Energy-aware checkpointing strategies

Guillaume Aupy, Anne Benoit, Mohammed El Mehdi Diouri, Olivier Glück, Laurent Lefèvre

Chapter 1
Energy-aware checkpointing strategies

Guillaume Aupy, Anne Benoit, Mohammed El Mehdi Diouri, Olivier Glück and Laurent Lefèvre

Abstract Future extreme-scale supercomputers will gather hundreds of million cores. The main problem that we address is energy consumption since such systems will consume enormous amount of energy. Besides that, we also need to overcome important challenges related to fault tolerance in such extreme-scale systems. Fault-tolerance protocols have different energy consumption depending on parameters like the platform characteristics, the application features and the number of processes used in the execution. Currently, in order to evaluate the power consumption of fault tolerant protocols in an given execution context, the only approach is to run the application with the different versions of fault tolerant protocols and monitor the energy consumption. In order to avoid this time and energy consuming process, we propose in this chapter a methodology in order to estimate the energy consumption of the fault-tolerance protocols used in High-Performance Computing applications. Our methodology relies on an energy calibration of the supercomputer and a user description of the execution setting. We evaluate the accuracy of the estimations with applications and scenarios executed on a real platform with energy consumption monitoring. Results show that the energy estimations that we are able to provide before the executions are highly accurate and allow the users to select the less energy consuming fault-tolerance protocol without pre-running the application.

1.1 Introduction

For decades, the computer science research community exclusively focused on performance, which resulted in highly powerful, but in turn, low efficient systems with a very high total cost of ownership (TCO) [28]. A significant research effort is focusing on the characteristics, features, and challenges of High Performance Computing (HPC) systems capable of reaching the Exaflop performance mark [18, 42]. The portrayed Exascale systems will necessitate billion way parallelism, resulting not only 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. Even if each independent component is quite reliable, the Mean Time Between Failures (MTBF) is expected to drop drastically. Executions of large parallel applications on these systems will have to tolerate a higher degree of errors and failures than in current systems. The de-facto general-purpose error recovery technique in high performance computing is checkpoint and rollback recovery. Such protocols employ checkpoints to periodically save the state of a parallel application, so that when an error strikes some process, the application can be restored into one of its former states. The most widely used protocol is coordinated checkpointing, where all processes periodically stop computing and synchronize to write critical application data onto stable storage. Coordinated checkpointing is well understood, at least in its blocking form (when no computing activity takes place during checkpoints), and good approximations of the optimal checkpoint interval exist; they are known as Young's and Daly's formula [44, 8]. While the future Exascale applications are not yet designed and developed, it is anticipated, from the current knowledge and observations of existing large systems, that fault tolerance is unavoidable at the post-Petascale era, since Exascale systems will experience various kind of faults many times per day [6].

While reliability is a major concern for Exascale, another key challenge is to minimize energy consumption, both for economic and environmental reasons. The HPC community has recently acknowledged that the energy efficiency of HPC systems is a major concern in designing future Exascale systems for the end of the decade [30, 22]. One of the most power-consuming components of today's systems is the processor: even when idle, it dissipates a significant fraction of the total power. However, for future Exascale systems, the power dissipated to execute I/O transfers is likely to play an even more important role, because the relative cost of communication is expected to dramatically increase, both in terms of latency and consumed energy [43]. An Exascale supercomputer will gather several millions of CPU cores running up to a billion trends to achieve a performance of 10^{18} FLoat Operations Per Second, and it will consume several megawatts. The energy consumption issue at the Exascale becomes even more worrying when we know that we already reach power consumptions higher than 17 MW at the Petascale while the DARPA set to 20 MW the threshold for Exascale supercomputers [42]. Hence, reducing the energy consumption of high-performance computing infrastructures is a major challenge for the next years in order to be able to move to the Exascale era. Nowadays, there exists a strong research effort towards energy-efficient supercomputers. Hardware provides part of the solution by exposing unceasingly more energy-efficient devices which also provide abilities that current operating systems can successfully leverage to save energy [31]. Mechanisms such as Dynamic Voltage Scaling (DVFS) or P-state management have also been used to develop power-aware user-level software [31, 38, 37].

Hence, dealing with errors and minimizing the energy consumption are two main challenges that should be addressed. However, fault tolerance and energy consump-

tion are interrelated: fault tolerance consumes energy and some energy reduction techniques can increase error and failure rates [20].

Very few papers consider the general problem of the interplay between energy consumption and fault-tolerance. Aupy et al. [1] discuss energy-aware checkpointing strategies for divisible tasks, using DVFS to reduce the energy consumption. Given a workload, they show how to decide how many chunks to use, what are the sizes of these chunks, and at which speed each chunk is executed. Tackling with HPC platforms, Diouri et al. [14] present the energy consumption of the three most important parts of fault-tolerance: message-logging, checkpointing and task coordination. Their first result is that task coordination is the most energy consuming part of fault-tolerance protocols. They also show that while it involves more power to store data on RAM, HDD logging is more energy consuming than RAM logging because of the logging duration. In a second paper, Diouri et al. [16] extend these results into a framework that predicts the energy consumption of a fault-tolerance protocol, allowing the user to choose amongst three fault-tolerance protocols: coordinated, uncoordinated, and hierarchical, depending on the application running on the platform. We detail the coordinated and uncoordinated protocols below. Finally, Meneses et al. [33] study the energy consumption of the coordinated periodic checkpointing protocol as a function of $\mathscr{P}_{\text{Static}}$ (the base power consumed when the platform is switched on) and $\mathscr{P}_{\text{Cal}}$ (the CPU overhead when the platform is active).

We identify two classes of fault-tolerance protocols: coordinated and uncoordinated protocols. Both coordinated and uncoordinated protocols rely on checkpointing regularly (each checkpoint interval) the global state of the application in order to restart it in case of failure from the last checkpoint instead of re-executing the whole application. The problem of checkpointing is to ensure a global coherent state of the system. A global state is considered as coherent if it does not contain messages that are received but that were not sent. Coordinated protocols (already discussed above) are currently the most used fault-tolerance protocols in high performance computing applications. In order to ensure the global coherent state of the system, the coordinated protocol relies on a coordination that consists of synchronizing all the processes before checkpointing [34]. Coordination may result in a huge waste in terms of performance. Indeed in order to synchronize all the processes, it is necessary to wait for all the inflight messages to be transmitted and received. Moreover, in case of failure with the coordinated protocol, all the processes have to be restarted from the last checkpoint even if a single process has crashed. This results in a huge waste in terms of energy consumption since all the processes even the non-crashed ones have to redo all the computations and the communications from the last checkpoint. The uncoordinated protocol with message logging addresses this issue by restarting only the failed processes. Thus, the power consumption in recovery is supposed to be much smaller than for coordinated checkpointing. However, in order to ensure a global coherent state of the system, all message logging protocols need to log all messages sent by all processes during the whole execution and this impacts the performance [3]. Hence, in case of failure, the non-crashed processes send to the crashed ones the messages that they have logged.

Our first main contribution in this chapter is to determine the optimal checkpointing interval in terms of energy consumption, for coordinated checkpointing. Section 1.2 presents a detailed analysis to compute this optimal checkpointing interval, considering two distinct objectives: minimizing the execution time and the energy consumption. Then, according to the determined checkpointing interval, the goal of the second main contribution is to allow supercomputer users to choose between the coordinated and the uncoordinated protocols before pre-executing the HPC application in a given execution context. To this end, we rely on a methodology that estimates the energy consumption of fault-tolerance protocols relying on a energy calibration of the execution platform and a description of execution parameters. Section 1.3 presents this methodology and shows how it enables supercomputer users to select the less energy consuming fault-tolerance protocol without pre-running the application. We conclude the chapter in Section 1.4.

1.2 Optimal checkpointing period: time vs. energy

This section deals with parallel scientific applications using non-blocking and periodic coordinated checkpointing to enforce resilience. We provide a model and detailed formulas for total execution time and consumed energy. We characterize the optimal period for both objectives, and we assess the range of time/energy trade-offs to be made by instantiating the model with a set of realistic scenarios for Exascale systems. We give a particular emphasis to I/O transfers, because the relative cost of communication is expected to dramatically increase, both in terms of latency and consumed energy, for future Exascale platforms.

In this section, we investigate trade-offs between execution time and energy consumption for the execution of parallel applications on future Exascale systems. The optimal period $\mathscr{T}_{\text{Time}}^{\text{opt}}$ given by Young's and Daly's formula [44, 8] will minimize (expected) execution time. However, this period $\mathscr{T}_{\text{Time}}^{\text{opt}}$ will not minimize energy consumption, mainly because the fraction of power $\mathscr{P}_{\text{Cal}}$ spent when computing (by the CPUs) is not the same as the fraction of power $\mathscr{P}_{\text{I/O}}$ spent when checkpointing. In particular, we revisit the work of Meneses, Sarood and Kalé [33] for checkpoint/restart, where formulas are given to compute the time-optimum and energy-optimum periods. However, our model is more precise: (i) we carefully assess the impact of the power consumption required for I/O activity, which is likely to play a key role at the Exascale; (ii) we consider non-blocking checkpointing that can be partially overlapped with computations; (iii) we give a more accurate analysis of the consumed energy.

Altogether, this section provides the following main contributions:

- We provide a refined analytical model to compute both the execution time and the consumed energy with a given checkpoint period. The model handles the case where checkpointing activity can be non-blocking, i.e., partially overlapped with computations.

- We provide analytical formulas to approximate the optimal period for time $\mathscr{T}_{\text{Time}}^{\text{opt}}$ as well as the optimal period for energy $\mathscr{T}_{\text{Energy}}^{\text{opt}}$, thereby refining and extending Daly [8] and Meneses, Sarood and Kalé [33] results to non-blocking checkpoints.
- We assess the range of time/energy trade-offs to be made by instantiating the model with a set of realistic scenarios for Exascale systems.

1.2.1 Model

In this section, we introduce all the model parameters. We start with parameters related to resilience (checkpointing) before moving to parameters related to energy consumption.

1.2.1.1 Checkpointing

We model coordinated checkpointing [7] where checkpoints are taken at regular intervals, after some fixed amount of work units have been performed. This corresponds to an execution partitioned into periods of duration T. Every period, a checkpoint of length C is taken.

An important question is whether checkpoints are blocking or not. On some architectures, we may have to stop executing the application before writing to the stable storage where the checkpoint data is saved; in that case checkpoint is fully blocking. On other architectures, checkpoint data can be saved on the fly into a local memory before the checkpoint is sent to the stable storage, 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 ω: during a checkpoint of duration C, the work that is performed is ωC work units. In other words, $(1-\omega)C$ work units are wasted due to checkpoint jitter disrupting the progress of computation. Here, $0 \leq \omega \leq 1$ is an arbitrary parameter. The case $\omega = 0$ corresponds to a fully blocking checkpoint, while $\omega = 1$ corresponds to a checkpoint totally overlapped with computations. All intermediate situations can be represented.

Next we have to account for failures. During t time units of execution, the expectation of the number of failures is $\frac{t}{\mu}$, where μ is the MTBF (Mean Time Between Failures) of the platform. Note that if the platform if made of N identical resources whose individual mean time between failures is μ_{ind}, then $\mu = \frac{\mu_{\text{ind}}}{N}$. This relation is agnostic of the granularity of the resources, which can be anything from a single CPU to a complex multi-core socket. When a failure strikes, there is a downtime of length D (time to reboot the resource or set up a spare), and then a recovery of length R (time to read the last stored checkpoint). The work executed by the application since the last checkpoint and before the failure needs to be re-executed. Clearly, the shorter the period T, the less work to re-execute, but also the more overhead due

to frequent checkpoints in a failure-free execution. The best trade-off when $\omega = 0$ (blocking checkpoint) is achieved for $T = \sqrt{2C\mu} + C$ (Young's formula [44]) or $T = \sqrt{2C(\mu + D + R)} + C$ (Daly's formula [8]). Both formulas are first-order approximations and valid only if all checkpoint parameters C, D and R are small in front of μ (and these formulas collapse if they become negligible). In Section 1.2.2, we show how to extend these formulas to the case of non-blocking checkpoints (see also [2] for more details).

1.2.1.2 Energy

To compute the energy consumption of the application, we need to consider the energy consumption of the different phases, and hence the power consumption at each time-step. To this purpose, we define:

- $\mathscr{P}_{\textbf{Static}}$**:** this is the base power consumed when the platform is switched on.
- $\mathscr{P}_{\textbf{Cal}}$**:** when the platform is active, we have to consider the CPU overhead in addition to the static power $\mathscr{P}_{\text{Static}}$.
- $\mathscr{P}_{\textbf{I/O}}$**:** similarly, this is the power overhead due to file I/O. This supplementary power consumption is induced by checkpointing, or when recovering from a failure.
- $\mathscr{P}_{\textbf{Down}}$**:** for coordinated checkpointing, when one processor fails, the rest of the machine stays idle. $\mathscr{P}_{\text{Down}}$ is the power consumption overhead when one machine is down, that may be incurred for instance by rebooting the machine. In general, we let $\mathscr{P}_{\text{Down}} = 0$.

