Optimal checkpointing period with replicated execution on heterogeneous platforms

Anne Benoit
LIP, ENS Lyon, France
anne.benoit@ens-lyon.fr

Aurélien Cavelan
LIP, ENS Lyon, France
aurelien.cavelan@ens-lyon.fr

Valentin Le Fèvre
LIP, ENS Lyon, France
valentin.le-fevre@ens-lyon.fr

Yves Robert
LIP, ENS Lyon, France & Univ. Tenn. Knoxville, USA
yves.robert@inria.fr

ABSTRACT

In this paper, we design and analyze strategies to replicate the execution of an application on two different platforms subject to failures, using checkpointing on a shared stable storage. We derive the optimal pattern size W for a periodic checkpointing strategy where both platforms concurrently try and execute W units of work before checkpointing. The first platform that completes its pattern takes a checkpoint, and the other platform interrupts its execution to synchronize from that checkpoint. We compare this strategy to a simpler on-failure checkpointing strategy, where a checkpoint is taken by one platform only whenever the other platform encounters a failure. We use first or second-order approximations to compute overheads and optimal pattern sizes, and show through extensive simulations that these models are very accurate. The simulations show the usefulness of a secondary platform to reduce execution time, even when the platforms have relatively different speeds: in average, over a wide range of scenarios, the overhead is reduced by 30%. The simulations also demonstrate that the periodic checkpointing strategy is globally more efficient, unless platform speeds are quite close.

ACM Reference format:
Anne Benoit, Aurélien Cavelan, Valentin Le Fèvre, and Yves Robert. 2017. Optimal checkpointing period
with replicated execution on heterogeneous platforms. In *Proceedings of FTXS, Washington, D.C., USA, 2017 (FTXS'17)*, 8 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

One of the most important challenges faced by large-scale computing systems is the frequent occurence of failures (a.k.a. fails-top errors) [3, 17]. Platform sizes have become so large that failures are likely to strike during the execution of an application. Consider the mean time between failures μ (usually denoted as MTBF) of a platform with p processors: μ decreases linearly with p, since $\mu = \frac{\mu_{\text{ind}}}{p}$, where μ_{ind} is the MTBF of each individual component (see Proposition 1.2 in [10]). For instance, with $\mu_{\text{ind}} = 10$ years and $p = 10^5$, we have $\mu \approx 50$ minutes, and it goes down to a failure every 5 minutes for $p = 10^6$.

The classical technique to deal with failures is to use a checkpoint-restart mechanism: the state of the application is periodically checkpointed on stable storage, and when a failure occurs, we can recover from the last valid checkpoint and resume the execution, rather than starting again from scratch. Checkpointing policies have been widely studied, see [10] for a survey of various protocols and the derivation of the Young/Daly formula [5, 19] for the optimal checkpointing period. Recent advances include multi-level approaches, or the use of SSD or NVRAM as secondary storage [3].

Another technique that has been advocated to deal with failures is *process replication*, where each process in a parallel MPI (Message Passing Interface) application is duplicated to increase the mean-time to interruption. More precisely, each processor of the platform is paired with a replica so that the execution can continue whenever one of them is struck by a failure. Given the high rate of failures expected in current systems, process replication is usually combined with a periodic checkpointing mechanism, as proposed in [8, 9, 16, 20] for HPC platforms, and in [11, 18] for grid computing. These approaches use *process replication*: each processor of the platform is paired with a replica so that the execution can continue whenever one is struck by a failure.

Another approach introduced in [4] is *group replication*, a technique that can be used whenever process replication is not available. Group replication is agnostic to the parallel programming model, and thus views the application as an unmodified black box. Group replication consists in executing multiple application instances concurrently. For example, two distinct p-process application instances could be executed on a $2p$-processor platform. Once an instance saves a checkpoint, the other instance can use this checkpoint immediately to "jump ahead" in its execution. A similar technique named *shadow replication* has been introduced in [12]. In this work, the authors consider a slow and a fast replica in order to both increase the reliability of the execution and reduce the associated energy consumption. When the fast replica fails, they increase the speed of the slow replica to complete the task under a targeted response time, and thus do not rely on checkpointing.

In this work, we extend group replication to the case of two different computing platforms executing concurrently and cooperating to the success of a given application. To the best of our knowledge, this scenario has not been explored yet. The two platforms share a set of remote disks, used as a stable storage for checkpointing. Typically, these platforms would be clusters, which may have different number of processors, and hence different MTBFs and execution

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
FTXS'17, Washington, D.C., USA
© 2017 ACM. 978-x-xxxx-xxxx-x/YY/MM…$15.00
DOI: 10.1145/nnnnnnn.nnnnnnn

speeds. Our goal is to determine the best way to have both platforms cooperate so that the execution time of the application is minimized. We design and analyze two strategies:

1. A periodic checkpointing strategy, where both platforms checkpoint periodically once they have executed a chunk of work of size W. Both platforms synchronize through the shared storage as soon as one of them has completed the execution of a chunk (at the time of the checkpoint). We provide a thorough analysis to express the overhead given the checkpointing period W, and we derive the size of the optimal pattern.

2. An on-failure checkpointing strategy, where each platform progresses at its own speed, and checkpoints only when a failure occurs on the other platform. Hence, when a failure occurs on one of the platforms (say platform A), the other one (platform B) checkpoints, and platform A gets a copy of this checkpoint to restart its execution at this point. Intuitively, if both platforms have the same speed, we will never roll back with this strategy, unless a failure occurs during checkpoint. We compare both strategies through extensive simulations, and show the gain (opr the absence thereof) compared to using a single platform. We also assess the accuracy of the model and of our first or second-order approximations.

The rest of the paper is organized as follows. We introduce the execution model in Section 2, and derive the optimal pattern for the periodic checkpointing strategy in Section 3. The analysis for the checkpoint-on-failure strategy is given in Section 4. Section 5 is devoted to the experimental evaluation. Finally, we provide concluding remarks and directions for future work in Section 6.

2 MODEL

We consider a black-box application and replicate its execution on two different computing platforms P_1 and P_2. The platforms may well be heterogeneous, with different processor numbers, different MTBF values and different execution speeds. Both platforms use the same stable storage system. A typical instance is the case of two clusters that share a set of storage disks. We assume that both executions can synchronize through checkpointing. Checkpoint time is C on either platform, and this includes the time to update the state of the application on the other platform. We make no further hypothesis: The checkpointing protocol can be single-level or multi-level, and the update of the application state on the other platform can take place either through the network or via the file system. Recovery time is R, independently of which platform has taken the last checkpoint.

