Unified Model for Assessing Checkpointing Protocols at Extreme-Scale

George Bosilca, Aurélien Bouteiller, Élisabeth Brunet, Franck Cappello, Jack Dongarra, Amina Guermouche, Thomas Hérault, Yves Robert, Frédéric Vivien, Dounia Zaidouni

Unified Model for Assessing Checkpointing Protocols at Extreme-Scale

George Bosilca, Aurélien Bouteiller, Elisabeth Brunet, Franck Cappello, Jack Dongarra, Amina Guermouche, Thomas Herault, Yves Robert, Frédéric Vivien, Dounia Zaidouni

RESEARCH REPORT

N° 7950

May 2012

Project-Teams GRAND-LARGE,ROMA

ISSN 0249-6399 ISRN INRIA/RR--7950--FR+ENG

Unified Model for Assessing Checkpointing Protocols at Extreme-Scale

George Bosilca[*], Aurélien Bouteiller[*], Elisabeth Brunet[†], Franck Cappello[‡§], Jack Dongarra[*], Amina Guermouche[‡¶], Thomas Herault[*], Yves Robert[¶*‖], Frédéric Vivien[‡¶], Dounia Zaidouni[‡¶]

Project-Teams GRAND-LARGE,ROMA

Research Report n° 7950 — May 2012 — 37 pages

Abstract: In this article, we present a unified model for several well-known checkpoint/restart protocols. The proposed model is generic enough to encompass both extremes of the checkpoint/restart space, from coordinated approaches to a variety of uncoordinated checkpoint strategies (with message logging). We identify a set of crucial parameters, instantiate them and compare the expected efficiency of the fault tolerant protocols, for a given application/platform pair. We then propose a detailed analysis of several scenarios, including some of the most powerful currently available HPC platforms, as well as anticipated Exascale designs. The results of this analytical comparison are corroborated by a comprehensive set of simulations. Altogether, they outline comparative behaviors of checkpoint strategies at very large scale, thereby providing insight that is hardly accessible to direct experimentation.

Key-words: Fault-tolerance, checkpointing, coordinated, hierarchical, model, exascale

[*] University of Tennessee Knoxville, USA
[†] Telecom SudParis, France
[‡] INRIA
[§] University of Illinois at Urbana Champaign, USA
[¶] LIP, École Normale Supérieure de Lyon, France
[‖] Institut Universitaire de France

RESEARCH CENTRE
GRENOBLE – RHÔNE-ALPES

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

Un modèle unifié pour l'évaluation des protocoles de checkpoint à très large échelle

Résumé : Nous présentons ici un modèle unifié de plusieurs protocoles de sauvegarde de points de reprise (*checkpoints*) et de redémarrage. Le modèle proposé est suffisamment générique pour contenir les deux extrêmes des techniques de checkpoint/restart, d'une approche coordonnée à toute une famille de stratégies non-coordonnées (avec enregistrement de messages).

Nous identifions un ensemble de paramètres cruciaux, les instancions et comparons l'espérance de l'efficacité des protocoles de tolérance aux pannes, pour un couple donné application/plate-forme. Nous proposons une analyse détaillée de plusieurs scénarios, incluant certaines des plates-formes de calcul existantes les plus puissantes, ainsi que des anticipations sur les futures plates-formes exascale. Les résultats de cette analyse sont corroborés par un ensemble de simulations. Ensemble, ces résultats illustrent le comportement relatif des différentes stratégies à large échelle, fournissant des enseignements qu'il serait très difficile, voire impossible, d'obtenir par l'expérimentation directe.

Mots-clés : Tolérance aux pannes, checkpoint, modèle, coordonné, hiérarchique, exascale

1 Introduction

A significant research effort is focusing on the outline, characteristics, features, and challenges of High Performance Computing (HPC) systems capable of reaching the Exaflop performance mark [1, 2, 3, 4]. The portrayed Exascale systems will necessitate, billion way parallelism, resulting in a massive increase in the number of processing units (cores), but also in terms of computing nodes. Considering the relative slopes describing the evolution of the reliability of individual components on one side, and the evolution of the number of components on the other side, the reliability of the entire platform is expected to decrease, due to probabilistic amplification. Executions of large parallel HPC applications on these systems will have to tolerate a higher degree of errors and failures than in current systems. Preparation studies forecast that standard fault tolerance approaches (*e.g.*, coordinated checkpointing on parallel file system) will lead to unacceptable overheads at Exascale. Thus, it is not surprising that improving fault tolerance techniques is one of the main recommendations isolated by these studies.

In this paper we focus on techniques for tolerating the ultimate effect of detected and uncorrectable hard and soft errors: the crash of processes (undetected errors, also known as silent errors, are out-of-scope of this analysis). There are two main ways of tolerating process crashes, without undergoing significant application code refactoring: replication and rollback recovery. An analysis of replication feasibility for Exascale systems was presented in [5]. In this paper we focus on rollback recovery, and more precisely on the comparison of checkpointing protocols.

There are three main families of checkpointing protocols: (i) coordinated checkpointing; (ii) uncoordinated checkpointing with message logging; and (iii) hierarchical protocols mixing coordinated checkpointing and message logging. The key principle in all these checkpointing protocols is that all data and states necessary to restart the execution are regularly saved in process *checkpoints*. Depending on the protocol, theses checkpoints are or are not guaranteed to form consistent recovery lines. When a failure occurs, appropriate processes *rollback* to their last checkpoints and resume execution.

Each protocol family has serious drawbacks. Coordinated checkpointing and hierarchical protocols suffer a waste of computing resources when living processes have to rollback and recover from a checkpoint, to help tolerate failures. These protocols may also lead to I/O congestion when too many processes are checkpointing at the same time. Message logging increases the memory consumption, the checkpointing time, and slows-down the failure-free execution when messages are logged. Our objective is to identify which protocol delivers the best performance for a given application on a given platform. While several criteria could be considered to make such a selection, we focus on the most widely used metric, namely, the expectation of the total parallel execution time.

Fault-tolerant protocols have different overheads in fault-free and recovery situations. These overheads depend on many factors (type of protocols, application characteristics, system features, etc.) that introduce complexity and limit the scope of experimental comparisons as they have been done several times in the past [6, 7]. In this paper, we approach the fault tolerant protocol comparison from an analytical perspective. Our objective is to provide a valid performance model covering the most suitable rollback recovery protocols for HPC executions. Our model captures many optimizations proposed in the literature, and can be used to explore the effects of novel optimizations, and highlight the critical parameters to be considered when evaluating a protocol. The main contributions of this paper are:

1. to provide a comprehensive model that captures many consistent rollback recovery protocols, including coordinated checkpoint, uncoordinated checkpoint, and the composite hierarchical hybrids;
2. to provide a closed-form formula for the waste of platform computing resources incurred by each checkpointing protocol. This formula is the key to assessing existing and new protocols, and constitutes the first tool that can help the community to compare protocols at very large scale, and to guide design decisions for given application/platform pairs;
3. to instantiate the model on several realistic scenarios involving state-of-the-art platforms, and future Exascale ones, thereby providing practical insight and guidance.

The rest of this paper is organized as follows. Section 2 details the characteristics of available rollback recovery approaches, and the tradeoff they impose on failure-free execution and recovery. We also briefly discuss related work in this section. In Section 3, we describe our model that partially unifies coordinated rollback recovery approaches, and effectively captures coordinated, partially and totally uncoordinated approaches as well as many of their optimizations. We then use the model to analytically assess the performance of rollback recovery protocols. We instantiate the model with realistic scenarios in Section 4, and we present case-study results in Section 6, before concluding and presenting perspectives.

2 Rollback Recovery Strategies

Rollback recovery addresses permanent (fail-stop) process failures, in the sense that a process reached a state where either it cannot continue for physical reasons or it detected that the current state has been corrupted and further continuation of the current computation is worthless. In order to mitigate the cost of such failures, processes save their state on persistent memory (remote node, disk, ...) by taking periodic *checkpoints*. When the state of a process is altered by a failure, a replacement resource is allocated, and a previous state of the process can be restored from a checkpoint. Clearly, to minimize the amount of lost computation, the checkpoint must be as close as possible to the failure point.

In a distributed system, many inter-dependent processes participate in the computation, and a consistent global state of the application not only captures the state of each individual process, but also captures the dependencies between these states carried by messages [8]. In the context of checkpoint-based rollback recovery, the state of processes after some have reloaded from a checkpoint forms the recovery line. Any message altering the deterministic behavior of the application, and crossing a recovery line, must be available for restarted processes, in order to maintain a consistent global state. Protocols taking checkpoints without a guarantee of consistency are subject to the domino effect, and may lead to a restart of the execution from the beginning. Such protocols are excluded from this study.

Ensuring a consistent recovery can be achieved by two approaches, whose key difference is the level of coordination imposed on the recovery procedure. On one extreme, *coordinating checkpoints*, where after a failure, the entire application rolls back to a known consistent global state. On the opposite extreme, only the failed process rolls back to a checkpoint, in which case the recovery line must be augmented with additional information in order to avoid inconsistent dependencies during the re-execution phase, a method known as *message logging* [9]. Recent advances in message logging [10, 11, 12] have led to composite algorithms, called *hierarchical checkpointing*, capable of partial coordination of checkpoints while retaining message logging capabilities to remove the need for a global restart.

The goal of the model presented in this paper, is to capture the largest possible spectrum of checkpointing techniques, in order to propose a holistic description of all viable approaches. Based on this unified model, an assessment between the different classical techniques can be made, allowing for a deeper understanding of the advantages and drawbacks of each particular approach.

2.1 Coordinated Checkpointing

Several algorithms have been proposed to coordinate checkpoints, the most commonly used being the Chandy-Lamport algorithm [8]. The main requirement to form a globally consistent recovery line, characterized by the absence of in-transit messages at the checkpoint line, is often implemented by completely silencing the network during checkpoint phases [13]. Coordinated algorithms possess the advantage of having almost no overhead outside of checkpointing periods, but require that every process, even if unaffected by failures, rolls back to its last checkpoint, as only this recovery line is guaranteed to be consistent. Arguably, the checkpoint and the recovery are costly operations, increasing the strain on the I/O subsystem.

2.2 Message Logging

Message Logging is a family of algorithms that provide a consistent recovery strategy from checkpoints taken at independent dates and without rolling back the surviving processes, a feature expected to increase resilience to adverse failure patterns [6]. As the recovery line includes the arbitrary state of processes that do not rollback to a checkpoint, the restarted processes must undergo a directed replay, in order to reach a state that is part of a consistent global state. Such a replay requires two supplementary bits of information, not captured by the traditional process checkpoint: 1) the *Event* log, which contains the outcome of all nondeterministic events that happened during the lost computation; and 2) the *Payload* log, which enables replaying reception of messages sent to restarted processes before the recovery line.

Payload copy can be highly optimized; in particular, the sender-based approach [9] permits the payload copy operation to be completely overlapped with the send operation itself. However, this extra data becomes part of the process checkpoint, and hence the amount of data to be stored on stable storage directly depends on the communication intensity of the application. On the other side, the fact that the recovery is a directed replay, rather than a complete coordinated restart, conveys two potential benefits. During the replay, message receptions are immediately available from the log and emissions are mostly discarded, hence the communication overhead is greatly reduced. As a result the replay of processes restarted from checkpoint may progress faster than the original execution [6]. Second, as the failure recovery is constrained to processes directly impacted, other processes may continue to progress concurrently with the recovery, until they need to interact with one of the replaying processes. However, it can be argued that for tightly coupled applications, very little computation is completed before depending (transitively) on a replaying process.

2.3 Hierarchical Checkpointing

Hierarchical checkpointing protocols are a recent refinement of fault tolerant protocols, gathering advantages from both coordinated checkpointing and message logging while minimizing the drawbacks. They are designed to avoid the global restart associated with coordinated checkpointing while drastically limiting the volume of message to log compared to message logging protocols [10, 12, 11]. These hierarchical schemes partition the processes of the application in groups, based on heuristics such as network proximity or communication intensity. Each group checkpoints independently, but processes belonging to the same group coordinate their checkpoints (and recovery), in order to spare some of the payload log. Communications between groups continue to incur payload logging. However, because processes belonging to a same group follow a coordinated checkpointing protocol, the payload of messages exchanged between processes of the same group is not needed during replay.

The optimizations driving the choice of the size and shape of groups are varied. A simple heuristic is to checkpoint as many processes as possible, simultaneously, without exceeding the capacity of the I/O system. In this case, groups do not checkpoint in parallel. Groups can also be formed according to hardware proximity and communication patterns. In such approaches, there may be opportunity for several groups to checkpoint concurrently. Without loss of generality, we consider that all groups of the application enter their checkpoint phase in turn, thus making the whole execution appear as a parallel work phase, followed by a checkpointing phase made of a sequence of checkpoints. The model we propose captures the intricacies of these grouping strategies, and we instantiate a variety of meaningful scenarios in Section 4.

2.4 Related work

The study of the optimal period of checkpoint for sequential jobs (or parallel jobs checkpointed in a coordinated way) has seen many studies presenting different order of estimates: see [14, 15], and [16, 17] that consider weibull distributions, or [18] that considers parallel jobs. The selection of the optimal checkpointing interval is critical to extract the best performance of any rollback-recovery protocol. However, although we use the same approach to find the optimal checkpoint interval, we focus our study on the comparison of different protocols that were not captured by the models these works considered.

The literature proposes different works as [19, 20, 21, 22, 23] on the modeling of coordinated checkpointing protocols. [24] focus on refining failures prediction; [20], and [19] focus on the optimized uses of the available resources: some may be kept in backup in order to replace the down ones and others may be even shutdown in order to decrease the failure risk or to prevent storage consumption by saving less checkpoints snapshots. [23] proposes a scalability model where they compare the impact of failures on application performance with and without coordinated checkpointing. The major difference with these works lays in the unified model for coordinated and hierarchical protocols, and the inclusion of more parameters (like recovery of the checkpoint transfer cost with overlapping computation), refinning the model.

