constant speed s. While we derive first-order approximations of the optimal solution for divisible load applications, we obtain exact optimal solutions for linear task chains. In addition, for linear task chains, we introduce a new execution scenario called MultiSpeed, which allows for changing the execution speed via DVFS. This advanced scenario uses two different speeds s and σ to execute the tasks in between two consecutive checkpoints (which we call a task segment). Within each segment, we use a speed s for the first execution, and a possibly different speed σ for all the re-executions after a fail-stop error or a silent error has occurred. Here, s can be considered as the regular speed, while σ corresponds to an adjusted speed to either speed up or slow down the re-executions, depending on the optimization objective. The speeds s and σ can be freely chosen among a set of K discrete speeds, and these speed pairs may well be different from one segment to another.

The main contributions of this paper are summarized as follows:

1. We introduce a general-purpose model to deal with both fail-stop and silent errors, combining the traditional checkpointing and rollback recovery strategies with verification mechanisms.
2. We express the objective function for all problems as a linear combination of execution time and consumed energy, in order to find optimal solutions for either time, or energy, or the bi-criteria problem.
3. We consider two resilience protocols: (i) VC-only, which uses only verified checkpoints, and (ii) VC+V, which uses both verified checkpoints and intermediate verifications.
4. For the divisible load application model, where checkpoints and verifications can be placed at any point in the execution of the application, we derive first-order approximations of the optimal checkpointing and verification periods.
5. For a linear chain of tasks, where checkpoints and verifications can be placed only at the end of the tasks, we provide optimal dynamic programming algorithms, both with a single speed and with an advanced scenario based on DVFS. In this later scenario, any two different speeds can be chosen within each segment between two checkpoints.
6. We conduct an extensive set of simulations to support the theoretical study and to assess the performance of each algorithm, hence demonstrating the quality and trade-off of our optimal algorithms under a wide range of parameter settings.

The rest of the paper is organized as follows. Section 2 provides an overview of related work. The next three sections deal with the main algorithmic contributions: Section 3 derives first-order approximations of the optimal checkpointing and verification periods for divisible load applications; Section 4 provides dynamic programming algorithms to solve exactly the problems for a linear chain of tasks; and Section 5 considers execution scenarios using different speeds for a linear chain of tasks. Then in Section 6, we report on a comprehensive set of simulations to assess the impact of each scenario and approach. Finally, we outline main conclusions and directions for future work in Section 7.

2 Related work

In this section, we discuss related work on fail-stop errors and silent errors, and finally we discuss the energy model and the impact of the execution speed on the error rate.

2.1 Fail-stop errors

The de-facto general-purpose error recovery technique in high performance computing is checkpoint and rollback recovery [10, 18]. Such protocols employ checkpoints to periodically save the state of a parallel application, so that when an error strikes some process, the application can be restored back to one of its former states. There are several families of checkpointing protocols, but they share a common feature: each checkpoint forms a consistent recovery line, i.e., when an error is detected, one can rollback to the last checkpoint and resume execution, after a downtime and a recovery time.

Many models are available to understand the behavior of checkpoint/restart [41, 12, 33, 8]. For a divisible load application where checkpoints can be inserted at any point in execution for a nominal cost C, there exist well-known formulas due to Young [41] and Daly [12] to determine the optimal checkpointing period. For an application composed of a linear chain of tasks, which is also the subject of this paper, the problem of finding the optimal checkpoint strategy, i.e., of determining which tasks to checkpoint, in order to minimize the expected execution time, has been solved by Toueg and Babaoglu [39], using a dynamic programming algorithm.

One major contribution of this paper is to extend both the Young/Daly formulas [41, 12] and the result of Toueg and Babaoglu [39] to deal with silent errors in addition to fail-stop errors, and to minimize a linear combination of time and energy rather than to focus solely on time. Therefore, we also consider using several discrete speeds instead of a single one.

2.2 Silent errors

Most traditional approaches maintain a single checkpoint. If the checkpoint file includes errors, the application faces an irrecoverable failure and must restart from scratch. This is because error detection latency is ignored in traditional rollback and recovery schemes, which assume instantaneous error detection (therefore mainly targeting fail-stop failures) and are unable to accommodate silent errors. We focus in this section on related work about silent errors. A comprehensive list of techniques and references is provided by Lu, Zheng and Chien [28].

Considerable efforts have been directed at error-checking to reveal silent errors. Error detection is usually very costly. Hardware mechanisms, such as ECC memory, can detect and even correct a fraction of errors, but in practice they are complemented with software techniques. The simplest technique is triple modular redundancy and voting [29], which induces a highly costly verification. For high-performance scientific applications, process replication (each process is equipped with a replica, and messages are quadruplicated) is proposed in the RedMPI library [21]. Elliot et al. [17] combine partial redundancy and checkpointing, and confirm the benefit of dual and triple redundancy. The drawback is that twice the number of processing resources is required (for dual redundancy). As already mentioned, an approach based on checkpointing and replication is proposed in [31], in order to detect and enable fast recovery of applications from both silent errors and hard errors.

Application-specific information can be very useful to enable ad-hoc solutions, which dramatically decrease the cost of detection. Many techniques have been advocated. They include memory scrubbing [25] and ABFT techniques [24, 7, 38], such as coding for the sparse-matrix vector multiplication kernel [38], and coupling a higher-order with a lower-order scheme for PDEs [6]. These methods can only detect an error but do not correct it. Self-stabilizing corrections after error detection in the conjugate gradient method are investigated by Sao and Vuduc [36]. Heroux and Hoemmen [22] design a fault-tolerant GMRES capable of converging despite silent errors. Bronevetsky and de Supinski [9] provide a comparative study of detection costs for iterative methods.

A nice instantiation of the checkpoint and verification mechanism that we study in this paper is provided by Chen [11], who deals with sparse iterative solvers. Consider a simple method such as the PCG, the Preconditioned Conjugate Gradient method: Chen's approach performs a periodic verification every d iterations, and a periodic checkpoint every $d \times c$ iterations, which is a particular case of the VC+V approach with equi-distance verifications. For PCG, the verification amounts to checking the orthogonality of two vectors and to recomputing and checking the residual. The cost of the verification is small in front of the cost of an iteration, especially when the preconditioner requires much more flops than a sparse matrix-vector product.

As already mentioned, our work is agnostic of the underlying error-detection technique and takes the cost of verification as an input parameter to the model.

2.3 Energy model and error rate

Modern processors are equipped with *dynamic voltage and frequency scaling* (DVFS) capability. The total power consumption is the sum of the static/idle power and the dynamic power, which is proportional to the cube of the processing speed s [40, 5], i.e., $P(s) = P_{idle} + \beta \cdot s^3$, where $\beta > 0$. A widely used reliability model assumes that radiation-induced transient faults (soft errors) follow a Poisson process with an average arrival rate λ. The impact of DVFS on the error rate is, however, not completely clear.

On the one hand, lowering the voltage/frequency is believed to have an adverse effect on the system reliability [14, 43]. In particular, many papers (e.g., [43, 42, 4, 13]) have assumed the following exponential error rate model:

$$\lambda(s) = \lambda_0 \cdot 10^{\frac{d(s_{max}-s)}{s_{max}-s_{min}}} , \tag{1}$$

where λ_0 denotes the average error rate at the maximum speed s_{max}, $d > 0$ is a constant indicating the sensitivity of error rate to voltage/frequency scaling, and s_{min} is the minimum speed. This model suggests that the error rate increases exponentially with decreased processing speed, which is a result of decreasing the voltage/frequency and hence lowering the circuit's critical charge (i.e., the minimum charge required to cause an error in the circuit).

On the other hand, the failure rates of computing nodes have also been observed to increase with temperature [34, 20, 23, 37], which generally increases together with the processing speed (voltage/frequency). As a rule of thumb, Arrenhius' equation when applied to microelectronic devices suggests that the error rate doubles for every 10°C increase in the temperature [20]. In general, the mean time between failure (MTBF) of a processor, which is the reciprocal of failure rate, can be expressed as [37]:

$$MTBF = \frac{1}{\lambda} = A \cdot e^{-b \cdot T} ,$$

where A and b are thermal constants, and T denotes the temperature of the processor. Under the reasonable assumption that higher operating voltage/frequency leads to higher temperature, this model suggests that the error rate increases with increased processing speed.

Clearly, the two models above draw contradictory conclusions on the impact of DVFS on error rates. In practice, the impact of the first model may be more evident, as the temperature dependency in some systems has been observed to be linear (or even not exist) instead of being exponential [16]. Generally speaking, the processing speed should have a composite effect on the average error rate by taking both voltage level and temperature into account. In the experimental section of this paper (Section 6), we adopt a trade-off model and modify Equation (1) to include the impact of temperature. We use

$$\lambda(s) = \lambda_{ref} \cdot 10^{\frac{d \cdot |s_{ref} - s|}{s_{max} - s_{min}}} , \tag{2}$$

where $s_{ref} \in [s_{min}, s_{max}]$ denotes the reference speed with the lowest error rate λ_{ref} among all possible speeds in the range. Equation (2) leads to a U-shaped curve where the error rate increases when the speed is either too high or too low.

3 Divisible load applications

In this section, we consider divisible load applications, for which checkpoints can be taken at any instant. In the presence of fail-stop errors only, the classical formula due to Young [41] and

Table 1: Notations for divisible load applications.

Protocols	
VC-ONLY	Single-chunk pattern with final verified checkpoint
VC+V	Multi-chunk pattern with intermediate verifications and final verified checkpoint
$T(s)$	Pattern period, or duration of computations of the pattern at speed s
k	Number of verifications inside a VC+V pattern ($k = 1$ for a VC-ONLY pattern)
Time	
$V(s)$	Time needed for verification at speed s
C	Time needed for checkpoint
R	Time needed for recovery
Error rates	
$\lambda^F(s)$	Fail-stop error rate for a given speed s
$\lambda^S(s)$	Silent error rate for a given speed s
$p^F(s, L)$	Probability of a fail-stop error during an execution of duration L at speed s
$p^S(s, L)$	Probability of a silent error during an execution of duration L at speed s
Energy	
P_{idle}	Static/idle power dissipated when the platform is switched on
$P_{cpu}(s)$	Dynamic power spent by operating the CPU at speed s
P_{io}	Dynamic power spent by I/O transfers (checkpoints and recoveries)
$E^V(s)$	Energy needed for verification at speed s
E^C	Energy needed for checkpoint
E^R	Energy needed for recovery

Daly [12] gives the optimal checkpointing period to minimize execution time. In this section, we first extend their result to include both fail-stop and silent errors (Section 3.2). Then, we further extend the approach to include intermediate verifications (Section 3.3). Finally, we show how to solve the bi-criteria problem where the objective is to minimize a linear combination of execution time and consumed energy (Section 3.4). We start by detailing the framework for divisible load applications in Section 3.1.

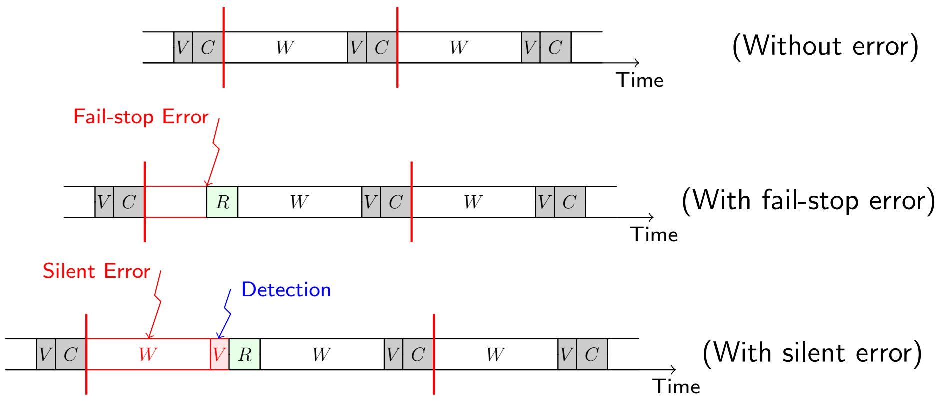


Figure 1: The VC-ONLY pattern executed at unit speed $s = 1$. The first figure shows the execution of a pattern without any error. The second figure shows that the execution is stopped immediately when a fail-stop error strikes, in which case the pattern is re-executed after a recovery. The third figure shows that the execution continues after a silent error strikes and it is detected by the verification at the end of the pattern. In this case, the pattern is also re-executed after a recovery.

3.1 Framework

In this section we introduce all model parameters for divisible load applications. For reference, main notations are summarized in Table 1. The platform is composed of p identical processors, which are subject to both fail-stop and silent-errors. A fundamental characteristic of the divisible application load model is that it allows us to view the platform as a single (very powerful but very error-prone) *macro-processor*, thereby providing a tractable abstraction of the problem.

Protocols

The application is partitioned into periodic patterns that repeat over time. Each pattern consists of some amount of work followed by a verified checkpoint, i.e., a verification immediately followed by a checkpoint. We consider two resilience protocols.

VC-ONLY: Placing only verified checkpoints. See Figure 1 for an illustration. The pattern consists of a single computational chunk of size W, which takes a time $T(s) = W/s$ to execute without any error at CPU speed s. The total duration of the pattern without any error is $T(s) + V(s) + C$, where $V(s)$ is the time to perform a verification at speed s, and C is the time to checkpoint. We assume that C does not depend upon s because

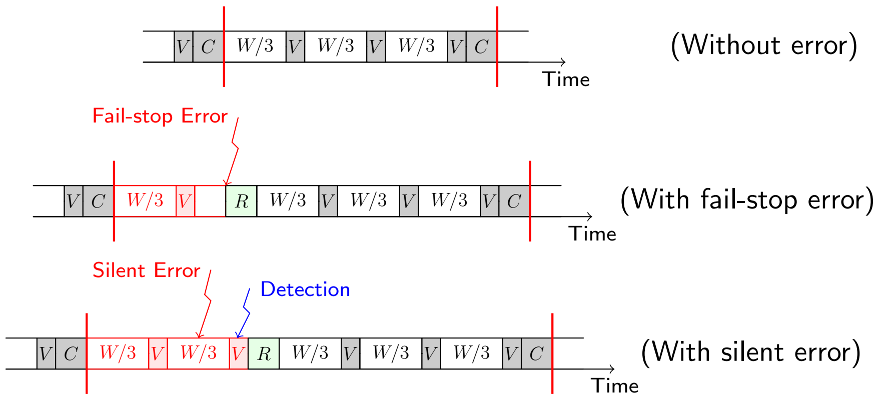


Figure 2: The VC+V pattern with $k = 3$ chunks executed at unit speed $s = 1$. The first figure shows the execution of a pattern without any error. The second figure shows that the execution is stopped immediately when a fail-stop error strikes anywhere in the pattern. The third figure shows that the execution continues after a silent error strikes and it is detected by an additional verification before the end of the pattern. In both cases, the pattern is re-executed after a recovery.

checkpointing time mainly depends on I/O operations. On the contrary, the verification time $V(s)$ could well depend on s (think of checksums to recompute).