Meneses, Sarood and Kalé [33] have a simpler model with two parameters, namely L, the base power (corresponding to $\mathscr{P}_{\text{Static}}$ with our notations), and H, the maximum power (corresponding to $\mathscr{P}_{\text{Static}} + \mathscr{P}_{\text{Cal}}$ with our notations). They use $\mathscr{P}_{\text{I/O}} = \mathscr{P}_{\text{Down}} = 0$.

In Section 1.2.2, we show how to compute the optimal period that minimizes the energy consumption. In Section 1.2.3, we instantiate the model with expected values for power consumption of Exascale platforms.

1.2.2 Optimal checkpointing period

We consider a parallel application whose execution time is $\mathscr{T}_{\text{base}}$ without any overhead due to the resilience method or the occurrence of failures. We compute the expectation $\mathscr{T}_{\text{final}}$ of the total execution time (accounting both for checkpointing and for failures) in Section 1.2.2.1, and the expectation $\mathscr{E}_{\text{final}}$ of the total energy consumed during this execution of length $\mathscr{T}_{\text{final}}$ in Section 1.2.2.2. We will compute the optimal period T that minimizes the objective, either $\mathscr{T}_{\text{final}}$ or $\mathscr{E}_{\text{final}}$.

1.2.2.1 Execution time

The total execution time $\mathscr{T}_{\text{final}}$ of the application depends on two sources of overhead. We first compute $\mathscr{T}_{\text{ff}}$, the time taken by a fault-free execution, thereby accounting only for the overhead due to periodic checkpointing. Then we compute $\mathscr{T}_{\text{fails}}$, the time lost due to failures. Finally, $\mathscr{T}_{\text{final}} = \mathscr{T}_{\text{ff}} + \mathscr{T}_{\text{fails}}$. We detail here both computations:

- The reasoning to derive $\mathscr{T}_{\text{ff}}$ is simple. We need to execute a total amount of work equal to $\mathscr{T}_{\text{base}}$. During each period of length T, there is an amount of time $T-C$ where only computations take place, and an amount of time C of checkpointing, where only a work ωC is done. Therefore, the total number of work units executed during a period of length T is $T-C+\omega C = T-(1-\omega)C$, and
$$\mathscr{T}_{\text{ff}} = \mathscr{T}_{\text{base}} \frac{T}{T-(1-\omega)C}.$$
- The reasoning to compute $\mathscr{T}_{\text{fails}}$ is the following. Since the mean time between two failures is μ, the average number of failures during execution is $\frac{\mathscr{T}_{\text{final}}}{\mu}$. For each failure, the time lost is expressed as:
  - $D+R$ for downtime and recovery;
  - a time ωC for the work that was done during the previous checkpoint and that has to be redone because it was not checkpointed (because of the failure);
  - with probability $\frac{T-C}{T}$, the failure happens while we are not checkpointing, and the time lost is on average $A = \frac{T-C}{2}$;
  - otherwise, with probability $\frac{C}{T}$, the failure happens while we are checkpointing, and the time lost is on average $B = T-C+\frac{C}{2} = T-\frac{C}{2}$.

  The time lost for each failure is
$$D+R+\omega C+\frac{T-C}{T}A+\frac{C}{T}B = D+R+\omega C+\frac{T}{2}.$$
  Finally,
$$\mathscr{T}_{\text{fails}} = \frac{\mathscr{T}_{\text{final}}}{\mu}\left(D+R+\omega C+\frac{T}{2}\right).$$

We are now ready to express the total execution time:

$$\begin{aligned}\mathscr{T}_{\text{final}} &= \mathscr{T}_{\text{ff}} + \mathscr{T}_{\text{fails}} \\ &= \mathscr{T}_{\text{base}} \frac{T}{T-(1-\omega)C} + \frac{\mathscr{T}_{\text{final}}}{\mu}\left(D+R+\omega C+\frac{T}{2}\right) \\ &= \frac{T}{(T-(1-\omega)C)\left(1-\frac{D+R+\omega C+T/2}{\mu}\right)}\mathscr{T}_{\text{base}} \\ &= \frac{T}{(T-a)\left(b-\frac{T}{2\mu}\right)}\mathscr{T}_{\text{base}},\end{aligned}$$

where $a = (1-\omega)C$ and $b = 1-\frac{D+R+\omega C}{\mu}$.

This equation is minimized for

$$\mathscr{T}_{\text{Time}}^{\text{opt}} = \sqrt{2(1-\omega)C(\mu-(D+R+\omega C))}. \tag{1.1}$$

When $\omega = 0$, we obtain an expression close to that of Young and Daly, but slightly different because they have less accurately approximated the total execution time. In the following, we let ALGOT be the checkpointing strategy that checkpoints with period $\mathscr{T}_{\text{Time}}^{\text{opt}}$.

1.2.2.2 Energy consumption

In order to compute the total energy consumption of the execution, we consider the different phases during which the different powers introduced in Section 1.2.1.2 are used:

- First, we consume $\mathscr{P}_{\text{Static}}$ during each time-step of the execution. Indeed, even when a node fails and is shutdown, we still pay for the power of all the other nodes, for the cooling system, etc. The corresponding energy cost is $\mathscr{T}_{\text{final}}\mathscr{P}_{\text{Static}}$.
- Next, let $\mathscr{T}_{\text{Cal}}$ be the time during which the CPU is used, inducing a power overhead $\mathscr{P}_{\text{Cal}}$. $\mathscr{T}_{\text{Cal}}$ includes the base work $\mathscr{T}_{\text{base}}$, and $\mathscr{T}_{\text{re-exec}}$, the work that must be re-executed after each failure (which we multiply by the number of failures $\mathscr{T}_{\text{final}}/\mu$):
  - with probability $\frac{T-C}{T}$, the failure does not happen during a checkpoint, and the work to re-execute is $A = \omega C + \frac{T-C}{2}$;
  - with probability $\frac{C}{T}$, the failure happens during the execution of a checkpoint, and the work to re-execute is $B = \omega C + T - C + \frac{\omega C}{2}$.

  We derive $\mathscr{T}_{\text{re-exec}} = \frac{T-C}{T}A + \frac{C}{T}B$, hence

  $$\mathscr{T}_{\text{re-exec}} = \omega C + \frac{T^2-C^2}{2T} + \frac{\omega C^2}{2T}.$$

  Finally, we have:

$$\mathscr{T}_{\text{Cal}} = \mathscr{T}_{\text{base}} + \frac{\mathscr{T}_{\text{final}}}{\mu}\left(\omega C + \frac{T^2 - C^2}{2T} + \frac{\omega C^2}{2T}\right).$$

The corresponding energy consumption is $\mathscr{T}_{\text{Cal}}\mathscr{P}_{\text{Cal}}$.

- Let $\mathscr{T}_{\text{I/O}}$ be the time during which the I/O system is used, inducing a power overhead $\mathscr{P}_{\text{I/O}}$. This time corresponds to checkpointing and recovery from failures.
  - The total number of checkpoints that are taken in a fault-free execution is equal to the number of periods, $\frac{\mathscr{T}_{\text{base}}}{T-(1-\omega)C}$, and the time taken by checkpoints is therefore $\frac{\mathscr{T}_{\text{base}}C}{T-(1-\omega)C}$.
  - For each failure, there is an additional overhead:
    1. the system needs to recover, which lasts R time-steps;
    2. with probability $\frac{T-C}{T}$, the failure does not happen during a checkpoint, and there is no additional I/O overhead;
    3. however, with probability $\frac{C}{T}$, the failure happens during a checkpoint, and the I/O time wasted is (in average) $\frac{C}{2}$.

  Altogether, we obtain

$$\mathscr{T}_{\text{I/O}} = \frac{\mathscr{T}_{\text{base}}C}{T-(1-\omega)C} + \frac{\mathscr{T}_{\text{final}}}{\mu}\left(R + \frac{C^2}{2T}\right).$$

  The corresponding energy consumption is $\mathscr{T}_{\text{I/O}}\mathscr{P}_{\text{I/O}}$.
- Finally, let $\mathscr{T}_{\text{Down}}$ be the total down time, incurring a power overhead $\mathscr{P}_{\text{Down}}$. We have

$$\mathscr{T}_{\text{Down}} = \frac{\mathscr{T}_{\text{final}}}{\mu}D,$$

  and the corresponding energy cost is $\mathscr{T}_{\text{Down}}\mathscr{P}_{\text{Down}}$. This term is only included for full generality, as we expect to have $\mathscr{P}_{\text{Down}} = 0$ in most scenarios.

The final expression for the total energy consumed is

$$\begin{aligned}
\mathscr{E}_{\text{final}} &= \mathscr{T}_{\text{Cal}}\mathscr{P}_{\text{Cal}} + \mathscr{T}_{\text{I/O}}\mathscr{P}_{\text{I/O}} + \mathscr{T}_{\text{Down}}\mathscr{P}_{\text{Down}} + \mathscr{T}_{\text{final}}\mathscr{P}_{\text{Static}} \\
&= \left(\mathscr{T}_{\text{base}} + \frac{\mathscr{T}_{\text{final}}}{\mu}\left(\omega C + \frac{T^2 - C^2}{2T} + \frac{\omega C^2}{2T}\right)\right)\mathscr{P}_{\text{Cal}} \\
&\quad + \left(\frac{\mathscr{T}_{\text{final}}}{\mu}\left(R + \frac{C^2}{2T}\right) + C\frac{\mathscr{T}_{\text{base}}}{T-(1-\omega)C}\right)\mathscr{P}_{\text{I/O}} \\
&\quad + \frac{\mathscr{T}_{\text{final}}}{\mu}D\mathscr{P}_{\text{Down}} + \mathscr{T}_{\text{final}}\mathscr{P}_{\text{Static}}.
\end{aligned}$$

It is important to understand that $\mathscr{T}_{\text{final}} \neq \mathscr{T}_{\text{Cal}} + \mathscr{T}_{\text{I/O}} + \mathscr{T}_{\text{Down}}$, unless $\omega = 0$. Indeed, CPU and I/O activities are overlapped (and both consumed) when checkpointing. To ease the derivation of the optimal period that minimizes $\mathscr{E}_{\text{final}}$, we introduce some notations and let $\mathscr{P}_{\text{Cal}} = \alpha\mathscr{P}_{\text{Static}}$, $\mathscr{P}_{\text{I/O}} = \beta\mathscr{P}_{\text{Static}}$, and $\mathscr{P}_{\text{Down}} = \gamma\mathscr{P}_{\text{Static}}$.

Re-using parameters $a = (1-\omega)C$ and $b = 1 - \frac{D+R+\omega C}{\mu}$ from Section 1.2.2.1, we obtain:

$$\frac{\mathscr{T}'_{\text{final}}}{\mathscr{T}_{\text{base}}} = \frac{-ab + \frac{T^2}{2\mu}}{(T-a)^2\left(b - \frac{T}{2\mu}\right)^2}, \quad \text{and}$$

$$\begin{aligned}\frac{\mathscr{E}'_{\text{final}}}{\mathscr{P}_{\text{Static}}} &= \frac{\mathscr{T}'_{\text{final}}}{\mu}\left(\alpha\omega C + \beta R + \gamma D + \frac{\alpha T}{2} - \frac{\alpha(1-\omega)C^2}{2T} + \frac{\beta C^2}{2T} + \mu\right) \\ &\quad + \frac{\mathscr{T}_{\text{final}}}{2\mu}\left(\alpha + \frac{\alpha(1-\omega)C^2}{T^2} - \frac{\beta C^2}{T^2}\right) - \frac{\beta C \mathscr{T}_{\text{base}}}{(T-(1-\omega)C)^2}.\end{aligned}$$

Then, letting $K = \frac{(T-a)^2\left(b-\frac{T}{2\mu}\right)^2}{\mathscr{P}_{\text{Static}}\mathscr{T}_{\text{base}}}$, we have:

$$\begin{aligned}K\mathscr{E}'_{\text{final}} &= \frac{-ab+\frac{T^2}{2\mu}}{\mu}\left((\alpha\omega C + \beta R + \gamma D + \mu) + \frac{\alpha T}{2} + \frac{\alpha(1-\omega)C^2}{2T} + \frac{\beta C^2}{2T}\right) \\ &\quad + \frac{(T-a)(b-\frac{T}{2\mu})}{2\mu}\left(\alpha + \frac{\alpha(1-\omega)C^2 - \beta C^2}{T}\right) - \beta C\left(b - \frac{T}{2\mu}\right)^2 \\ &= T^3\left(\frac{1}{4\mu} - \frac{1}{4\mu}\right) + T^2\left(\frac{\alpha\omega C + \beta R + \gamma D}{2\mu^2} + \frac{b+\frac{a}{2\mu}}{2\mu} - \frac{\beta C}{4\mu^2} + \frac{1}{2\mu}\right) \\ &\quad + T\left(-\frac{ab}{2\mu} - \frac{ab}{2\mu} + \frac{\beta C b}{\mu} - 2\frac{(\alpha(1-\omega)-\beta)C^2}{4\mu^2}\right) - \beta C b^2 \\ &\quad - \frac{ab(\alpha\omega C + \beta R + \gamma D + \mu)}{\mu} - \left(\frac{b}{2\mu} - \frac{a}{4\mu^2}\right)(\alpha(1-\omega)-\beta)C^2 \\ &\quad + \frac{1}{T}\left((\alpha(1-\omega)-\beta)\frac{C}{2\mu} - (\alpha(1-\omega)-\beta)\frac{C}{2\mu}\right) \\ &= T^2\left(\frac{\alpha\omega C + \beta R + \gamma D}{2\mu^2} + \frac{b}{2\mu} + \frac{a-\beta C}{4\mu^2} + \frac{1}{2\mu}\right) \\ &\quad + T\left(\frac{(\beta C - a)b}{\mu} - 2\frac{(\alpha(1-\omega)-\beta)C^2}{4\mu^2}\right) \\ &\quad - \frac{ab(\alpha\omega C + \beta R + \gamma D + \mu)}{\mu} - \beta C b^2 \\ &\quad + \left(\frac{b}{2\mu} + \frac{a}{4\mu^2}\right)(\alpha(1-\omega)-\beta)C^2\,.\end{aligned}$$

Let $\mathscr{T}^{\text{opt}}_{\text{Energy}}$ be the only positive root of this quadratic polynomial in T: $\mathscr{T}^{\text{opt}}_{\text{Energy}}$ is the value that minimizes $\mathscr{E}_{\text{final}}$. In the following, we let ALGOE be the checkpointing strategy that checkpoints with period $\mathscr{T}^{\text{opt}}_{\text{Energy}}$.