We partition the execution of the application into *periodic patterns*, i.e., computational units that repeat over time. Each pattern includes W units of work (we also say a chunk of size W) and ends with a checkpoint. With a single platform, the optimal pattern length is well-known and obeys the Young/Daly formula [5, 19]. With two platforms executing concurrently, both platforms execute the pattern concurrently, and repeat until success. Once a platform succeeds, the other one stops executing and synchronizes on checkpoint. Computing the optimal pattern length turns out a challenging problem in this case.

We assume that failures independently strike the platforms with an Exponential distribution. Platform P_1 has failure rate λ_1, which means its MTBF (Mean Time Between Failures) is $\mu_1 = \frac{1}{\lambda_1}$. Similarly, P_2 has failure rate λ_2, and MTBF $\mu_2 = \frac{1}{\lambda_2}$. We let σ_1 be the execution speed of the application on platform P_1, and σ_2 be the speed on P_2. We assume that P_1 is the fast platform, so that $\sigma_1 \geq \sigma_2$.

The expected execution time of the pattern is $\mathbb{E}(P)$. Letting $T_1 = \frac{W}{\sigma_1}$, we note that $\mathbb{E}(P) > T_1 + C$, the failure-free time on the fast platform. An optimal pattern is defined as the one minimizing the ratio $\frac{\mathbb{E}(P)}{T_1}$, or equivalently the ratio $\mathbb{H}(P) = \frac{\mathbb{E}(P)}{T_1} - 1$. This latter ratio is the relative overhead paid for executing the pattern. The smaller this overhead, the faster the progress of the execution. For the theoretical analysis, we assume that checkpoint and recovery are failure-free, because this assumption does not modify the dominant terms of the overhead (see the companion research report [2] for details), but for the simulations, we do account for failures striking anytime. Finally, to be able to write Taylor expansions, we also let λ be the global failure rate and write $\lambda_1 = \alpha_1\lambda$ and $\lambda_2 = \alpha_2\lambda$, with $\alpha_1 + \alpha_2 = 1$.

3 OPTIMAL PATTERN

In this section, we show how to derive the optimal pattern length. The derivation is quite technical, and a summary of the results is provided in Section 3.2.

3.1 Expected execution time

Consider a pattern P of size W, and let $\mathbb{E}(P)$ denote the expected execution time of the pattern. Because we assume that checkpoints are failure-free, we have $\mathbb{E}(P) = \mathbb{E}(W) + C$, where $\mathbb{E}(W)$ is the expected time to execute a chunk of size W. We start with some background on well-known results on $\mathbb{E}(W)$ with a single platform P_1, before moving on to our problem with two platforms. With a single platform P_1, let $T_1 = \frac{W}{\sigma_1}$ and $p_1 = 1 - e^{-\lambda_1 T_1}$ be the probability of a failure on P_1 while attempting to execute the chunk of size W. We can write

$$\mathbb{E}(W) = (1 - p_1)T_1 + p_1(\mathbb{E}^{\text{lost}} + R + \mathbb{E}(W)).$$

The first term corresponds to a successful execution, while the second term accounts for a failure striking during execution, with expected time lost $\mathbb{E}^{\text{lost}}$, recovery time R and calling $\mathbb{E}(W)$ recursively to restart from scratch. We know from [10] that $\mathbb{E}^{\text{lost}} = \frac{1}{\lambda_1} - \frac{T_1}{e^{\lambda_1 T_1 - 1}}$, and after simplification we get $\mathbb{E}(W) = (\frac{1}{\lambda_1} + R)(e^{\lambda_1 T_1} - 1)$ (see [10]for details). We aim at minimizing the pattern overhead $\mathbb{H}(P) = \frac{E(P)}{T_1} - 1$. To get a first-order approximation, we assume that $\lambda_1 W$ is small so that we can expand $p_1 = 1 - e^{\lambda_1 T_1}$ into

$$p_1 = \lambda_1 \frac{W}{\sigma_1} + \frac{1}{2}\left(\lambda_1 \frac{W}{\sigma_1}\right)^2 + o\left(\left(\lambda_1 \frac{W}{\sigma_1}\right)^2\right).$$

We then derive that $\mathbb{H}(P) = \frac{C\sigma_1}{W} + \frac{\lambda_1 W}{2\sigma_1} + o(\sqrt{\lambda_1})$. The first two-terms show that $W_{\text{opt}} = \Theta(\lambda_1^{-1/2})$ and we retrieve the Young/Daly formula $W_{\text{opt}} = \sigma_1\sqrt{\frac{2C}{\lambda_1}}$. For the optimal pattern, we have $\mathbb{H}_{\text{opt}} = \sqrt{2C\lambda_1} + o(\sqrt{\lambda_1})$.

Equipped with these results for a single platform, we can now tackle the problem with two platforms. We will need a second-order approximation of the form

$$\mathbb{H}(P) = \frac{C\sigma_1}{W} + \beta\left(\lambda\frac{W}{\sigma_1}\right) + \gamma\left(\lambda\frac{W}{\sigma_1}\right)^2 + \delta\lambda + o\left((\lambda W)^2\right),$$

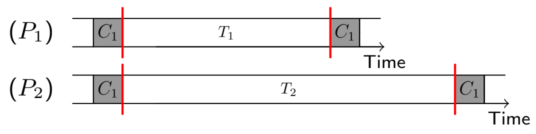

Figure 1: I_0 – no failure on P_1 (there can be failures on P_2); P_1 always finishes first.

where $\lambda = \lambda_1 + \lambda_2$ is the total failure rate, and β, γ and δ are constants that we derive below. With a single platform, we had $\beta = \frac{1}{2}$. With two platforms, we obtain a complicated expression for β, whose value will always be nonnegative. If β is strictly positive and above a reasonable threshold, we will proceed as above and be satisfied with the first-order approximation that gives $W_{\text{opt}} = \sigma_1\sqrt{\frac{C}{\beta\lambda}} = \Theta(\lambda^{-1/2})$. However, if β is zero or close to zero, we need to resort to the second-order expansion to derive an accurate approximation of W_{opt}. In particular, when P_1 and P_2 are same-speed platforms, we find that $\beta = 0$, $\gamma > 0$ and $W_{\text{opt}} = \Theta(\lambda^{-2/3})$.

As above, let $\mathbb{E}(W)$ be the expected time to execute a chunk of size W with both platforms. Let $T_1 = \frac{W}{\sigma_1}$ and $p_1 = 1 - e^{-\lambda_1 T_1}$ as before. We write $\mathbb{E}(W) = \sum_{i=0}^{\infty} p_1^i(1-p_1)\mathbb{E}_i$, where $\mathbb{E}_i$ denotes the expected time to execute W successfully, knowing that there were i failures on P_1 before P_1 executes the chunk successfully. We point out that we condition $\mathbb{E}_i$ on the number of failures on P_1, independently on what is happening on P_2. In other words, we let P_1 execute until success, but we do account for the fact that P_2 may have completed before P_1 when computing $\mathbb{E}_i$. Similarly, letting $T_2 = \frac{W}{\sigma_2}$ and $p_2 = 1 - e^{-\lambda_2 T_2}$ be the probability of a failure on P_2, we write $\mathbb{E}_i = \sum_{j=0}^{\infty} p_2^j(1-p_2)\mathbb{E}_{i,j}$, where $\mathbb{E}_{i,j}$ denotes the expected execution time of the pattern, knowing there were i failures on P_1 and j failures on P_2 before both platforms execute successfully.