VC+V: Placing additional verifications. See Figure 2 for an illustration. The pattern consists of k computational chunks of size W/k, which also takes $T(s) = W/s$ to execute without any error at speed s. Each chunk ends with a verification, and the last chunk ends with a verified checkpoint. Now the total duration of the pattern without any error is $T(s) + kV(s) + C$.

For both protocols, we say that $T(s)$ is the *period* of the pattern, because it corresponds to the same amount of useful work. However, the total duration of the VC-ONLY pattern is smaller for an error-free execution. This is balanced by the fact that silent errors may be detected earlier on with the VC+V pattern, thereby decreasing the re-execution time whenever a silent error has struck.

Error rates

As already mentioned, we consider two types of errors: *fail-stop* and *silent*. The arrivals of both fail-stop errors and silent errors on the platform follow exponential distribution with average rates $\lambda^F(s)$ and $\lambda^S(s)$, respectively, where s denotes the CPU speed. The variation of the error rates as a function of the speed s is discussed in Section 2. Note that the error rates are aggregated onto the macro-processor, so to speak: if each of the p processors has an individual fail-stop error rate $\lambda^F_{ind}(s)$, then the macro-processor has a fail-stop error rate $\lambda^F(s) = p\lambda^F_{ind}(s)$ [26, Proposition 1.2] (and similarly for silent errors).

For an execution of duration L, the probability of a fail-stop error is $p^F(s, L) = 1 - e^{-\lambda^F(s)L}$, and that of a silent error is $p^S(s, L) = 1 - e^{-\lambda^S(s)L}$. We assume that both error rates are in the same order, i.e., $\lambda^F(s) = \Theta(\lambda(s))$ and $\lambda^S(s) = \Theta(\lambda(s))$, where $\lambda(s) =$

$\lambda^F(s) + \lambda^S(s) = 1/\mu(s)$ denotes the reciprocal of the platform MTBF running at speed s while accounting for both error sources.

When an error (of any source) strikes, we roll-back and recover from the previous checkpoint (or from the original data for the first pattern). Let R denote the recovery time. We assume that errors only strike during computations, and not during I/O transfers (checkpoints and recoveries) nor verifications.

Optimizing for time

The TIME-VC-ONLY optimization problem is to determine the optimal period $T(s)$ to minimize the expected overhead in the execution time in the presence of errors. The overhead is the ratio $\frac{Time(T(s))}{T(s)}$ of the expected execution time $Time\,(T(s))$ over the base time $T(s)$. The TIME-VC+V optimization problem is defined similarly, but we also need to compute the optimal number k of verifications inside the pattern. Recall that for both problems, the overhead is due to two different sources: the error-free overhead (verifications and checkpoints) and the error-induced overhead (recovery and re-execution after an error). The impact of both overhead sources must obey a delicate trade-off, which makes both optimization problems challenging.

Optimizing for energy

Another important optimization objective is energy consumption. Altogether, the total power consumption of the macro-processor is p times the power consumption of each individual resource. It is decomposed into three different components:

- P_{idle}, the static power dissipated when the platform is on (even idle);
- $P_{cpu}(s)$, the dynamic power spent by operating the CPU at speed s;
- P_{io}, the dynamic power spent by I/O transfers (checkpoints and recoveries).

During checkpointing and recovery, we assume a dedicated (constant) power consumption, while during computation and verification, the power consumption depends upon the operating speed s. Assume w.l.o.g. that there is no overlap between CPU operations and I/O transfers. Then the total energy consumed during the execution of a pattern at speed s can be expressed as

$$Energy = T_{cpu}(s)(P_{idle} + P_{cpu}(s)) + T_{io}(P_{idle} + P_{io}) \ ,$$

where $T_{cpu}(s)$ is the total time spent on computing and verifying, and T_{io} is the total time spent on I/O transfers (checkpointing and recovering). The energy consumed to checkpoint is $E^C = C(P_{idle} + P_{io})$, to recover is $E^R = R(P_{idle} + P_{io})$, and to verify at speed s is $E^V(s) = V(s)(P_{idle} + P_{cpu}(s))$. Just as for time, resilience has a double cost in terms of energy consumption: the error-free overhead (verifications and checkpoints) and the error-induced overhead (recovery and re-execution after an error).

Bi-criteria problem

The most general problem can be expressed as a linear combination of execution time and energy consumption, and it is addressed in Section 3.4. We first tackle the problem of minimizing the expected execution time, both for the VC-ONLY protocol and for the VC+V protocol.

3.2 The Time-VC-Only problem

In this section, we deal with the VC-ONLY protocol and we aim at minimizing the expected execution time.

Theorem 1. *For a divisible load application subject to both fail-stop and silent errors, a first-order approximation of the optimal checkpointing period to minimize the expected execution overhead at speed s with the* VC-ONLY *protocol is*

$$T^*(s) = \sqrt{\frac{2(V(s)+C)}{\lambda^F(s)+2\lambda^S(s)}} \ .$$

Note that when silent errors are not considered, i.e., $\lambda^S(s) = 0$ and $V(s) = 0$, we retrieve the original Young/Daly formula $T^*(s) = \sqrt{\frac{2C}{\lambda^F(s)}}$ [41, 12].

Proof. With both fail-stop and silent errors, let $T(s)$ denote the checkpointing period. Silent errors can occur at any time during the computation but we only detect them after the pattern has been executed. Thus, we always have to pay $T(s) + V(s)$, the time needed to execute a segment between two consecutive checkpoints and to verify the result. If the verification fails, which happens with probability $p^S(s, T(s))$, a silent error has occurred and we have to recover from the last checkpoint and start anew.

Things are different when accounting for fail-stop errors, because the application will stop immediately when a fail-stop error occurs, even in the middle of the computation. Let $T_{lost}(s)$ denote the expected time lost during the execution of a segment between two consecutive checkpoints if a fail-stop error strikes, and it can be expressed as

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

where $\mathbb{P}(X = x)$ denotes the probability that a fail-stop error strikes at time x. By definition, we have $p^S(s, T(s)) = \mathbb{P}(X < T(s)) = 1 - e^{-\lambda^F(s)T(s)}$. Integrating by parts, we can get

$$T_{lost}(s) = \frac{1}{\lambda^F(s)} - \frac{T(s)}{e^{\lambda^F(s)T(s)} - 1} \ . \tag{3}$$

When accounting for both fail-stop and silent errors, we consider fail-stop errors first. If the application stops, with probability $p^F(s, T(s))$, then we do not need to perform a verification since we must do a recovery anyway. If no fail-stop error strikes during the execution, with probability $1 - p^F(s, T(s))$, we proceed with the verification and check for silent errors. If there is no error, we are done and we pay the cost for the checkpoint. Therefore, the expected execution time for a segment between two consecutive checkpoints (accounting for the cost of the checkpoint itself) is given by

$$\begin{aligned}
Time\,(T(s)) = {} & p^F(s, T(s))\,(T_{lost}(s) + R + Time\,(T(s))) \\
& + \left(1 - p^F(s, T(s))\right)\Big(T(s) + V(s) + p^S(s, T(s))\,(R + Time\,(T(s))) \\
& \qquad + \left(1 - p^S(s, T(s))\right)C\Big) \ .
\end{aligned}$$

When plugging $p^F(s,T(s))$, $p^S(s,T(s))$ and $T_{lost}(s)$ into the above equation, we get

$$Time\,(T(s)) = e^{\lambda^S(s)T(s)}\left(\frac{e^{\lambda^F(s)T(s)}-1}{\lambda^F(s)} + V(s)\right) + \left(e^{(\lambda^F(s)+\lambda^S(s))T(s)}-1\right)R + C\ . \tag{4}$$

Now, we are interested in the value of $T(s)$ that minimizes the overhead $\frac{Time(T(s))}{T(s)}$ with respect to the error-free and checkpoint-free execution time $T(s)$. Using Taylor expansion to approximate $e^{\lambda T} \approx 1 + \lambda T + \frac{\lambda^2 T^2}{2}$ up to the second-order term, we get the following first-order approximation for the overhead:

$$\frac{Time\,(T(s))}{T(s)} = 1 + \left(\frac{\lambda^F(s)}{2} + \lambda^S(s)\right)T(s) + \frac{V(s)+C}{T(s)}$$
$$+ \lambda^S(s)V(s) + \left(\lambda^F(s)+\lambda^S(s)\right)R + o\left(\lambda(s)\right)\ .$$

Differentiating the above expression with respect to $T(s)$, we find that $T^*(s) = \sqrt{\frac{2(V(s)+C)}{\lambda^F(s)+2\lambda^S(s)}}$ minimizes the overhead, which nicely extends Young/Daly's result to include both fail-stop and silent errors. We stress that, as in Young/Daly's formula, this result is a first-order approximation, which is valid only if all resilience parameters C, R and $V(s)$ are small in front of both MTBF values, namely $1/\lambda^F(s)$ for fail-stop errors and $1/\lambda^S(s)$ for silent errors. □

3.3 The Time-VC+V problem

In this section, we aim at finding the optimal parameters for the VC+V protocol. The following theorem shows the optimal checkpointing period as well as the optimal number of verifications inside the pattern.

Theorem 2. *For a divisible load application subject to both fail-stop and silent errors, a first-order approximation of the optimal checkpointing period to minimize the expected execution overhead with the* VC+V *protocol is* $T^*(s) = \bar{k}^* t^*(s)$*, where* $\bar{k}^*$ *denotes the optimal number of verifications in the pattern and* $t^*(s)$ *denotes the optimal length of each chunk inside the pattern. Here, we have*

$$t^*(s) = \sqrt{\frac{2(V(s)+C/\bar{k}^*)}{\bar{k}^*\lambda^F(s) + (\bar{k}^*+1)\lambda^S(s)}}\ ,$$

and $\bar{k}^*$ *is equal to either* $\max(1, \lfloor k^* \rfloor)$ *or* $\lceil k^* \rceil$ *(whichever leads to the smallest overhead), where* k^* *is given by*

$$k^* = \sqrt{\frac{\lambda^S(s)}{\lambda^F(s)+\lambda^S(s)} \cdot \frac{C}{V(s)}}\ .$$

Note that when there is only one verification (or chunk) in a pattern, i.e., $\bar{k}^* = 1$, we have $t^*(s) = \sqrt{\frac{2(V(s)+C)}{\lambda^F(s)+2\lambda^S(s)}} = T^*(s)$, retrieving the result of Theorem 1.

Proof. Consider a pattern with k chunks (hence k verifications), each of length $t(s)$. The total length of the pattern is thus $T(s) = kt(s)$. For convenience, we write $p^F(s)$ instead of $p^F(s,t(s)) = 1 - e^{-\lambda^F(s)t(s)}$, which is the probability that a fail-stop error strikes when executing a chunk. We also use $p^S(s)$ for $p^S(s,t(s)) = 1 - e^{-\lambda^S(s)t(s)}$, the corresponding probability with silent errors. The probability that neither type of error occurs during the execution of a chunk is then given by $q(s) = (1-p^F(s))(1-p^S(s)) = e^{-(\lambda^F(s)+\lambda^S(s))T(s)}$. The expected execution time for the whole pattern can be expressed recursively by enumerating the failure possibilities for all its chunks as follows:

$$\begin{aligned} \mathit{Time}\,(T(s)) \;=\; & \sum_{i=1}^{k} q(s)^{i-1}\Big(p^F(s)\Big((i-1)\,(t(s)+V(s)) + t_{lost}(s) + R + \mathit{Time}\,(T(s))\Big) \\ & \qquad + \big(1-p^F(s)\big)\,p^S(s)\Big(i\,(t(s)+V(s)) + R + \mathit{Time}\,(T(s))\Big)\Big) \\ & + q(s)^k\Big(k\,(t(s)+V(s)) + C\Big)\ , \end{aligned} \tag{5}$$

where $t_{lost}(s)$ is the expected time lost during the execution of a chunk knowing that a fail-stop error has struck. According to Equation (3), we have $t_{lost}(s) = \frac{1}{\lambda^F(s)} - \frac{t(s)}{e^{\lambda^F(s)t(s)}-1}$.