More scare papers present the modeling of uncoordinated or hierarchical checkpointing. [25] models a *periodic* checkpointing on *fault-aware* parallel tasks that do not communicate. From our point of view, this specificity does not match the uncoordinated checkpointing with message logging we consider. In this paper, the three families of checkpointing protocols are targeted : the coordinated, the uncoordinated and the hierarchical ones. To the best of our knowledge, it is the first attempt at providing a unified model for this large spectrum of protocols.

3 Model and Analytical Assessment

In this section, we formally state the unified model, together with the closed-form formula for the waste optimization problem. We start with the description of the abstract model (Section 3.1). Processors are partitioned into G groups, where each group checkpoints independently and periodically. To help follow the technical derivation of the waste, we start with one group (Section 3.2) before tackling the general problem with $G \geq 1$ groups (Section 3.3). We even deal with a simplified model with $G \geq 1$ before tackling the fully general model, which requires three additional parameters (pay-load overhead, faster execution replay after a failure, and increase in checkpoint size due to message logging). We end up with a complicated formula that characterizes the waste of resources due to checkpointing. This formula can be instantiated to account for all the checkpoint protocols described in Section 2, see Section 4 for examples. Note that in all scenarios, we model the behavior of tightly coupled applications, meaning that no computation can progress on the entire platform as long as the recovery phase of a group with a failing processor is not completed.

3.1 Abstract model

In this section, we detail the main parameters of the model. We consider an application that executes on p_{total} processors subject to failures.

Units– To avoid introducing several conversion parameters, we instantiate all the parameters of the model in seconds. The failure inter-arrival times, the durations of a downtime, checkpoint, or recovery are all expressed in seconds. Furthermore, we assume (without loss of generality) that one work unit is executed in one second, when all processors are computing at full rate. One work-unit may correspond to any relevant application-specific quantity. When a processor is slowed-down by another activity related to fault-tolerance (writing checkpoints to stable storage, logging messages, etc.), one work-unit takes longer than a second to complete.

Failures and MTBF– The platform consists of p_{total} identical processors. We use the term "processor" to indicate any individually scheduled compute resource (a core, a socket, a cluster node, etc) so that our work is agnostic to the granularity of the platform. These processors are subject to failures. Exponential failures are widely used for theoretical studies, while Weibull or log-normal failures are representative of the behavior of real-world platforms [26, 27, 28, 29]. The mean time between failures of a given processor is a random variable with mean (*MTBF*) μ (expressed in seconds). Given the MTBF of one processor, it is difficult to compute, or even approximate, the failure distribution of a platform with p_{total} processors, because it is the *superposition* of p_{total} independent and identically distributed distributions (with a single processor). However, there is an easy formula for the MTBF of that distribution, namely $\mu_p = \frac{\mu}{p_{total}}$.

In our theoretical analysis, we do not assume to know the failure distribution of the platform, except for its mean value (the MTBF). Nevertheless, consider any time-interval $\mathcal{I} = [t, t+T]$ of length T and assume that a failure strikes during this interval. We can safely state that the probability for the failure to strike during any sub-interval $[t', t'+X] \subset \mathcal{I}$ of length X is $\frac{X}{T}$. Similarly, we state that the expectation of the time m at which the failure strikes is $m = t + \frac{T}{2}$. Neither of these statements rely on some specific property of the failure distribution, but instead are a direct consequence of averaging over all possible interval starting points, that will correspond to the beginning of checkpointing periods, and that are independent of failure dates.

Tightly-coupled application– We consider a tightly-coupled application executing on the p_{total} processors. Inter-processor messages are exchanged throughout the computation, which can only progress if all processors are available. When a failure strikes some processor, the application is missing one resource for a certain period of time, the *downtime*. Then, the application recovers from the last checkpoint (*recovery* time) before it re-executes the work done since that checkpoint and up to the failure. Under a hierarchical scenario, the useful work resumes only when the faulty group catches up with the overall state of the application at failure time. Many scientific applications obey to the previous scheme. Typically, the tightly-coupled application will be an iterative application with a global synchronization point at the end of each iteration. However, the fact that inter-processor information is exchanged continuously or at given synchronization steps (as in BSP-like models [30]) is irrelevant: in steady-state mode, all processors must be available concurrently for the execution to actually progress. While the tightly-coupled assumption may seem very constraining, it captures the fact that processes in the application depend on each other, and progress can be guaranteed only if all processes are present to participate to the computation.

Blocking or non-blocking checkpoint– There are various scenarios to model the cost of checkpointing, so we use a very flexible model, with several parameters to instantiate. The first question is whether checkpoints are blocking or not. In some architectures, we may have to stop executing the application before writing to the stable storage where checkpoint data is saved; in that case checkpoint is fully blocking. In other architectures, checkpoint data can be saved on the fly into a local memory before the checkpoint is sent to the resilient disk,while computation can resume progress; in that case, checkpoints can be fully overlapped with computations. To deal with all situations, we introduce a slow-down factor α: during a checkpoint of duration C, the work that is performed is αC work units, instead of C work-units if only computation takes place. In other words, $(1-\alpha)C$ work-units are wasted due to checkpoint jitter perturbing the progress of computation. Here, $0 \leq \alpha \leq 1$ is an arbitrary parameter. The case $\alpha = 0$ corresponds to a fully blocking checkpoint, while $\alpha = 1$ corresponds to a fully overlapped checkpoint, and all intermediate situations can be represented.

Periodic checkpointing strategies– For the sake of clarity and tractability, we focus on periodic scheduling strategies where checkpoints are taken at regular intervals, after some fixed amount of work-units have been performed. This corresponds to an infinite-length execution partitioned into periods of duration T. We partition T into $T = W + C$, where W is the amount of time where only computations take place, while C corresponds to the amount of time where checkpoints are taken. The total amount of work units that are executed during a period of length T is thus $\textsc{Work} = W + \alpha C$ (recall that there is a slow-down due to the overlap). In a failure-free environment, the *waste* of computing resources due to checkpointing is (see Figure 1):

$$\textsc{Waste} = \frac{T - \textsc{Work}}{T} = \frac{(1-\alpha)C}{T} \tag{1}$$

As expected, if $\alpha = 1$ there is no overhead, but if $\alpha < 1$ (actual slowdown, or even blocking if $\alpha = 0$), checkpointing comes with a price in terms of performance degradation.

For the time being, we do not further quantify the length of a checkpoint, which is a function of several parameters. Instead, we proceed with the abstract model. We envision several scenarios in Section 4, only after setting up the formula for the waste in a general context.

Processor groups– As described above, we assume that the platform is partitioned into G groups of same size. Each group contains q processors (hence $p_{total} = Gq$). For the sake of the presentation, we first compute the waste when $G = 1$ before discussing the case where $G \geq 1$. When $G = 1$, we speak of a *coordinated* scenario, and we simply write C, D and R for the duration of a checkpoint, downtime and recovery. When $G \geq 1$, we speak of a *hierarchical* scenario. Each group includes q processors and checkpoints independently and sequentially in time $C(q)$. Similarly, we use $D(q)$ and $R(q)$ for the durations of the downtime and recovery. Of course, if we let $G = 1$ in the (more general) *hierarchical* scenario, we retrieve the value of the waste for the coordinated scenario. As already mentioned, we derive a general expression for the waste for both scenarios, before further specifying the values of $C(q)$, $D(q)$, and $R(q)$ as a function of q and the various architectural parameters under study.

3.2 Waste for the coordinated scenario ($G = 1$)

The goal of this section is to compute a formula for the expected waste in the coordinated scenario where $G = 1$. Recall that the waste is the fraction of time that the processors do not compute at full rate, either because they are checkpointing, or because they recover from a failure. Recall too that we write C, D, and R for the checkpoint, downtime, and recovery using a single group of p_{total} processors.

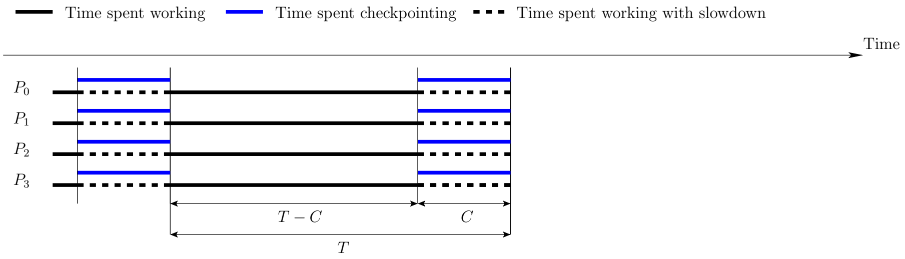


Figure 1: Illustrating the waste due to checkpointing in a failure-free environment: there is a slowdown of duration C at the end of every period of length $T = W + C$. The total work that is executed during such a period is $\textsc{Work} = W + \alpha C$ work units.

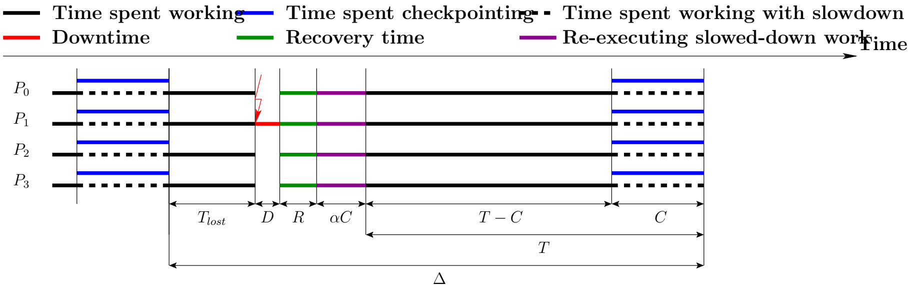


Figure 2: Coordinated checkpoint: illustrating the waste when a failure occurs during the work phase.

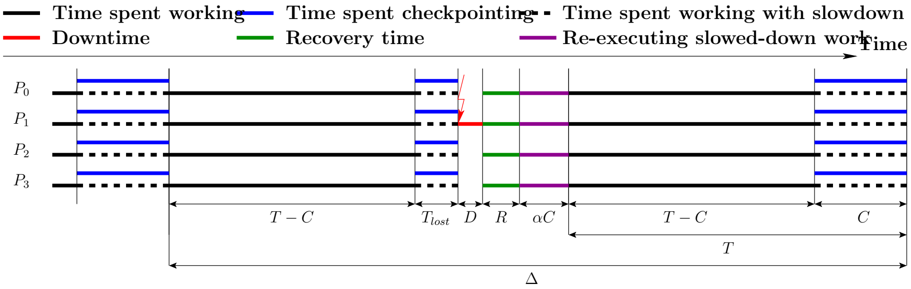


Figure 3: Coordinated checkpoint: illustrating the waste when a failure occurs during the checkpoint phase.

We obtain the following equation for the waste, which we explain below, and illustrate with Figures 2 and 3:

$$\begin{aligned}\textsc{Waste}_{coord} &= \frac{(1-\alpha)C}{T} &(2)\\ &+ \frac{1}{\mu_p}\frac{T-C}{T}\left[R+D+\alpha C+\frac{T-C}{2}\right] &(3)\\ &+ \frac{1}{\mu_p}\frac{C}{T}\left[R+D+\alpha C+T-C+\frac{C}{2}\right] &(4)\end{aligned}$$

- (2) is the portion of the execution lost in checkpointing, even during a fault-free execution, see Equation (1).
- (3) is the overhead of the execution time due to a failure during a work interval (see Figure 2). $D+R$ is the duration of the downtime and recovery. We add the time needed to re-execute the work that had already completed during the period, and that has been lost due to the failure. First, we need to re-execute the work done in parallel with the last checkpoint (of duration C). This takes a time αC since no checkpoint activity is taking place during that replay. Then we re-execute the work done in the work-only area, which had a duration of T_{lost}. On average, the failure happens in the middle of the interval of length $T-C$, hence the term T_{lost} has expected value $T_{lost} = \frac{T-C}{2}$. The previous quantity is weighted by its probability: when a fault occurs, it occurs during the work period with probability $(T-C)/T$. Finally, this execution overhead occurs each time a fault occurs, that is, on average, once every μ_p.
- (4) is the overhead due to a failure during a checkpoint (see Figure 3). The reasoning is similar. Just as before, we re-execute the work done in parallel with the previous checkpoint (of duration αC), then the work done in the work-only area (of duration $T-C$), and finally the work done during the checkpoint (of duration T_{lost} as it is re-executed while checkpoints are taken). In expectation, $T_{lost} = C/2$. We weight with the probability of the event, now equal to C/T.

After simplification of Equations (2) to (4), we get:

$$\textsc{Waste}_{\text{coord}} = \frac{(1-\alpha)C}{T} + \frac{1}{\mu_p}\left(D+R+\frac{T}{2}+\alpha C\right) \tag{5}$$

We point out that Equation (5) is valid only when $T \ll \mu_p$: indeed, we made a first-order approximation when implicitly assuming that we do not have more than one failure during the same period. In fact, the number of failures during a period of length T can be modeled as a Poisson process of parameter $\frac{T}{\mu_p}$; the probability of having $k \geq 0$ failures is $\frac{1}{k!}(\frac{T}{\mu_p})^k e^{-\frac{T}{\mu_p}}$. Hence the probability of having two or more failures is $\pi = 1-(1+\frac{T}{\mu_p})e^{-\frac{T}{\mu_p}}$. Enforcing the constraint $T \leq 0.1\mu_p$ leads to $\pi \leq 0.005$, hence a valid approximation when bounding the period range accordingly.

In addition to the previous constraint, we must enforce the condition $C \leq T$, by construction of the periodic checkpointing policy. Without the constraint

$C \leq T \leq 0.1\mu_p$, the optimal checkpointing period $\mathbb{T}^*$ that minimizes the expected waste in Equation (5) is $\mathbb{T}^* = \sqrt{2\mu_p C(1-\alpha)}$. However, this expression for $\mathbb{T}^*$ (which is known as Young's approximation [14] when $\alpha = 0$) may well be out of the admissible range. Finally, note that the optimal waste may never exceed 1, since it represents the fraction of time that is "wasted". In this latter case, the application no longer makes progress.