As a side note, let us emphasize the differences with the approach of Meneses, Sarood and Kalé [33] when restricting to the case $\omega = 0$ (because they only consider the blocking variant). For each failure, they consider that:

- energy lost due to re-execution is $\frac{T-2C}{2}\mathscr{P}_{\text{Cal}}$, while we have $\left(\frac{T-C}{T}\left(\frac{T-C}{2}\right) + \frac{C}{T}(T-C)\right)\mathscr{P}_{\text{Cal}} = \frac{T^2-C^2}{2T}\mathscr{P}_{\text{Cal}}$;
- energy lost due to I/O is $C\mathscr{P}_{\text{I/O}}$, while we have $\frac{C^2}{2T}\mathscr{P}_{\text{I/O}}$.

Theses differences come from our more detailed analysis of the impact of the failure location, which can strike either during the computation phase, or during the checkpointing phase, of the whole period.

1.2.3 Experiments

In this section, we instantiate the previous model with scenarios taken from current projections for Exascale platforms [18, 42, 43, 23]. We choose realistic values for all model parameters: this includes all types of power consumption ($\mathscr{P}_{\text{Static}}$, $\mathscr{P}_{\text{Cal}}$, $\mathscr{P}_{\text{I/O}}$ and $\mathscr{P}_{\text{Down}}$), all checkpoint parameters (C, R, D and ω), and the platform MTBF μ. We start with a word of caution: our choices for these parameters may be somewhat arbitrary, and do not cover the whole range of scenarios that can be investigated. However, a key feature of our model is its robustness: as long as μ is reasonably large in front of checkpoint times, the model is able to accurately predict the best period for execution time and for energy consumption.

The power consumption of an Exascale machine is capped to 20 Mega-watts. With 10^6 nodes, this represents a nominal power of 20 watts per node. Let us express all power values in watts. A reasonable scenario is to assume that half this power is used for operating the platform, hence to let $\mathscr{P}_{\text{Static}} = 10$. The overhead due to computing would represent the other half, hence $\mathscr{P}_{\text{Cal}} = 10$. As for communications and I/Os, which are expected to cost an order of magnitude more than computing [43], we take an overhead of 100, hence $\mathscr{P}_{\text{I/O}} = 100$. A key parameter for the experimental study is the ratio

$$\rho = \frac{\mathscr{P}_{\text{Static}} + \mathscr{P}_{\text{I/O}}}{\mathscr{P}_{\text{Static}} + \mathscr{P}_{\text{Cal}}} = \frac{1+\beta}{1+\alpha}. \tag{1.2}$$

With our values, we get $\rho = 5.5$. Note that if we used $\mathscr{P}_{\text{Static}} = 5$ and kept the same overheads 10 and 100 for computing and I/O respectively, we would get $\mathscr{P}_{\text{Cal}} = 10$, $\mathscr{P}_{\text{I/O}} = 100$, and $\rho = 7$. These two representative values of ρ ($\rho = 5.5$ and $\rho = 7$) are emphasized by vertical arrows in the plots below on Figure 1.1. As for $\mathscr{P}_{\text{Down}}$, the power during downtime, we use $\mathscr{P}_{\text{Down}} = 0$, meaning that during downtime we only account for the static power $\mathscr{P}_{\text{Static}}$ of the processors that are idle.

The Jaguar platform, with $N = 45,208$ processors, is reported to have experienced about one fault per day [46], which leads to an individual (processor) MTBF μ_{ind} equal to $\frac{45,208}{365} \approx 125$ years. Therefore, we set the individual (processor) MTBF to $\mu_{\text{ind}} = 125$ years. Letting the total number of processors N vary from $N = 219,150$ to $N = 2,191,500$ (future Exascale platforms), the platform MTBF μ varies from $\mu = 300$ min (5 hours) down to $\mu = 30$ min. The experiments use resilience parameters that are representative of current and forthcoming large-scale platforms [23, 5]. We take $C = R = 10$ min, $D = 1$ min, and $\omega = 1/2$.

On Figures 1.1, 1.2, and 1.3, we evaluate the impact of the ratio ρ (see Equation (1.2)) on the gain in energy and loss in time of ALGOE with respect to ALGOT. The general trend is that using ALGOE can lead to significant gains in energy at the price of a small increase in execution time.

We then study in Figures 1.4 and 1.5 the scalability of the approach on forthcoming platforms. We set the duration of the complete checkpoint and rollback (C and R, respectively) to 1 minute, independently of the number of processors, and we let

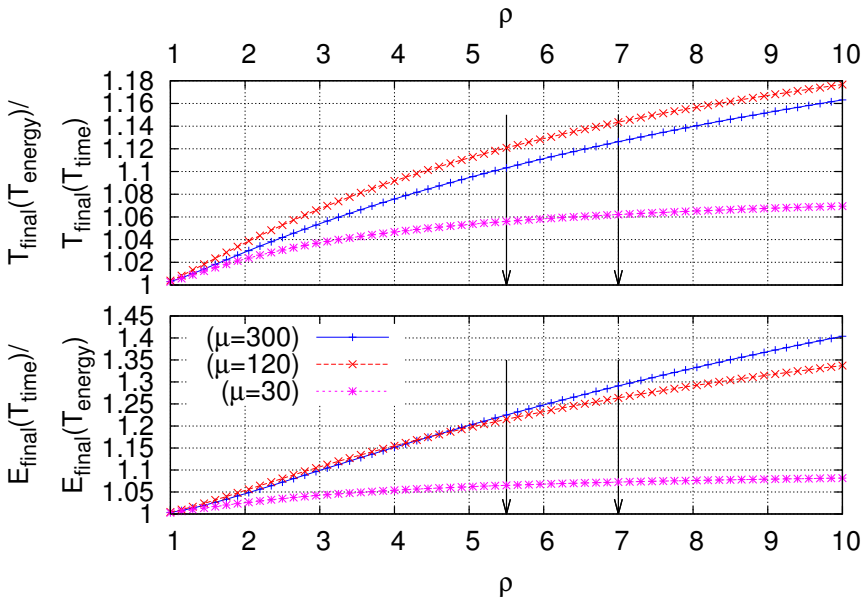

Fig. 1.1: Time and energy ratios as a function of ρ, with $C = R = 10$ min, $D = 1$ min, $\gamma = 0$, $\omega = 1/2$, and various values for μ

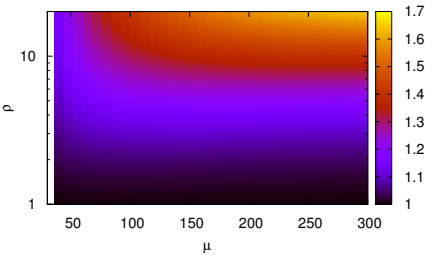

Fig. 1.2: Ratios of the different strategies with $C = R = 10$ min, $D = 1$ min, $\gamma = 0$, $\omega = 1/2$ as a function of μ and ρ: Energy ratio of ALGOT over ALGOE

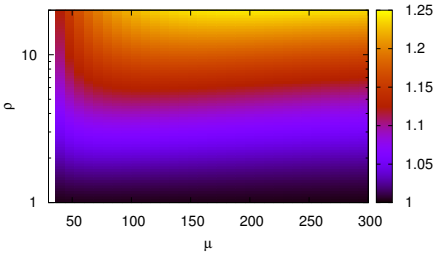

Fig. 1.3: Ratios of the different strategies with $C = R = 10$ min, $D = 1$ min, $\gamma = 0$, $\omega = 1/2$ as a function of μ and ρ: Execution time ratio of ALGOE over ALGOT

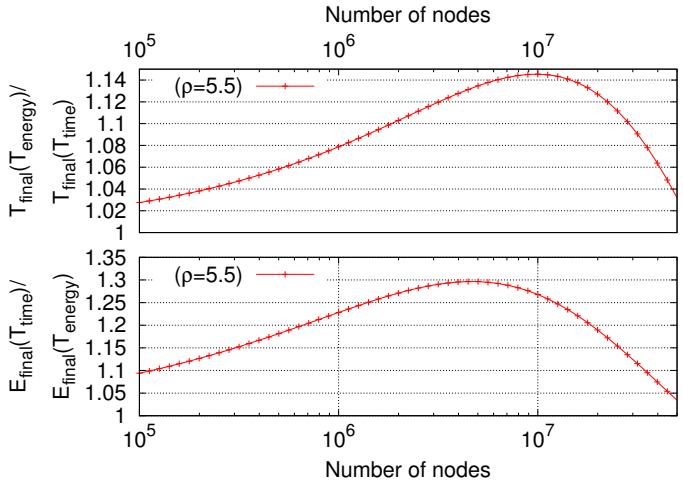

Fig. 1.4: Ratios of total energy and time for the two period strategies, as a function of the number of nodes, with $\mu = 120$ min for 10^6 nodes, $C = R = 1$ min, $D = 0.1$ min, $\gamma = 0$, $\omega = 1/2$: Time and energy ratios, as a function of the number of nodes, when $\rho = 5.5$

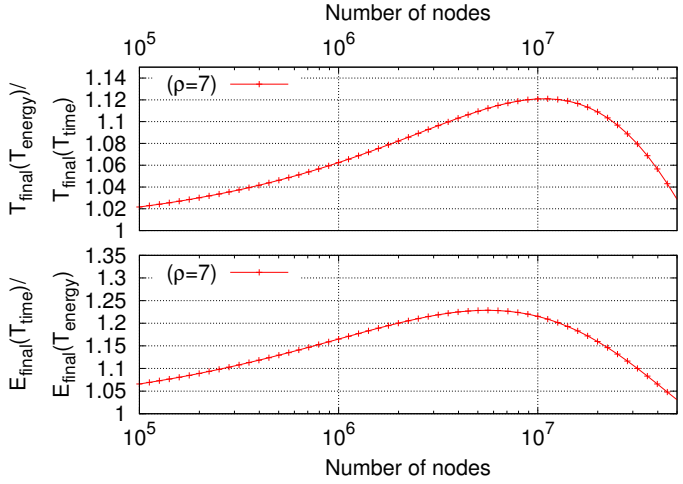

Fig. 1.5: Ratios of total energy and time for the two period strategies, as a function of the number of nodes, with $\mu = 120$ min for 10^6 nodes, $C = R = 1$ min, $D = 0.1$ min, $\gamma = 0$, $\omega = 1/2$: Time and energy ratios, as a function of the number of nodes, when $\rho = 7$

the downtime D equal to 0.1 minutes. It is reasonable to consider that checkpoint storage time will not increase with the number of nodes in the future, but on the contrary will remain constant. Indeed, system designers are studying a couple of alternative approaches. One consists of providing each computing node with local storage capability, ensuring through hardware mechanisms that this storage will remain available during a failure of the node. Another approach consists of using the memory of the other processors to store the checkpoint, pairing nodes as "buddies", thus allowing to take advantage of the high bandwidth capability of the high speed network to design a scalable checkpoint storage mechanism [47, 35, 19, 40].

The MTBF for 10^6 nodes is set to 2 hours, and this value scales linearly with the number of components. Given these parameters, Figures 1.4 and 1.5 show (i) the execution time ratio of ALGOE over ALGOT, and (ii) the energy consumption ratio of ALGOT over ALGOE, both as a function of the number of nodes. Figures 1.4 and 1.5 confirm the important gain in energy that can be achieved, namely up to 30% for a time overhead of only 12%. When the number of nodes gets very high (up to 10^8), then we observe that both energy and time ratios converge to 1. Indeed, when C becomes of the order of magnitude of the MTBF, then both periods $\mathscr{T}_{\text{Time}}^{\text{opt}}$ and $\mathscr{T}_{\text{Energy}}^{\text{opt}}$ become close to C to account for the higher failure rate.

1.2.4 Summary

In this section, we have provided a detailed analysis to compute the optimal checkpointing period, when the checkpointing activity can be partially overlapped with computations. We have considered two distinct objectives: either the goal is to minimize the total execution time, or it is to minimize the total energy consumption. Because of the different power consumption overheads due to computations and I/Os, we obtain different optimal periods.