Let $h = \sum_{i=1}^{k} iq(s)^{i-1}$, and $q(s)h = \sum_{i=1}^{k} iq(s)^{i}$, so we have $(1-q(s))h = \sum_{i=1}^{k} q(s)^{i-1} - kq(s)^k = \frac{1-q(s)^k}{1-q(s)} - kq(s)^k$, hence $h = \frac{1-q(s)^k}{(1-q(s))^2} - \frac{kq(s)^k}{1-q(s)}$. Substituting h and $t_{lost}(s)$ into Equation (5) and simplifying it, we get

$$\begin{aligned} \mathit{Time}\,(T(s)) \;=\; & \frac{1-q(s)^k}{1-q(s)}\left(p^F(s) + (1-p^F(s))p^S(s)\right)\left(R + \mathit{Time}\,(T(s))\right) \\ & + \frac{(1-q(s)^k)p^F(s)}{1-q(s)}\left(\frac{1}{\lambda^F(s)} - \frac{t(s)}{e^{\lambda^F(s)t(s)}-1}\right) + q(s)^k C \\ & + (t(s)+V(s))\Bigg(\frac{p^F(s)(q(s)-q(s)^k) + (1-p^F(s))p^S(s)(1-q(s)^k)}{(1-q(s))^2} \\ & \qquad - \frac{(k-1)p^F(s)q(s)^k + k(1-p^F(s))p^S(s)q(s)^k}{1-q(s)}\Bigg) \\ \;=\; & (1-q(s)^k)(R + \mathit{Time}\,(T(s))) + \frac{(1-p^F(s))(1-q(s)^k)}{1-q(s)}(t(s)+V(s)) \\ & + \frac{(1-q(s)^k)p^F(s)}{1-q(s)}\left(\frac{1}{\lambda^F(s)} - \frac{t(s)}{e^{\lambda^F(s)t(s)}-1}\right) + q(s)^k C\ , \end{aligned}$$

which leads to

$$\begin{aligned} \mathit{Time}\,(T(s)) = & \frac{q(s)^{-k}-1}{1-q(s)}\left((1-p^F(s))(t(s)+V(s)) + p^F(s)\left(\frac{1}{\lambda^F(s)} - \frac{t(s)}{e^{\lambda^F(s)t(s)-1}}\right)\right) \\ & + \left(q(s)^{-k}-1\right)R + C\ . \end{aligned}$$

Applying Taylor expansion to the equation above by approximating $e^{\lambda t} \approx 1 + \lambda t + \frac{\lambda^2 t^2}{2}$ and

$$\begin{aligned} \frac{e^{k\lambda t}-1}{1-e^{-\lambda t}} \;&\approx\; \frac{k\lambda t\left(1+\frac{k\lambda t}{2}\right)}{\lambda t\left(1-\frac{\lambda t}{2}\right)} \\ &\approx\; k\left(1 + \frac{(k+1)\lambda t}{2}\right)\ , \end{aligned}$$

we can approximate, up to the first-order term, the overhead of executing the pattern as

$$\begin{aligned}\frac{\mathit{Time}\,(T(s))}{T(s)} &= 1+\frac{k\lambda^F(s)+(k+1)\lambda^S(s)}{2}t(s)+\frac{V(s)+C/k}{t(s)}+\left(\lambda^F(s)+\lambda^S(s)\right)R\\ &\quad+\frac{(k+1)\lambda^S(s)+(k-1)\lambda^F(s)}{2}V(s)+o\left(\lambda(s)\right)\ .\end{aligned}\tag{6}$$

Differentiating Equation (6) with respect to $t(s)$, we find that $t^*(s)=\sqrt{\frac{2(V(s)+C/k)}{k\lambda^F(s)+(k+1)\lambda^S(s)}}$ minimizes the overhead.

Now, substituting $t^*(s)$ back into Equation (6), we get

$$\frac{\mathit{Time}\,(T(s))}{T(s)}=1+\sqrt{2\left(xk+y+\frac{z}{k}\right)}+o\left(\sqrt{\lambda(s)}\right)\ ,\tag{7}$$

where $x=V(s)(\lambda^F(s)+\lambda^S(s))$, $y=C(\lambda^F(s)+\lambda^S(s))+V(s)\lambda^S(s)$, $z=C\lambda^S(s)$. Differentiating Equation (7) with respect to k, we find that $k^*=\sqrt{\frac{z}{x}}=\sqrt{\frac{\lambda^S(s)}{\lambda^F(s)+\lambda^S(s)}\cdot\frac{C}{V(s)}}$ minimizes the overhead. Since the number of verifications in a pattern must be an integer, the optimal strategy uses either $\max(1,\lfloor k^*\rfloor)$ or $\lceil k^*\rceil$ verifications, whichever leads to a smaller value for the overhead $\frac{\mathit{Time}(T(s))}{T(s)}$.

Again, as for the optimal checkpointing period in the VC-ONLY protocol, the values of $t^*(s)$ and k^* are first-order approximations. The results are valid only if all resilience parameters C, R and $V(s)$ are small in front of the MTBF values of the platform. □

Let us now consider a simple example. Suppose $\lambda^F(s)=0.001$, $\lambda^S(s)=0.002$, $C=R=20$ and $V(s)=1$. Using the VC-ONLY protocol, the optimal checkpointing period is given by $T^*(s)\approx 91.65$, which results in an execution overhead of $\frac{\mathit{Time}(T(s))}{T(s)}\approx 1.56$. If the VC+V protocol is used instead, the optimal value of k^* is given by $\sqrt{\frac{z}{x}}\approx 3.6515$, which leads to the optimal number of verifications $\bar{k}^*=3$ and optimal chunk length $t^*(s)\approx 37.33$. The pattern checkpointing period in this case is $T(s)=3t^*(s)\approx 111.99$ and the execution overhead becomes $\frac{\mathit{Time}(T(s))}{T(s)}\approx 1.51$. This example demonstrates the advantage of using additional verifications for coping with both fail-stop and silent errors.

3.4 Bi-criteria problem

In this section, we take energy consumption into consideration, and aim at optimizing a linear combination of execution time and energy consumption, i.e.,

$$a\cdot\mathit{Time}+b\cdot\mathit{Energy}\ ,\tag{8}$$

where a and b are the weights associated with time and energy, respectively. Indeed, optimizing a linear combination of two objectives is a common approach that has been widely adopted by the literature for many bi-criteria optimization problems (see, e.g., [15, 19, 1, 27]). In our case, setting $b=0$ reduces to minimizing the execution time as considered in the previous sections, while setting $a=0$ amounts to minimizing energy consumption alone. Different values of the weights a and b allow for investigating various user-defined trade-offs.

The corresponding optimization problems are called TIMEENERGY-VC-ONLY for the VC-ONLY protocol, and TIMEENERGY-VC+V for the VC+V protocol.

Theorem 3. *Consider a divisible load application subject to both fail-stop and silent errors, with the objective to minimize a linear combination of expected execution time and energy consumption as shown in Equation* (8).
(i) In the TimeEnergy-VC-Only *problem, the optimal checkpointing period is*

$$T^*(s) = \sqrt{\frac{2(V(s)+C_e(s))}{\lambda^F(s)+2\lambda^S(s)}} \ .$$

(ii) In the TimeEnergy-VC+V *problem, the optimal length of each chunk is*

$$t^*(s) = \sqrt{\frac{2(V(s)+C_e(s)/\bar{k}^*)}{\bar{k}^*\lambda^F(s)+(\bar{k}^*+1)\lambda^S(s)}} \ ,$$

where $\bar{k}^*$ denotes the optimal number of verifications in the pattern, and it is equal to either $\max(1, \lfloor k^* \rfloor)$ *or* $\lceil k^* \rceil$, *where the value of k^* is*

$$k^* = \sqrt{\frac{\lambda^S(s)}{\lambda^F(s)+\lambda^S(s)} \cdot \frac{C_e(s)}{V(s)}} \ .$$

Here, we define $C_e(s) = \frac{a+b(P_{idle}+P_{io})}{a+b(P_{idle}+P_{cpu}(s))}C$.

Note that $C_e(s)$ is a time/energy ratio that depends on both parameters a and b, and reduces to $C_e(s) = C$ when $b = 0$, in accordance with Theorems 1 and 2. When $a = 0$, we have $C_e(s) = \frac{P_{idle}+P_{io}}{P_{idle}+P_{cpu}(s)}C = \frac{E^C}{P_{idle}+P_{cpu}(s)}$.

sketch. The proof is similar to those of Theorems 1 and 2. Below, we only sketch the proof for the VC-only protocol.

Let $T(s)$ denote the checkpointing period and let $G(s) = aT(s) + bT(s)(P_{idle} + P_{cpu}(s)) = (a + b(P_{idle} + P_{cpu}(s)))\,T(s)$ denote the value of the objective function for the period in an error-free and checkpoint-free execution. Following the proof of Theorem 1, we compute the expected cost to execute a segment between two checkpoints as follows:

$$G(T(s)) = (a + b(P_{idle} + P_{cpu}(s)))\,F(T(s)) \ , \tag{9}$$

where $F(T(s))$ is given by

$$F(T(s)) = e^{\lambda^S(s)T(s)}\left(\frac{e^{\lambda^F(s)T(s)}-1}{\lambda^F(s)} + V(s)\right) + \left(e^{(\lambda^F(s)+\lambda^S(s))T(s)}-1\right)R_e(s) + C_e(s) \ , \tag{10}$$

with $R_e(s) = \frac{a+b(P_{idle}+P_{io})}{a+b(P_{idle}+P_{cpu}(s))}R$ and $C_e(s) = \frac{a+b(P_{idle}+P_{io})}{a+b(P_{idle}+P_{cpu}(s))}C$.

Now, considering the execution overhead, we can get

$$\begin{aligned}\frac{G(T(s))}{G(s)} &= \frac{F(T(s))}{T(s)}\\ &= 1 + \left(\frac{\lambda^F(s)}{2} + \lambda^S(s)\right)T(s) + \frac{V(s)+C_e(s)}{T(s)}\\ &\quad + \lambda^S(s)V(s) + \left(\lambda^F(s)+\lambda^S(s)\right)R_e(s) + o\left(\lambda(s)\right) \ .\end{aligned}$$

Differentiating the above expression with respect to $T(s)$, we find that $T^*(s) = \sqrt{\frac{2(V(s)+C_e(s))}{\lambda^F(s)+2\lambda^S(s)}}$ minimizes the overhead. This result is analogous to the one in Theorem 1 and further extends Young/Daly's formula to cover energy consumption in the optimization objective. The optimal parameters for the VC+V protocol can be similarly derived and are omitted here. □

4 Optimal algorithms for a linear chain of tasks

This section is the counterpart of the previous one for a linear chain of tasks. Rather than divisible load applications, we consider linear workflows. The main differences are the following:

- Checkpoints and verifications have to be placed at the end of some tasks, while they could be freely located for divisible load applications. Hence it is not possible to have periodic patterns any longer.
- The goal now is to minimize total execution time (or makespan), or total energy, or a linear combination, by judiciously placing verified checkpoints (both VC-ONLY and VC+V protocols) and intermediate verifications (VC+V protocol).
- Owing to the specific structure of linear chains, we are able to provide exact optimal solutions, not just first-order approximations.

For a linear chain of tasks subject to fail-stop errors only, Toueg and Babaoglu [39] give an optimal algorithm to compute the best checkpointing positions in order to minimize expected execution time. In this section, we extend their results to include both fail-stop and silent errors, and to handle intermediate verifications. In other words, both VC-ONLY and VC+V protocols are studied, for time and/or energy optimization. The organization of this section follows that of divisible load applications. We start by detailing the framework for linear chains in Section 4.1. Then, we present optimal algorithms for TIME-VC-ONLY (Section 4.2) and TIME-VC+V (Section 4.3) problems, before addressing the bi-criteria optimization problems TIMEENERGY-VC-ONLY and TIMEENERGY-VC+V (Section 4.4).

4.1 Framework

We consider application workflows whose task graph is a linear chain $T_1 \to T_2 \cdots \to T_n$. Here n is the number of tasks, and each task T_i is weighted by its computational cost w_i. For reference, all additional notations for a linear chain are summarized in Table 2.

The time to compute tasks T_i to T_j at speed s is $T_{i,j}(s) = \frac{1}{s}\sum_{k=i}^{j} w_i$ and the corresponding energy is $E_{i,j}(s) = T_{i,j}(s)(P_{idle} + P_{cpu}(s))$. The time to checkpoint (the output of) task T_i is C_i, the time to recover from (the checkpoint of) task T_i is R_i, and the time to verify (the output of) task T_i at speed s is $V_i(s)$. We define $p^F_{i,j}(s) = p(s, T_{i,j}(s))$ to be the probability that a fail-stop error strikes when executing from T_i to T_j, and define $p^S_{i,j}(s) = p(s, T_{i,j}(s))$ similarly for silent errors.

Finally, the energy to checkpoint task T_i is $E^C_i = C_i(P_{idle} + P_{io})$, to recover from task T_i is $E^R_i = R_i\ (P_{idle} + P_{io})$, and to verify task T_i at speed s is $E^V_i(s) = V_i(s)(P_{idle} + P_{cpu}(s))$.

Table 2: Additional notations for a linear chain of tasks.

Tasks	
$\{T_1, T_2, \ldots, T_n\}$	Set of n tasks
w_i	Computational cost of task T_i
Time	
$T_{i,j}(s)$	Time needed to execute tasks T_i to T_j at speed s
$V_i(s)$	Time needed to verify task T_i at speed s
C_i	Time needed to checkpoint task T_i
R_i	Time needed to recover from task T_i
$p^F_{i,j}(s)$	Probability that a fail-stop error strikes when executing tasks T_i to T_j at speed s
$p^S_{i,j}(s)$	Probability that a silent error strikes when executing tasks T_i to T_j at speed s
Energy	
$E_{i,j}(s)$	Energy needed to execute tasks T_i to T_j at speed s
$E^V_i(s)$	Energy needed to verify task T_i at speed s
E^C_i	Energy needed to checkpoint task T_i
E^R_i	Energy needed to recover from task T_i

4.2 The Time-VC-Only problem

Theorem 4. *The* TIME-VC-ONLY *problem can be solved by a dynamic programming algorithm in $O(n^2)$ time.*

Proof. We define $Time_C(j,s)$ to be the optimal expected time to successfully execute tasks $T_1, \ldots, T_j$ at speed s, where T_j has a verified checkpoint, and there are possibly other verified checkpoints from T_1 to T_{j-1}. Note that we always verify and checkpoint the last task T_n to save the final result, so the goal is to find $Time_C(n,s)$.

To compute $Time_C(j,s)$, we formulate the following dynamic program by trying all possible locations for the last checkpoint before T_j (see Figure 3):

$$Time_C(j,s) = \min_{0 \le i < j} \Big\{ Time_C(i,s) + T_C(i+1,j,s) \Big\} + C_j \ ,$$

where $T_C(i,j,s)$ is the expected time to successfully execute the tasks T_i to T_j, provided that T_{i-1} and T_j are both verified and checkpointed, while no other task in between is verified nor checkpointed. Note that we also account for the checkpointing cost C_j for task T_j, which is not included in the definition of T_C. To initialize the dynamic program, we define $Time_C(0,s) = 0$.