3.3 Waste for the hierarchical scenario ($G \geq 1$)

In this section, we compute the expected waste for the hierarchical scenario. We have G groups of q processors, and we let $C(q)$, $D(q)$, and $R(q)$ be the duration of the checkpoint, downtime, and recovery for each group. We assume that the checkpoints of the G groups take place in sequence within a period (see Figure 4). We start by generalizing the formula obtained for the coordinated scenario before introducing several new parameters to the model.

3.3.1 Generalizing previous scenario with $G \geq 1$

We obtain the following intricate formula for the waste, which we explain term by term below, and illustrate with Figures 4 to 7:

$$\textsc{Waste}_{\text{hierarch}} = \frac{T - \textsc{Work}}{T} + \frac{1}{\mu_p}\left(D(q) + R(q) + \textsc{Re-Exec}\right) \tag{6}$$

$$\textsc{Work} = T - (1-\alpha)GC(q) \tag{7}$$

$$\textsc{Re-Exec} =$$

$$\frac{T - GC(q)}{T}\frac{1}{G}\sum_{g=1}^{G}\left[(G-g+1)\alpha C(q) + \frac{T-GC(q)}{2}\right] \tag{8}$$

$$+ \frac{GC(q)}{T}\frac{1}{G^2}\sum_{g=1}^{G}\Bigg[\tag{9}$$

$$\sum_{s=0}^{g-2}(G-g+s+2)\alpha C(q) + T - GC(q) \tag{10}$$

$$+ G\alpha C(q) + T - GC(q) + \frac{C(q)}{2} \tag{11}$$

$$+ \sum_{s=1}^{G-g}(s+1)\alpha C(q)\Bigg] \tag{12}$$

- The first term in Equation (6) represents the overhead due to checkpointing during a fault-free execution (same reasoning as in Equation (1)), and the second term the overhead incurred in case of failure. Failures strike every μ_p seconds on average; each of them induces a downtime of duration $D(q)$, a recovery of duration $R(q)$, and the re-execution of some amount of work, which is captured in the term Re-Exec.

- (7) provides the amount of work units executed within a period of length T, namely $T - GC(q)$ at full speed, and $\alpha GC(q)$ during the G checkpoints.

Note that when given a group checkpoints, all groups endure the same slowdown, because the application is tightly-coupled and can only progress at the pace of the slowest resource.

- (8) represents the time needed for re-executing the work when the failure happens in a work-only area, i.e., during the first $T - GC(q)$ seconds of the period. Thanks to message-logging, only the failing group must rollback and re-execute some work. The re-execution time depends upon the group which is hit by the failure, and we average over all groups: see Figure 4. If the failure hits group g, where $1 \leq g \leq G$, then we need to re-execute the work done in parallel with the last checkpoint of all groups following g, including itself. There are $G - g + 1$ such groups, hence the term $(G - g + 1)\alpha C(q)$. Just as for the coordinated approach, note that the work that has been executed in time $(G - g + 1)C(q)$ is now re-executed faster, since no checkpoint activity is taking place during that replay. Then, on average, the failure happens in the middle of the interval of length $T - GC(q)$, hence the term T_{lost} has expected value $T_{lost} = \frac{T-GC(q)}{2}$. The total time needed for re-execution is then weighted by the probability for the failure to happen during the work-only area, namely $\frac{T-GC(q)}{T}$.
- (9) deals with the case where the fault happens during a checkpoint, i.e. during the last $GC(q)$ seconds of the period (hence the first term that represents the probability of this event). There are G^2 cases to average upon. Indeed, assume that the failure hits group g, where $1 \leq g \leq G$. The failure took place while one of the G groups was checkpointing. We distinguish three cases, depending upon whether the latter group is a group preceding g, group g itself, or a group following g.
  - (10) is for the case when the fault happens before the checkpoint of group g. More precisely, see Figure 5: there are s groups that have completed their checkpoints, and the failure took place during the checkpoint of group $s + 1$, where $0 \leq s \leq g - 2$. The amount of wasted time is $\Delta - T$, with the notations of the figure, and is explained as follows: (i) the work done in parallel with the last checkpoint of all groups following g, including g itself, which amounts to $(G - g + 1)\alpha C(q)$, just as before; (ii) the work done during the first $T - GC(q)$ seconds of the period; and (iii) the work done while the $s + 1$ first groups were checkpointing, re-executed faster, namely in time $(s + 1)\alpha C(q)$. Note that we do not need to take the expected value of the term T_{lost} (which is $\frac{C(q)}{2}$), because terms cancel in the equation, as shown in the figure. Note also that we assume that the downtime of the struck group starts after the end of the current checkpoint (by group $s + 1$), while some (short) overlap might be possible in theory.
  - (11) is for the case when the fault happens during the checkpoint of group g, see Figure 6. This case is quite similar to the previous case, and we retrieve the quantity $(G - g + 1)\alpha C(q) + (T - GC(q)) + (s + 1)\alpha C(q)$, with $s = g - 1$. As outlined in the figure, in contrast with the previous case, the faulty group is the one that is currently checkpointing: instead of re-executing the corresponding work at a faster

rate, we have lost half of the checkpoint duration: the expectation of T_{lost} is $\frac{C(q)}{2}$.

- (12) is the case when the fault happens after the checkpoint of group g, during the checkpoint of group $g+s$, where $g+1 \leq g+s \leq G$. See Figure 7: in this last case, the incurred overhead is much smaller. Because the failure took place after the checkpoint of group g, we only re-execute, at a faster rate, the work that was done during the checkpoints of groups g to $g+s$, which amounts to a time $((g+s) - g + 1)\alpha C(q)$. Note that we made the same simplification as for case (10), assuming no overlap between the downtime and the end of the current checkpoint.

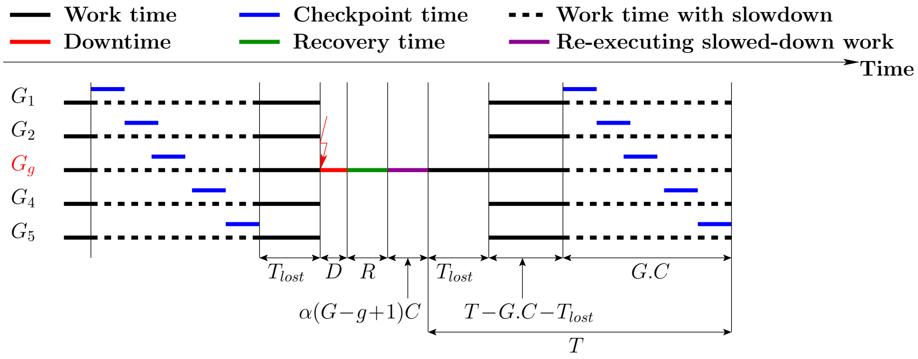


Figure 4: Hierarchical checkpoint: illustrating the waste when a failure occurs during the work phase.

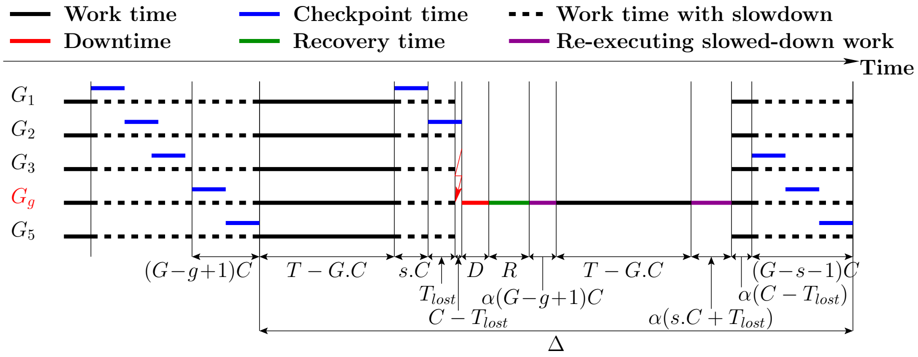


Figure 5: Hierarchical checkpoint: illustrating the waste when a failure occurs during the checkpoint phase, and before the checkpoint of the failing group.

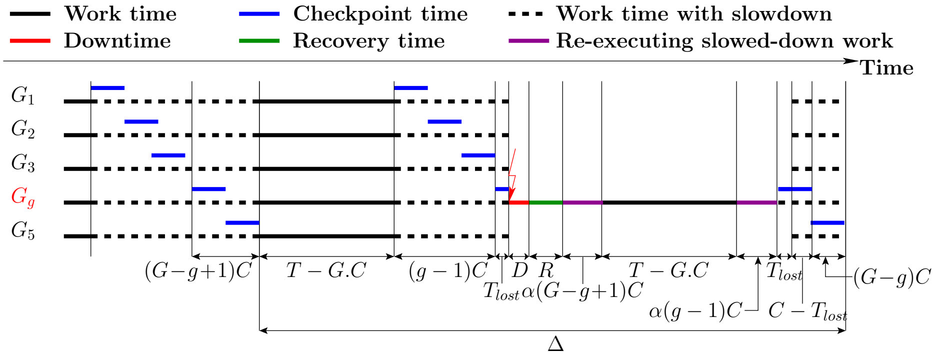


Figure 6: Hierarchical checkpoint: illustrating the waste when a failure occurs during the checkpoint phase, and during the checkpoint of the failing group.

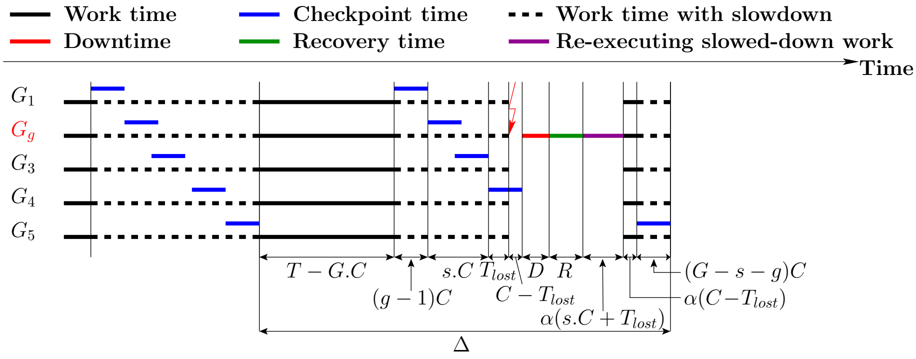


Figure 7: Hierarchical checkpoint: illustrating the waste when a failure occurs during the checkpoint phase, and after the checkpoint of the failing group.

After simplification (using a computer algebra software), we obtain:

$$\textsc{Waste}_{\text{hierarch}} = \frac{1}{2\mu_p T} \times \left(\begin{array}{l} T^2 \\ +GC(q)\big[(1-\alpha)(2\mu_p - T) + (2\alpha - 1)C(q)\big] \\ +T\big[2(D(q)+R(q)) + (\alpha+1)C(q)\big] \\ +(1-2\alpha)C(q)^2 \end{array} \right) \tag{13}$$

Of course this expression reduces to Equation (5) when $G = 1$. Just as for the coordinated scenario, we enforce the constraint

$$GC(q) \leq T \leq 0.1\mu_p \tag{14}$$

The first condition is by construction of the periodic checkpointing policy, and the second is to enforce the validity of the first-order approximation, assuming at most one failure per period.

3.3.2 Refining the model

We introduce three new parameters to refine the model when the processors have been partitioned into several groups. These parameters are related to the impact of message logging on execution, re-execution, and checkpoint image size, respectively.

Impact of message logging on execution and re-execution– With several groups, inter-group messages need to be stored in local memory as the execution progresses, and event logs must be stored in a reliable storage, so that the recovery of a given group, after a failure, can be done independently of the other groups. This induces an overhead, which we express as a slowdown of the execution rate: instead of executing one work-unit per second, the application executes only λ work-units, where $0 < \lambda < 1$. Typical values for λ are said to be $\lambda \approx 0.98$, meaning that the overhead due to payload messages is only a small percentage [31, 11].

On the contrary, message logging has a positive effect on re-execution after a failure, because inter-group messages are stored in memory and directly accessible after the recovery (as explained in Section 2.2). Our model accounts for this by introducing a speed-up factor ρ during the re-execution. Typical values for ρ lie in the interval [1..2], meaning that re-execution time can be reduced up to by half for some applications [6].

Fortunately, the introduction of λ and ρ is not difficult to account for in the expression of the expected waste: in Equation (6), we replace $\textsc{Work}$ by $\lambda\textsc{Work}$ and $\textsc{Re-Exec}$ by $\frac{\textsc{Re-Exec}}{\rho}$ and obtain

$$\textsc{Waste}_{\text{hierarch}} = \frac{T - \lambda\textsc{Work}}{T} + \frac{1}{\mu_p}\left(D(q) + R(q) + \frac{\textsc{Re-Exec}}{\rho}\right) \tag{15}$$

where the values of $\textsc{Work}$ and $\textsc{Re-Exec}$ are unchanged, and given by Equations (7) and (8 – 12) respectively.