Theorem 3.1. *The expected execution time of a pattern $\mathbb{E}(P)$ of length W, whose execution is replicated on two platforms P_1 and P_2, is $\mathbb{E}(P) = \mathbb{E}(W) + C$, where*

$$\begin{aligned}
\mathbb{E}(W) = (1-p_1)T_1 & \qquad (I_0)\\
+ p_1(1-p_1-p_2)\mathbb{E}_{1,0} & \qquad (I_1)\\
+ p_1p_2\mathbb{E}_{1,1} & \qquad (I_2)\\
+ p_1^2\mathbb{E}_{2,0} & \qquad (I_3)\\
+ O(\lambda^3W^4)\ ,
\end{aligned}$$

Here I_0, I_1, I_2 and I_3 denote the four possible outcomes of the execution (up to two failures), with their associated probability.

The proof of Theorem 3.1 is technical; due to a lack of space, we refer the reader to the companion research report [2] for additional details. The complicated part is to compute approximation of I_0, I_1, I_2 and I_3. We explain how to do so for I_0 and I_1, which are the easy ones. I_2 and I_3 correspond to two failures (one on each platform for I_2, two on P_1 for I_3) and are detailed in [2].

Computing I_0 (Figure 1). Let I_0 denote the expected execution time associated with having no failures on P_1. With probability $(1-p_1)$, P_1 finishes faster than P_2 in T_1 time, and we can write $I_0 = (1-p_1)T_1$. Using Taylor expansions to approximate p_1 to

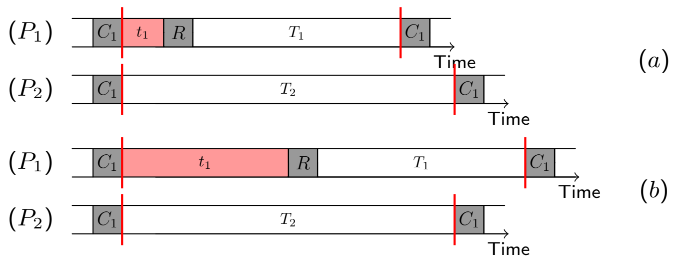

Figure 2: I_1 – there is one failure on P_1; depending on the failure arrival time t_1, P_1 finishes either first (a) or last (b).

$\lambda_1 T_1 + \frac{\lambda_1^2T_1^2}{2} + o(\lambda^2T_1^2)$, we can write:

$$I_0 = \left(1 - \lambda_1T_1 - \frac{\lambda_1^2T_1^2}{2} + o(\lambda^2T_1^2)\right)T_1 = T_1 - \lambda_1T_1^2 - \frac{\lambda^2T_1^3}{2} + o(\lambda^2T_1^3)\ .$$

Computing I_1 (Figure 2). Let I_1 denote the expected execution time when having exactly one failure on P_1. Letting $X \sim exp(\lambda_1)$ denote the failure inter-arrival time, we have:

$$\begin{aligned}
I_1 &= p_1(1-p_1-p_2)\int_0^\infty \mathbb{P}(X=t|X\le T_1)\min(t+R+T_1,T_2)dt\\
&= p_1(1-p_1-p_2)\frac{1}{\mathbb{P}(X\le T_1)}\int_0^{T_1}\mathbb{P}(X=t)\min(t+R+T_1,T_2)dt\ .
\end{aligned}$$

By definition, $\mathbb{P}(X \le T_1) = p_1$ and $\mathbb{P}(X = t) = \lambda_1 e^{-\lambda_1 t}$, therefore:

$$I_1 = (1-p_1-p_2)\int_0^{T_1}\lambda_1 e^{-\lambda_1 t}\min(t+R+T_1,T_2)dt\ .$$

Note that $\min(t+R+T_1,T_2)$ is in order of $O(W)$. Using Taylor series to approximate p_1 to $\lambda_1T_1 + o(\lambda W)$, p_2 to $\lambda_2T_2 + o(\lambda W)$, $e^{-\lambda_1 t}$ to $1 - \lambda_1 t + o(\lambda t)$ and keeping second-order terms only, we can get:

$$\begin{aligned}
I_1 &= \lambda_1(1-\lambda_1T_1-\lambda_2T_2)\int_0^{T_1}(1-\lambda_1T_1)\min(t+R+T_1,T_2)dt\\
&\quad + o(\lambda^2W^3)\ .
\end{aligned}$$

The minimum depends on which platform finishes first. We know that $t + R + T_1 \le T_2 \iff t \le T_2 - T_1 - R$, so that we break the integral into two parts to address both cases, as follows:

$$\begin{aligned}
I_1 &= \lambda_1(1-\lambda_1T_1-\lambda_2T_2)\Bigg(\int_0^{T_2-T_1-R}(1-\lambda_1t)(t+R+T_1)dt\\
&\qquad + \int_{T_2-T_1-R}^{T_1}(1-\lambda_1t)T_2\Bigg)dt + o(\lambda^2W^3)\ ,
\end{aligned}$$

where $T_2 - T_1 - R$ must be both positive and less that T_1. Finally, let $r_1 = \max(\min(T_2 - T_1 - R, T_1), 0)$, and we can write:

$$\begin{aligned}
I_1 &= \lambda_1(1-\lambda_1T_1-\lambda_2T_2)\Bigg(\int_0^{r_1}(1-\lambda_1t)(t+R+T_1)dt\\
&\qquad + \int_{r_1}^{T_1}(1-\lambda_1t)T_2dt\Bigg) + o(\lambda^2W^3)\ .
\end{aligned}$$

Finally, note that I_1 depends on the value of r_1 as follows:

$$r_1 = \begin{cases} T_2 - T_1 - R, & \text{if } 0 \le T_2 - T_1 - R \le T_1\\ T_1, & \text{if } T_2 - T_1 - R > T_1\\ 0, & \text{otherwise.}\end{cases}$$

Assuming R is small in front T_1 and T_2, we derive:

$$r_1 = \begin{cases} T_2 - T_1 - R, & \text{if } 1 \le \frac{\sigma_1}{\sigma_2} \le 2 \\ T_1, & \text{if } 2 < \frac{\sigma_1}{\sigma_2}. \end{cases}$$