For convenience, we assume that there is a virtual task T_0 that is always verified and checkpointed, with a recovery cost $R_0 = 0$. According to Equation (4) but without counting the checkpointing time at the end, the expected time needed to execute tasks T_i to T_j for each (i,j) pair with $i \le j$ is given by

$$T_C(i,j,s) = e^{\lambda^S(s)T_{i,j}(s)} \left(\frac{e^{\lambda^F(s)T_{i,j}(s)} - 1}{\lambda^F(s)} + V_j(s) \right) + \left(e^{(\lambda^F(s)+\lambda^S(s))T_{i,j}(s)} - 1 \right) R_{i-1} \ . \quad (11)$$

We can now compute $Time_C(j,s)$ for all $j = 1, \ldots, n$. For the complexity, the computation of $T_C(i,j,s)$ for all (i,j) pairs with $i \le j$ takes $O(n^2)$ time. The computation of the dynamic programming table for $Time_C(j,s)$ also takes $O(n^2)$ time, as $Time_C(j,s)$ depends on at most j other entries in the same table. Therefore, the overall complexity is $O(n^2)$, and this concludes the proof. □

We point out that our solution also improves upon Toueg and Babaoglu's original algorithm [39], which has complexity $O(n^3)$. They provide an improved $O(n^2)$ algorithm only for the special case where $C_i > C_j$ implies $R_i \geq R_j$, while our algorithm returns the optimal solution regardless of the values of C and R.

4.3 The Time-VC+V problem

Theorem 5. *The* Time-VC+V *problem can be solved by a dynamic programming algorithm in $O(n^3)$ time.*

Note that adding intermediate verifications between two verified checkpoints creates an additional step in the dynamic programming algorithm, leading to a higher computational complexity.

Proof. In the Time-VC-Only problem, we were only allowed to place verified checkpoints. Here, we can add intermediate verifications. The main idea is to replace T_C in the dynamic programming algorithm of Theorem 4 by another expression $Time_V(i,j,s)$, which denotes the optimal expected time to successfully execute from task T_i to task T_j (and to verify it), provided that T_{i-1} has a verified checkpoint and only single verifications are allowed within tasks $T_i,\dots,T_{j-1}$. Furthermore, we use $Time_{VC}(j,s)$ to denote the optimal expected time to successfully execute the first j tasks, where T_j has a verified checkpoint, and there are possibly other verified checkpoints and single verifications before T_j. The goal is to find $Time_{VC}(n,s)$. The dynamic program to compute $Time_{VC}(j,s)$ can be formulated as follows (see Figure 4):

$$Time_{VC}(j,s) = \min_{0\leq i<j} \Big\{ Time_{VC}(i,s) + Time_V(i+1,j,s) \Big\} + C_j \ .$$

In particular, we try all possible locations for the last checkpoint before T_j, and for each location T_i, we compute the optimal expected time $Time_V(i+1,j,s)$ to execute tasks T_{i+1} to T_j with only single verifications in between. We also account for the checkpointing time C_j, which is not included in the definition of $Time_V$. By initializing the dynamic program with $Time_{VC}(0,s)=0$, we can then compute the optimal solution as in the Time-VC-Only problem.

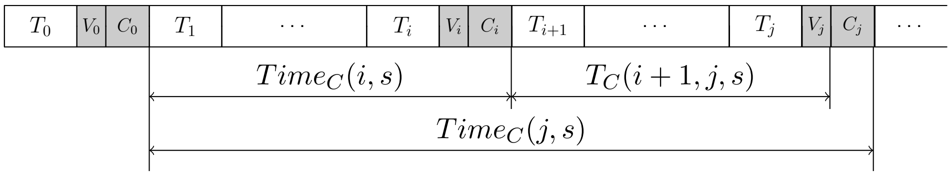


Figure 3: Illustration of the dynamic programming formulation for $Time_C(j,s)$.

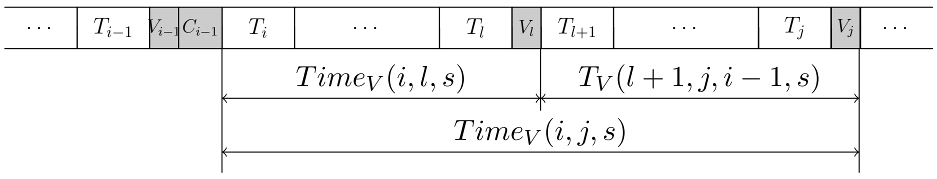


Figure 5: Illustration of the dynamic programming formulation for $Time_V(i,j,s)$.

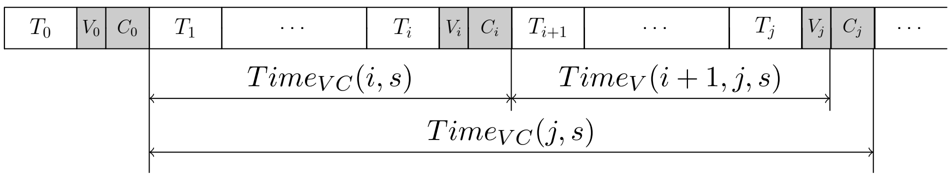


Figure 4: Illustration of the dynamic programming formulation for $Time_{VC}(j,s)$.

It remains to compute $Time_V(i,j,s)$ for each (i,j) pair with $i \leq j$. To this end, we formulate another dynamic program by trying all possible locations for the last single verification before T_j (see Figure 5):

$$Time_V(i,j,s) = \min_{i-1 \leq l < j} \left\{ Time_V(i,l,s) + T_V(l+1,j,i-1,s) \right\} ,$$

where $T_V(i,j,l_c,s)$ is the expected time to successfully execute all the tasks from T_i to T_j (and to verify T_j), knowing that if an error strikes, we can recover from T_{l_c}, which is the last task before T_i to have a verified checkpoint.

First, note that if we account for fail-stop errors only, we do not need to perform any single verification, and hence the problem becomes simply the Time-VC-Only problem. When accounting for both fail-stop and silent errors, we can apply the same method as in the proof of Theorem 1. Specifically, if a fail-stop error strikes between two verifications, we directly perform a recovery from task T_{l_c} and redo the entire computation from T_{l_c+1} to T_j, which contains a single verification after T_{i-1} and possibly other single verifications between T_{l_c+1} and T_{i-2}. This is done by calling $Time_V(l_c+1, i-1, s)$ first, and then $T_V(i,j,l_c,s)$ recursively. Otherwise, if no fail-stop error occurs during the execution from task T_i to task T_j, we check for silent errors by performing a verification on task T_j. If a silent error occurs, we perform the same recovery as before. Altogether, we have the following expression:

$$\begin{aligned} T_V(i,j,l_c,s) &= p^F_{i,j}(s) \left(T_{lost_{i,j}}(s) + R_{l_c} + Time_V(l_c+1, i-1, s) + T_V(i,j,l_c,s) \right) \\ &+ (1 - p^F_{i,j}(s)) \Big(T_{i,j}(s) + V_j(s) + p^S_{i,j}(s) \left(R_{l_c} + Time_V(l_c+1, i-1, s) + T_V(i,j,l_c,s) \right) \Big) , \end{aligned} \tag{12}$$

where $T_{lost_{i,j}}(s)$ denotes the expected time lost when executing tasks T_i to T_j if a fail-stop error strikes and, according to Equation (3), it is given by

$$T_{lost_{i,j}}(s) = \frac{1}{\lambda^F(s)} - \frac{T_{i,j}(s)}{e^{\lambda^F(s) T_{i,j}(s)} - 1} . \tag{13}$$

Solving $T_V(i,j,l_c,s)$ from Equation (12) above, we can get

$$T_V(i,j,l_c,s) = e^{\lambda^S(s)T_{i,j}(s)}\left(\frac{e^{\lambda^F(s)T_{i,j}(s)}-1}{\lambda^F(s)} + V_j(s)\right) + \left(e^{(\lambda^F(s)+\lambda^S(s))T_{i,j}(s)}-1\right)\left(R_{l_c} + Time_V(l_c+1,i-1,s)\right) \ .$$

Note that $T_V(i,j,l_c,s)$ depends on the value of $Time_V(l_c+1,i-1,s)$, except when $l_c+1=i$, in which case we initialize $Time_V(i,i-1,s)=0$. Hence, in the dynamic program, $Time_V(i,j,s)$ can be expressed as a function of $Time_V(i,l,s)$ for all $l=i-1,\cdots,j-1$.

Finally, the complexity is dominated by the computation of the second dynamic programming table for $Time_V(i,j,s)$, which contains $O(n^2)$ entries and each entry depends on at most n other ones. Hence, the overall complexity of the algorithm is $O(n^3)$, which concludes the proof. □

4.4 Bi-criteria problem

Theorem 6. *Consider a chain of tasks with the objective of minimizing a linear combination of execution time and energy consumption as shown in Equation* (8)*. The* TimeEnergy-VC-Only *problem can be solved in $O(n^2)$ time and the* TimeEnergy-VC+V *problem can be solved in $O(n^3)$ time.*

sketch. The proof is similar to the case when minimizing time alone. Below, we only sketch the proof for the VC-only protocol.

Let $Cost_C(j,s)$ denote the optimal expected cost (with combined time and energy) to successfully execute tasks $T_1,\ldots,T_j$, where T_j has a verified checkpoint, and there are possibly other verified checkpoints from T_1 to T_{j-1}. Let $G_C(i,j,s)$ denote the expected cost to successfully execute all the tasks from T_i to T_j without any checkpoint and verification in between, while T_{i-1} and T_j are both verified and checkpointed. The following dynamic program can be formulated for the VC-only protocol:

$$Cost_C(j,s) = \min_{0\le i<j}\left\{Cost_C(i,s) + G_C(i+1,j,s)\right\} + G_j^C \ ,$$

where $G_j^C = aC_j + bE_j^C = \left(a + b\left(P_{idle}+P_{io}\right)\right)C_j$ denotes the combined cost to checkpoint task T_j. The goal is to find $Cost_C(n,s)$.

According to Equations (9) and (10), the combined cost to execute from task T_i to task T_j, but without considering the checkpointing cost at the end, is given by

$$\begin{aligned} G_C(i,j,s) &= \left(a + b\left(P_{idle}+P_{cpu}(s)\right)\right)e^{\lambda^S(s)T_{i,j}(s)}\left(\frac{e^{\lambda^F(s)T_{i,j}(s)}-1}{\lambda^F(s)} + V_j(s)\right) \\ &\quad + \left(a + b\left(P_{idle}+P_{io}\right)\right)\left(e^{\left(\lambda^F(s)+\lambda^S(s)\right)T_{i,j}(s)}-1\right)R_{i-1} \ . \end{aligned}$$

Clearly, the computational complexity is $O(n^2)$, which is the same as that of the time minimization algorithm shown in Theorem 4. The dynamic programming algorithm for the VC+V protocol can be similarly constructed and is omitted here. □

Table 3: Additional notations for DVFS.

Speeds	
$S = \{s_1, s_2, \ldots, s_K\}$	Set of K discrete computing speeds (DVFS)
$s \in S$	Regular speed
$\sigma \in S$	Re-execution speed
Execution scenarios	
SingleSpeed	All tasks execute and re-execute at speed s (Section 4)
ReExecSpeed	All tasks first execute at speed s and then re-execute at speed σ
MultiSpeed	Tasks are partitioned into segments. Each segment executes with any speed in S for first execution and a (possibly different) speed for re-execution

5 DVFS for a linear chain of tasks

We extend the optimal algorithms for a linear chain to the case where several speeds are available. When computing (including verification), we use DVFS to change the speed of the processors, and assume a set $S = \{s_1, s_2, \ldots, s_K\}$ of K discrete computing speeds. During checkpointing and recovery, we assume a dedicated (constant) power consumption. The formula to compute the total energy consumed during the execution of the chain becomes

$$Energy = \sum_{i=1}^{K} T_{cpu}(s_i)(P_{idle} + P_{cpu}(s_i)) + T_{io}(P_{idle} + P_{io}) \ ,$$

where $T_{cpu}(s_i)$ is the time spent on computing at speed s_i, and T_{io} is the total time spent on I/O transfers. We introduce two new execution scenarios:

ReExecSpeed: There are two (possibly different) speeds, s for the first execution of each task, and σ for any potential re-execution.

MultiSpeed: The workflow chain is partitioned into segments delimited by verified checkpoints. For each of these segments, we can freely choose a speed for the first execution, and a (possibly different) speed for any ulterior execution, among the set of K speeds in S. Note that these speeds may well vary from one segment to another.

The design of optimal algorithms for the ReExecSpeed scenario enables to assess the impact of a simple speed change, and it paves the way to the most general and flexible execution scenario, MultiSpeed, which makes full use of the potential of having K different speeds. For reference, we use SingleSpeed to denote the execution scenario with a unique speed, which we have discussed in Section 4 for a linear chain of tasks. The main notations for this section are summarized in Table 3.

5.1 The ReExecSpeed Scenario

In the ReExecSpeed scenario, we are given two speeds s and σ, where s is the regular speed and σ is the re-execution speed. The regular speed s is used for the first execution of the tasks, while σ is used for all subsequent re-executions in case of failure during the first execution. Due to the use of two speeds, the analysis is considerably more involved than the SingleSpeed scenario.

5.1.1 The Time-VC-Only problem

We need to derive two independent expressions to compute the expected execution time for both VC-ONLY and VC+V protocols. The first expression is for the first execution of the tasks with the first speed s until the first error is encountered. Once the first error strikes, we recover from the last checkpoint and start re-executing the tasks with the second speed σ until we reach the next checkpoint. This latter expression is essentially the same as when we used a single speed s, but with speed σ instead. The following theorem shows the result.