We have instantiated the formulas with values derived from current and future Exascale platforms, and we have studied the impact of the power overhead due to I/O activity on the gains in time and energy. With current values, we can save more than 20% of energy with an MTBF of 300 min, at the price of an increase of 10% in the execution time. The maximum gains are expected for a platform with between 10^6 and 10^7 processors (up to 30% energy savings).

Our analytical model is quite flexible and can easily be instantiated to investigate scenarios that involve a variety of resilience and power consumption parameters.

1.3 Energy-aware fault-tolerance protocols for HPC applications: A methodology based on energy estimation

Although some devices allow to measure the power and energy consumption of a protocol [12], measuring the energy consumption requires always to run the pro-

cotol at a large scale and this in all execution contexts. To reduce the number of measurements, we must be able to estimate accurately the energy consumption of a protocol, for any execution context and for any experimental platform. The advantage of such energy estimation is to evaluate the energy consumption of a protocol without pre-executing in each execution context, and in this order to be able to choose the fault-tolerance protocol that consumes less energy.

In order to adapt the energy estimations to the execution platform, we need to collect a set of power measurements of the nodes of the platform during the various operations that compose the fault-tolerance protocols. However, we learned from [13], that the nodes of a same cluster can have an heterogeneous idle power consumption while they have the same extra power consumption due to the execution of an giving operation. We deduce that we need to measure the idle power consumption of each node of a same cluster but we need to measure the extra power consumption due to an operation only for each type of node. Moreover, in order to estimate the energy consumption according to the execution platform, we also need to measure the execution time of each operation on this platform. However, we have shown in [13] that the nodes of a cluster are homogeneous in terms of performance. We deduce that we do not need to measure the execution time due to an operation for each type of node. In order to adapt the energy estimations to the execution context, our estimation approach is also based on a description of the parameters execution provided by the user.

In this section, we explain our estimation methodology from the identification of the operations found in a fault-tolerance protocol to the energy estimation models of these operation, through a description of the calibration and the execution parameters that we need . We apply each step of our methodology to fault-tolerance protocols [15, 17]. In Section 1.3.1, we identify the various operations in the considered fault-tolerance protocols. Section 1.3.2 presents our methodology for calibrating the power consumption and the execution time of the identified operations. Section 1.3.3 shows how we estimate the energy consumption of the different operations by relying on the energy calibration and the different execution parameters. In Section 1.3.4, we evaluate the precision of the estimates for the considered fault-tolerance protocols by comparing then to the real energy measurements. In Section 1.3.5, we show how such energy estimations can be used in order to choose the energy-aware fault-tolerance protocol. Section 1.3.6 presents the conclusions of this section.

1.3.1 Identifying operations in fault-tolerance protocols

The first step of our methodology consists of identifying the various operations that we find in the different fault-tolerance protocols. An operation is a task that the fault-tolerance protocol may need to perform several times during the execution of an application.

As described in Section 1.1, we study the two families of fault-tolerance protocols: coordinated and uncoordinated protocols. For each of these two families, we distinguish two major phases: on the one hand, the checkpointing that occurs during a fault free execution (i.e., without failure) of an application, and on the other hand, the recovery which occurs whenever a failure occurs. In our study, we focus on the checkpointing phase.

We consider an application using the fault-tolerance protocol that is running on N nodes with p processes per node and where p is identical in the N nodes. In fault-tolerance protocols, we identify the following operations:

- Checkpointing: performed in both coordinated and uncoordinated protocols, it consists in storing a snapshot image of the current application state that can be later on used for restarting the execution in case of failure. In our study, we consider the system level checkpointing at the system level and not checkpointing at the application level. Such a choice is motivated by the fact that not all the applications embed global checkpointing and that we cannot select the optimal checkpointing interval with the applicative checkpointing. We consider the checkpointing provided in the Berkeley Lab Checkpoint/Restart library (BLCR), and available in the MPICH2 implementation. In checkpointing, the basic operation is to write a checkpoint of V_{data} size on a reliable media storage. For our study, we consider only the HDD since RAM is not reliable.
- Message logging: performed in uncoordinated protocols, it consists in saving on each sender process the messages sent on a specific storage medium (RAM, HDD, NFS, ...). In case of failure, thanks to message logging, only the crashed processes need to restart. In message logging, the basic operation is to write the message of V_{data} size on a given media storage. For our study, we consider the RAM and the HDD.
- Coordination: performed in coordinated protocols, it consists in synchronizing the processes before taking the checkpoints. If some processes have inflight messages at the coordination time, all the other ones are actively polling until these messages are sent. This ensures that there will be no orphan messages: messages sent before taking the checkpoints but received after checkpointing. When there is no more inflight message, all the processes exchange a synchronization marker. In coordination, the basic operations are the active polling during the transmission of inflight messages of V_{data} and the synchronization of $N \times p$ processes that occurs when there is no more inflight message.

In order to estimate the energy consumption of these operations, we need to take into account a large set of parameters. These operations are associated to parameters that depend not only on the protocols but also on the application features, and on the hardware used. Thus, in order to estimate accurately the energy consumption due to a specific implementation of a fault-tolerance protocol, the estimator needs to take into consideration all the protocol parameters (checkpointing interval, checkpointing storage destination, etc.), all the application specifications (number of processes, number and size of messages exchanged, volume of data written/read

by each process, etc.) and all the hardware parameters (number of cores per node, memory architecture, type of hard disk drives, etc.).

- fault-tolerance and application parameters: checkpointing interval, checkpointing storage destination, number of processes, number and size of messages exchanged between processes, type of storage media used (RAM, HDD, NFS, etc), volume of data written/read by each process, etc.
- hardware parameters: number of nodes, number of sockets per node, number of cores per socket, network topology, memory architecture, network technologies (Infiniband, Gigabit Ethernet, proprietary solutions, etc), type of hard disk drives (SSD, SATA, SCSI, etc), etc.

We consider that a parameter is a variable of our estimator only if a variation of this parameter generates a significant variation of the energy consumption while all the other parameters are fixed. It is necessary to calibrate the execution platform by taking into account all the parameters to estimate the energy consumption.

1.3.2 Energy calibration methodology

Energy consumption depends strongly on the hardware used in the execution platform. For instance, the energy consumption of checkpointing depends on the checkpointing storage destination (SSD, SATA, SCSI, etc.), on the read and write speeds and on the access times to the resource. The goal of the calibration process is to gather energy knowledge of all the identified operations according to the hardware used in the supercomputer. To this end, we gather the information about the energy consumption of the operations by running a set of benchmarks allowing to collect at set of power measurements and execution times of the various operations. The goal of such calibration approach is to adapt to the supercomputer used, the energy evaluations computed from the theoretical estimation models, and this in order to make our energy estimations accurate on any supercomputer, regardless of specifications. Although this knowledge base has a significant size, it needs to be done only occasionally, for example when there is a change in the hardware (like a new hard disk drive).

To estimate the energy consumption of a node performing an operation op, we need to obtain the power consumption of the node during the execution of op and the execution time of this operation. We know from [13] that the nodes from a same cluster are homogeneous in terms of performance. Therefore, we do need to measure and estimate the execution time due to an operation only for each type of nodes. Thus, the energy $\xi_{op}^{Node_i}$ consumed by a node i performing an operation op is:

$$\xi_{op}^{Node_i} = \rho_{op}^{Node_i} \cdot t_{op}$$

Analogously, the energy consumption $\xi_{op}^{Switch_j}$ of a (*switch*) j during the operation op is:

$$\xi_{op}^{Switch_j} = \rho_{op}^{Switch_j} \cdot t_{op}$$

t_{op} is the time required to perform op by un type of nodes.
$\rho_{op}^{Node_i}$ is the power consumed by the node i during t_{op}.
$\rho_{op}^{Switch_j}$ is the power consumed by the switch j during t_{op}.
As a consequence, in order to calibration the energy consumption, we need a calibrator for the power consumption described in Section 1.3.2.1 and a calibrator for the execution time described in Section 1.3.2.2.

1.3.2.1 Calibration of the power consumption ρ_{op}

We showed in [13] that the power consumption of a node i performing an operation op is composed of a static part, $\rho_{idle}^{Node_i}$, which is the power consumption of the node i when it is idle and a dynamic part $\Delta\rho_{op}^{Node_i}$, which is the extra power cost related to the operation op. We have shown that $\rho_{idle}^{Node_i}$ can be different even for identical nodes from homogeneous clusters. Therefore, we measure $\rho_{idle}^{Node_i}$ for each node i. We also have shown in [13] that $\Delta\rho_{op}^{Node_i}$ is the same for identical nodes running the same operation op. Consequently, we measure $\Delta\rho_{op}^{Node_i}$, for each operation op, once for each type of nodes,

In [13], we have also highlighted that the number p of processes used per node may influence the power consumed by the node. Therefore, we need to measure $\rho_{op}^{Node_i}(p)$ for every operation op and for different values of p. Thus, the power consumption $\rho_{op}^{Node_i}(p)$ of a node i during an operation op using p processes of this node is:

$$\rho_{op}^{Node_i}(p) = \rho_{idle}^{Node_i} + \Delta\rho_{op}^{Node}(p)$$

Analogously, the power consumption $\rho_{op}^{Switch_j}$ of a switch j during the operation op is:

$$\rho_{op}^{Switch_j} = \rho_{idle}^{Switch_j} + \Delta\rho_{op}^{Switch}$$

$\rho_{idle}^{Node_i}$ (or $\rho_{idle}^{Switch_j}$) is the power consumption of a node i (or of a switch j) when it is idle (i.e., switched on but executed nothing except the operating system) and $\Delta\rho_{op}^{Node_i}$ (or $\Delta\rho_{op}^{Switch_j}$) is the extra power consumption due to the execution of the operation op.

In order to compute $\Delta\rho_{op}^{Node}(p)$ (or $\Delta\rho_{op}^{Switch}$), we measure $\rho_{op}^{Node_i}(p)$ (or $\rho_{op}^{Switch_j}$) for a given node i (or a switch j) and subtract the static part of the power consumption which corresponds to the idle power consumption of the node i (or switch j). We measure $\rho_{op}^{Node_i}(p)$ (or $\rho_{op}^{Switch_j}$) by making the operation last a few seconds.

Therefore, it is an *mean* extra power consumption because it is computed from the average of several power measurements (one every second).

Moreover, $\rho_{op}^{Node_i}(p)$ and so $\Delta\rho_{op}^{Node}(p)$ may vary depending on the number p of processes used by the node i. Therefore, we need to calculate $\Delta\rho_{op}^{Node}(p)$ and thus to measure $\rho_{op}^{Node_i}(p)$ for different values of p in order to be able to estimate $\Delta\rho_{op}^{Node}(p)$ for a number p of processes executing the operation op. To do this, we should be able to know how $\Delta\rho_{op}^{Node}(p)$ evolves according to p (and this for each type of nodes). We do not know such information a priori. To this end, we rely on four possible models presented in the table below:

linear	$\Delta\rho_{op}^{Node}(p) = \alpha p + \beta$
logarithmic	$\Delta\rho_{op}^{Node}(p) = \alpha ln(p) + \beta$
power	$\Delta\rho_{op}^{Node}(p) = \beta p^{\alpha}$
exponential	$\Delta\rho_{op}^{Node}(p) = \alpha^{p} + \beta$

For each type of nodes, we measure $\Delta\rho_{op}^{Node}(p)$ for five different numbers of processes:

- the smallest possible value p denoted p_{min}, which is equal to 1;
- the highest possible value p denoted p_{max}, which corresponds to the number of cores available in the node;
- the median value denoted p_2 which corresponds to half of the number of cores available in the node;
- the number p_1 which is located in the middle of the interval $[p_{min}; p_2]$;
- the number p_3 which is located in the middle of the interval $[p_2; p_{max}]$

Then, we determine thanks to the least squares method [41] the coefficients (α and β) of each of the four models according to the five measured values for $\Delta\rho_{op}^{Node}(p)$. We compute the coefficient of determination R^2 corresponding to each of the four adjusted models obtained with the least squares method. We consider $\Delta\rho_{op}^{Node}(p)$ evolves according to the adjusted model for which the coefficient of determination is the highest one (i.e., that is to say, the closest to 1) .

For our measurements of $\Delta\rho_{op}^{Node}(p)$ (deduced from the measurements of $\rho_{op}^{Node_i}(p)$), we use an external wattmeter capable to provide us the mean power measurements with a sufficiently high frequency (1 Hz). We have shown in [12] that the OMEGAWATT wattmeter is a good candidate to collect such power measurements.

1.3.2.2 Calibration of the execution time t_{op}

The execution time t_{op} depends on one or many parameters according to the operation op. To take into account the possible effects of congestion, we consider that the number p of the same process node performing the same operation simultaneously op is a parameter to consider in our calibration of t_{op}. For example, this may occur

if multiple processes on the same node try to write data simultaneously on the local hard drive.

To calibrate t_{op}, we need to measure the execution time by varying different parameters. To do this, we measure t_{op} for five values uniformly distributed between the minimum and maximum for each parameter (while fixing all the other parameters). The five values of each parameter are chosen similarly to what we have previously reported with the parameter p for the calibration of $\Delta\rho_{op}^{Node}(p)$.