Impact of message logging on checkpoint size– Message logging has an impact on the execution and re-execution rates, but also on the size of the checkpoint. Because inter-group messages are logged continuously, the size of the checkpoint increases with the amount of work that is executed before a checkpoint. Consider the hierarchical scenario with G of q processors. Without message logging, the checkpoint time of each group is $C_0(q)$, and to account for the increase in checkpoint size due to message logging, we write the equation

$$C(q) = C_0(q)(1 + \beta\lambda\textsc{Work}) \Leftrightarrow \beta = \frac{C(q) - C_0(q)}{C_0(q)\lambda\textsc{Work}} \tag{16}$$

As before, $\lambda\textsc{Work} = \lambda(T - (1-\alpha)GC(q))$ (see Equation (7)) is the number of work units, or application iterations, completed during the period of duration T, and the parameter β quantifies the increase in the checkpoint image size per work unit, as a proportion of the application footprint. Typical values of β are given in the examples of Section 4. Combining with Equation (16), we derive the value of $C(q)$ as

$$C(q) = \frac{C_0(q)(1 + \beta\lambda T)}{1 + GC_0(q)\beta\lambda(1-\alpha)} \tag{17}$$

The first constraint in Equation (14), namely $GC(q) \leq T$, now translates into

$$\frac{GC_0(q)(1 + \beta\lambda T)}{1 + GC_0(q)\beta\lambda(1-\alpha)} \leq T$$

which leads to

$$GC_0(q)\beta\lambda\alpha \leq 1 \text{ and } T \geq \frac{GC_0(q)}{1 - GC_0(q)\beta\lambda\alpha} \tag{18}$$

4 Case Studies

In this section, we instantiate the previous model to evaluate different case studies. We propose three generic scenarios for the checkpoint protocols, three application examples that provide different values for the parameter β, and four platform instances.

4.1 Checkpointing algorithm scenarios

Coord-IO– The first scenario considers a coordinated approach, where the duration of a checkpoint is the time needed for the p_{total} processors to write the memory footprint of the application onto stable storage. Let Mem denote this memory, and b_{io} represents the available I/O bandwidth. Then we have $C = C_{\text{Mem}}$, where $C_{\text{Mem}} = \frac{\text{Mem}}{\mathrm{b}_{io}}$.

In most cases we have equal write and read speed access to stable storage, and we let $R = C = C_{\text{Mem}}$, but in some cases we have different values, for example with the K-Computer (see Table 1). As for the downtime, the value D is the expectation of the duration of the downtime. With a single processor, the downtime has a constant value, but with several processors, the duration of the downtime is very difficult to compute: a processor can fail while another one is down, thereby leading to cascading downtimes. The exact value of the downtime with several processors is unknown, even for failures distributed according to an exponential law, but an upper bound can be provided (see [32] for details). In most practical cases, the lower bound of the downtime of a single processor is expected to be very accurate, and we use a constant value for D in our case studies.

HIERARCH-IO– The second scenario uses a number of relatively large groups. Typically, these groups are composed so as to take advantage of the application communication pattern [11, 12]. For instance, if the application executes on a 2D-grid of processors, a natural way to create processor groups is to have one group per row (or column) of the grid. If all processors of a given row belong to the same group, horizontal communications are intra-group communications and need not to be logged. Only vertical communications are inter-group communications and, as such, need to be logged.

With large groups, there are enough processors within each group to saturate the available I/O bandwidth, and the G groups checkpoint sequentially. Hence the total checkpoint time without message logging, namely $GC_0(q)$, is equal to that of the coordinated approach. This leads to the simple equation

$$C_0(q) = \frac{C_{\text{Mem}}}{G} = \frac{\text{Mem}}{G\text{b}_{io}} \tag{19}$$

where Mem denotes the memory footprint of the application, and b_{io} the available I/O bandwidth. Similarly as before, we let $R(q)$ for the recovery (either equal to $C(q)$ or not), and use a constant value $D(q) = D$ for the downtime.

HIERARCH-PORT– The third scenario investigates the possibility of having a large number of very small groups, a strategy proposed to take advantage of hardware proximity and failure probability correlations [10]. However, if groups are reduced to a single processor, a single checkpointing group is not sufficient to saturate the available I/O bandwidth. In this strategy, multiple groups of q processors are allowed to checkpoint simultaneously in order to saturate the I/O bandwidth. We define q_{min} as the smallest value such that

$$\text{q}_{\text{min}}\text{b}_{port} \geq \text{b}_{io} \tag{20}$$

Here b_{port} is the network bandwidth of a single processor. In other words, q_{min} is the minimal size of groups so that Equation (19) holds.

Small groups typically imply logging more messages (hence a larger growth factor of the checkpoint per work unit β, and possibly a larger impact on computation speed λ). Coming back to an application executing on a 2D-grid of processors, twice as many communications will be logged (assuming a symmetrical communication pattern along each grid direction). However, let us ompare recovery times in the HIERARCH-PORT and HIERARCH-IO strategies; assume that $R_0(q) = C_0(q)$ for simplicity. In both cases Equation (19) holds, but the number of groups is significantly larger for HIERARCH-PORT, thereby ensuring a much shorter recovery time.

4.2 Application examples

We study the increase in checkpoint size due to message logging by detailing three application examples: 2D- and 3D-stencil computations, and linear algebra kernels such as matrix product. These examples are typical scientific applications that execute on 2D-or 3D-processor grids, but they exhibit a different increase rate parameter β, as shown below.

2D-STENCIL– We first consider a 2D-stencil computation: a real matrix of size $n \times n$ is partitioned across a $p \times p$ processor grid, where $p^2 = p_{total}$. At each iteration, each matrix element is averaged with its 8 closest neighbors, which requires rows and columns that lie at the boundary of the partition to be exchanged (it is easy to generalize to larger update masks). Each processor holds a matrix block of size $b = n/p$, and sends four messages of size b (one in each grid direction) at each iteration. Then each element is updated, at the cost of 9 double floating-point operations. The (parallel) work for one iteration is thus WORK $= \frac{9b^2}{\mathrm{s}_p}$, where s_p is the speed of one processor.

With the COORD-IO scenario, $C = C_{\mathrm{Mem}} = \frac{\mathrm{Mem}}{\mathrm{b}_{io}}$. Here Mem $= 8n^2$ (in bytes), since there is a single (double real) matrix to store. As already mentioned, a natural (application-aware) group partition is with one group per row (or column) of the grid, which leads to $G = q = p$. Such large groups correspond to the HIERARCH-IO scenario, with $C_0(q) = \frac{C_{\mathrm{Mem}}}{G}$. At each iteration, vertical (inter-group) communications are logged, but horizontal (intra-group) communications are not logged. The size of logged messages is thus $2pb = 2n$ for each group. If we checkpoint after each iteration, $C(q) - C_0(q) = \frac{2n}{\mathrm{b}_{io}}$, and we derive from Equation (16) that $\beta = \frac{2np\mathrm{s}_p}{n^2 9b^2} = \frac{2\mathrm{s}_p}{9b^3}$. We stress that the value of β is unchanged if groups checkpoint every k iterations, because both $C(q) - C_0(q)$ and WORK are multiplied by a factor k. Finally, if we use small groups of size $\mathrm{q}_{\min}$, we have the HIERARCH-PORT scenario. We still have $C_0(q) = \frac{C_{\mathrm{Mem}}}{G}$, but now the value of β has doubled since we log twice as many communications.

MATRIX-PRODUCT– Consider now a typical linear-algebra kernel involving several matrix products. For each matrix-product, there are three matrices involved, so Mem $= 24n^2$ (in bytes) and $C = C_{\mathrm{Mem}} = \frac{\mathrm{Mem}}{\mathrm{b}_{io}}$ for the COORD-IO scenario. Just as before, each matrix is partitioned along a 2D-grid of size $p \times p$, but now each processor holds three matrix blocks of size $b = n/p$. Consider Cannon's algorithm [33] which has p steps to compute a product. At each step, each processor shifts one block vertically and one block horizontally, and the work is WORK $= \frac{2b^3}{\mathrm{s}_p}$. In the HIERARCH-IO scenario with one group per grid row, only vertical messages ar logged, so that $C(q) - C_0(q) = \frac{b^2}{\mathrm{b}_{io}}$. We derive that $\beta = \frac{\mathrm{s}_p}{6b^3}$. Again, β is unchanged if groups checkpoint every k steps, or every matrix product ($k = p$). In the COORD-PORT scenario with groups of size $\mathrm{q}_{\min}$, the value of β is doubled. In both scenarios, we have $C_0(q) = \frac{C_{\mathrm{Mem}}}{G}$ (but many more groups in the latter).

3D-STENCIL– This application is similar to 2D-STENCIL, but exhibits larger values of β. We have a 3D matrix of size n partitioned across a 3D-grid of size p, where $8n^3 =$ Mem and $p^3 = p_{total}$. Each processor holds a cube of size $b = n/p$. At each iteration, each pixel is averaged with its 27 closest neighbors, so that WORK $= \frac{27b^3}{\mathrm{s}_p}$. Each processor sends the six faces of its cube, one in each direction. In addition to the COORD-IO scenario, there are now three hierarchical scenarios: A) HIERARCH-IO-PLANE where groups are horizontal planes, of size p^2. Only vertical communications are logged, which represents

Name	Number of cores	Number of processors p_{total}	Number of cores per processor	Memory per processor	I/O Network Bandwidth (b_{io})		I/O Bandwidth per processor (b_{port})	
					Write	Read	Write	Read
Titan	299,008	18,688	16	32GB	300GB/s	300GB/s	20GB/s	20GB/s
K-Computer	705,024	88,128	8	16GB	96GB/s	150GB/s	20GB/s	20GB/s
Exascale Slim	1,000,000,000	1,000,000	1,000	64GB	1TB/s	1TB/s	200GB/s	200GB/s
Exascale Fat	1,000,000,000	100,000	10,000	640GB	1TB/s	1TB/s	400GB/s	400GB/s

Table 1: Basic characteristics of platforms used to feed the model.

Name	Scenario	G (q_{min} if app.)	Checkpoint Saving Time	Checkpoint Loading Time	β for 2D-Stencil	β for Matrix-Product
Titan	Coord-IO	1	2,048s	2,048s	/	/
	Hierarch-IO	136	15s	15s	0.0001098	0.0004280
	Hierarch-Port	1,246 ($q_{min} = 15$)	1.6s	1.6s	0.0002196	0.0008561
K-Computer	Coord-IO	1	14,688s	9,400s	/	/
	Hierarch-IO	296	50s	32s	0.0002858	0.001113
	Hierarch-Port	17,626 ($q_{min} = 5$)	0.83s	0.53s	0.0005716	0.002227
Exascale-Slim	Coord-IO	1	64,000s	64,000	/	/
	Hierarch-IO	1,000	64s	64s	0.0002599	0.001013
	Hierarch-Port	200,000 ($q_{min} = 5$)	0.32s	0.32s	0.0005199	0.002026
Exascale-Fat	Coord-IO	1	64,000s	64,000	/	/
	Hierarch-IO	316	217s	217s	0.00008220	0.0003203
	Hierarch-Port	33,333 ($q_{min} = 3$)	1.92s	1.92s	0.00016440	0.0006407

Table 2: Parameters G, q_{min}, C, R, $C(q)$, $R(q)$ and β for all platform/scenario combinations with 2D-Stencil and Matrix-Product. The equation $C_0(q) = C/G$ always hold.

two faces per processor. We derive $\beta = \frac{2s_p}{27b^3}$; B) Hierarch-IO-Line where groups are lines, of size p. Twice as many communications are logged, which represents four faces per processor. We derive $\beta = \frac{4s_p}{27b^3}$; C) Hierarch-Port with groups of size q_{min}. All communications are logged, which represents six faces per processor. We derive $\beta = \frac{6s_p}{27b^3}$. Note that the order of magnitude of b is the cubic root of the memory per processor for 3D-Stencil, while it was its square root for 2D-Stencil and Matrix-Product, so β will be larger for 3D-Stencil than for the other two applications.

4.3 Platforms

We consider multiple platforms, existing or envisioned, that represent state-of-the-art targets for HPC applications. Table 1 presents the basic characteristics of the platforms we consider. The machine named Titan represents the fifth phase of the Jaguar supercomputer, as presented by the Oak Ridge Leadership Computing Facility[1]. The cumulated bandwidth of the I/O network is targeted to top out at 1 MB/s/core, resulting in 300GB/s for the whole system. We consider that all existing machines are limited for a single node output by the bus capacity, at approximately 20GB/s. The K-Computer machine, hosted by Riken in Japan, is the fastest supercomputer of the Top 500 list at the time of writing. Its I/O characteristics are those presented during the Lustre File System User's Group meeting, in April, 2011 [34], with the same bus limitation for a single node maximal bandwidth. The two exa-scale machines represent the two most likely scenarios envisioned by the International Exascale Software Project community [1], the largest variation being on the number of cores a single node should host. For all platforms, we let the speed of one core be 1

[1] `http://www.olcf.ornl.gov/computing-resources/titan/`

Gigaflops, and we derive the speed of one processor s_p by multiplying by the number of cores.

4.4 Parameters

Tables 2 and 3 summarize key parameters for all platform/scenario combinations. In all instances, we use the following default values: $\rho = 1.5$, $\lambda = 0.98$ and $\alpha = 0.3$. It turns out that these latter parameters have very little impact on the results. The impact of the different parameters are shown in figures 9, 10, 11, 12, 13, 14.

Name	Scenario	G	β for 3D-Stencil
Titan	Coord-IO	1	/
	Hierarch-IO-Plane	26	0.001476
	Hierarch-IO-Line	658	0.002952
	Hierarch-Port	1,246	0.004428
K-Computer	Coord-IO	1	/
	Hierarch-IO-Plane	45	0.003422
	Hierarch-IO-Line	1,980	0.006844
	Hierarch-Port	17,626	0.010266
Exascale-Slim	Coord-IO	1	/
	Hierarch-IO-Plane	100	0.003952
	Hierarch-IO-Line	10,000	0.007904
	Hierarch-Port	200,000	0.011856
Exascale-Fat	Coord-IO	1	/
	Hierarch-IO-Plane	47	0.001834
	Hierarch-IO-Line	2,154	0.003668
	Hierarch-Port	33,333	0.005502

Table 3: Parameters G and β for all platform/scenario combinations with 3D-Stencil. The equation $C_0(q) = C/G$ always hold (see Table 2 for values of C).

5 Results from the model

This section covers the results of our unified model on the previously described scenarios (one for coordinated checkpointing and two for hierarchical checkpointing) applied to four platforms, two that reflect existing top entries of Top500, and two on envisioned Exascale machines. In order to allow fellow researchers access to the model, results and scenarios proposed in this paper, we made our computation spreadsheet publicly available. `https://www.dropbox.com/s/m7b2q6vu4y1b2wc/Maple.tar.gz`.