Theorem 7. *For the* REEXECSPEED *scenario, the* TIME-VC-ONLY *problem can be solved by a dynamic programming algorithm in* $O(n^2)$ *time.*

Proof. The proof extends that of Theorem 4. To account for two speeds, we replace $Time_C(j,s)$ with $Time_{C_{re}}(j,s,\sigma)$, which denotes the optimal expected time to successfully execute the tasks T_1 to T_j, where T_j has a verified checkpoint. Similarly, we replace $T_C(i,j,s)$ with $T_{C_{re}}(i,j,s,\sigma)$ as the expected time to successfully execute the tasks T_i to T_j, where both T_{i-1} and T_j are verified and checkpointed. The goal is to find $Time_{C_{re}}(n,s,\sigma)$, and the dynamic program is formulated as follows:

$$Time_{C_{re}}(j,s,\sigma) = \min_{0 \le i < j} \left\{ Time_{C_{re}}(i,s,\sigma) + T_{C_{re}}(i+1,j,s,\sigma) \right\} + C_j \ .$$

Note that the checkpointing cost after T_j is included in $Time_{C_{re}}(j,s,\sigma)$ but not in $T_{C_{re}}(i,j,s,\sigma)$. We initialize the dynamic program with $Time_{C_{re}}(0,s,\sigma) = 0$.

To compute $T_{C_{re}}(i,j,s,\sigma)$ for each (i,j) pair with $i \le j$, we need to distinguish the first execution (before the first error) and all potential re-executions (after at least one error). Let $T_{C_{first}}(i,j,s)$ denote the expected time to execute the tasks T_i to T_j for the very first time before the first error is encountered, and let $T_C(i,j,\sigma)$ denote the expected time to successfully execute the tasks T_i to T_j in the re-executions.

While $T_C(i,j,\sigma)$ is given by Equation (11) but using speed σ, $T_{C_{first}}(i,j,s)$ can be computed by considering two possible scenarios: (i) a fail-stop error has occurred during the execution from T_i to T_j, in which case we lose $T_{lost_{i,j}}(s)$ time as given in Equation (13); (ii) there is no fail-stop error, in which case the execution time is $T_{i,j}(s) + V_j(s)$ regardless of whether silent errors occur. Note that in both cases, we do not account for the re-executions, as they are handled by T_C separately (with the second speed). Therefore, we have

$$T_{C_{first}}(i,j,s) = p^F_{i,j}(s)\left(\frac{1}{\lambda^F(s)} - \frac{T_{i,j}(s)}{e^{\lambda^F(s)T_{i,j}(s)}-1}\right) + \left(1-p^F_{i,j}(s)\right)\left(T_{i,j}(s)+V_j(s)\right) \ . \quad (14)$$

Let $p^E_{i,j}(s)$ denote the probability that at least one error is detected in the first execution of the tasks from T_i to T_j at speed s. Since we account for both silent and fail-stop errors, and we can only detect silent errors if no fail-stop error has occurred, we have $p^E_{i,j}(s) = p^F_{i,j}(s) + \left(1-p^F_{i,j}(s)\right)p^S_{i,j}(s)$. If no error strikes during the first execution, then the time to execute from T_i to T_j is exactly $T_{C_{first}}(i,j,s)$, which means that all the tasks have been executed successfully with the first speed. If at least one error occurs, which happens with probability $p^E_{i,j}(s)$, then $T_{C_{first}}(i,j,s)$ is the time lost trying to execute the tasks with the first speed. In this case, we need to recover from the last checkpoint and use $T_C(i,j,\sigma)$ to re-execute all the tasks from T_i to T_j with the second speed until we pass the next checkpoint. Therefore, we have

$$T_{C_{re}}(i,j,s,\sigma) = T_{C_{first}}(i,j,s) + p^E_{i,j}(s)\left(R_{i-1} + T_C(i,j,\sigma)\right) \ . \quad (15)$$

Despite the two steps needed to compute $T_{C_{re}}(i,j,s,\sigma)$, the complexity remains the same as in the SingleSpeed scenario (Theorem 4). This concludes the proof. $\square$

5.1.2 The Time-VC+V problem

With the VC+V protocol, we place intermediate verifications between two verified checkpoints. Because two speeds are used in the ReExecSpeed scenario, we will place two sets of intermediate verifications. The first set is used during the first execution of the tasks until the first error is encountered, in which case we recover from the last checkpoint and start re-executing the tasks using the second set of verifications with the second speed. The problem is to find the best positions for the verified checkpoints as well as the best positions for the two sets of verifications in order to minimize the total execution time.

Because two sets of intermediate verifications need to be placed, we formulate two independent dynamic programs to determine their respective optimal positions. The overall complexity, however, remains the same as with a single speed. The following theorem shows the result.

Theorem 8. *For the* ReExecSpeed *scenario, the* Time-VC+V *problem can be solved by a dynamic programming algorithm in* $O(n^3)$ *time.*

Proof. We follow the same reasoning as in the VC-only protocol (see Section 5.1.1). Here, we replace $Time_{C_{re}}(j,s,\sigma)$ with $Time_{VC_{re}}(j,s,\sigma)$, and replace $T_{C_{re}}(i,j,s,\sigma)$ with $T_{VC_{re}}(i,j,s,\sigma)$. Note that both expressions follow the new re-execution model and account for both sets of intermediate verifications. The goal is to find $Time_{VC_{re}}(n,s,\sigma)$, and the dynamic program is formulated as follows:

$$Time_{VC_{re}}(j,s,\sigma) = \min_{0\le i<j} \Big\{ Time_{VC_{re}}(i,s,\sigma) + T_{VC_{re}}(i+1,j,s,\sigma) \Big\} + C_j \ .$$

Note that the checkpointing cost after T_j is included in $Time_{VC_{re}}(j,s,\sigma)$ but not in $T_{VC_{re}}(i,j,s,\sigma)$. We initialize the dynamic program with $Time_{VC_{re}}(0,s,\sigma)=0$.

To compute $T_{VC_{re}}(i,j,s,\sigma)$, where both T_{i-1} and T_j are verified and checkpointed, we again consider two parts: (1) the optimal expected time $Time_{V_{first}}(i,j,s)$ to execute the tasks T_i to T_j in the first execution using the first set of intermediate verifications with speed s; (2) the optimal expected time $Time_V(i,j,\sigma)$ to successfully execute the tasks T_i to T_j in all subsequent re-executions using the second set of single verifications with speed σ. Similarly to the proof of the Time-VC-Only problem in Section 5.1.1, $T_{VC_{re}}(i,j,s,\sigma)$ always includes the cost $Time_{V_{first}}(i,j,s)$ regardless of whether a error strikes during the first execution. Let $p^E_{i,j}(s)$ denote the probability that at least one error is detected in the first execution, and it is again given by $p^E_{i,j}(s) = p^F_{i,j}(s) + \Big(1-p^F_{i,j}(s)\Big)\, p^S_{i,j}(s)$. If an error indeed strikes during the first execution, then we need to recover from the last checkpoint and use $Time_V(i,j,\sigma)$ to re-execute all the tasks from T_i to T_j with the second speed until we pass the next checkpoint. Therefore, we have

$$T_{VC_{re}}(i,j,s,\sigma) = Time_{V_{first}}(i,j,s) + p^E_{i,j}(s)\,(R_{i-1} + Time_V(i,j,\sigma)) \ . \qquad (16)$$

Here, $Time_V(i,j,\sigma)$ follows the same dynamic programming formulation as in Section 4.3 but using speed σ. $Time_{V_{first}}(i,j,s)$, on the other hand, denotes the optimal expected time to execute the tasks T_i to T_j at speed s until the first error strikes. Hence, it should not include the recovery cost nor the re-executions. The following describes a dynamic programming formulation to compute $Time_{V_{first}}(i,j,s)$:

$$Time_{V_{first}}(i,j,s) = \min_{i-1 \le l < j} \left\{ Time_{V_{first}}(i,l,s) + \left(1 - p^E_{i,l}(s)\right) \left(T_{V_{first}}(l+1,j,s)\right) \right\} ,$$

where $p^E_{i,l}(s) = p^F_{i,l}(s) + (1 - p^F_{i,l}(s))p^S_{i,l}(s)$ is the probability that at least one error is detected when executing the tasks T_i to T_l, and $T_{V_{first}}(i,j,s)$ denotes the expected time to execute the tasks T_i to T_j with both T_{i-1} and T_j verified. In particular, the computation of $T_{V_{first}}(i,j,s)$ is exactly the same as that of $T_{C_{first}}(i,j,s)$ given in Equation (14). In this dynamic program, we include the second term only when no error has happened during the first term, otherwise we have to recover and re-execute the tasks with the second speed, which is handled by $Time_V(i,j,\sigma)$. Finally, we initialize this dynamic program with $Time_{V_{first}}(i,i-1,s) = 0$ for all $i = 1,\ldots,n$.

The complexity is dominated by the computation of $Time_V(i,j,s)$ and $Time_{V_{first}}(i,j,s)$, both of which take $O(n^3)$ time. Therefore, the overall complexity remains the same as in the SingleSpeed scenario (Theorem 5). □

5.2 The MultiSpeed Scenario

In this section, we investigate the most flexible scenario, MultiSpeed, which is built upon the ReExecSpeed scenario, to get even more control over the expected execution time but at the cost of a higher complexity. Instead of having two fixed speeds, we are given a set $S = \{s_1, s_2, \cdots, s_K\}$ of K discrete speeds. We call a sequence of tasks between two checkpoints a *segment* of the chain, and we allow each segment to use one speed for the first execution, and a second speed for all potential re-executions. The two speeds can well be different for different segments.

5.2.1 The Time-VC-Only problem

Theorem 9. *For the* MultiSpeed *scenario, the* Time-VC-Only *problem can be solved by a dynamic programming algorithm in $O(n^2K^2)$ time.*

Proof. The proof is built upon that of Theorem 7 for the ReExecSpeed scenario. Here, we use $Time_{C_{mul}}(j)$ to denote the optimal expected time to successfully execute tasks T_1 to T_j, where T_j has a verified checkpoint and there are possibly other verified checkpoints in between. Also, we use $T_{C_{mul}}(i,j)$ to denote the optimal expected time to successfully execute the tasks T_i to T_j, where both T_{i-1} and T_j are verified and checkpointed. In both expressions, the two execution speeds for each segment can be arbitrarily chosen from the discrete set S. The goal is to find $Time_{C_{mul}}(n)$, and the dynamic program can be formulated as follows:

$$Time_{C_{mul}}(j) = \min_{0 \le i < j} \left\{ Time_{C_{mul}}(i) + T_{C_{mul}}(i+1,j) \right\} + C_j ,$$

which is initialized with $Time_{C_{mul}}(0) = 0$. Recall that $T_{C_{re}}(i,j,s,\sigma)$ from the ReExecSpeed scenario (see Equation (15)) already accounts for two speeds that are fixed. We can use it to compute $T_{C_{mul}}(i,j)$ by trying all possible speed pairs:

$$T_{C_{mul}}(i,j) = \min_{s,\sigma \in S} T_{C_{re}}(i,j,s,\sigma) .$$

The complexity is now dominated by the computation of $T_{C_{mul}}(i,j)$ for all (i,j) pairs with $i \leq j$, and it takes $O(n^2K^2)$ time. After $T_{C_{mul}}(i,j)$ is computed, the dynamic programming table can then be filled in $O(n^2)$ time. □

5.2.2 The Time-VC+V problem

Theorem 10. *For the* MultiSpeed *scenario, the* Time-VC+V *problem can be solved by a dynamic programming algorithm in $O(n^3K^2)$ time.*

Proof. The proof is similar to that of the Time-VC-Only problem in Theorem 9. Here, we replace $Time_{C_{mul}}(j)$ with $Time_{VC_{mul}}(j)$ and replace $T_{C_{mul}}(i,j)$ with $T_{VC_{mul}}(i,j)$. Again, the two expressions denote the optimal execution times with the best speed pair chosen from S for each segment. The goal is to find $Time_{VC_{mul}}(n)$, and the dynamic program is formulated as follows:

$$Time_{VC_{mul}}(j) = \min_{0 \leq i < j} \left\{ Time_{VC_{mul}}(i) + T_{VC_{mul}}(i+1,j) \right\} + C_j \ ,$$

which is initialized with $Time_{VC_{mul}}(0) = 0$. We can compute $T_{VC_{mul}}(i,j)$ from $T_{VC_{re}}(i,j,s,\sigma)$ (see Equation (16)) by trying all possible speed pairs:

$$T_{VC_{mul}}(i,j) = \min_{s,\sigma \in S} T_{VC_{re}}(i,j,s,\sigma) \ .$$

The complexity is still dominated by the computation of $T_{VC_{mul}}(i,j)$, which amounts to computing $T_{VC_{re}}(i,j,s,\sigma)$ for all (i,j) pairs and for K^2 possible pairs of speeds (see Theorem 8). Therefore, the overall complexity is $O(n^3K^2)$. □

5.3 Bi-criteria problems

Results nicely extend to the bi-criteria problems. We state the following theorem without proof, because it is similar to that of Theorem 6.

Theorem 11. *Consider a chain of tasks with the objective of minimizing a linear combination of execution time and energy consumption as shown in Equation (8).*
(i) In the ReExecSpeed *scenario, the* TimeEnergy-VC-Only *problem can be solved in $O(n^2)$ time and the* TimeEnergy-VC+V *problem can be solved in $O(n^3)$ time.*
(ii) In the MultiSpeed *scenario, the* TimeEnergy-VC-Only *problem can be solved in $O(n^2K^2)$ time and the* TimeEnergy-VC+V *problem can be solved in $O(n^3K^2)$ time.*

6 Simulations

We conduct simulations to evaluate the performance of the dynamic programming algorithms under different execution scenarios and parameter settings. We instantiate the model parameters with realistic values taken from the literature, and we point out that the code for all algorithms and simulations is publicly available at `http://graal.ens-lyon.fr/~yrobert/failstop-silent`, so that interested readers can build relevant scenarios of their choice.