3.2 Expected overhead

Theorem 3.2. *The expected overhead of a pattern $\mathbb{H}(P)$, whose execution is replicated on two independent platforms P_1 and P_2 is*

$$\mathbb{H}(P) = \frac{C\sigma_1}{W} + \beta\left(\lambda\frac{W}{\sigma_1}\right) + \gamma\left(\lambda\frac{W}{\sigma_1}\right)^2 + \delta\lambda + o\left((\lambda W)^2\right) , \quad (1)$$

where $\lambda_1 = \alpha_1\lambda$ and $\lambda_2 = \alpha_2\lambda$ with $\alpha_1 + \alpha_2 = 1$. The values of the constants β, γ and δ are provided by the following case analysis:

Case 1: $1 \le \frac{\sigma_1}{\sigma_2} \le 2$.

$$\beta = \frac{\alpha_1}{2}\frac{-\sigma_1^2 + 4\sigma_1\sigma_2 - 3\sigma_2^2}{\sigma_2^2} ,$$

$$\gamma = \frac{\alpha_1^2}{2}\frac{\sigma_1^2 - 3\sigma_1\sigma_2 + 2\sigma_2^2}{\sigma_2^2} + \frac{\alpha_1\alpha_2}{3}\frac{2\sigma_1^3 - 9\sigma_1^2\sigma_2 + 12\sigma_1\sigma_2^2 - 4\sigma_2^3}{\sigma_2^3} ,$$

$$\delta = R\frac{\sigma_1 - \sigma_2}{\sigma_2} .$$

Case 2: $2 \le \frac{\sigma_1}{\sigma_2} < 3$.

$$\beta = \frac{\alpha_1}{2}, \gamma = \frac{\alpha_1^2}{6}\frac{\sigma_1^3 - 9\sigma_1^2\sigma_2 + 27\sigma_1\sigma_2^2 - 26\sigma_2^3}{\sigma_2^3}, \delta = \alpha_1 R .$$

Case 3: $3 \le \frac{\sigma_1}{\sigma_2}$.

$$\beta = \frac{\alpha_1}{2}, \gamma = \alpha_1^2, \delta = \alpha_1 R .$$

The optimal checkpointing period W_{opt} can be obtained by solving the following third-degree equation numerically:

$$\frac{\partial\mathbb{H}(P)}{\partial W} = -\frac{C\sigma_1}{W^2} + \beta\frac{\lambda}{\sigma_1} + 2\gamma\frac{\lambda W}{\sigma_1^2} = 0 . \quad (2)$$

See [2] for a proof. For cases 2 and 3 (where $\sigma_1 \ge 2\sigma_2$), we have $\beta = \frac{\alpha_1}{2}$. If we use the first-order approximation, we neglect the last two terms with γ and δ in $\mathbb{H}(P)$. Then we obtain $W_{\text{opt}} = \sigma_1\sqrt{\frac{C}{\beta\lambda}}$, a similar formula as with a single platform. We experimentally check the accuracy of the first-order approximation in Section 5. On the contrary for case 1 (where $\sigma_1 \ge 2\sigma_2$), we have $\beta = \frac{\alpha_1}{2}(\frac{\sigma_1}{\sigma_2} - 1)(3 - \frac{\sigma_1}{\sigma_2}) \ge 0$ but $\beta = 0 \iff \sigma_1 = \sigma_2$. We can still use the first-order approximation when β is not too close to 0. For same-speed platforms, we need to use the second-order approximation:

Theorem 3.3. *For same-speed platforms ($\sigma_2 = \sigma_1$), the expected overhead is*

$$\mathbb{H}(P) = \frac{C\sigma_1}{W} + \frac{\alpha_1\alpha_2\lambda^2 W^2}{3\sigma_1^2} + o(\lambda^2 W^2) . \quad (3)$$

and the associated optimal checkpointing period is

$$W_{\text{opt}} = \sigma_1\sqrt[3]{\frac{3C}{2\alpha_1\alpha_2\lambda^2}} . \quad (4)$$

It is striking to note that $W_{\text{opt}} = \Theta(\lambda^{-2/3})$ for same speed platforms, instead of $W_{\text{opt}} = \Theta(\lambda^{-1/2})$. Finally, with two identical platforms ($\alpha_1 = \alpha_2 = \frac{1}{2}$ and $\lambda = 2\lambda_1$), we obtain $W_{\text{opt}} = \sigma_1\sqrt[3]{\frac{3C}{2\lambda_1^2}}$.

4 ON-FAILURE CHECKPOINTING

In this section, we present another strategy., where the work is no longer divided into periodic patterns. We only checkpoint when a failure strikes either platform. More precisely, when a failure f strikes one platform, we use the other platform to checkpoint the work, so that both platforms can resume their execution from this checkpoint, in a synchronized fashion. This scheme is exposed to the risk of having a second failure f' striking the other platform during its checkpoint, which would cause to roll-back and re-execute from the previous checkpoint (which was taken right after the failure preceding f, which may be a long time ago). Such a risk can be neglected in most practical settings. As before, we assume that failures do not strike during checkpoints.

Intuitively, this checkpoint-on-failure strategy is appealing, because we checkpoint a minimum number of times. And when a failure strikes the slow platform P_2, we do not roll-back. However, when a failure strikes the fast platform P_1, we have to roll-back to the state of P_2. Altogether, we expect this strategy to work better when platform speeds are close. We will experimentally assess the checkpoint-on-failure strategy in Section 5.

4.1 Expected execution time

Let $\mathbb{E}(A)$ denote the expected time needed to execute the application successfully, and let $T_{base} = \frac{W_{base}}{\sigma_1}$ denote the total execution time of the application on the fast platform P_1, without any resilience mechanism nor failures. Here W_{base} denotes the total amount of work of the application.

Theorem 4.1. *The expected execution time of the application is*

$$\mathbb{E}(A) = T_{base} + \frac{T_{base}}{\mu}\left(C + \alpha_1\left(\mu\frac{\sigma_1 - \sigma_2}{\sigma_1}\right)\right) . \quad (5)$$

where $\mu = \frac{1}{\lambda}$ is the MTBF.

Proof. We first consider the case of two identical platforms, i.e. $\sigma_1 = \sigma_2$ and $\lambda_1 = \lambda_2 = \frac{\lambda}{2}$. In this case, as soon as a failure occurs on either platform, the other one immediately checkpoints, and both platforms synchronize on this checkpoint, before resuming execution. In other words, the execution never rolls back, and no work is ever lost.

Now, in order to compute the expected execution time, we need to account for the time needed to execute the entire application T_{base}, as well as the time lost due to failures. When a failure occurs, we only need to account for the time C to checkpoint and synchronize. In addition, we can estimate the expected number of failures as $\frac{T_{base}}{\mu}$ in average, and we write $\mathbb{E}(A) = T_{base} + \frac{T_{base}}{\mu}C$.