We start with some words of caution. First, the applications used for this evaluation exhibit properties that makes them a difficult case for hierarchical checkpoint/restart techniques. These applications are communication intensive, which leads to a noticeable impact on performance (due to message logging). In addition, their communication patterns create logical barriers that make them tightly-coupled, giving a relative advantage to all coordinated checkpointing methods (due to the lack of independent progress). However, these applications are more representative of typical HPC applications than loosely-coupled (or even independent) jobs, and their communication-to-computation ratio tends to zero with the problem size (full weak scalability). Next, we point out that the theoretical values used in the model instances are overly optimistic, based on the values released by the constructors and not on measured values. Finally, we stress that the horizontal axis of all figures is the processor MTBF μ, which ranges from 1 year to 100 years, a choice consistent with the usual representation in the community. In the following discussion, we often refer to the platform MTBF μ_p, (see Section 3.1).

Due to the large time required to dump the total memory of the K-Computer or of the Exascale platforms, we assume that a checkpoint incurs a transfer of only a fraction of the total memory (1% for the K-Computer, 1% and 0.1% for Exascale). Otherwise the waste is almost equal to 1 for all scenarios with these machines. While checkpointing only a small fraction of the memory is possible only in some circumstances (employing incremental or application-based checkpointing), here it is mandatory to ensure any application progress – a first (negative) result of our model.

On platforms exhibiting characteristics similar to today's top entries in the Top500, namely Titan and K-Computer, we encounter a quite familiar environment (Figure 8(a)). Clearly, the key factors impacting the balance between coordinated and hierarchical protocols are the communication intensity of the applications (2D-Stencil, Matrix-Product, and 3D-Stencil), and the I/O capabilities of the system. On both platforms, the coordinated protocol has a slow startup, preventing the application from progressing when the platform MTBF μ_p is under a system limit directly proportional to the time required to save the coordinated checkpoint. Hierarchical protocols have a quicker startup. However, the cost of logging the messages and the extra checkpoint data is detrimental to the hierarchical protocols when there are fewer failures (even considering the most promising approaches). The vertical segments on the graphs correspond to cut-off values where we enforce the condition $T \leq 0.1\mu_p$ (see Equation (14)). Values of μ for which no waste is reported correspond to configurations for which no period can satisfy Equation (14).

The Exascale platforms exhibit a similar behavior. In order to evaluate the impact of the checkpoint size, here we provide results with a checkpointing size of 1% and 0.1% of the total memory (Figure 8(b) and 8(c)). As expected, by reducing the checkpoint size, the applications start progressing earlier for both protocols. In most cases hierarchical protocols seem more suitable for such a type of platforms. While they struggle when the communication intensity increases (the case of the 3D-Stencil), they provide limited waste for all the other cases. Overall, Exascale-Fat leads to a smaller waste (or better resource usage) than Exascale-Slim, because it has 10 times fewer processors of same MTBF.

6 Validation of the model

The simulation engine generates a random trace of errors (parameterized by an Exponential failure distribution or a Weibull one with a shape parameter of 0.7 or 0.5). On this error trace, we simulate the behavior of fault tolerance protocols. In the simulator, there is no assumption on when errors can happen: an error can strike a processor while another or the same processor is already subject to a failure and a recovery phase.

We arbitrarily set the duration of an application failure-free execution to 4 days on each process. We measure the actual execution time of each application on each platform and on each error trace using a time-out of one year: if an execution does not complete before the one year deadline, we consider it never completes and do not report any result for this particular platform/application setting. From the simulated execution times, we compute the actual average waste.

Figure 15 reports the waste for various application/platform scenarios for a Weibull failure distribution with $k = 0.7$. Results for an Exponential failure distribution and for a Weibull with $k = 0.5$, are provided in Figures 16 and 17 . Each point on the graphs is an average over 20 randomly generated instances.

Overall, Figures 15, 16 and 17 present similar trends and conclusions as 8. The main differences are seen for low MTBFs, in the vicinity of cut-offs values for Figure 8. There, either the first-order assumption could no longer be satisfied, or coordinated protocols were assessed not to allow for any application progress. Simulations show that, in these extreme settings, our analytical study was pessimistic: coordinated protocols have indeed very bad performance, but often applications still make progress (albeit at an unrealistic pace); coordinated protocols have an advantage over hierarchical protocols for slightly lower MTBFs than predicted. However, the simulations validate the relative performance of the different protocols.

For each scenario and each protocol, we plot (in solid line) the average waste for the checkpointing period computed with our model (the one minimizing Equation (15)). In Figures 15, 16 and 17 we also plot (in dotted line) the average waste obtained for the best checkpointing period (BestPer), numerically found by generating, and evaluating through simulations, a set of 480 periods representative of a very large neighborhood of the period computed with our model. The very good matching between solid and dotted lines show that our model enables to compute near-optimal checkpointing periods.

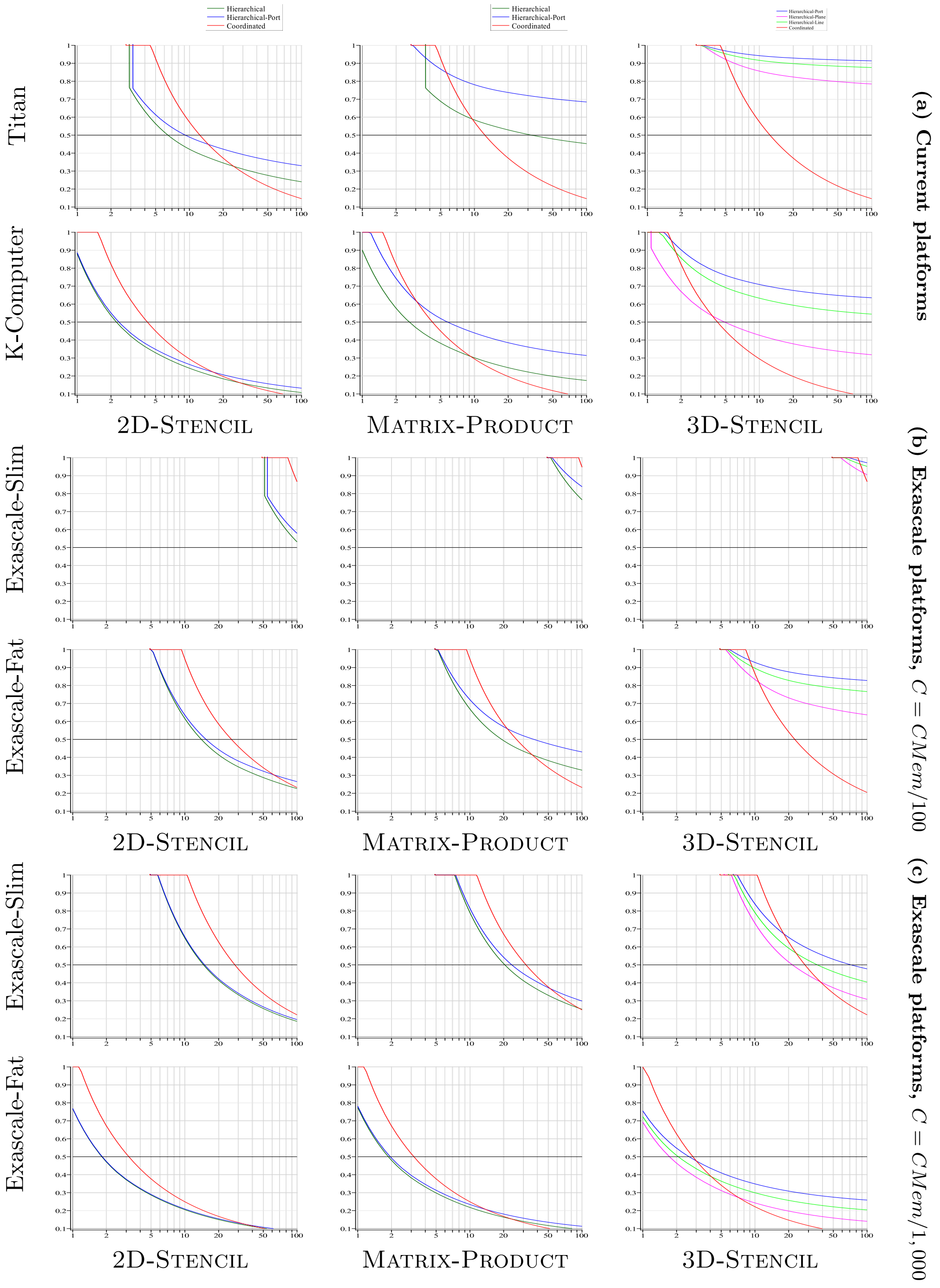
The comparison between Figures 15, 16 and 17 shows some differences. First, Figure 16 shows better results compared to 15 and 17. This is because, for the same MTBF value, the probability of failure for an exponential distribution is smaller than the one for a Weibull distribution with the two values of k. This is also the reason why the coordinated checkpointing protocol outperforms the hierarchical one for a smaller MTBF value in Figure 16 than in Figures 15 and 17. The same remarks can be done if we compare Figures 15 and 17 since the probability of failure is smaller when $k = 0.7$.

7 Conclusion

Despite the increasing importance of fault tolerance in achieving sustained, predictable performance, the lack of models and predictive tools has restricted the analysis of fault tolerant protocols to experimental comparisons only, which are painfully difficult to realize in a consistent and repeated manner. This paper introduces a comprehensive model of rollback recovery protocols that encompasses a wide range of checkpoint/restart protocols, including coordinated checkpoint and an assortment of uncoordinated checkpoint protocols (based on message logging). This model provides the first tool for a *quantitative* assessment of all these protocols. Instantiation on future platforms enables the investigation and understanding of the behavior of fault tolerant protocols at scales currently inaccessible. The results presented in Section 5, and corroborated by Section 6, highlight the following tendencies:

- Hardware properties will have tremendous impact on the efficiency of future platforms. Under the early assumptions of the projected Exascale systems, rollback recovery protocols are mostly ineffective. In particular, significant efforts are required in terms of I/O bandwidth to enable any type of rollback recovery to be competitive. With the appropriate provision in I/O (or the presence of distributed storage on nodes), rollback recovery can be competitive and significantly outperform full-scale replication [5] (which by definition cannot reach more than 50% efficiency).
- Under the assumption that I/O network provision is sufficient, the reliability of individual processors has a minor impact on rollback recovery efficiency. This suggests that most research efforts, funding and hardware provisions should be directed to I/O performance rather than improving component reliability in order to increase the scientific throughout of Exascale platforms.
- The model outlines some realistic ranges where hierarchical checkpointing outperforms coordinated checkpointing, thanks to its faster recovery from individual failures. This early result had already been outlined experimentally at smaller scales, but it was difficult to project at future scales. Our study provides a theoretical foundation and a quantitative evaluation of the drawbacks of checkpoint/restart protocols at Exascale; it can be used as a first building block to drive the research field forward, and to design platforms with specific resilience requirements.

Finally, as we are far from a comprehensive understanding of future Exascale applications and platform characteristics, we hope that the community will be interested in instantiating our publicly available model with other scenarios and case-studies.

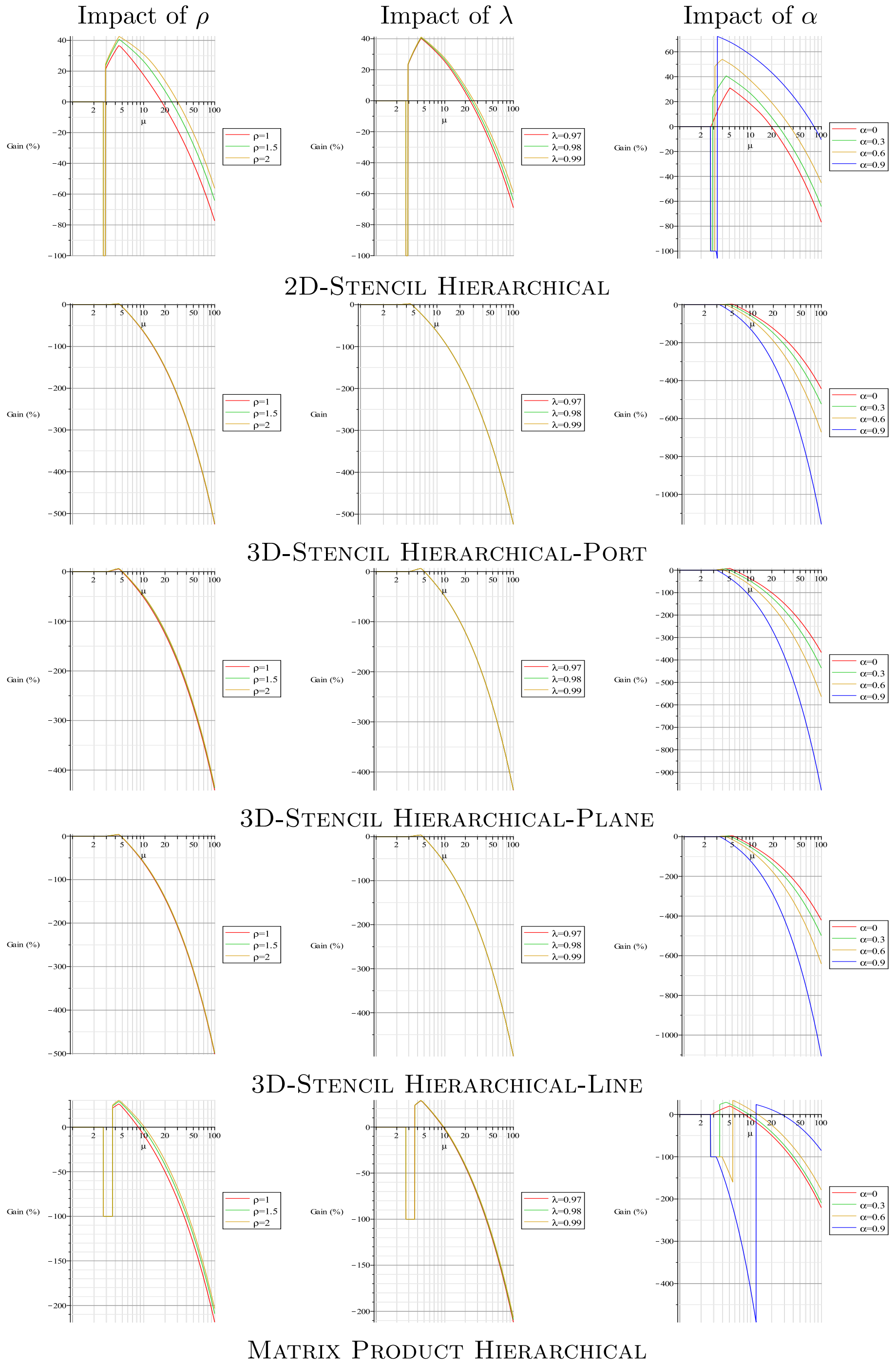


(a) Current platforms

(b) Exascale platforms, $C = CMem/100$

(c) Exascale platforms, $C = CMem/1,000$

Figure 8: Model: waste as a function MTBF μ (years per processor).