6.1 Simulation settings

We generate a linear chain with different number n of tasks while keeping the total computational cost at $W = 5 \times 10^4$ seconds $\approx$ 14 hours. The total amount of computation is distributed among the tasks in three different patterns:

1. *Uniform:* all tasks share the same cost W/n, as in matrix multiplication or in some iterative stencil kernels.
2. *Decrease:* task T_i has cost $\alpha \cdot (n+1-i)^2$, where $\alpha \approx 3W/n^3$. This quadratically decreasing function resembles some dense matrix solvers, e.g., by using LU or QR factorization.
3. *HighLow:* a set of identical tasks with large cost is followed by tasks with small cost. This distribution is created to distinguish the performance of different execution scenarios. In the default setting, we assume that 10% of the tasks are large and they contain 60% of the total computational cost. We will also vary these parameters to evaluate their impact on performance.

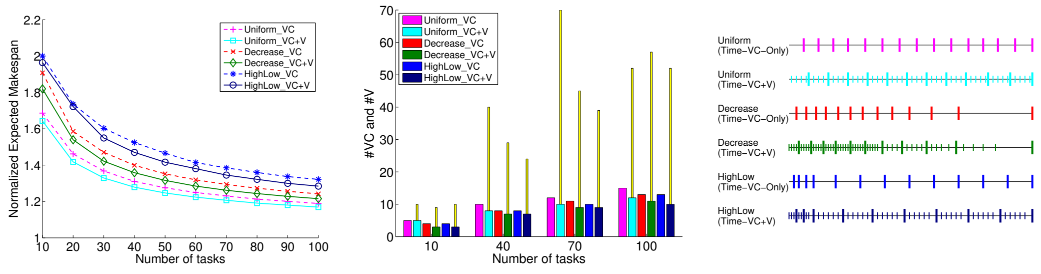
We adopt the set of speeds from the Intel Xscale processor. Following [35], the normalized set of speeds is $S = \{0.15, 0.4, 0.6, 0.8, 1\}$ and the fitted power function is given by $P(s) = 1550s^3 + 60$. From the discussion in Section 2.3, we assume the following model for the average error rate of fail-stop errors:

$$\lambda^F(s) = \lambda^F_{ref} \cdot 10^{\frac{d \cdot |s_{ref} - s|}{s_{max} - s_{min}}} , \tag{17}$$

where $s_{ref} \in [s_{min}, s_{max}]$ denotes the reference speed with the lowest error rate λ^F_{ref} among all possible speeds in the range. The above equation allows us to account for higher fail-stop error rates when the CPU speed is either too low or too high. In the simulations, the reference speed is set to be $s_{ref} = 0.6$ with an error rate of $\lambda^F_{ref} = 10^{-5}$ for fail-stop errors, and the sensitivity parameter is set to be $d = 3$. These parameters represent realistic settings reported in the literature [2, 4, 42], and they correspond to $0.83 \sim 129$ errors over the entire chain of computation depending on the processing speed chosen.

For silent errors, we assume that the error rate is related to that of the fail-stop errors by $\lambda^S(s) = \eta \cdot \lambda^F(s)$, where $\eta > 0$ is the relative parameter. To achieve realistic scenarios, we try to vary η to assess the impact of both error sources on the performance. However, we point out that our approach is completely independent of the evolution of the error rates as a function of the speed. In a practical setting, we are given a set of discrete speeds and two error rates for each speed, one for fail-stop errors and one for silent errors. This is enough to instantiate our model.

In addition, we define cr to be the ratio between the checkpointing/recovery cost and the computational cost for the tasks, and define vr to be the ratio between the verification cost and the computational cost. By default, we execute the tasks using the reference speed s_{ref}, and we set $\eta = 1$, $cr = 1$ and $vr = 0.01$. This setting corresponds to the case where fail-stop and silent errors have similar probabilities to strike the system. Moreover, the tasks have costly checkpoints (same order of magnitude as the computational costs) and lightweight verifications (average cost 1% of computational costs); examples of such tasks are data-oriented kernels processing large files and checksumming for verification. We will also vary these parameters to study their impact.



(a) (b) (c)

Figure 6: Impact of n and cost distribution on the performance of the Time-VC-Only and Time-VC+V algorithms. In (b), the thick bars represent the number of verified checkpoints and the yellow thin bars represent the total number of verifications. In (c), the number of tasks is fixed at $n = 100$. The long vertical bars mark the positions of the verified checkpoints within the task chain, whereas the short vertical bars mark the positions of the additional verifications.

6.2 Results

We first focus on the SingleSpeed scenario and the minimization of the execution time (Section 6.2.1). Then, we discuss in Section 6.2.2 results when energy is minimized instead of time, and for a linear combination of time and energy, still with SingleSpeed. Finally, Section 6.2.3 shows the gains achieved by using several speeds, through the ReExecSpeed and MultiSpeed scenarios for a linear chain of tasks. A summary of results is given in Section 6.2.4.

6.2.1 The SingleSpeed scenario for execution time

The first set of simulations is devoted to the evaluation of the time optimization algorithms in the SingleSpeed scenario.

Impact of number of tasks and cost distribution Figure 6(a) shows the expected execution time, normalized by the error-free execution time at the reference speed, i.e.,

$$Time_{ref} = \frac{W}{s_{ref}} , \tag{18}$$

with different number n of tasks and different cost distributions. The results show that having more tasks reduces the expected execution time (for a fixed total computation), since it enables the algorithms to place more checkpoints and verifications, as can be seen in Figure 6(b). The distribution that renders a larger variation in the costs of the tasks create more difficulty in the placement of checkpoints/verifications, thus resulting in worse execution time. Figure 6(c) shows, for $n = 100$ tasks, that the checkpoints and verifications are placed evenly within the task chain for the *Uniform* distribution, while for *Decrease* and *HighLow* distributions, they are placed more densely at the beginning, where large tasks need to be checkpointed and/or verified for better resilience. The figure also compares the performance of the Time-VC-Only algorithm with that of Time-VC+V algorithm. The latter, being more flexible, leads to improved execution time under all cost distributions. Because of the additionally placed verifications, it also reduces the number of checkpoints in the optimal solution.

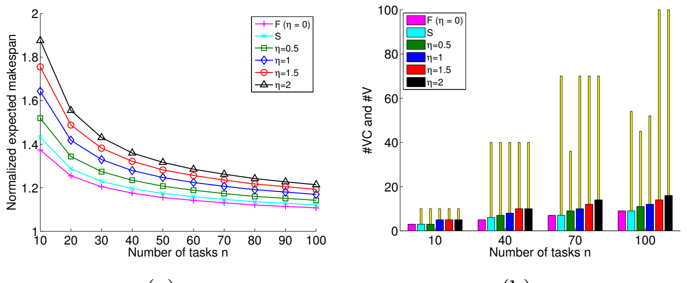


Figure 7: Impact of η on the performance for the *Uniform* distribution. F denotes fail-stop error only and S denotes silent error only.

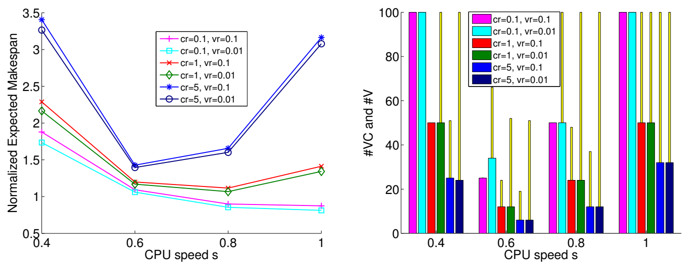
In the rest of this section, we mainly focus on the evaluation of the Time-VC+V algorithm.

Impact of error mode and relative ratio Figure 7(a) compares the performance of the Time-VC+V algorithm for the *Uniform* distribution under different error modes, namely, fail-stop (F) only, silent (S) only, and fail-stop plus silent with different values of η. As silent errors are harder to detect and hence to deal with, the S-only case leads to larger execution times than the F-only case. In the presence of both types of errors, the execution times become worse with larger η, i.e., with increased rate for silent errors, despite the algorithm's effort to place more checkpoints and more verifications, as shown in Figure 7(b). Similar results (not shown) are also observed for the other cost distributions.

In the subsequent simulations, we concentrate on $n = 100$ tasks in the presence of both fail-stop and silent errors with $\eta = 1$.

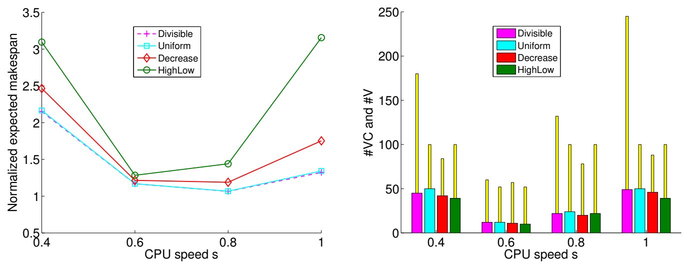
Impact of checkpointing and verification ratios Figure 8(a) presents the impact of checkpointing/recovery ratio (cr) and verification ratio (vr) on the performance of the Time-VC+V algorithm under different CPU speeds for the *Uniform* distribution. For a given speed, a small cr (or vr) enables the algorithm to place more checkpoints (or verifications), which leads to a better execution time. Moreover, the performance degrades significantly as the CPU speed is set below the reference speed s_{ref}, because the error rate increases exponentially. A higher CPU speed also increases the error rate, but it improves the execution time, at least for small values of cr, by executing the tasks faster with more checkpoints. Finally, if the checkpointing cost is on par with the verification cost (e.g., $cr = 0.1$), reducing the verification cost can additionally increase the number of checkpoints (e.g., at $s = 0.6$), since each checkpoint also has a verification cost associated with it. For a high checkpointing cost, however, reducing the verification cost no longer influences the algorithm's checkpointing decisions.

Comparison with divisible load application Figure 9(a) compares the execution time of the Time-VC+V algorithm for the three linear task distributions with that of the periodic checkpointing and verification algorithm for a divisible load application. Figure 9(b) also shows the number of checkpoints and verifications placed in each case. Note that, for the divisible load application, the total computational cost, the checkpointing cost and the verification cost are set to be the same as the corresponding costs of a discrete task under the



(a) (b)

Figure 8: Impact of *cr* and *vr* on the performance with different CPU speeds for the *Uniform* distribution. Speed $s = 0.15$ yields extremely large execution time, which is omitted in the figure.



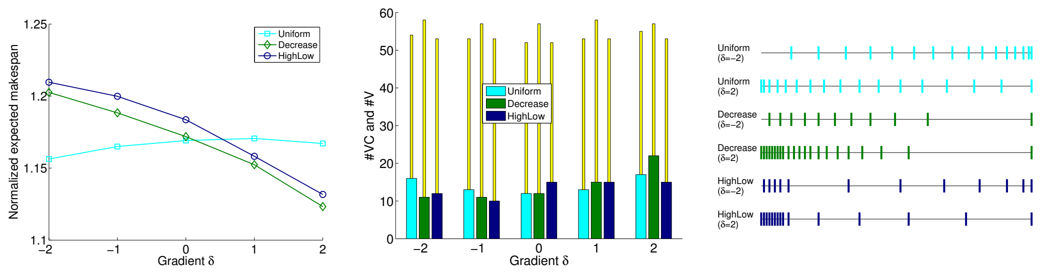
(a) (b)

Figure 9: Performance comparison of the TIME-VC+V algorithm and the periodic checkpointing and verification algorithm for divisible load application.

Uniform distribution. We see that the execution time for uniform tasks is almost identical to that of the divisible load under all CPU speeds, while the performance degrades significantly for the other two distributions with larger variations in the costs of the tasks. Similar results (not shown) are also observed when comparing the TIME-VC-ONLY algorithm with the periodic checkpointing algorithm for divisible load. Moreover, because a divisible load application does not impose restrictions in the checkpointing and verification positions, there tends to be more verifications (or checkpoints in the case of periodic checkpointing algorithm) than for discrete tasks, especially when the CPU speed is further away from the reference speed, and hence the error rate is high.

In view of these results, we could imagine the following greedy algorithm as an alternative to the TIME-VC-ONLY and TIME-VC+V algorithms for a linear chain of tasks with *Uniform* cost: position the next checkpoint or verification as soon as the time spent on computing since the last checkpoint or verification exceeds the optimal periods given by Theorems 1 and 2. The results here suggest that this linear-time algorithm (with complexity $O(n)$) would give a good approximation to the optimal solution (returned by the TIME-VC-ONLY algorithm with complexity $O(n^2)$ or the TIME-VC+V algorithm with complexity $O(n^3)$).

Performance with independent checkpointing cost We now consider the case where the checkpointing costs are independent of the tasks' computational costs. To assess the impact, we generate different patterns by varying the checkpointing costs linearly within the task chain while keeping the sum a constant. Specifically, we use the following function to



(a) (b) (c)

Figure 10: Impact of δ on the performance when the checkpointing cost for task T_i is given by $C_i = C\left(1 + \frac{\delta}{n}(i - \frac{n}{2})\right)$, where $C = \frac{W}{n}$.

generate the checkpointing cost $C_i = C\left(1 + \frac{\delta}{n}(i - \frac{n}{2})\right)$ for each task T_i $(1 \leq i \leq n)$, where $C = W/n$ denotes the average checkpointing cost. Here, δ represents the gradient of the linear function: $\delta = 0$ means that all tasks have the same checkpointing cost, $\delta > 0$ increases the checkpointing cost as more tasks are processed, and $\delta < 0$ decreases the checkpointing cost.

Figure 10 shows the performance of the TIME-VC+V algorithm for different values of δ under the reference speed $s_{ref} = 0.6$. When δ increases, the execution time for tasks with *Uniform* computational cost is barely affected, while significant improvements are observed for the other two distributions. Indeed, the algorithms for the *Decrease* and the *HighLow* distributions place more checkpoints at the beginning of the task chain, where the large tasks are located. Therefore, reducing the checkpointing costs for these tasks decreases the execution overhead and hence the overall execution time.