This is fine for two identical platforms. However, when failure rates and speeds differ, there are two cases: (i) a failure strikes the fast platform P_1. Then platform P_2 checkpoints, but because it is slower than P_1, P_1 needs to rollback and we lose the extra amount of work that P_1 has computed since the last failure and synchronization; (ii) a failure strikes the slow platform P_2. Then platform P_1 checkpoints, and because it is faster, P_2 will roll-forward instead, catching up with the execution of P_1.

Assuming failures are Exponentially distributed, and given that a failure (from either platform) strikes during the execution of the segment, the probability that the failure belongs to a particular platform is proportional to the failure rate of that platform [13], i.e.

Name	Titan	Cori	K computer	Trinity	Theta
Speed (PFlops)	17.6	14.0	10.5	8.1	5.1
MTBF (s)	50,000	100,000			

Table 1: Summary of parameters used for simulations for each platform.

the probability that the failure belongs to P_1 and P_2 are $\frac{\lambda_1}{\lambda} = \alpha_1$ and $\frac{\lambda_2}{\lambda} = \alpha_2$, respectively.

In order to compute the expected execution time, we first need to account for T_{base}, which is the time to execute the application once, without failures. Then, when a failure strikes, either it strikes P_2, with probability α_2, and we only lose the time to checkpoint C; or it strikes P_1, with probability α_1, and we lose the difference between the amount of work executed on P_1 and P_2 since the last synchronization. In average, the last synchronization was when the last failure occurred, that is μ time-steps ago. During that time, P_1 and P_2 have executed $\mu\sigma_1$ and $\mu\sigma_2$ units of work, respectively, and we have lost $\mu\frac{\sigma_1-\sigma_2}{\mu}$ due to the failure. Altogether, we can write:

$$\mathbb{E}(A) = T_{base} + \frac{T_{base}}{\mu}\left(C + \alpha_1\left(\mu\frac{\sigma_1 - \sigma_2}{\sigma_1}\right)\right).$$

□

4.2 Expected overhead

Theorem 4.2. *The expected overhead is*

$$\mathbb{H}(A) = \frac{C}{\mu} + \alpha_1\left(\frac{\sigma_1 - \sigma_2}{\sigma_1}\right). \quad (6)$$

Proof. Let $\mathbb{H}(A) = \frac{\mathbb{E}(A)}{T_{base}} - 1$. We write:

$$\mathbb{H}(A) = \frac{1}{\mu}\left(C + \alpha_1\left(\mu\frac{\sigma_1 - \sigma_2}{\sigma_1}\right)\right).$$

Then, simplifying, we obtain Equation (6). □

5 EXPERIMENTAL EVALUATION

In this section, we conduct a set of simulations, whose goal is three-fold: (i) assess the accuracy of the proposed models; (ii) compare the performance of the two replication strategies in different scenarios; and (iii) evaluate the performance improvement of the approach over classical periodic checkpointing with a single platform.

5.1 Simulation setup

This section describes the parameters used for the simulations. First, we set $R = C$ in all cases. This is a common assumption [6, 14, 15], even though in practice the recovery cost can be expected to be smaller than the checkpoint cost [6, 7]. to a read (recovery) and a write (checkpoint) operation, and they take approximately the same amount of time. Then, we set the other parameters according to real behaviors on today's supercomputers. Because the typical failure rate for the most powerful Top500 platforms [1] is around 1 or 2 failures per day, we choose $\mu_1 = 50,000s \approx 14h$ and $\mu_2 = 100,000s \approx 28h$. The speeds were set using the *Rmax* value (maximum performance achieved when executing LINPACK) in PFlops of Top500 platforms (list of November 2016). We always set $\sigma_1 = 17.6$ (units in Petaflops, corresponding to the Titan platform), and we build four different cases aiming at having different $\frac{\sigma_1}{\sigma_2}$ ratios: σ_2 can be either 14.0 (Cori), 10.5 (K computer), 8.1 (Trinity) or 5.1 (Theta). We also have two possible configurations for the checkpointing (and recovery) time: a small checkpoint of 60 seconds and a large checkpoint of 1800 seconds. Overall, the parameters used by default for each platform are summarized in Table 1.

For each experiment, we setup the simulator with the resilience parameters λ_1, λ_2, C and R, and we compute the optimal pattern length W_{opt}, which is obtained by solving Equation 1 numerically. The total amount of work in the simulation is fixed to be $1000W_{\text{opt}}$, and each simulation is repeated 1000 times. All the figures report the optimal overhead $\mathbb{H}_{\text{opt}}$ as a function of some parameter. The solid lines are simulation results: green for the fastest machine alone (with Young/Daly period), blue for the periodic checkpoint strategy, red for the on-failure checkpoint strategy. The dashed lines are model predictions: blue for the periodic checkpoint strategy, red for the on-failure checkpoint strategy. The simulator is publicly available at http://perso.ens-lyon.fr/aurelien.cavelan/replication-ftxs.zip.

5.2 Accuracy of the models

In this section, we study the accuracy of the models and we assess the usefulness of the second-order approximation by comparing the results obtained with both first and second-order formulas. We take the fastest machine Titan and let its speed σ_1 vary, while keeping all other parameters fixed. Hence we always have $\mu_1 = 50,000s$ and four possible second platforms (Cori, K-computer, Trinity, Theta) whose parameters are given in Table 1.

Figure 3 presents the evolution of the overhead as a function of σ_1 varying from σ_2 to $5\sigma_2$, and using a checkpointing time of $60s$ (left), and $1800s$ (right). We observe that the model matches very well the results of the simulations: the maximum relative error is 5% with $C = 1800s$, and is within 0.2% with $C = 60s$. The latter result is expected: we do not account for failures during checkpoints t in the analysis, hence the approximation gets less accurate as checkpoint time increases.

For each value of σ_1 varying from σ_2 to $5\sigma_2$, we set β, γ and δ in Equation 1, according to the ratio $\frac{\sigma_1}{\sigma_2}$, which shows the accuracy of the formula in all three cases. Finally, we note that the overhead increases with larger speeds σ_1, but the expected throughput (time per unit of work) keeps decreasing.

Regarding on-failure checkpointing, we observe that the precision of the formula quickly degrades with larger σ_1, because it does not take into account failures that can occur during the re-execution work, which corresponds to the factor $\mu(\frac{\sigma_1-\sigma_2}{\sigma_1})$ in Equation 6. Note that this factor grows when σ_1 increases (or when σ_2 decreases), and it is not surprising to find that the overhead is always underestimated when the two speeds are quite different.