2D-Stencil Hierarchical

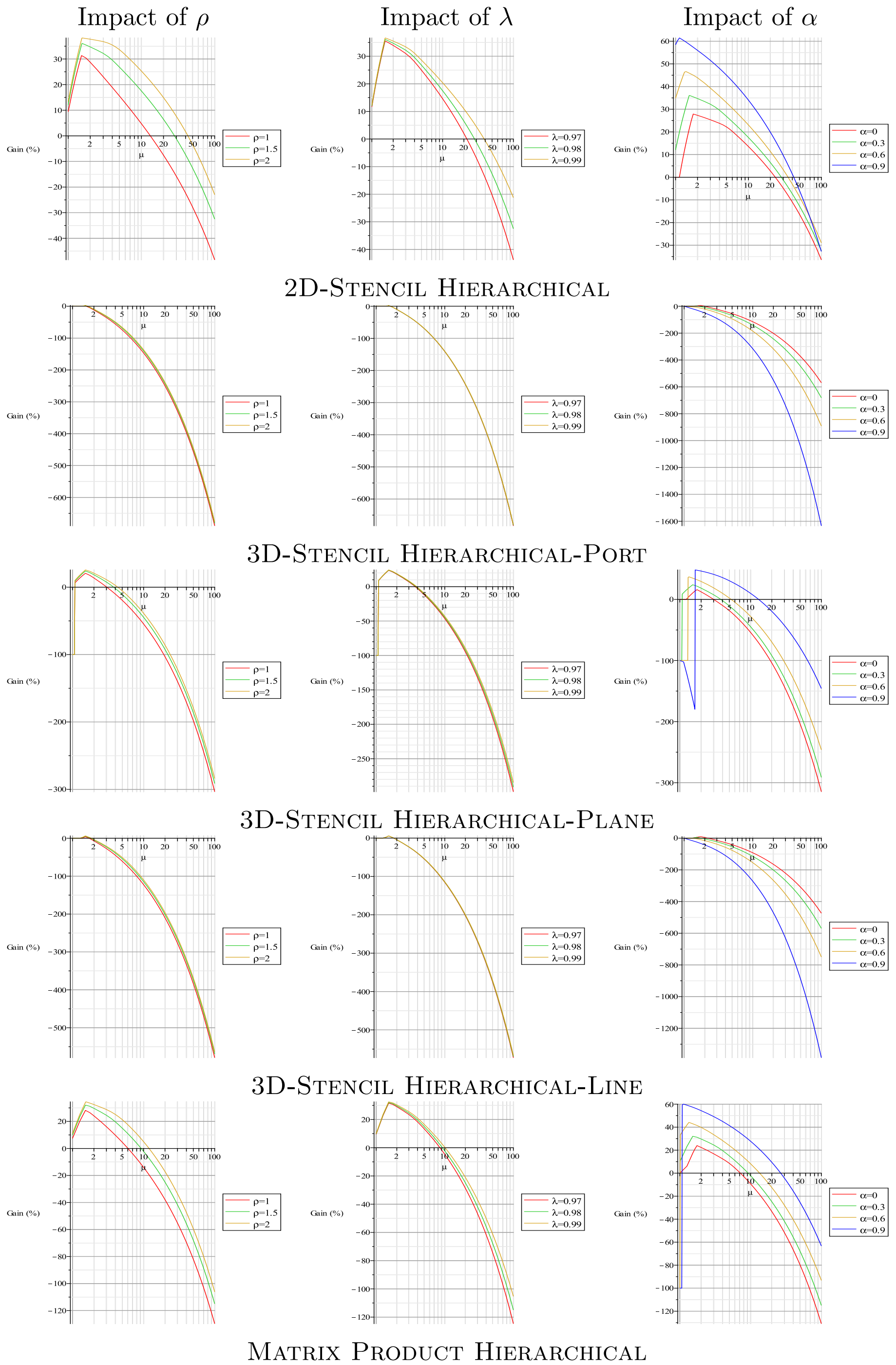
3D-Stencil Hierarchical-Port

3D-Stencil Hierarchical-Plane

3D-Stencil Hierarchical-Line

Matrix Product Hierarchical

Figure 9: Impact of the parameters ρ, λ, and α on the relative gain of the hierarchical protocols with respect to the coordinated one on the Titan platform.



2D-Stencil Hierarchical

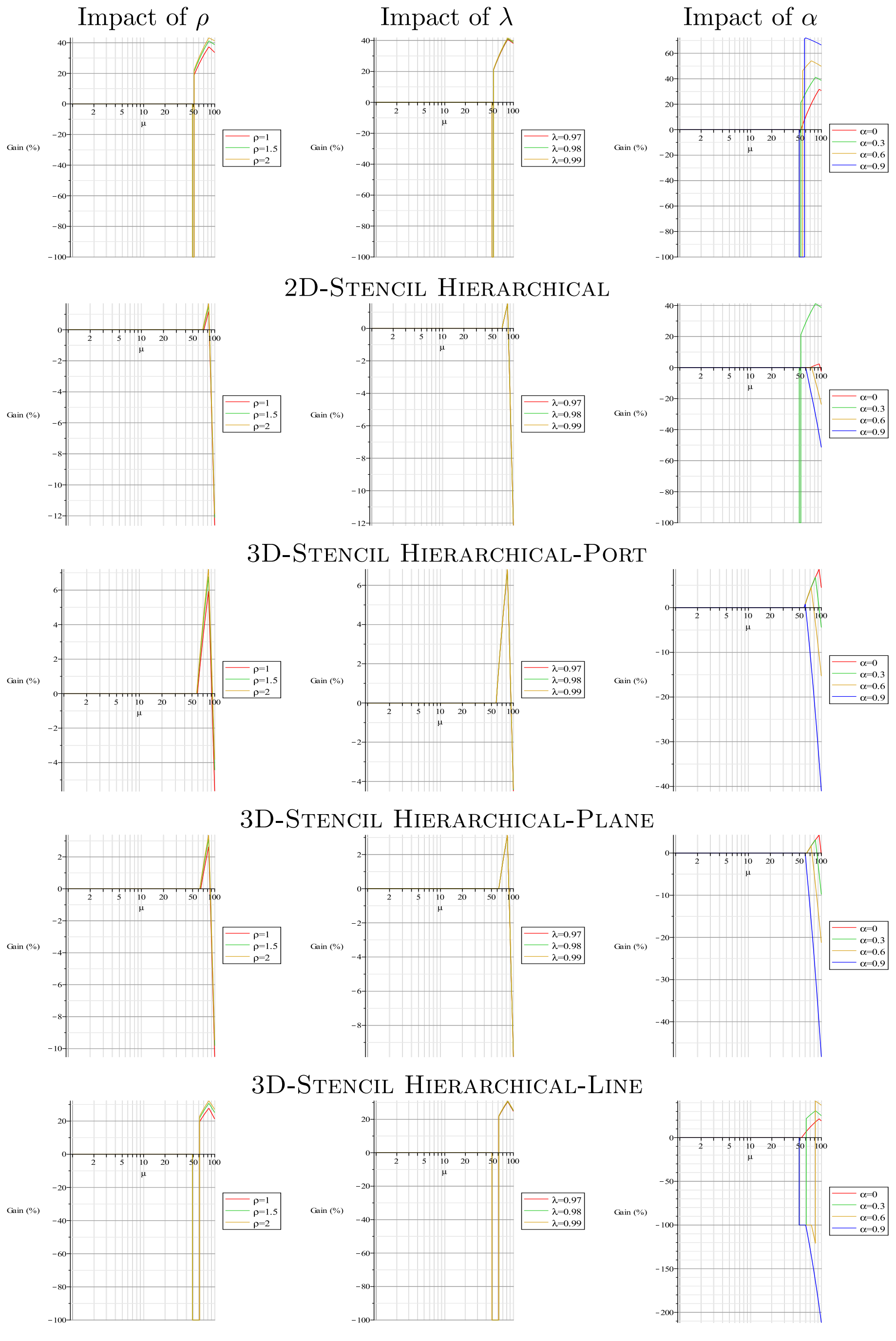
3D-Stencil Hierarchical-Port

3D-Stencil Hierarchical-Plane

3D-Stencil Hierarchical-Line

Matrix Product Hierarchical

Figure 10: Impact of the parameters ρ, λ, and α on the relative gain of the hierarchical protocols with respect to the coordinated one on the K-computer platform.



Impact of ρ

Impact of λ

Impact of α

2D-Stencil Hierarchical

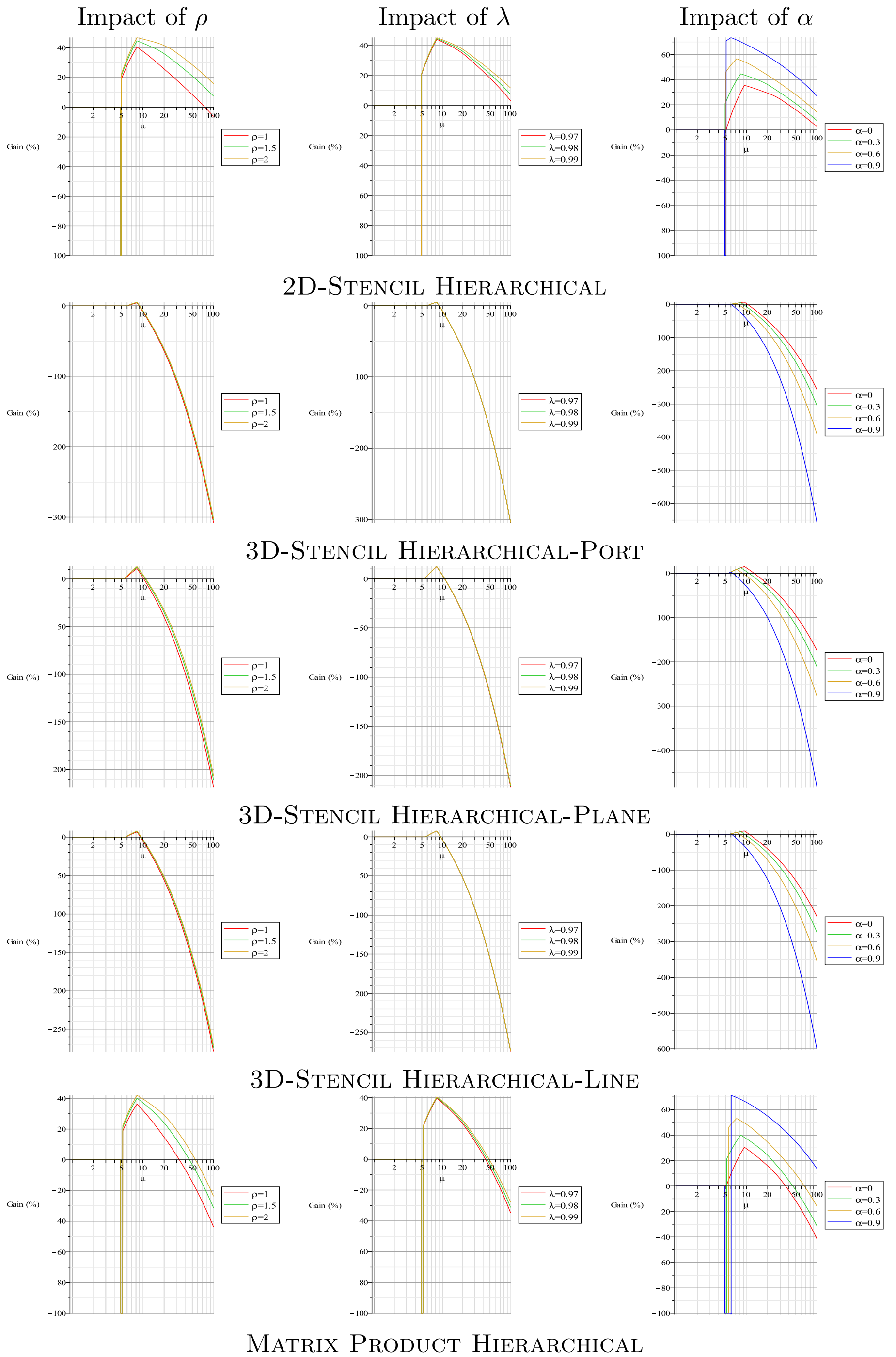
3D-Stencil Hierarchical-Port

3D-Stencil Hierarchical-Plane

3D-Stencil Hierarchical-Line

Matrix Product Hierarchical

Figure 11: Impact of the parameters ρ, λ, and α on the relative gain of the hierarchical protocols with respect to the coordinated one on the Exascale-Slim platform with $C_{\text{Mem}} = C_{\text{Mem}}/100$.