We consider two cases:

1. "Known model" case: we know a model a model where t_{op} evolves with respect to the parameters. We know it from the literature, with the knowledge of the algorithm used in the operation op or resource requested by the operation op. In this case, we determine the coefficients of the theoretical model using the least squares method [41] based on the values of the five parameters.
2. "No model known" case: we do not know how t_{op} evolves with respect to the parameters. In this case, for each parameter, we proceed to the determination by the adjusted least squares method as presented for the calibration of $\Delta\rho_{op}^{Node}(p)$ relying on the four models (linear, logarithmic, exponential and power).

To measure t_{op}, we instrument the code of the algorithm or the protocol of the operation op, in order to obtain the corresponding execution time. To ensure that the calibration of the execution time is accurate, we realize each measurement 30 times and we compute the mean value of the 30 measurements.

1.3.2.3 Models used for the execution times of the identified fault-tolerance operations

In this section, we describe the models used for the execution times of each operation of the fault-tolerance protocols. For each operation op, t_{op} depends on different parameters.

We remind that the calibration of t_{op} is required for each type of nodes. In other words, we do not need to calibrate t_{op} on all nodes when they are all identical.

For each type of nodes, the time $t_{checkpointing}$ required for checkpointing a volume of data V_{data} is:

$$t_{checkpointing}(p,V_{data}) = t_{access}(p) + t_{transfer}(p,V_{data}) = t_{access}(p) + \frac{V_{data}}{r_{transfer}(p)}$$

Similarly, the time $t_{logging}$ required to log a message with a size equal to V_{data} is:

$$t_{logging}(V_{data}) = t_{access} + t_{transfer}(V_{data}) = t_{access} + \frac{V_{data}}{r_{transfer}}$$

p is the number of processes within the same node simultaneously trying to perform the checkpointing operation. t_{access} is the time required to access the storage media where the checkpoint will be saved or the message logged. $t_{transfer}$ is the time

required to write data size V_{data} on the storage medium. $r_{transfer}$ is the transmission rate when writing on storage medium.

In the case of checkpointing, t_{access} and $r_{transfer}$ (and $t_{transfer}$) depend on the number p of processes per node since the p processes save their checkpoints simultaneously on the same storage media as the frequency of checkpoints writing is the same for all processes of an application.

A message is logged on a storage medium once it has been sent by a process of the node through the network interface used by the node. Thus, if several processes of the node try to send messages, there will be a traffic congestion at the network interface and the time for the current message will overlap the time of writing the message previously sent. In other words, this means that we consider that we can not find themselves in a situation where multiple messages are logged simultaneously by p processes of the node. Therefore, in the case of message logging, t_{access} and $r_{transfer}$ (and $t_{transfer}$) do not depend on the number p process per node.

As explained in Section 1.3.2.2, we measure $t_{checkpointing}$ considering both p and V_{data} parameters.

We know the theoretical model of $t_{checkpointing}$ based on V_{data} so for this parameter, we proceed to the determination of the coefficients of the theoretical model as explained in the case "with known model" (Section 1.3.2.2).

For p parameter, we do not have theoretical model giving $t_{checkpointing}$ based on p and therefore proceed as explained in the case of "no known model" (Section 1.3.2.2).

Regarding $t_{logging}$, it depends only on V_{data} and we have the theoretical model giving $t_{logging}$ depending on this parameter. So we proceed as explained in the "with known model" case.

We calibrate $t_{checkpointing}$ and $t_{logging}$ with respect to various storage media available on each node of the platform (RAM, local hard disk, flash SSD, etc..).

As we consider checkpointing at system-level, coordinated protocol requires a coordination between all processes.

The execution time for coordination between all processes is:

$$\begin{aligned} t_{coordination}(N,p,V_{data}) &= t_{polling}(V_{data}) + t_{synchro}(N,p) \\ &= \frac{V_{data}}{R_{transfer}} + t_{synchro}(N,p) \end{aligned}$$

p is the number of processes of the node i trying to perform coordination. $t_{synchro}(N,p)$ is the time required to exchange a marker synchronization between all processes. $t_{synchro}(N,p)$ depends on the number of nodes and the number of processes per node involved in the synchronization. We do not have a theoretical model for $t_{synchro}(N,p)$ neither in terms of N nor based on p. For the calibration, we proceed as explained in the "without known model" case (Section 1.3.2.2). $t_{polling}(V_{data})$ is the time required to finish transmitting the messages being transmitted at the time of coordination. In other words, $t_{polling}(V_{data})$ is equal to the time required to transfer the larger application message. $R_{transfer}$ is the transmission rate in the network infrastructure used for the platform.

Regarding the polling time, $t_{polling}(V_{data})$, we have a theoretical model giving $t_{polling}(V_{data})$. For the calibration, we proceed as in the "known model" case for V_{data} parameter.

1.3.3 Energy estimation methodology

We have previously described how we realize the energy calibration. Once the calibration is done, the estimator is able to provide estimates of the energy consumed by the various operations identified for fault-tolerance protocols. Figure 1.6 shows the framework components related to the estimation of the energy consumed.

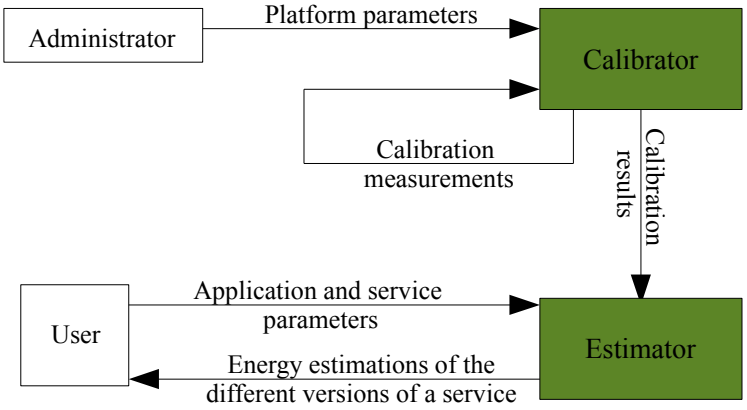

Fig. 1.6: Framework to estimate the energy consumption of fault-tolerance protocols

We can now describe how to estimate the energy consumed by each of the identified operations. To this end, we rely on the parameters provided by the user and the data measured by our calibrator.

Once the administrator has provided the hardware settings of the platform, the calibrator performs the various steps required to build the knowledge base on the power consumption and the execution time of the various identified operations. Then, based on the calibration results and a description of the application (the application memory size, etc..) and runtime parameters (number of nodes used, number of processes per node, etc..) provided by the user, the estimator calculates the energy consumption of different fault-tolerance protocols.

The parameters that we get from the user for the estimation depend on each operation to estimate. In case these parameters correspond to the values that we have measured during calibration, estimation directly uses these values to calculate

the energy consumed by the operation. If this is not the case, that is to say, if there is a lack of measurement points in the calibrator, the estimator uses the models created with the least squares method [41] during calibration.

This section describes how we estimate the energy consumed by each operation identified in the considered fault-tolerance protocols. For this, we show the necessary information: the parameters provided by the user and the data measured by our calibrator.

1.3.3.1 Checkpointing

To estimate the energy consumption of checkpointing, the estimator gets from the user the total memory size required by the application to run, the number of nodes N and the number p of processes per node.

From this information, the estimator calculates the average memory size V_{memory}^{mean} required by each process (total memory size divided by the number of processes). Then the estimator gets from the calibrator the extra power consumption $\Delta\rho_{checkpointing}(p)$ and the execution time $t_{checkpointing}(p, V_{memory}^{mean})$ depending on the models obtained by the least squares method in the step of the calibration.

It also gets the measurement $\rho_{idle}^{Node_i}$ for each node i. We denote respectively by $\xi_{checkpointing}^{Node_i}(p)$ and $\rho_{checkpointing}^{Node_i}(p)$ the energy consumption and the average power consumption of each node i performing checkpointing. The estimation of the energy consumption of a single checkpointing is given by:

$$\begin{aligned}
E_{checkpointing} &= \sum_{i=1}^{N} \xi_{checkpointing}^{Node_i}(p) \\
&= \sum_{i=1}^{N} \rho_{checkpointing}^{Node_i}(p) \cdot t_{checkpointing}(p, V_{memory}^{mean})) \\
&= t_{checkpointing}(p, V_{memory}^{mean}) \cdot \sum_{i=1}^{N} (\rho_{idle}^{Node_i} + \Delta\rho_{checkpointing}(p)) \\
&= t_{checkpointing}(p, V_{memory}^{mean}) \cdot (N \cdot \Delta\rho_{checkpointing}(p) \\
&\quad + (\sum_{i=1}^{N} \rho_{idle}^{Node_i}))
\end{aligned}$$

1.3.3.2 Message logging

To estimate the energy consumption of message logging, the estimator gets from the user the number of nodes N, the number p of processes per node, the number and total size of all messages sent during the application that he wants to run.

With this information, the estimator calculates the average volume V_{data}^{mean} of data sent and therefore logged on each node (total size of all messages sent divided by the

number of nodes N). Then, the estimator gets from the calibrator the extra power consumption $\Delta\rho_{logging}$ and the execution time $t_{logging}(p, V_{data}^{mean})$ depending on the models obtained with least squares method in the step of the calibration. It also receives the measurement of $\rho_{idle}^{Node_i}$ for each node i.

The estimation of the energy consumption of messages logging is given by:

$$\begin{aligned} E_{logging} &= \sum_{i=1}^{N} \xi_{logging}^{Node_i}(p) \\ &= \sum_{i=1}^{N} \rho_{logging}^{Node_i}(p) \cdot t_{logging}(V_{data}^{mean}) \\ &= t_{logging}(V_{data}^{mean}) \cdot \sum_{i=1}^{N} (\rho_{idle}^{Node_i} + \Delta\rho_{logging}(p)) \\ &= t_{logging}(V_{data}^{mean}) \cdot (N \cdot \Delta\rho_{logging}(p) + (\sum_{i=1}^{N} \rho_{idle}^{Node_i})) \end{aligned}$$

1.3.3.3 Coordination

We remind that the coordination is divided into two phases: the active polling during the transmission of the inflight messages followed by the synchronization of all processes. To estimate the energy consumption of the coordination, the estimator calculates the average message size $V_{message}^{mean}$ as the total size of messages divided by the total number of messages exchanged. The estimator also uses the total number of nodes N and the number of processes per node p. Then the estimator gets from the calibrator the extra power consumption $\Delta\rho_{sync}(p)$ and the execution time $t_{sync}(N, p)$ depending on the models obtained with the least squares method in calibration step. It also receives the measurement $\rho_{idle}^{Node_i}$ for each node i. The estimation of the energy consumption $E_{synchro}$ of synchronization is given by:

$$\begin{aligned} E_{synchro} &= \sum_{i=1}^{N} \xi_{synchro}^{Node_i}(N, p) \\ &= \sum_{i=1}^{N} \rho_{synchro}^{Node_i}(p) \cdot t_{synchro}(N, p) \\ &= t_{synchro}(N, p) \cdot \sum_{i=1}^{N} (\rho_{idle}^{Node_i} + \Delta\rho_{synchro}(p)) \\ &= t_{synchro}(N, p) \cdot (N \cdot \Delta\rho_{synchro}(p) + (\sum_{i=1}^{N} \rho_{idle}^{Node_i})) \end{aligned}$$

Regarding active polling, the estimator gets from the calibrator the extra power consumption $\Delta\rho_{polling}(p)$ and the execution time $t_{polling}(N, p, V_{message}^{mean})$ depending on the models obtained with least squares method in the calibration step. The esti-

mation of the energy consumption of active polling is given by:

$$\begin{aligned} E_{polling} &= \sum_{i=1}^{N} \xi_{polling}^{Node_i}(N,p) \\ &= \sum_{i=1}^{N} \rho_{polling}^{Node_i}(p) \cdot t_{polling}(V_{message}^{mean}) \\ &= t_{polling}(V_{message}^{mean}) \cdot \sum_{i=1}^{N} (\rho_{idle}^{Node_i} + \Delta\rho_{polling}(p)) \\ &= t_{polling}(V_{message}^{mean}) \cdot (N \cdot \Delta\rho_{polling}(p) + (\sum_{i=1}^{N} \rho_{idle}^{Node_i})) \end{aligned}$$

The estimator computes the energy consumption of coordination as follows:

$$E_{coordination} = E_{polling} + E_{synchro}$$

1.3.4 Validation of the estimations

To validate our estimations, we perform various real applications of high performance computing with different fault-tolerance protocols on a homogeneous cluster of the experimental distributed platform for large-scale computing, Grid'5000 [4], then we compare the energy consumption actually measured to the energy consumption evaluated by our estimator. For the experiments to validate our estimations, we used a cluster of the Grid'5000 distributed platform. The cluster we used for our experiments offers 16 identical nodes Dell R720. Each node contains 2 Intel Xeon CPU 2.3 GHz, with 6 cores each; 32 GB of memory; a 10 Gigabit Ethernet network; a SCSI hard disk with a storage capacity of 598 GB. We monitor this cluster with an energy- sensing infrastructure of external wattmeters from the SME Omegawatt. This energy-sensing infrastructure, which was also used in [11], enables to get the instantaneous consumption in Watts, at each second for each monitored node [10]. Logs provided by the energy-sensing infrastructure are displayed lively and stored into a database, in order to enable users to get the power and the energy consumption of one or more nodes between a start date and an end date. We ran each experiment 30 times and computed the mean value over the 30 values. We use the same notations as in previous sections: N is the number of nodes, p is the number of processes and op denotes one of the identified operations (checkpointing, message logging, etc.).