Next in Figure 4, we compare the simulated and theoretical overheads obtained with the first and second-order approximations. Note that the plot colors have a different meaning in this figure. The difference is small when using small checkpoint time (left), but when the two speeds get close and the checkpoint cost is high (right), the first-order approximation collapses and the theoretical overhead increases dramatically ($\mathbb{H}_{\text{opt}} = 0.5$). This is because the coefficient in $O(\lambda W)$ tends to 0, and the first-order approximation used to get W_{opt} is not valid anymore. However, we show that

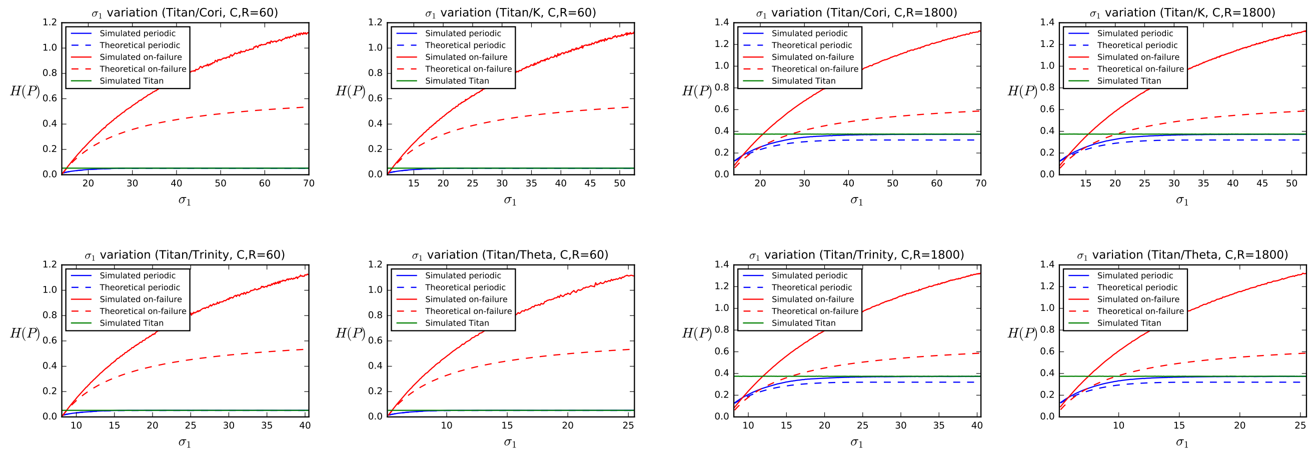

Figure 3: Evolution of overhead when σ_1 varies with $C = R = 60s$ on the left and $C = R = 1800s$ on the right.

using the second-order approximation, (i.e. considering additional terms in $O(\lambda^2 W^2)$) still yields good results ($\mathbb{H}_{\text{opt}} = 0.128$).

5.3 Comparison of the two strategies

In this section, we compare the overhead with the two strategies against that with a single platform. Coming back to Figure 3, we make two observations. First, when the ratio between σ_1 and σ_2 is large (greater than 2 with a small checkpoint C, somewhat higher when C increases), using a periodic pattern with replication is the same as using the fast platform only: the slow platform is not useful. Second, when this ratio between between σ_1 and σ_2 increases, the on-failure checkpointing strategy becomes worse than using the fast platform alone, especially with small checkpoint costs (left). This can be explained as follows: we wait for a failure on the slow platform to checkpoint the work done by the fast platform. But given the value of μ_2, the slow platform is struck less frequently than the fast one, hence we often lose a lot of work (remember we lose $\mu(\sigma_1 - \sigma_2)$ units of work when a failure strikes on P_1).

Figures 5 to 7 show the evolution of the overhead when parameters μ_1, μ_2 and C, R vary. Overall, we observe again that the work lost when a failure occurs on P_1 is important with the on-failure checkpointing strategy, whose overhead strongly depends upon on the second platform used. For instance, the overhead for $\mu_1 = 10,000s$ and $C = 60s$ goes from 0.236 (using Cori) to 1.81 (using Theta), whereas the overhead of the periodic checkpointing remains small (between 0.074 and 0.125). This observation is confirmed by Figure 6, where the overhead increases when the number of faults actually decreases on the slow platform!

We see the benefits of using replication when looking at Figure 5. When μ_1 becomes small ($10,000s$, or 8.6 failures per day), the overhead with a single platform (green) increases a lot, while the overhead with the periodic strategy (blue) increases only a little, even when the second platform is twice slower than the first one. For instance we have an overhead of 1.36 for P_1 alone when $C = 1800s$, whereas we get 0.894 when using P_1 in conjunction with Trinity, i.e. a reduction of 34%. However, when the second platform gets too slow, the improvement brought by the use of P_2 is only meaningful when the checkpointing cost is large: on Figure 7, we get 15% of improvement if $C \geq 10s$ with Cori, if $C \geq 760s$ with K, if $C \geq 4460s$ with Trinity, and more than $5000s$ with Theta.

Figure 8 presents the case of same-speed platforms. In this case, for all parameter choices (C, R, μ_1, μ_2), it is interesting to see that on-failure checkpointing is the best strategy, while it was less efficient than periodic checkpointing in almost all the other scenarios that we considered. This can be explained by the fact that there is no work lost at all with this strategy, except when there is a failure during a checkpoint.

5.4 Summary

We summarize simulation results as follows:

- The model is very accurate, as long as the resilience parameters remain reasonably small.
- On-failure checkpointing is generally less efficient than periodic checkpointing, except when the speeds of the two platforms are equal ($\sigma_2 = \sigma_1$).
- If P_2 is really too slow compared to P_1 ($\sigma_2 < \frac{\sigma_1}{2}$) or if the checkpointing cost is small, there is little reason to use a second platform.
- In all other cases ($\frac{\sigma_1}{2} \leq \sigma_2 < \sigma_1$), the periodic checkpointing strategy reduces the overhead by 30% in average, and up to 90% in some particular cases.

6 CONCLUSION

This work has addressed group replication for a black-box application executing on two heterogeneous platforms. We designed and thoroughly analyzed two strategies, periodic checkpointing and on-failure checkpointing. For periodic checkpointing, we have been able to analytically derive the best pattern length, using either first-order or second-order approximations. These results nicely extend the Young/Daly formula.