2D-Stencil Hierarchical

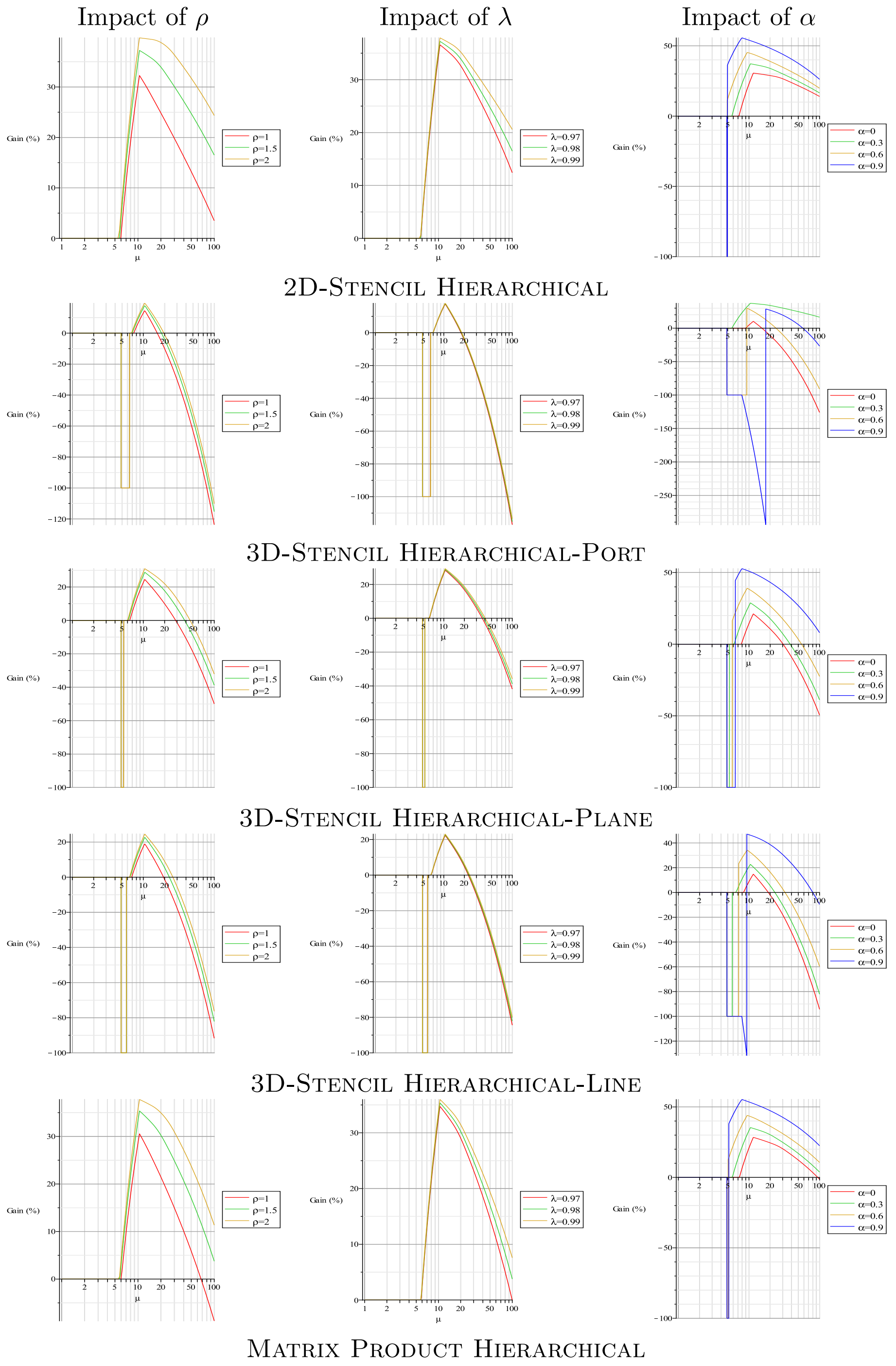
3D-Stencil Hierarchical-Port

3D-Stencil Hierarchical-Plane

3D-Stencil Hierarchical-Line

Matrix Product Hierarchical

Figure 12: Impact of the parameters ρ, λ, and α on the relative gain of the hierarchical protocols with respect to the coordinated one on the Exascale-Fat platform with $C_{\text{Mem}} = C_{\text{Mem}}/100$.



2D-Stencil Hierarchical

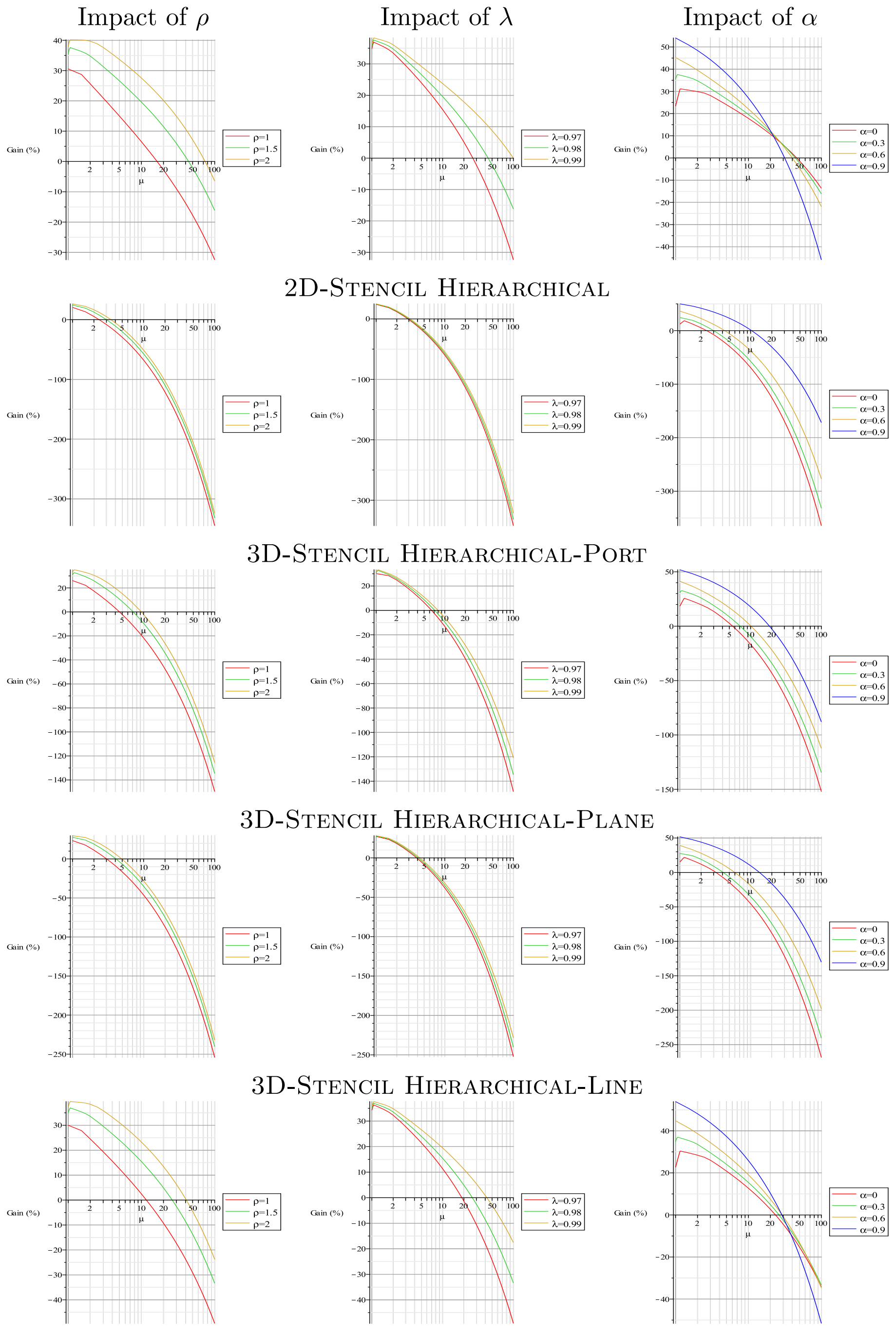
3D-Stencil Hierarchical-Port

3D-Stencil Hierarchical-Plane

3D-Stencil Hierarchical-Line

Matrix Product Hierarchical

Figure 13: Impact of the parameters ρ, λ, and α on the relative gain of the hierarchical protocols with respect to the coordinated one on the Exascale-Slim platform (with $C_{\text{Mem}} = C_{\text{Mem}}/1000$).



2D-Stencil Hierarchical

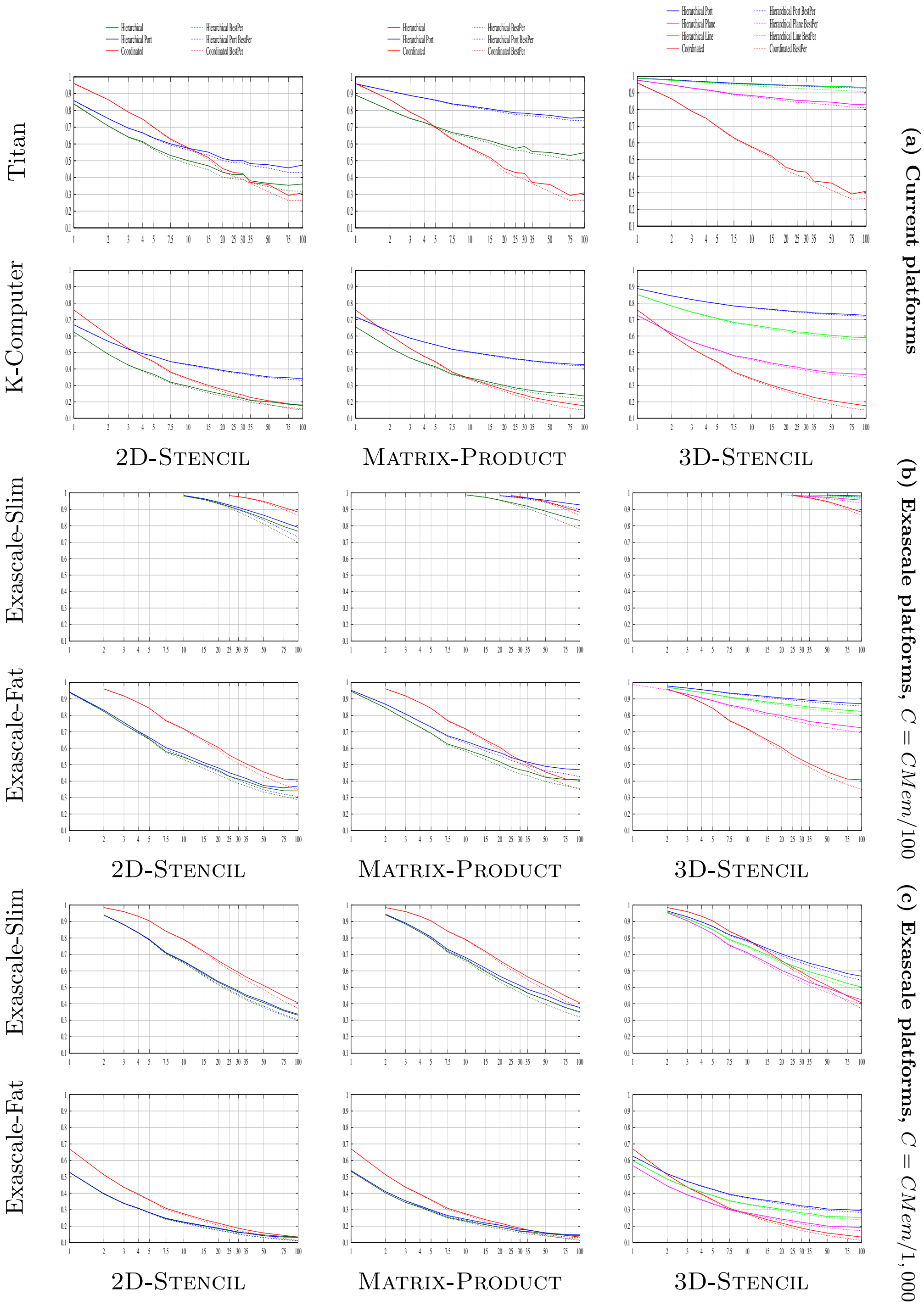
3D-Stencil Hierarchical-Port

3D-Stencil Hierarchical-Plane

3D-Stencil Hierarchical-Line

Matrix Product Hierarchical

Figure 14: Impact of the parameters ρ, λ, and α on the relative gain of the hierarchical protocols with respect to the coordinated one on the Exascale-Fat platform (with $C_{\text{Mem}} = 1000$).