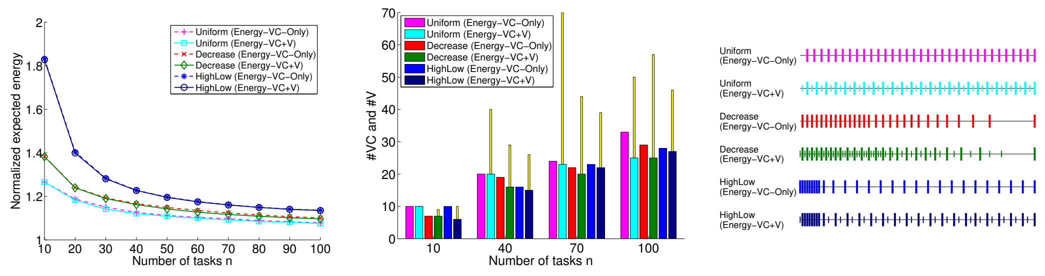
6.2.2 The SingleSpeed scenario for energy and energy-time trade-off

This set of simulations evaluates the energy optimization algorithms (i.e., obtained by setting $a = 0$ in Equation (9), and denoted as ENERGY-VC-ONLY and ENERGY-VC+V) as well as the energy-time trade-off, in the SINGLESPEED scenario. The default power parameters are set to be $P_{idle} = 60$ and $P_{cpu}(s) = 1550s^3$, according to [35]. The dynamic power consumption P_{io} due to I/O is equal to the dynamic power of the CPU at the lowest discrete speed 0.15. We vary these parameters to study their impact.

Impact of number of tasks and cost distribution Figure 11(a) shows the expected energy consumption of the energy optimization algorithm, normalized by the error-free energy consumption at the reference speed, i.e.,

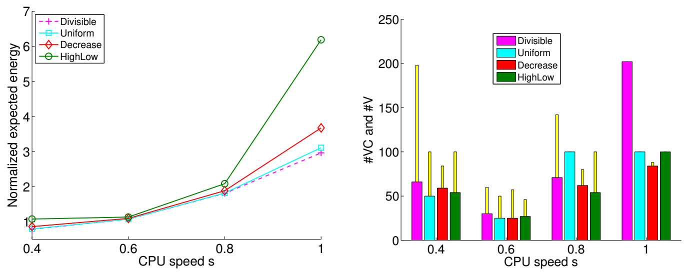
$$Energy_{ref} = \frac{W(P_{idle} + P_{cpu}(s_{ref}))}{s_{ref}} , \tag{19}$$

As with the time optimization algorithms (see Figure 6), a larger number of tasks improves the performance, while a larger variation in the tasks' costs worsens the performance. Unlike the case with time, the performance difference between ENERGY-VC-ONLY and ENERGY-VC+V is less evident. The reason is that the checkpointing cost is much smaller in terms of energy consumption, so more checkpoints are placed (compare Figures 11(b) and 6(b)), which reduces the number of additional verifications required and hence their benefits.



(a) (b) (c)

Figure 11: Impact of n and cost distribution on the performance of the Energy-VC-Only and Energy-VC+V algorithms. In (c), the number of tasks is fixed at $n = 100$.

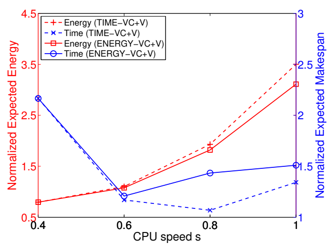


(a) (b)

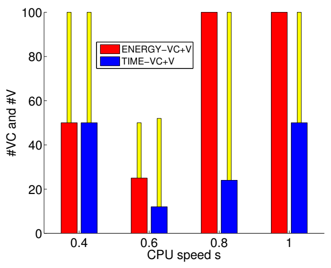
Figure 12: Performance comparison of the Energy-VC+V algorithm and the periodic checkpointing and verification algorithm for divisible load application.

Comparison with divisible load application Figure 12 compares the performance of the Energy-VC+V algorithm for $n = 100$ tasks with that of the periodic checkpointing and verification algorithm for a divisible load application. Similarly to the execution time case (see Figure 9), the energy consumed for tasks with *Uniform* distribution is very close to that for the divisible load application, which admits more verifications and/or checkpoints due to the flexible application model. In addition, the optimal energy is achieved by setting the CPU speed below the reference, i.e, at $s = 0.4$. This is in contrast to the Time-VC+V algorithm, which achieves the optimal execution time at $s = 0.8$, a higher speed than the reference.

Energy-time trade-off We now study the energy-time trade-off exhibited by the Energy-VC+V and Time-VC+V algorithms. Figure 13 compares their performance in terms of both time and energy when executing $n = 100$ tasks with the *Uniform* distribution. At speed $s = 0.4$, the power consumed by the CPU is still comparable to the I/O power. This yields the same number of checkpoints placed by the two algorithms, which in turn leads to the same performance for time and energy. As the speed s increases, the I/O power becomes relatively cheaper, so Energy-VC+V tends to place more checkpoints to improve the energy at the expense of execution time, and the performance difference of the two algorithms becomes more obvious. The result indicates that running at the reference speed $s = 0.6$ offers a good trade-off with reasonable performance in both energy and time, while running at $s = 0.4$ suffers from a large execution time and running at $s = 0.8$ has a large energy consumption.

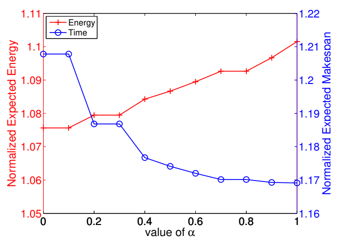


(a)

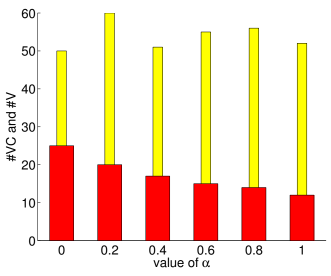


(b)

Figure 13: Performance of the ENERGY-VC+V and TIME-VC+V algorithms with different CPU speeds.



(a)



(b)

Figure 14: Performance of the TIMEENERGY-VC+V algorithm with different values of α.

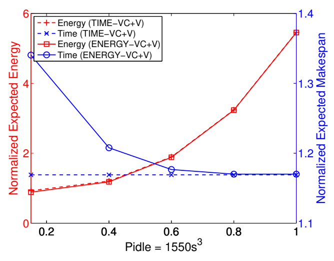
Linear combination of time and energy To further understand the energy-time trade-off, we evaluate the TIMEENERGY-VC+V algorithm that minimizes a linear combination of the two objectives. To make sure that the values of time and energy are in the same range, both quantities are normalized by their respective reference values shown in Equations (18) and (19). Thus, the weights in Equation (8) are set as follows:

$$a = \frac{\alpha}{Time_{ref}} \ , \tag{20}$$

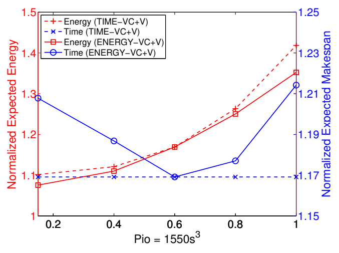
$$b = \frac{1-\alpha}{Energy_{ref}} \ , \tag{21}$$

where $\alpha \in [0,1]$ gives the relative importance of time in the optimization. Figure 14 shows the performance of the resulting algorithm for different values of α at the reference speed $s = 0.6$. We can see that a larger value of α reduces the execution time by placing fewer checkpoints, which adversely increases the energy consumption, and vice versa. Clearly, the behavior of the algorithm evolves from ENERGY-VC+V to TIME-VC+V as α increases from 0 to 1, and one can choose the value of α according to the desired level of trade-off between energy and time.

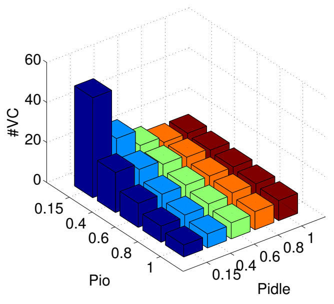
Impact of P_{idle} and P_{io} Figures 15(a) and 15(b) show the performance of the TIME-VC+V and ENERGY-VC+V algorithms by varying P_{idle} and P_{io} separately according to the dynamic power function $1550s^3$, while keeping the other one at the smallest CPU power, i.e., $1550 \cdot 0.15^3$. The CPU speed is fixed at $s = 0.6$. Figure 15(c) further shows the number of



(a)



(b)



(c)

Figure 15: Impact of P_{idle} and P_{io} on the performance of the ENERGY-VC+V and TIME-VC+V algorithms at the reference speed $s = 0.6$. The number of checkpoints placed by ENERGY-VC+V with different P_{io}, P_{idle} values ($= 1550s^3$) is shown in (c), while TIME-VC+V always places 11 checkpoints in this simulation.

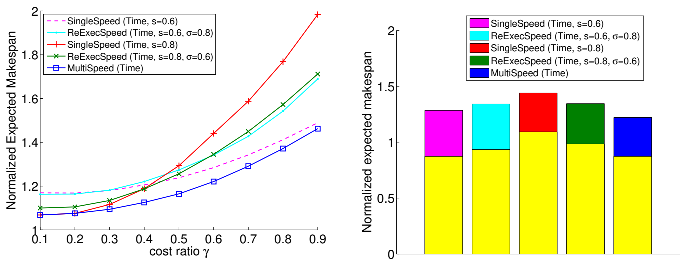
checkpoints placed by the ENERGY-VC+V algorithm at different P_{idle} and P_{io} values. The TIME-VC+V algorithm is not affected by these two parameters, so it always places the same number of checkpoints (11 in this simulation) and results in the same execution time, while the energy consumed increases with P_{idle} and P_{io}.

Setting the smallest value for both parameters creates a big gap between the CPU and I/O power consumptions. This leads to a larger number of checkpoints placed by the ENERGY-VC+V algorithm, which improves its energy consumption at the expense of execution time. Increasing P_{idle} closes this gap and hence reduces the number of checkpoints, which leads to the performance convergence of the two algorithms for both energy and time. While increasing P_{io} has the same effect, a larger value than $P_{cpu} = 1550 \cdot 0.6^3$ further reduces the number of checkpoints placed by ENERGY-VC+V below 11, since checkpointing starts to be less energy-efficient. This again gives ENERGY-VC+V advantage in terms of energy but degrades its performance in time.

6.2.3 The ReExecSpeed and MultiSpeed scenarios

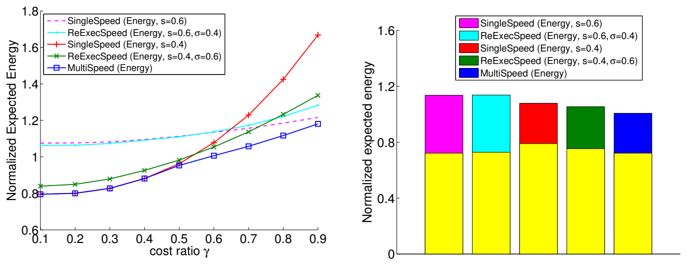
This last set of simulations evaluates the REEXECSPEED and MULTISPEED scenarios for execution time, energy consumption, and a linear combination of time and energy. To distinguish them from the SINGLESPEED scenario, we consider the *HighLow* distribution, which yields a larger variance among the computational costs of the tasks. In the simulations, we again focus on the VC+V algorithms for $n = 100$ tasks, and vary the *cost ratio* γ, which is the percentage of the large tasks' computational cost in the total computational cost.

Comparison of different execution scenarios Figure 16(a) compares the execution time of the TIME-VC+V algorithms under the three execution scenarios. For the SINGLESPEED and REEXECSPEED scenarios, only $s = 0.6$ and $s = 0.8$ are drawn, since the other speeds lead to much larger execution time. For small cost ratios, no task has a very large computational cost, so the distribution is close to *Uniform*. In this case, the faster speed $s = 0.8$ offers the best performance despite its higher error rate, as we have seen in Figure 9(a). When the cost ratio increases, tasks with larger cost start to emerge. At the high error rate of $s = 0.8$, these tasks will experience much more errors and re-executions, so their execution time will



(a) (b)

Figure 16: Performance comparison of the TIME-VC+V algorithms in MULTISPEED, REEXECSPEED and SINGLESPEED scenarios for $n = 100$ tasks under *HighLow* distribution. In (b), the cost ratio is fixed at $\gamma = 0.6$, and the yellow part at the bottom of each bar represents the expected execution time for the large tasks.



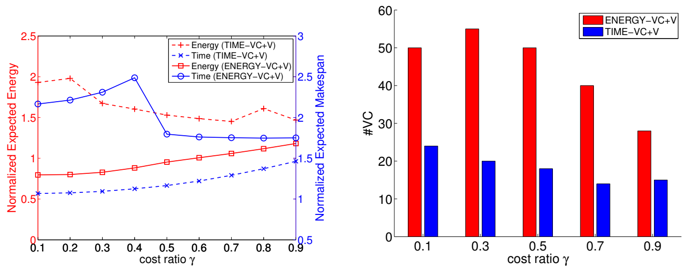
(a) (b)

Figure 17: Performance comparison of the ENERGY-VC+V algorithms in MULTISPEED, REEXECSPEED and SINGLESPEED scenarios for $n = 100$ tasks with *HighLow* distribution. In (b), the cost ratio is fixed at $\gamma = 0.6$, and the yellow part at the bottom of each bar represents the expected energy consumption for the large tasks.

dominate the overall execution time. Therefore, for large cost ratios, $s = 0.6$ becomes the best speed due to its smaller error rate, which was also observed in Figure 9(a) under the *HighLow* distribution.