1.3.4.1 Calibration results of the platform

In this section, we present some of the calibration results on the considered platform according to the methodology described in Section 1.3.2. The considered platform is composed only of identical nodes: thus there is only one type of nodes. They are interconnected using a single network switch.

Calibrating the power consumption

First, we measure the idle power consumption ρ^i_{idle} for each node i of the experimental cluster. Figure 1.7 shows the idle power consumption of the 16 nodes belonging to the considered cluster. From this figure, even if the cluster is composed of homogeneous nodes, we notice the need to calibrate the electrical power when idle of each node.

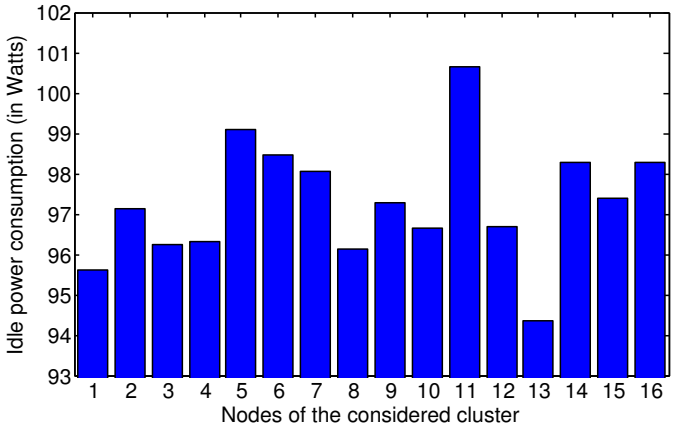

Fig. 1.7: Idle power consumption of the nodes of the cluster *Taurus*

For each identified operation op and for each of the cluster nodes, we calibrate with OMEGAWATT the average additional cost of electrical power due to the op operation, $\Delta\rho_{op}(p)$, as explained in Section 1.3.2.1. Since each node of the *Taurus* cluster has 12 processing cores, the five values of p we choose to calibrate $\Delta\rho_{op}(p)$ are 1, 4, 6, 9 and 12 processes per node. Figure 1.8 shows the measurements $\Delta\rho_{op}(p)$ for the five values of p and for each operation op identified in fault-tolerance protocols.

We note in Figure 1.8 that for some operations, $\Delta\rho_{op}(p)$ does vary depending on the number of cores per node that perform the same operation. For some operations, such as checkpointing $\Delta\rho_{op}(p)$ is almost a constant function of p. For $\Delta\rho op(p)$

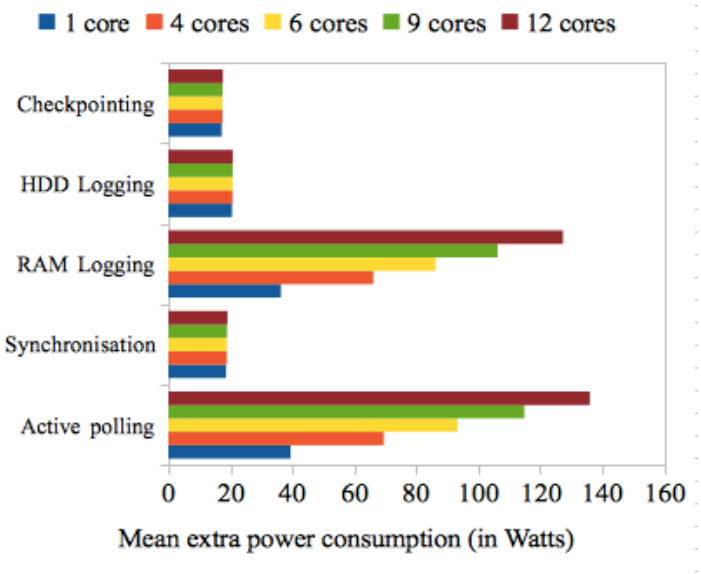

Fig. 1.8: Extra power consumption of fault-tolerance operations

of these operations, we obtain one of the four models of the calibrator (see section 1.3.2.1) with a coefficient α very close to 0 and a value of β very close to the constant value of $\Delta\rho_{op}(p)$ (i.e., that is to say, quasi-stationary model). For example, the model of $\Delta\rho_{checkpointing}(p)$ adjusted by the least squares method for the five values of p is:

$$\Delta\rho_{checkpointing}(p) = 17.22 \cdot p^{0.0084271}$$

Although the fitted model is a power model, the very low coefficient α implies that $\Delta\rho_{checkpointing}(p)$ is a quasi-stationary function p. The coefficient of determination R^2 corresponding to this model is 0.976, which is very close to 1.

For other operations such as *RAM logging*, $\Delta\rho_{op}(p)$ increases with p. For example, the model of $\Delta\rho_{RAM\_logging}(p)$ obtained in the calibration is:

$$\Delta\rho_{RAM\_logging}(p) = 35.237 \cdot p^{0.50158}$$

The fact that α is very close to 0.5 means that $\Delta\rho_{RAM\_logging}(p)$ is almost expressed in terms of $\sqrt{p}$. The coefficient of determination R^2 corresponding to this model is 0.992, which is also very close to 1.

In addition, we measure the energy consumption when idle of 10 Gigabit Ethernet switch for 300 seconds followed by the electrical power during heavy network traffic for 300 seconds. To measure its electrical power when idle, we ensure that there is no network traffic by turning off all nodes that are interconnected by the network switch. To measure its electrical power during heavy network traffic, we

run `iperf` in server mode on one of the nodes and `iperf` in client mode on all other interconnected nodes. Figure 1.9 shows the electrical measurements.

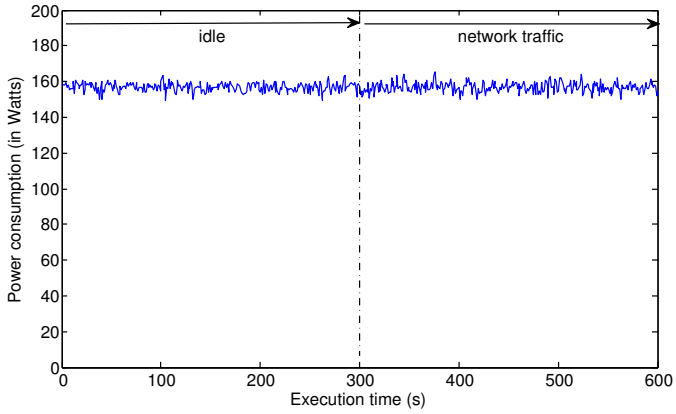

Fig. 1.9: Power consumption of the switch when idle for 300 seconds and with an intense network load for 300 seconds

From Figure 1.9, we note that the electric power switch remains almost constant throughout the duration of the experiment. In other words, the electrical power network switch does not vary depending on the network traffic. This means that $\Delta\rho_{op}^{Switch}$ is (almost) equal to 0 for all operations ($\forall op, \rho_{op}^{Switch_j} = \rho_{idle}^{Switch_j}$). A recent study [32, 26] confirms this fact in evaluating and demonstrating that the electrical power of multiple network devices is not affected by network traffic. That said, even if the electrical power of a network switch would depend on network traffic, our approach to calibration would allow to take into account in measuring $\Delta\rho_{op}^{Switch}$ for each operation op.

Calibration of the execution time

Based on the methodology presented in Section 1.3.2.2, we calibrate the execution time for each operation on each type of node of the experimental platform.

To calibrate the execution time of checkpointing on local hard drive, we consider a variable number of cores per node simultaneously checkpointing and we measure the time for different sizes of checkpoints V_{data} for one node of the experimental platform. Each node process saves a checkpoint with a size equal to V_{data}. In other words,when there are p processes that save checkpoints simultaneously a volume of $p \cdot V_{data}$ is saved on the local hard drive. Figure 1.10 shows the measured time for checkpointing on a node of the experimental platform. As explained in 1.3.2.2, we choose 1, 4, 6, 9 and 12 processes per node for the five values of p and 0 MB, 500

MB, 1000 MB, 1500 MB and 2000 MB for the five values of V_{data}. The choice of 2000 MB as the maximum size of checkpoint is motivated by the fact that each node has only 32 GB of memory that can be shared by 12 processing cores. For different values of p, Figure shows how evolves $t_{checkpointing}$ with respect to V_{data}.

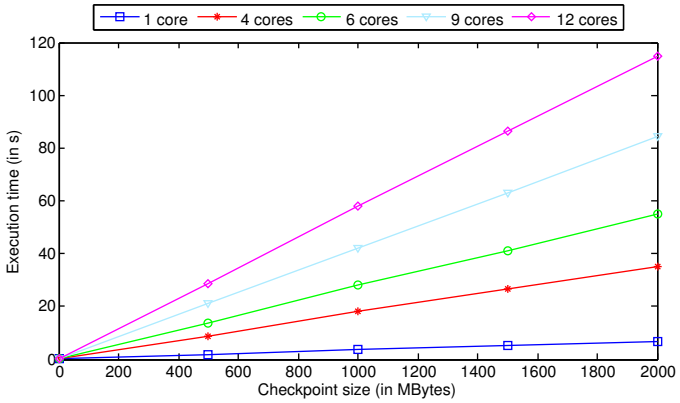

Fig. 1.10: Calibration of checkpointing on local hard drive

First, we observe that the curves have a linear trend according to V_{data} for p fixed. For example, for $p = 4$, the model for $t_{checkpointing}$ adjusted by the least squares method from the five values of V_{data}:

$$t_{checkpointing}(4, V_{data}) = \frac{1}{0.56569 \cdot 10^9} \cdot V_{data} + 0.09433 \cdot 10^{-3}$$

We also note that for V_{data} fixed $t_{checkpointing}$ increases when p grows this is because of the congestion of the input-output generated by concurrent access by p process on local hard drive. For $V_{data} = 1000MB$ the model of $t_{checkpointing}$ adjusted by the least squares method from the five values of p:

$$t_{checkpointing}(p, 1000Mo) = 4.91359 \cdot p - 1.5026$$

If for example we want to estimate the time $t_{checkpointing}(3, 800MB)$, that is to say, for values of p and V_{data} which booth are not belonging to the five measured values, we calculate:

- on one side: $t_{checkpointing}(1, 800Mo)$, $t_{checkpointing}(4, 800Mo)$, $t_{checkpointing}(6, 800Mo)$, $t_{checkpointing}(9, 800Mo)$ et $t_{checkpointing}(12, 800Mo)$, respectively from the equations $t_{checkpointing}(1, V_{data})$, $t_{checkpointing}(4, V_{data})$, $t_{checkpointing}(6, V_{data})$, $t_{checkpointing}(9, V_{data})$ and $t_{checkpointing}(12, V_{data})$;
- on the other side: $t_{checkpointing}(3, 0Mo)$, $t_{checkpointing}(3, 500Mo)$,

$t_{checkpointing}(3,1000Mo)$, $t_{checkpointing}(3,1500Mo)$, $t_{checkpointing}(3,2000Mo)$, respectively from the equations $t_{checkpointing}(p,0Mo)$, $t_{checkpointing}(p,500Mo)$, $t_{checkpointing}(p,1000Mo)$, $t_{checkpointing}(p,1500Mo)$, $t_{checkpointing}(p,2000Mo)$.

From the calculated values $t_{checkpointing}(1,800Mo)$, $t_{checkpointing}(4,800Mo)$, $t_{checkpointing}(6,800Mo)$, $t_{checkpointing}(9,800Mo)$ and $t_{checkpointing}(12,800Mo)$, we determine by the least squares method, the model giving $t_{checkpointing}(p,800Mo)$ as a function of p (as explained in section 1.3.2.2) and calculate the determination coefficient R^2 corresponding to the adjusted model.

Similarly, from the values $t_{checkpointing}(3,0Mo)$, $t_{checkpointing}(3,500Mo)$, $t_{checkpointing}(3,1000Mo)$, $t_{checkpointing}$(3, 1500 Mo), $t_{checkpointing}$(3, 2000 Mo), we determine the model giving $t_{checkpointing}(3,V_{data})$ as a function of V_{data} and calculate the determination coefficient R^2 corresponding to the thereby adjusted model. Then between $t_{checkpointing}(p,800Mo)$ and $t_{checkpointing}(3,V_{data})$, we choose the model for which the determination coefficient is the closest to 1. Then we calculate $t_{checkpointing}(3,800Mo)$ with the choosen model.

Figure 1.11 presents the execution time for message logging in RAM and on a HDD. To calibrate the execution time of message logging on memory or on disk, we measure the time for different message sizes V_{data} for one node of the experimental platform. The values choosen for V_{data} are 0 Ko, 500 Ko, 1000 Ko, 1500 Ko et 2000 Ko. As explained in section 1.3.2.3, we do not need to calibrate $t_{logging}$ as a function of p because the processes do not write simultaneously the messages on the medium storage due to the contention during message sending. We measure the execution time when a single process ($p = 1$) of the node executes the message logging operation.