Simulations show that the model is quite accurate. As expected, when the platform speeds have different orders of magnitude, it is better to use only the fast platform. However, periodic checkpointing is useful for a wide range of speeds, and generally more

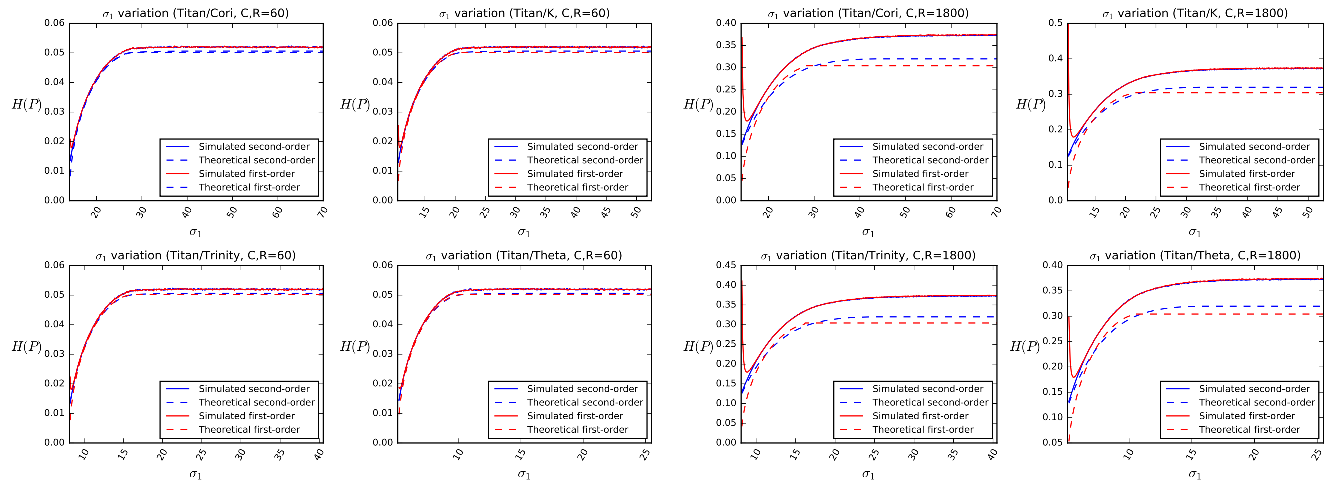

Figure 4: Comparison of overhead using first-order approximation and second-order approximation when σ_1 varies, with $C = R = 60s$ on the left and $C = R = 1800s$ on the right.

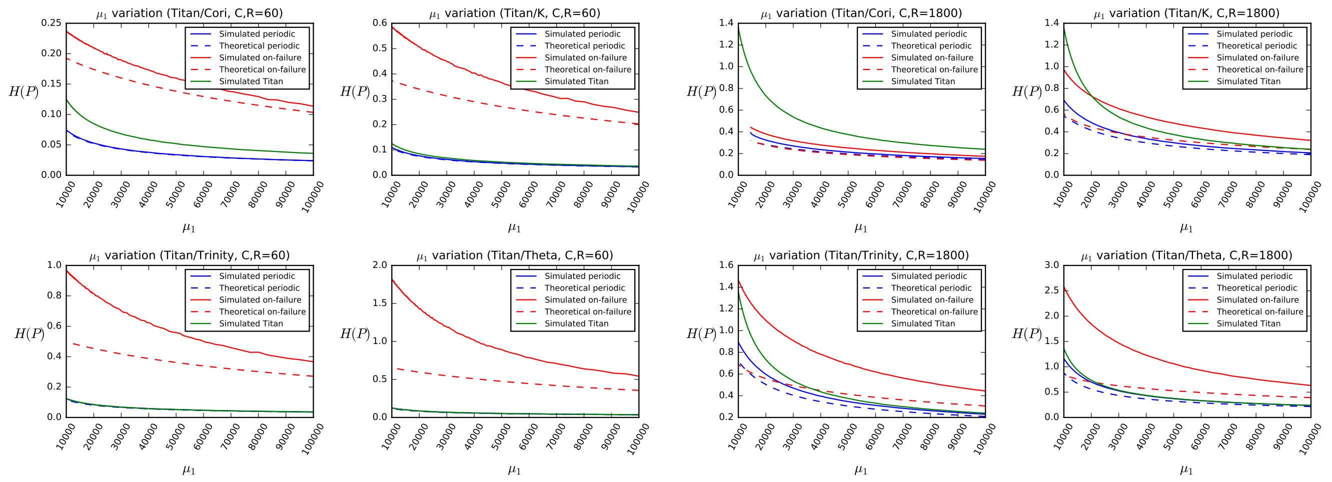

Figure 5: Evolution of overhead when μ_1 varies with $C = R = 60s$ on the left and $C = R = 1800s$ on the right.

efficient than on-failure checkpointing. The latter strategy is to be preferred only when the platform speeds are close.

Future work will be devoted to extending replication with heterogeneous platforms to deal with more complex applications, such as scientific workflows arranged as linear chains or fork-join graphs. Another interesting direction is to study the bi-criteria problem with energy consumption as a second metric, in addition to total execution time, in order to better assess the cost of replication.

REFERENCES

[1] 2016. Top500 Supercomputer Sites. (November 2016). https://www.top500.org/lists/2016/11/.
[2] Anne Benoit, Aurélien Cavelan, Valentin Le Fèvre, and Yves Robert. 2017. *Optimal checkpointing period with replicated execution on heterogeneous platforms.* Research report RR-9047. INRIA.
[3] Franck Cappello, Al Geist, William Gropp, Sanjay Kale, Bill Kramer, and Marc Snir. 2014. Toward Exascale Resilience: 2014 update. *Supercomputing frontiers and innovations* 1, 1 (2014).
[4] Henri Casanova, Marin Bougeret, Yves Robert, Frédéric Vivien, and Dounia Zaidouni. 2014. Using group replication for resilience on exascale systems. *Int. Journal of High Performance Computing Applications* 28, 2 (2014), 210–224.
[5] J. T. Daly. 2006. A higher order estimate of the optimum checkpoint interval for restart dumps. *Future Generation Comp. Syst.* 22, 3 (2006), 303–312.
[6] Sheng Di, Mohamed Slim Bouguerra, Leonardo Bautista-Gomez, and Franck Cappello. 2014. Optimization of multi-level checkpoint model for large scale HPC applications. In *IPDPS*. IEEE.
[7] Sheng Di, Yves Robert, Frédéric Vivien, and Franck Cappello. 2017. Toward an optimal online checkpoint solution under a two-level HPC checkpoint model. *IEEE Trans. Parallel Distributed Systems* 28, 1 (2017).
[8] C. Engelmann, H. H. Ong, and S. L. Scorr. 2009. The case for modular redundancy in large-scale highh performance computing systems. In *PDCN*. IASTED.
[9] K. Ferreira, J. Stearley, J. H. III Laros, R. Oldfield, K. Pedretti, R. Brightwell, R. Riesen, P. G. Bridges, and D. Arnold. 2011. Evaluating the Viability of Process Replication Reliability for Exascale Systems. In *SC'11*. ACM.
[10] Thomas Hérault and Yves Robert (Eds.). 2015. *Fault-Tolerance Techniques for High-Performance Computing*. Springer Verlag.
[11] Troy Leblanc, Rakhi Anand, Edgar Gabriel, and Jaspal Subhlok. 2009. VolpexMPI: An MPI Library for Execution of Parallel Applications on Volatile Nodes. In *16th European PVM/MPI Users' Group Meeting*. Springer-Verlag, 124–133.