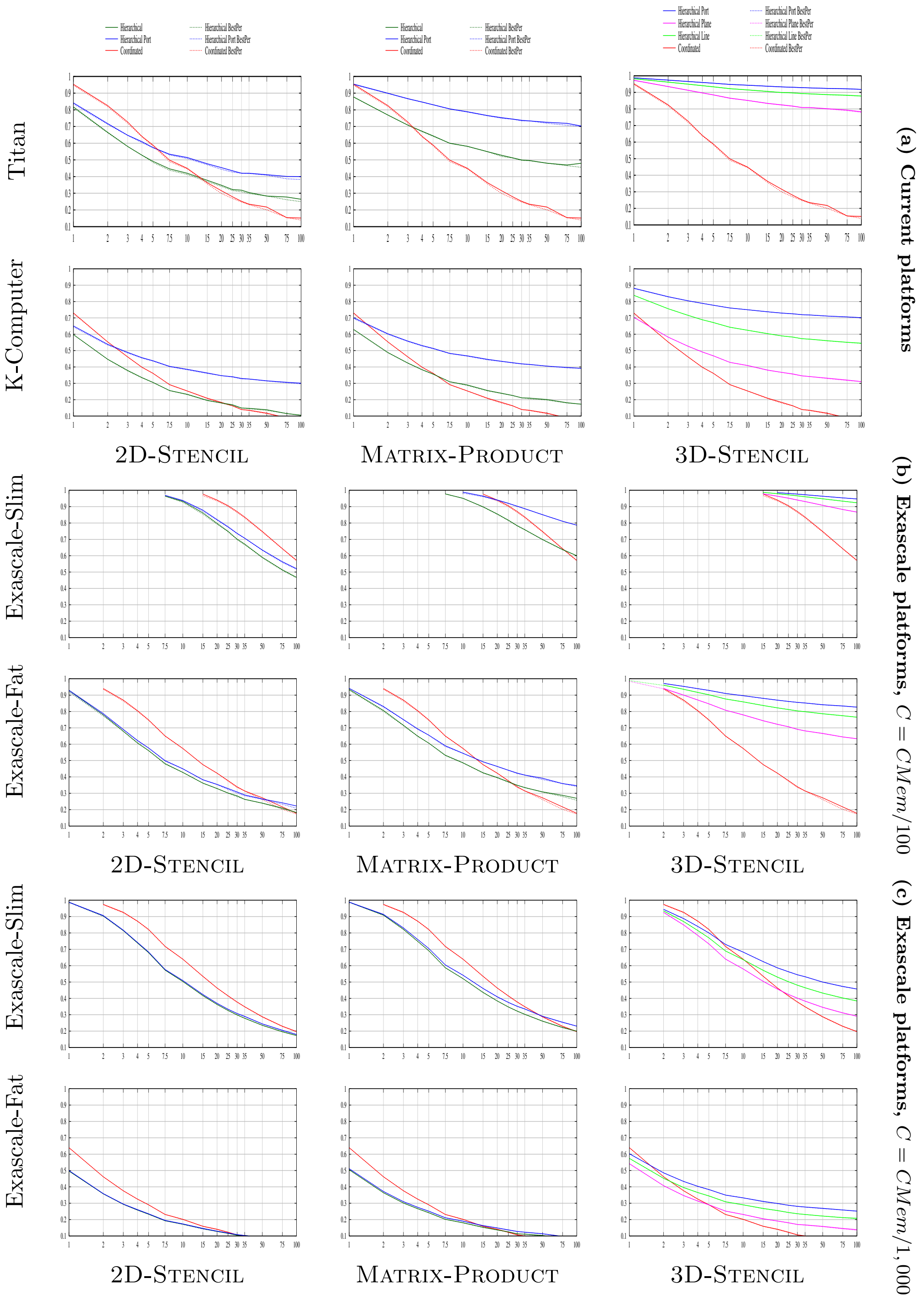


(a) Current platforms

(b) Exascale platforms, $C = CMem/100$

(c) Exascale platforms, $C = CMem/1,000$

Figure 15: Simulations: waste as a function MTBF μ (years per processor), for a Weibull failure distribution with shape parameter $k = 0.7$.

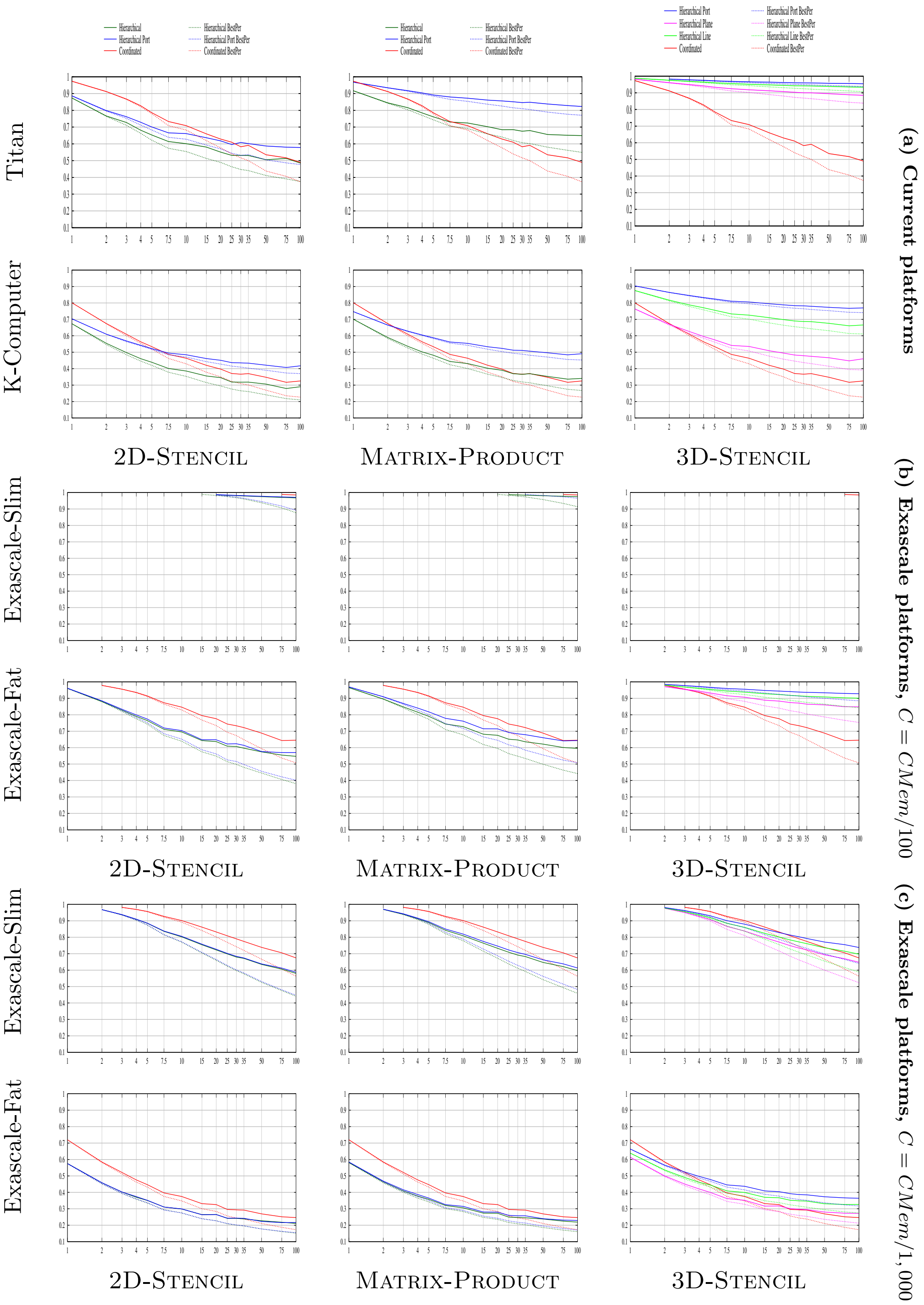


(a) Current platforms

(b) Exascale platforms, $C = CMem/100$

(c) Exascale platforms, $C = CMem/1,000$

Figure 16: Simulations: waste as a function MTBF μ (years per processor), for an Exponential failure distribution.



(a) Current platforms

(b) Exascale platforms, $C = CMem/100$

(c) Exascale platforms, $C = CMem/1,000$

Figure 17: Simulations: waste as a function MTBF μ (years per processor), for a Weibull failure distribution with shape parameter $k = 0.5$.

References

[1] J. Dongarra, P. Beckman, P. Aerts, F. Cappello, T. Lippert, S. Matsuoka, P. Messina, T. Moore, R. Stevens, A. Trefethen, and M. Valero, "The international exascale software project: a call to cooperative action by the global high-performance community," *Int. J. High Perform. Comput. Appl.*, vol. 23, no. 4, pp. 309–322, 2009.

[2] EESI, "The European Exascale Software Initiative," 2011, http://www.eesi-project.eu/pages/menu/homepage.php.

[3] V. Sarkar *et al.*, "Exascale software study: Software challenges in extreme scale systems," 2009, white paper available at: http://users.ece.gatech.edu/mrichard/ExascaleComputingStudyReports/ECSS%20report%20101909.pdf.

[4] S. Ashby *et al.*, "The opportunities and challenges of exascale computing," 2010, white paper available at: http://science.energy.gov/~/media/ascr/ascac/pdf/reports/Exascale_subcommittee_report.pdf.

[5] K. Ferreira, J. Stearley, J. H. I. Laros, R. Oldfield, K. Pedretti, R. Brightwell, R. Riesen, P. G. Bridges, and D. Arnold, "Evaluating the Viability of Process Replication Reliability for Exascale Systems," in *Proc. ACM/IEEE Conf. on Supercomputing*, 2011.

[6] A. Bouteiller, T. Herault, G. Krawezik, P. Lemarinier, and F. Cappello, "MPICH-V: a multiprotocol fault tolerant MPI," *IJHPCA*, vol. 20, no. 3, pp. 319–333, 2006.

[7] S. Rao, L. Alvisi, H. M. Viny, and D. C. Sciences, "Egida: An extensible toolkit for low-overhead fault-tolerance," in *In Symposium on Fault-Tolerant Computing.* Press, 1999, pp. 48–55.

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

[9] S. Rao, L. Alvisi, and H. M. Vin, "The cost of recovery in message logging protocols," in *17th Symposium on Reliable Distributed Systems (SRDS).* IEEE CS Press, October 1998, pp. 10–18.

[10] A. Bouteiller, T. Herault, G. Bosilca, and J. J. Dongarra, "Correlated set coordination in fault tolerant message logging protocols," in *Proc. of Euro-Par'11 (II)*, ser. LNCS, vol. 6853. Springer, 2011, pp. 51–64.

[11] A. Guermouche, T. Ropars, M. Snir, and F. Cappello, "HydEE: Failure Containment without Event Logging for Large Scale Send-Deterministic MPI Applications," in *IPDPS'12.* IEEE, 2012.

[12] C. L. M. Esteban Meneses and L. V. Kalé, "Team-based message logging: Preliminary results," in *Workshop Resilience in Clusters, Clouds, and Grids (CCGRID 2010).*, 2010.

[13] J. S. Plank, "Efficient Checkpointing on MIMD Architectures," Ph.D. dissertation, Princeton University, jun 1993.

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

[15] J. T. Daly, "A higher order estimate of the optimum checkpoint interval for restart dumps," *FGCS*, vol. 22, no. 3, pp. 303–312, 2004.

[16] Y. Ling, J. Mi, and X. Lin, "A variational calculus approach to optimal checkpoint placement," *IEEE Trans. Computers*, pp. 699–708, 2001.

[17] T. Ozaki, T. Dohi, H. Okamura, and N. Kaio, "Distribution-free checkpoint placement algorithms based on min-max principle," *IEEE TDSC*, pp. 130–140, 2006.

[18] M. Bougeret, H. Casanova, M. Rabie, Y. Robert, and F. Vivien, "Checkpointing strategies for parallel jobs," INRIA, France, Research Report 7520, Jan. 2011, available at http://graal.ens-lyon.fr/~fvivien/.

[19] J. S. Plank and M. G. Thomason, "Processor allocation and checkpoint interval selection in cluster computing systems," *Journal of Parallel and Distributed Computing*, vol. 61, p. 1590, 2001.

[20] H. Jin, Y. Chen, H. Zhu, and X.-H. Sun, "Optimizing HPC Fault-Tolerant Environment: An Analytical Approach," in *39th Int. Conf. Parallel Processing (ICPP)*, 2010, pp. 525 –534.

[21] L. Wang, P. Karthik, Z. Kalbarczyk, R. Iyer, L. Votta, C. Vick, and A. Wood, "Modeling Coordinated Checkpointing for Large-Scale Supercomputers," in *Proceedings of ICDSN'05*, 2005, pp. 812–821.

[22] R. Oldfield, S. Arunagiri, P. Teller, S. Seelam, M. Varela, R. Riesen, and P. Roth, "Modeling the impact of checkpoints on next-generation systems," in *Proceedings of IEEE MSST'07*, 2007, pp. 30 –46.

[23] Z. Zheng and Z. Lan, "Reliability-aware scalability models for high performance computing," in *Proc. of IEEE Cluster'09*, 2009, pp. 1 –9.

[24] M.-S. Bouguerra, D. Trystram, and F. Wagner, "Complexity Analysis of Checkpoint Scheduling with Variable Costs," *IEEE Transactions on Computers*, vol. 99, no. PrePrints, 2012.

[25] M. Wu, X.-H. Sun, and H. Jin, "Performance under failures of high-end computing," in *Proc. ACM/IEEE Conf. on Supercomputing*, 2007.

[26] T. Heath, R. P. Martin, and T. D. Nguyen, "Improving cluster availability using workstation validation," *SIGMETRICS Perf. Eval. Rev.*, vol. 30, no. 1, pp. 217–227, 2002.

[27] B. Schroeder and G. A. Gibson, "A large-scale study of failures in high-performance computing systems," in *Proc. of DSN*, 2006.

[28] Y. Liu, R. Nassar, C. Leangsuksun, N. Naksinehaboon, M. Paun, and S. Scott, "An optimal checkpoint/restart model for a large scale high performance computing system," in *IPDPS'08*. IEEE, 2008, pp. 1–9.

[29] E. Heien, D. Kondo, A. Gainaru, D. LaPine, B. Kramer, and F. Cappello, "Modeling and tolerating heterogeneous failures in large parallel systems," in *Proc. ACM/IEEE Conf. on Supercomputing*, 2011.

[30] L. G. Valiant, "A bridging model for parallel computation," *Commun. ACM*, vol. 33, no. 8, pp. 103–111, 1990.

[31] A. Bouteiller, G. Bosilca, and J. Dongarra, "Redesigning the message logging model for high performance," *Concurrency and Computation: Practice and Experience*, vol. 22, no. 16, pp. 2196–2211, 2010.

[32] M. Bougeret, H. Casanova, Y. Robert, F. Vivien, and D. Zaidouni, "Using group replication for resilience on exascale systems," INRIA, Research report RR-7876, February 2012.

[33] L. E. Cannon, "A cellular computer to implement the Kalman filter algorithm," Ph.D. dissertation, Montana State University, 1969.

[34] S. Sumimoto, "An Overview of Fujitsu's Lustre Based File System," Lustre Filesystem Users' Group Meeting, Orlando, USA., April 2011.

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