We observe that the curves have a linear trend and this as well for message logging on RAM on local hard drive. The message logging time on local hard drive is higher than the RAM one and this regardless of the size of the logged message. Similarly to checkpointing, we get the following adjusted models for $t_{logging}$:

$$\begin{aligned} \text{In RAM} &: t_{logging}(V_{data}) = \tfrac{1}{4.4342\cdot 10^9}\cdot V_{data} + 0.0426\cdot 10^{-3} \\ \text{On the local HDD} &: t_{logging}(V_{data}) = \tfrac{1}{1.0552\cdot 10^9}\cdot V_{data} + 0.0858\cdot 10^{-3} \end{aligned}$$

Regarding coordination, we need to calibrate the time of the synchronization as well as the transfer time of a message.

To calibrate the synchronization time $t_{synchro}(N,p)$ of Np process, we measure this time for different values of N and for different values of p The measured values of p and N are chosen as explained in Section 1.3.2.2. In our case, the measured values of p are 1, 4, 6, 9 and 12 while the measured values for N are 1, 4, 8, 12 and 16. Figure 1.12 presents the synchronization time measured by the calibrator. For example, point 4 cores / 8 nodes is the time required to synchronize 32 processes 32 uniformly distributed over 8 nodes. First, we find that the time to synchronize processes located on the same node is lower than for processes located on different nodes. Indeed, it requires much less time to synchronize processes located on the same node than for processes located on different nodes. The transmission rate of the network is much lower than the transmission rate within a single node.

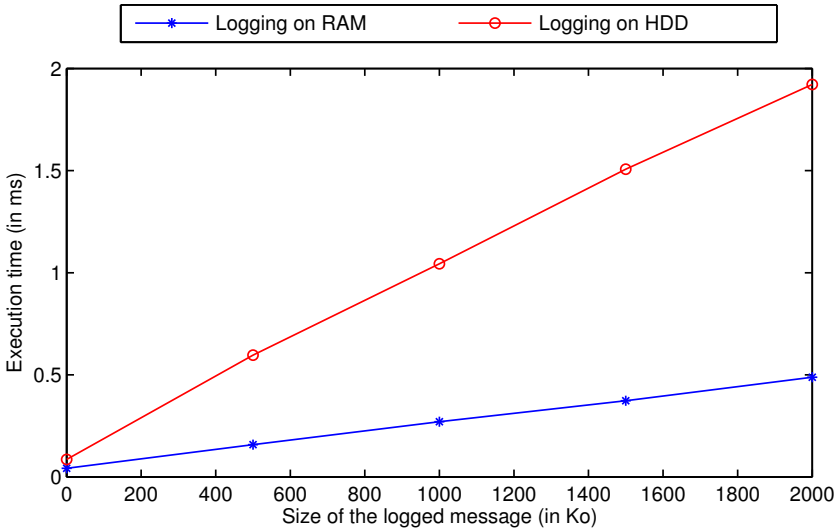

Fig. 1.11: Calibration of message logging on RAM and local disk

For example, for $p = 4$, the model for $t_{synchro}$ adjusted by the least squares method from the five values of N is:

$$t_{synchro}(N,4) = 0.0103757 \cdot ln(N) + 0.00445945$$

For $N = 8$, the model for $t_{synchro}$ adjusted by the least squares method from the five values of N is:

$$t_{synchro}(8,p) = 0.00443799 \cdot ln(p) + 0.02225942$$

If for example we want to estimate the time $t_{synchro}(N,p)$, that is to say, for values of N and p which booth are not belonging to the five measured values, then we proceed in a manner similar to that explained for $t_{checkpointing}(p,V_{data})$.

We calibrate the time needed to transfer a message (i.e., the active polling occurring at the time of coordination) on the experimental platform by varying the of size V_{data} of the message to transfer. To do this, we measure the execution time $t_{polling}(V_{data})$ to transfer a message sent using *MPI_Send* by a process located on a given node to a process on a different node. In the general case, we must make this measurement for each pair of processes at different levels of the network hierarchy, ie for two processes that need to cross a single network switch, then for two processes that need to cross two network switches, etc. In our experimental platform, a single network switch interconnects all nodes so we only need to measure the time for a couple of processes on different nodes.

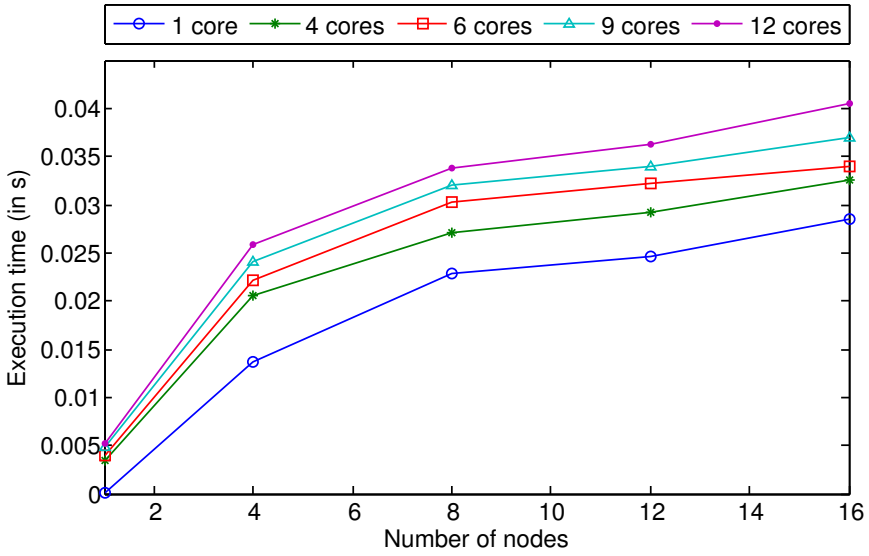

Fig. 1.12: Calibration of the synchronization time of the experimental platform

To calibrate the execution time to transfer a message over the network , we measure the time for different message sizes V_{data} for a couple of processes located on two separate nodes. The values chosen for V_{data} are 0 KB, 500 KB, 1000 KB, 1500 KB and 2000 KB. On Figure 1.13, we present the calibration of the transfer time of a message.

The measured transfer time depends linearly on the size of the message transferred. Similarly to checkpointing, we get the following adjusted model for $t_{polling}$:

$$t_{polling}(V_{data}) = \frac{1}{0.60148 \cdot 10^9} \cdot V_{data} + 3.6222 \cdot 10^{-3}$$

1.3.4.2 Accuracy of the estimations

In this section, we seek to compare the energy consumption achieved by our estimator once the calibration is made (but before executing the application) to the energy actually measured by the meters OMEGAWATT during the execution of the application.

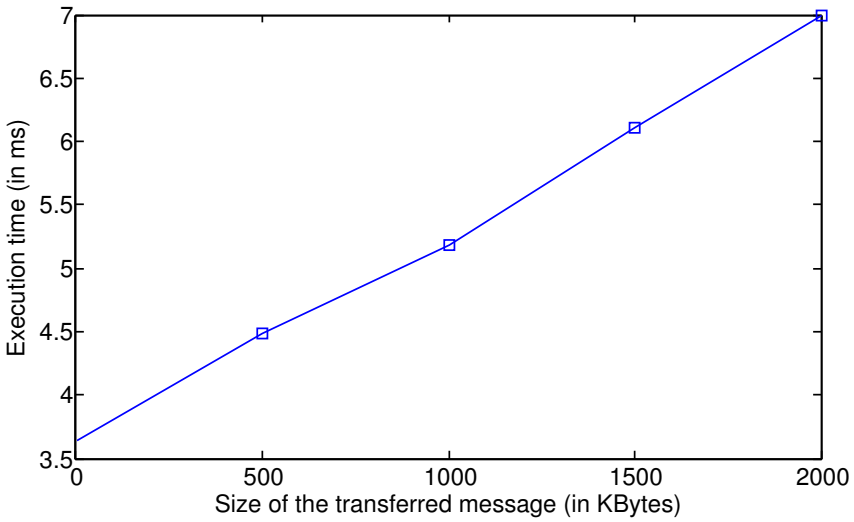

Fig. 1.13: Calibration of the active polling for the experimental platform

We consider four HPC applications: CM1[1] with a 2400x2400x40 resolution and 3 NAS[2] of class D (SP, BT, et EP) executed on 144 processes (i.e., 12 nodes with 12 cores per node) of the considered cluster.

With the infrastructure of external wattmeter OMEGAWATT, we measure for each application the energy consumption during the execution of the application with and without activation of the fault-tolerance protocols. Specifically, we instrumented the source code implementations of the different protocols of fault-tolerance in order to enable/disable each of the operations described above: checkpointing, message logging (on local disk or RAM disk) and coordination. Thus, we obtain the actual energy consumption for each operation. Each energy measurement is performed 30 times, and we consider the average values.

As concerns the uncoordinated protocol, we estimated and measured the energy consumption of all message logging. As concerns the coordinated protocol, we estimated and measured the energy consumption of a single checkpointing and therefore for one single coordination. To measure the energy consumption of a single checkpointing, we used a checkpoint interval greater than half of the application duration. Thus, the first (and only) checkpoint will occur in the second half of the application.

In Figure 1.14, we show the energy estimations for different operations identified in the protocols of fault tolerance. In Figure 1.15, we show the relative differences (in percent) between the estimated and the actual energy consumption. Figure 1.15

[1] Cloud Model 1: `http://www.mmm.ucar.edu/people/bryan/cm1/`

[2] NAS: `http://www.nas.nasa.gov/publications/npb.html`

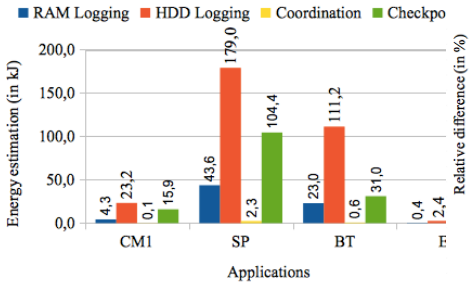

Fig. 1.14: Energy estimations (in kJ) of operations related to fault tolerance

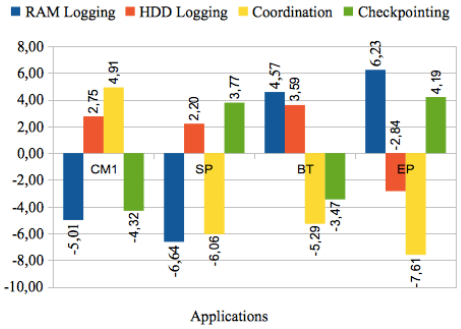

Fig. 1.15: Relative differences (in %) between the estimated and measured energy consumption of the operations related to fault tolerance

shows that the energy estimations provided in Figure 1.14 are accurate. Indeed, the relative differences between the estimated and measured energy consumption is low. The worst estimate shows a gap of 7.6 % compared to the measured coordination with EP value. The average deviation of all tests is 4.9 %.

In comparison with message logging and checkpointing, we find that we estimate a little less coordination. This is due to the fact that this process takes much less time than message logging. This is also due to the fact that this operation is evaluated from the estimated two sub-operations ($t_{polling}$ et $t_{synchro}$) which generates more inaccuracies in our estimation We will show in Section 1.3.5 how such energy estimations can reduce energy consumption related to protocols of fault tolerance when they are known before pre-executing the application.

1.3.5 Energy-aware choice of checkpointing protocols

In this section, we show how we can rely on energy estimations in order to reduce the energy consumption of the different fault-tolerance protocols executed with high-performance computing applications. The fault-tolerance protocol that consumes less energy may change depending on the considered application. The energy estimation that we are able to provide allows the users to choose the best fault-tolerance protocol in terms of energy consumption according to the execution context. By making such choice, the user is able to reduce the energy consumption of the executed fault-tolerance protocols.

The two families of fault-tolerance protocols that we considered are the coordinated and the uncoordinated protocols. We consider the 4 HPC applications that we studied in Section 1.3.4.2: CM1 with a resolution of 2400x2400x40 and 3 NAS

in Class D (SP, BT, and EP) running over 144 processes (i.e., 12 nodes with 12 cores per node). For each application and for each fault-tolerance protocol, we estimate the energy consumption by considering the different operations that we have identified in Section 1.3.1. First, we highlight that the energy consumption of an fault-tolerance operation depends highly on the application. Then, we show how the energy estimations of the different operations identified in Section 1.3.1 help the user in the choice of the fault-tolerance protocol that consumes the less energy.

Figure 1.14 shows that energy consumption of the operations are not the same from one application to another. For instance, the energy consumption of RAM logging in SP is more than 10 times the one in CM1. This is because CM1 exchanges much less messages compared to SP. Another example is that checkpointing in CM1 is more than 20 times the one in EP. Indeed, the execution time of CM1 is much higher than EP so the number of checkpoints is more important in CM1. Moreover, the volume of data to checkpoint is more important in CM1 as it involves a more important volume of data in memory.

We can obtain the overall energy estimation of the entire fault-tolerant protocols by summing the energy consumptions of the operations considered in each protocol. For fault free uncoordinated checkpointing, we add the energy consumed by checkpointing to the energy consumption of message logging. For fault free coordinated checkpointing, we add the energy consumed by checkpointing to the energy consumption of coordinations.