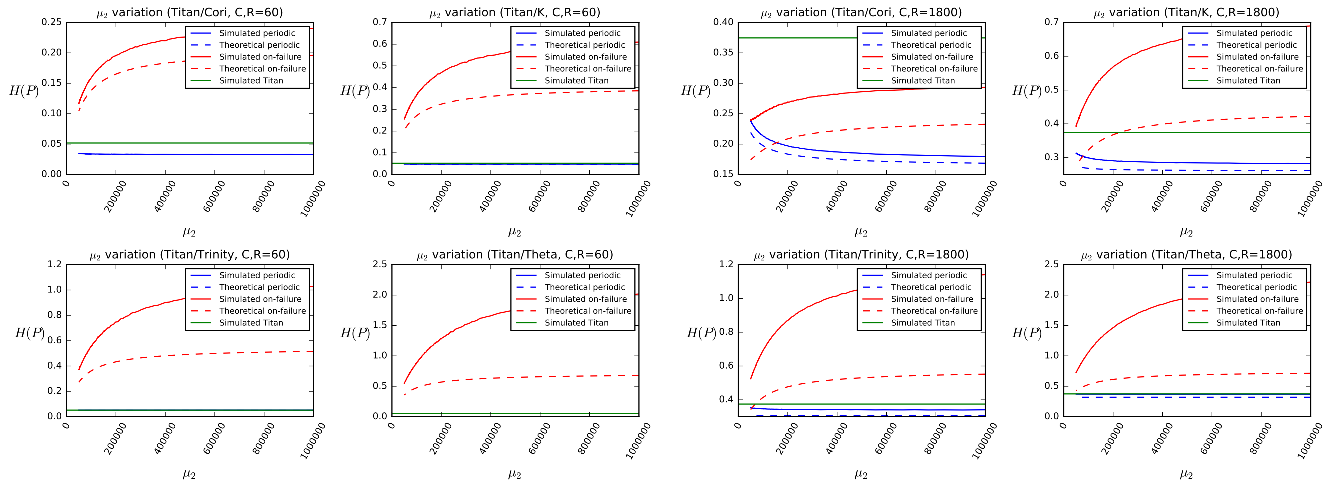

Figure 6: Evolution of overhead when μ_2 varies with $C = R = 60s$ on the left and $C = R = 1800s$ on the right.

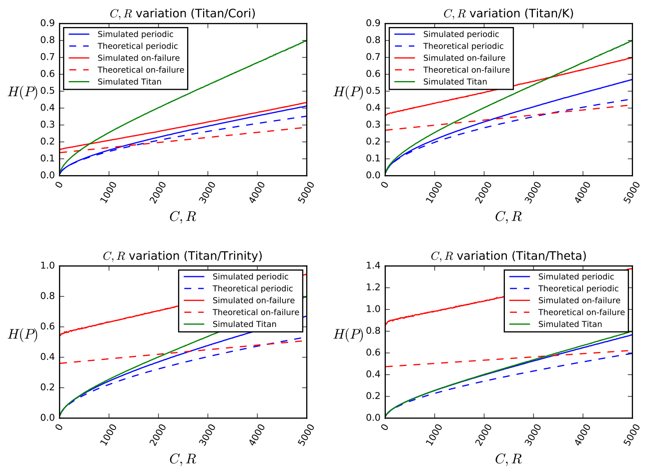

Figure 7: Evolution of overhead when C and R vary.

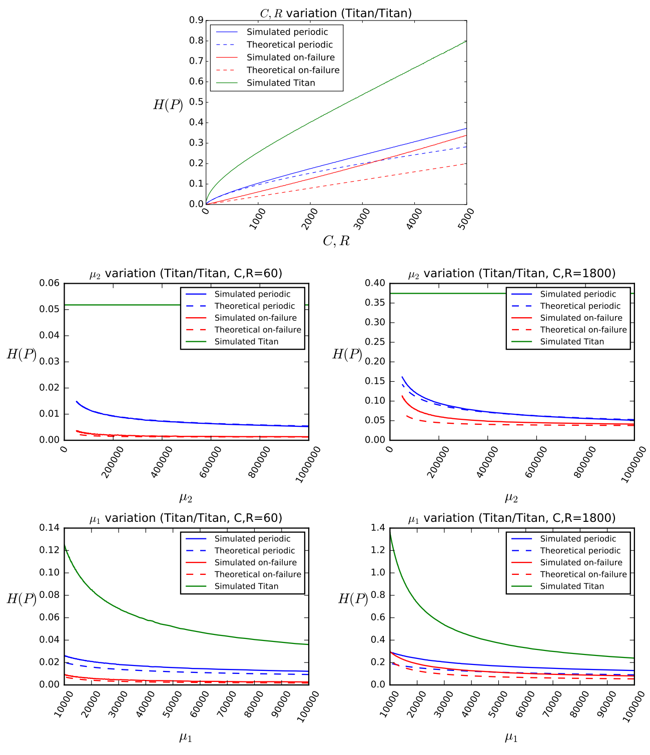

Figure 8: Evolution of overhead when parameters vary, using two same-speed platforms.

[12] B. Mills, T. Znati, and R. Melhem. 2014. Shadow Computing: An energy-aware fault tolerant computing model. In *Int. Conf. on Computing, Networking and Communications (ICNC)*. IEEE, 73–77.

[13] Michael Mitzenmacher and Eli Upfal. 2005. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press.

[14] Adam Moody, Greg Bronevetsky, Kathryn Mohror, and Bronis R. de Supinski. 2010. Design, Modeling, and Evaluation of a Scalable Multi-level Checkpointing System. In *SC*. ACM.

[15] Francesco Quaglia. 2001. A Cost Model for Selecting Checkpoint Positions in Time Warp Parallel Simulation. *IEEE Trans. Parallel Dist. Syst.* 12, 4 (2001), 346–362.

[16] B. Schroeder and G. A. Gibson. 2007. Understanding Failures in Petascale Computers. *Journal of Physics: Conference Series* 78, 1 (2007).

[17] M. Snir and et al. 2014. Addressing Failures in Exascale Computing. *Int. J. High Perform. Comput. Appl.* 28, 2 (2014), 129–173.

[18] S. Yi, D. Kondo, B. Kim, G. Park, and Y. Cho. 2010. Using Replication and Checkpointing for Reliable Task Management in Computational Grids. In *SC'10*. ACM.

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

[20] Z. Zheng and Z. Lan. 2009. Reliability-aware scalability models for high performance computing. In *Cluster Computing*. IEEE.