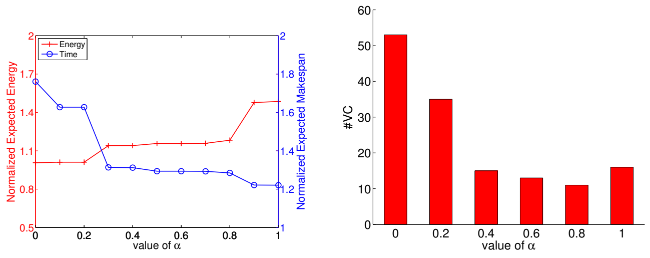
In the REEXECSPEED scenario, we observe that the best re-execution speed σ, regardless of the initial speed s, is similarly determined by the computational costs of the tasks, which are in turn decided by the cost ratio. Figure 16(b) shows, for cost ratio $\gamma = 0.6$, that setting $\sigma = 0.6$ improves the execution of the big tasks but degrades the performance of the small tasks. On the other hand, setting $\sigma = 0.8$ helps the small tasks but hurts the big tasks. These simulations suggest that, despite the ability to select a more appropriate speed for the re-executions, the REEXECSPEED scenario presents limited benefits compared to the best performance achievable in the SINGLESPEED scenario. The MULTISPEED scenario, with its flexibility to choose different speeds depending on the costs of the tasks, offers clear performance gains. The advantage is especially evident at medium cost ratio, where a good mix of large and small tasks coexist, a situation that is very hard to cope with by using fixed speed(s).

Similar results can also be observed for the ENERGY-VC+V algorithms in the three scenarios, which are shown in Figure 17. Note that, in terms of energy consumption, speed $s = 0.4$ is more suitable for small tasks due to its better power efficiency, while speed $s = 0.6$



(a) (b)

Figure 18: Impact of γ on the performance of the Energy-VC+V and Time-VC+V algorithms in the MultiSpeed scenario.



(a) (b)

Figure 19: Performance of the TimeEnergy-VC+V algorithm with different values of α in the MultiSpeed scenario.

is more suitable for big tasks due to its lower error rate. Again, the most flexible MultiSpeed scenario is able to choose between the two speeds depending on the costs of the tasks, and hence it offers the best overall performance.

Energy-time trade-off in the MultiSpeed scenario Figure 18 shows the performance of the Energy-VC+V and Time-VC+V algorithms for both time and energy in the MultiSpeed scenario. Since small cost ratios favor speed $s = 0.4$ for energy and $s = 0.8$ for time, the two algorithms experience a large performance difference, by more than 100% in both execution time and energy consumption. Increasing the cost ratio creates more computationally demanding tasks, which need to be executed at speed $s = 0.6$ in order to optimize both objectives as it incurs fewer errors. This closes the performance gap of the two algorithms as well as the number of checkpoints placed by them, because the total computational cost in the small tasks shrinks and fewer checkpoints are required among them.

Figure 19 shows the performance of the TimeEnergy-VC+V algorithm for minimizing a linear combination of time and energy in the MultiSpeed scenario. Again, the weights are set according to Equations (20) and (21), the same as the SingleSpeed scenario. We can clearly observe the energy-time trade-off as α is varied from one extreme to the other. The result suggests that setting $\alpha \in [0.3, 0.8]$ seems to offer a good compromise between energy and time, as both quantities turn out to be not too far away from their respective optimal values.

6.2.4 Summary

We have evaluated and compared various algorithms under different execution scenarios, resilience protocols, and parameter settings. In general, the algorithms under the most flexible VC+V and MultiSpeed scenario provide the best overall performance, which in practice translates to shorter execution time or lower energy consumption.

For tasks with similar computational cost as in the *Uniform* distribution, the SingleSpeed algorithm, or the greedy approximation in the context of divisible load application, could provide comparable solutions with lower computational complexity. The ReExecSpeed algorithms provide marginal benefit compared to SingleSpeed, but clear performance gains are observed from the MultiSpeed algorithms, especially for tasks with very different costs. The results also show that the optimal solutions are often achieved by processing around the reference speed that yields the least number of failures.

For the complexity of computing the optimal solutions, we point out that application workflows rarely exceed a few tens of tasks. In such practical contexts, even the most advanced algorithms have a very fast execution time, of a few seconds. To give a number, the Time-VC+V algorithm in the MultiSpeed scenario requires less than one second to execute for 100 tasks and 5 speed levels on a 3.7Ghz single-core processor. Hence, all our algorithms can be applied to determine the optimal checkpointing and verification locations with almost negligible cost.

7 Conclusion

In this paper, we have presented a general-purpose solution that combines checkpointing and verification mechanisms to cope with fail-stop errors and silent data corruptions. We have extended the classical formula of Young/Daly for a divisible load application while incorporating both resilience techniques in the presence of both error sources. By using dynamic programming, we have devised polynomial-time algorithms that decide the optimal checkpointing and verification locations for a linear chain of tasks. The algorithms can be applied to several execution scenarios to minimize either the expected execution time, or energy consumption, or a linear combination of both objectives. The results are supported by a set of extensive simulations, which demonstrate the quality and trade-off of our optimal algorithms under a wide range of parameter settings.

Further work will be devoted to using DVFS for divisible load applications. However, computing the optimal checkpointing period when different speeds are used, seems to be quite a challenging problem. Another future direction is to extend our study from linear chains to other application workflows, such as tree graphs, fork-join graphs, series-parallel graphs, or even general directed acyclic graphs (DAGs).

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, by the LABEX MILYON (ANR-10- LABX-0070) of Université de Lyon, within the program "Investissements d'Avenir" (ANR-11-IDEX-0007) operated by the French National Research Agency (ANR), and by the PIA ELCI project. Yves Robert is with Institut Universitaire de France.

References

[1] S. Albers and H. Fujiwara. Energy-efficient algorithms for flow time minimization. *ACM Transactions on Algorithms*, 3(4), 2007.

[2] I. Assayad, A. Girault, and H. Kalla. Tradeoff exploration between reliability, power consumption, and execution time for embedded systems. *International Journal on Software Tools for Technology Transfer*, 15(3):229–245, 2013.

[3] G. Aupy, A. Benoit, T. Hérault, Y. Robert, F. Vivien, and D. Zaidouni. On the combination of silent error detection and checkpointing. In *Proceedings of the 19th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC)*, pages 11–20, 2013.

[4] G. Aupy, A. Benoit, and Y. Robert. Energy-aware scheduling under reliability and makespan constraints. In *Proceedings of the International Conference on High Performance Computing (HiPC)*, pages 1–10, 2012.

[5] N. Bansal, T. Kimbrel, and K. Pruhs. Speed scaling to manage energy and temperature. *Journal of the ACM*, 54(1):3:1–3:39, 2007.

[6] A. R. Benson, S. Schmit, and R. Schreiber. Silent error detection in numerical time-stepping schemes. *The International Journal of High Performance Computing Applications*, DOI: 10.1177/1094342014532297, 2014.

[7] G. Bosilca, R. Delmas, J. Dongarra, and J. Langou. Algorithm-based fault tolerance applied to high performance computing. *J. Parallel Distrib. Comput.*, 69(4):410–416, 2009.

[8] M. Bougeret, H. Casanova, M. Rabie, Y. Robert, and F. Vivien. Checkpointing strategies for parallel jobs. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pages 1–11, 2011.

[9] G. Bronevetsky and B. de Supinski. Soft error vulnerability of iterative linear algebra methods. In *Proceedings of the International Conference on Supercomputing (ICS)*, pages 155–164, 2008.

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

[11] Z. Chen. Online-ABFT: An online algorithm based fault tolerance scheme for soft error detection in iterative methods. In *Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP)*, pages 167–176, 2013.

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

[13] A. Das, A. Kumar, B. Veeravalli, C. Bolchini, and A. Miele. Combined DVFS and mapping exploration for lifetime and soft-error susceptibility improvement in MPSoCs. In *Proceedings of the Conference on Design, Automation & Test in Europe (DATE)*, pages 61:1–61:6, 2014.

[14] A. Dixit and A. Wood. The impact of new technology on soft error rates. In *IEEE International on Reliability Physics Symposium (IRPS)*, pages 5B.4.1–5B.4.7, 2011.

[15] D. R. Dooly, S. A. Goldman, and S. D. Scott. On-line analysis of the tcp acknowledgment delay problem. *Journal of the ACM*, 48(2):243–273, 2001.

[16] N. El-Sayed, I. A. Stefanovici, G. Amvrosiadis, A. A. Hwang, and B. Schroeder. Temperature management in data centers: Why some (might) like it hot. *SIGMETRICS Perform. Eval. Rev.*, 40(1):163–174, 2012.

[17] J. Elliott, K. Kharbas, D. Fiala, F. Mueller, K. Ferreira, and C. Engelmann. Combining partial redundancy and checkpointing for HPC. In *Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 615–626, 2012.

[18] E. N. M. Elnozahy, L. Alvisi, Y.-M. Wang, and D. B. Johnson. A survey of rollback-recovery protocols in message-passing systems. *ACM Computing Survey*, 34:375–408, 2002.

[19] A. Fabrikant, A. Luthra, E. Maneva, C. H. Papadimitriou, and S. Shenker. On a network creation game. In *Proceedings of the Twenty-second Annual Symposium on Principles of Distributed Computing*, PODC '03, pages 347–351, 2003.

[20] W.-C. Feng. Making a case for efficient supercomputing. *Queue*, 1(7):54–64, Oct. 2003.

[21] 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 *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, page 78, 2012.

[22] M. Heroux and M. Hoemmen. Fault-tolerant iterative methods via selective reliability. Research report SAND2011-3915 C, Sandia National Laboratories, 2011.

[23] C.-H. Hsu and W.-C. Feng. A power-aware run-time system for high-performance computing. In *Proceedings of the ACM/IEEE Supercomputing Conference (SC)*, pages 1–9, 2005.

[24] K.-H. Huang and J. A. Abraham. Algorithm-based fault tolerance for matrix operations. *IEEE Trans. Comput.*, 33(6):518–528, 1984.

[25] A. A. Hwang, I. A. Stefanovici, and B. Schroeder. Cosmic rays don't strike twice: understanding the nature of dram errors and the implications for system design. *SIGARCH Comput. Archit. News*, 40(1):111–122, 2012.

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

[27] M. Kargar, A. An, and M. Zihayat. Efficient bi-objective team formation in social networks. In *Proceedings of the 2012 European Conference on Machine Learning and Knowledge Discovery in Databases - Volume Part II*, ECML PKDD'12, pages 483–498, 2012.

[28] G. Lu, Z. Zheng, and A. A. Chien. When is multi-version checkpointing needed? In *Proceedings of the 3rd Workshop on Fault-tolerance for HPC at extreme scale (FTXS)*, pages 49–56, 2013.

[29] R. E. Lyons and W. Vanderkulk. The use of triple-modular redundancy to improve computer reliability. *IBM J. Res. Dev.*, 6(2):200–209, 1962.

[30] A. Moody, G. Bronevetsky, K. Mohror, and B. R. d. Supinski. Design, Modeling, and Evaluation of a Scalable Multi-level Checkpointing System. In *Proc. of the ACM/IEEE SC Conf.*, pages 1–11, 2010.

[31] X. Ni, E. Meneses, N. Jain, and L. V. Kalé. ACR: Automatic Checkpoint/Restart for Soft and Hard Error Protection. In *Proc. Int. Conf. High Performance Computing, Networking, Storage and Analysis*, SC '13. ACM, 2013.

[32] T. O'Gorman. The effect of cosmic rays on the soft error rate of a DRAM at ground level. *IEEE Trans. Electron Devices*, 41(4):553–557, 1994.

[33] T. Ozaki, T. Dohi, H. Okamura, and N. Kaio. Distribution-free checkpoint placement algorithms based on min-max principle. *IEEE Transactions on Dependable and Secure Computing*, 3(2):130–140, 2006.

[34] M. Patterson. The effect of data center temperature on energy efficiency. In *Proceedings of 11th Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems*, pages 1167–1174, 2008.

[35] N. B. Rizvandi, A. Y. Zomaya, Y. C. Lee, A. J. Boloori, and J. Taheri. Multiple frequency selection in dvfs-enabled processors to minimize energy consumption. In A. Y. Zomaya and Y. C. Lee, editors, *Energy-Efficient Distributed Computing Systems*. John Wiley & Sons, Inc., Hoboken, NJ, USA, 2012.

[36] P. Sao and R. Vuduc. Self-stabilizing iterative solvers. In *Proceedings of the Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA)*, 2013.

[37] O. Sarood, E. Meneses, and L. V. Kale. A 'cool' way of improving the reliability of HPC machines. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, pages 58:1–58:12, 2013.

[38] M. Shantharam, S. Srinivasmurthy, and P. Raghavan. Fault tolerant preconditioned conjugate gradient for sparse linear system solution. In *Proceedings of the ACM International Conference on Supercomputing (ICS)*, pages 69–78, 2012.

[39] S. Toueg and Ö. Babaoglu. On the optimum checkpoint selection problem. *SIAM J. Comput.*, 13(3):630–649, 1984.

[40] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *Proceedings of the 36th Annual Symposium on Foundations of Computer Science (FOCS)*, page 374, 1995.

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

[42] B. Zhao, H. Aydin, and D. Zhu. Reliability-aware dynamic voltage scaling for energy-constrained real-time embedded systems. In *Proceedings of the IEEE International Conference on Computer Design (ICCD)*, pages 633–639, 2008.

[43] D. Zhu, R. Melhem, and D. Mosse. The effects of energy management on reliability in real-time embedded systems. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 35–40, 2004.

[44] J. Ziegler, H. Muhlfeld, C. Montrose, H. Curtis, T. O'Gorman, and J. Ross. Accelerated testing for cosmic soft-error rate. *IBM J. Res. Dev.*, 40(1):51–72, 1996.

[45] J. Ziegler, M. Nelson, J. Shell, R. Peterson, C. Gelderloos, H. Muhlfeld, and C. Montrose. Cosmic ray soft error rates of 16-Mb DRAM memory chips. *IEEE Journal of Solid-State Circuits*, 33(2):246–252, 1998.

[46] J. F. Ziegler, H. W. Curtis, H. P. Muhlfeld, C. J. Montrose, and B. Chin. Ibm experiments in soft fails in computer electronics. *IBM J. Res. Dev.*, 40(1):3–18, 1996.

RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

Inovallée

655 avenue de l'Europe Montbonnot

38334 Saint Ismier Cedex

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

ISSN 0249-6399