Both of uncoordinated and coordinated protocols rely on checkpointing. To obtain a coherent global state, checkpointing is combined with message logging in uncoordinated protocols and with coordination in coordinated protocols. Therefore, to compare coordinated and uncoordinated protocols from an energy consumption point of view, we compare the energy cost of coordinations to message logging. In our experiments we considered message logging either in RAM or in HDD. Coordination will consume as much as there are still bulked messages that are being transferred at the moments of the processes synchronization. Message logging will consume as much as the number and the size of exchanged messages during the application are important.

Figure 1.14 shows that from one application to another the less energy consuming protocol is not always the same. In general, determining the less consuming protocol depends on the trade-off between the volume of logged data and the coordination cost. For BT, SP and CM1, the less energy consuming protocol is the coordinated protocol (Coordination values on Figure 1.14 lower to the RAM and HDD logging values) since the volume of data to log for these applications is relatively important and leads to a higher energy consumption. Oppositely, the less energy consuming fault-tolerance protocol for EP is the uncoordinated one.

These conclusions are specific to the case where there is only one checkpointing and so one coordination during the execution of these applications. If the user is interested in more reliability, and this specifically for the applications that last long (several hours), he should choose a smaller checkpoint interval and so a higher number of checkpointing and coordinations. This checkpoint interval can influence the choice of the fault-tolerance protocol that consumes the less energy. Indeed, if

for instance during the execution of SP, there are more than 19 checkpointing and therefore more than 19 coordinations, the energy consumption of coordinations will be higher than the one of RAM logging. As a consequence, as opposed to what we have seen previously, it would be better to use the uncoordinated protocol to reduce the energy consumption of fault tolerance.

This checkpoint interval can be selected by considering the models that define the optimal interval: the one that enables to maximize the reliability by minimizing the performance degradation [45, 9].

In case we use a higher number of processes for the execution of a same application, the energy consumption of coordination will be more important. However, the energy consumption of message logging may also increase since there may be more communications with an increased number of processes. Therefore, there will be more message to send and so more message to log.

Thus, by providing such energy estimations before executing the HPC application, we help the user to select the best fault-tolerance protocol in terms of energy consumption depending on the number of checkpoints that he would like to perform during the execution of his application.

1.3.6 Summary

In this section, we presented an approach to accurately estimate the energy consumption of fault-tolerance protocols. We focused on the phase without failure. We considered the case of coordinated protocols and uncoordinated protocols.

Our estimation approach is to first identify the operations that we find in the different fault-tolerance protocols. Then, in order to adapt our theoretical models to the specificities of the considered platform, we perform an energy calibration that consists in gathering a set of measurements of the electrical power and execution times of each of the identified operations. To calibrate the considered platform, the calibrator collects parameters describing the execution platform, such as the number of nodes or the number of cores per node. With this calibration, energy estimations that we provide can adapt to any platform. Once the calibration is complete, the estimator is based on the calibration results as well as a description of the execution context to provide an estimation of the energy consumption of the fault-tolerance protocols.

We have shown in this section that our energy estimations are accurate for each fault-tolerance operation. Indeed, comparing the energy measurements for each operation to energy estimations that we are able to provide, we have shown that the relative differences were small. The relative differences between the estimates and energy measures are equal to 4.9% on average and do not exceed 7.6%.

Furthermore, we described the way to use our estimations in order to consume less energy. By providing energy consumption estimations before the execution of the application, we showed that it is possible to choose the fault-tolerance protocol

which is consuming the less energy for a particular application in a given execution context.

1.4 Conclusion

In this chapter, we have focused on the combination of two of the main challenges faced by Exascale systems: resilience and energy consumption. Even though these challenges have mainly been tackled independently, they are strongly interrelated. We have reviewed the literature on energy-aware checkpointing strategies and detailed two main contributions.

The first contribution consists in a detailed analysis to compute the optimal checkpointing period for a coordinated checkpointing protocol where the checkpointing activity can be partially overlapped with computations. We have considered different power consumption overheads for computations and I/Os in order to represent real-life systems in an accurate way. Experiments have shown that we can save more than 20% in energy, at the price of an increase of 10% in the execution time.

Then, we have considered both coordinated and uncoordinated protocols and explained how to estimate the energy consumption of these protocols. Energy estimations were shown to be accurate, and they can be used to reduce the energy consumption by allowing the user to use the protocol best suited for a particular application.

Besides, thanks to the energy estimations, understanding the energy behavior of the different fault-tolerance protocols allows us consider other solutions in order to reduce the energy consumption of a fault-tolerance protocol. Indeed, by predicting the idle periods and the active polling periods, we would be able to apply some power saving capabilities such as slowing down resources (like DVFS [27, 21, 29, 24]) or even shutting down [39, 25] some components if these idle or active polling periods are long enough [36].

Acknowledgements

This work was supported in part by the ANR *RESCUE* project and the Joint Laboratory for Petascale Computing between Inria and the University of Illinois at Urbana Champaign. Some experiments presented in this chapter were carried out using the Grid'5000 experimental testbed, being developed under the INRIA ALADDIN development action with support from CNRS, RENATER and several Universities as well as other funding bodies (see `https://www.grid5000.fr`).

References

[1] G. Aupy, A. Benoit, R. Melhem, P. Renaud-Goud, and Y. Robert. Energy-aware checkpointing of divisible tasks with soft or hard deadlines. In *International Green Computing Conference (IGCC)*, pages 1–8, 2013.

[2] G. Bosilca, A. Bouteiller, E. Brunet, F. Cappello, J. Dongarra, A. Guermouche, T. Hérault, Y. Robert, F. Vivien, and D. Zaidouni. Unified model for assessing checkpointing protocols at extreme-scale. *Concurrency and Computation: Practice and Experience*, October 2013. To be published. Also available as INRIA research report 7950 at `graal.ens-lyon.fr/~yrobert`.

[3] A. Bouteiller, G. Bosilca, and J. Dongarra. Redesigning the message logging model for high performance. *Concurrency and Computation: Practice and Experience*, 22(16):2196–2211, 2010.

[4] F. Cappello, E. Caron, M. J. Daydé, F. Desprez, Y. Jégou, P. V.-B. Primet, E. Jeannot, S. Lanteri, J. Leduc, N. Melab, G. Mornet, R. Namyst, B. Quétier, and O. Richard. Grid'5000: A Large Scale, Reconfigurable, Controlable and Monitorable Grid Platform. In *IEEE/ACM Grid 2005, Seattle, Washington, USA*, Nov. 2005.

[5] F. Cappello, H. Casanova, and Y. Robert. Preventive migration vs. preventive checkpointing for extreme scale supercomputers. *Parallel Processing Letters*, 21(2):111–132, 2011.

[6] F. Cappello, A. Geist, B. Gropp, S. Kale, B. Kramer, and M. Snir. Toward exascale resilience. *International Journal of High Performance Computing Applications*, 23:374–388, November 2009.

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

[8] J. T. Daly. A higher order estimate of the optimum checkpoint interval for restart dumps. *FGCS*, 22(3):303–312, 2004.

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

[10] M. Dias de Assuncao, J.-P. Gelas, L. Lefèvre, and A.-C. Orgerie. The green grid5000: Instrumenting a grid with energy sensors. In *5th International Workshop on Distributed Cooperative Laboratories: Instrumenting the Grid (INGRID 2010)*, Poznan, Poland, May 2010.

[11] M. Dias de Assuncao, A.-C. Orgerie, and L. Lefèvre. An analysis of power consumption logs from a monitored grid site. In *IEEE/ACM International Conference on Green Computing and Communications (GreenCom-2010)*, pages 61–68, Hangzhou, China, Dec. 2010.

[12] M. E. M. Diouri, M. F. Dolz, O. Glück, L. Lefèvre, P. Alonso, S. Catalán, R. Mayo, , and E. S. Quintana-Ortí. Solving some mysteries in power monitoring of servers: Take care of your wattmeters! In *Energy Efficiency in Large Scale Distributed Systems (EE-LSDS), Vienna, Austria, April, 22-24 2013*, 2013.

[13] M. E. M. Diouri, O. Glück, and L. Lefèvre. Your Cluster is not Power Homogeneous: Take Care when Designing Green Schedulers! In *4th IEEE International Green Computing Conference (IGCC), Arlington, VA USA*, June 2013.

[14] M. e. M. Diouri, O. Gluck, L. Lefèvre, and F. Cappello. Energy considerations in checkpointing and fault tolerance protocols. In *DSNW*, pages 1–6. IEEECSP, 2012.

[15] M. E. M. Diouri, O. Glück, L. Lefèvre, and F. Cappello. ECOFIT: A Framework to Estimate Energy Consumption of Fault Tolerance protocols during HPC executions. In *13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, Delft, Netherlands, May 2013.

[16] M. e. M. Diouri, O. Gluck, L. Lefevre, and F. Cappello. Ecofit: A framework to estimate energy consumption of fault tolerance protocols for HPC applications. In *CCGRID*, pages 522–529. IEEECSP, 2013.

[17] M. E. M. Diouri, G. L. Tsafack Chetsa, O. Glück, L. Lefèvre, J.-M. Pierson, P. Stolf, and G. Da Costa. Energy efficiency in high-performance computing with and without knowledge of applications and services. *International Journal of High Performance Computing Applications (IJHPCA)*, 2013. To appear in 2013.

[18] 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. Journal of High Performance Computing Applications*, 23(4):309–322, 2009.

[19] J. Dongarra, T. Hérault, and Y. Robert. Revisiting the double checkpointing algorithm. In *15th Workshop on Advances in Parallel and Distributed Computational Models APDCM 2013*. IEEE Computer Society Press, 2013.

[20] E. N. M. Elnozahy, L. Alvisi, Y.-M. Wang, and D. B. Johnson. A Survey of Rollback-Recovery Protocols in Message-Passing Systems. *ACM Computing Surveys*, 34(3):375–408, 2002.

[21] M. Etinski, J. Corbalan, J. Labarta, and M. Valero. Utilization driven power-aware parallel job scheduling. *Computer Science - Research and Development*, 25(3-4):207–216, 2010.

[22] W.-c. Feng, X. Feng, and R. Ge. Green supercomputing comes of age. *IT Professional*, 10(1):17 –23, 2008.

[23] 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 *Proceedings of the 2011 ACM/IEEE Conference on Supercomputing*, 2011.

[24] V. W. Freeh, D. K. Lowenthal, F. Pan, N. Kappiah, R. Springer, B. Rountree, and M. E. Femal. Analyzing the energy-time trade-off in high-performance computing applications. *IEEE Trans. Parallel Distrib. Syst.*, 18(6):835–848, 2007.

[25] F. Hermenier, N. Loriant, and J.-M. Menaud. Power Management in Grid Computing with Xen. In *Frontiers of High Performance Computing and Net-*

working - ISPA 2006 Workshops, ISPA 2006 International Workshops, FHPCN, XHPC, S-GRACE, GridGIS, HPC-GTP, PDCE, ParDMCom, WOMP, ISDF, and UPW, volume 4331 of *Lecture Notes in Computer Science*, pages 407–416, Sorrento, Italy, December 4-7 2006.

[26] H. Hlavacs, G. Da Costa, and J.-M. Pierson. Energy consumption of residential and professional switches. In *IEEE CSE*, 2009.

[27] Y. Hotta, M. Sato, H. Kimura, S. Matsuoka, T. Boku, and D. Takahashi. Profile-based optimization of power performance by using dynamic voltage scaling on a pc cluster. In *Proceedings of the 20th International in Parallel and Distributed Processing Symposium, IPDPS 2006*, 2006.

[28] C. hsing Hsu, W. chun Feng, and J. S. Archuleta. Towards efficient supercomputing: A quest for the right metric. In *Proceedings of the High Performance Power-Aware Computing Workshop*, 2005.

[29] C.-H. Hsu and W. chun Feng. A power-aware run-time system for high-performance computing. In *Supercomputing, 2005. Proceedings of the ACM/IEEE SC 2005 Conference*, 2005.

[30] J. Dongarra et al. The international ExaScale software project roadmap. *Int. J. of High Performance Computing & Applications*, 25(1), 2011.

[31] J. H. Laros III, K. T. Pedretti, S. M. Kelly, W. Shu, and C. T. Vaughan. Energy based performance tuning for large scale high performance computing systems. In *Proceedings of the 2012 Symposium on High Performance Computing*, HPC '12, pages 6:1–6:10, San Diego, CA, USA, 2012.

[32] P. Mahadevan, P. Sharma, S. Banerjee, and P. Ranganathan. A power benchmarking framework for network devices. In *NETWORKING 2009 Conference, Aachen, Germany, May 11-15, 2009.*, pages 795–808, 2009.

[33] E. Meneses, O. Sarood, and L. V. Kalé. Assessing Energy Efficiency of Fault Tolerance Protocols for HPC Systems. In *Proceedings of the 2012 IEEE 24th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD 2012)*, New York, USA, October 2012.

[34] R. H. B. Netzer and J. Xu. Necessary and sufficient conditions for consistent global snapshots. *IEEE Transactions on Parallel and Distributed Systems*, 6(2):165–169, 1995.

[35] X. Ni, E. Meneses, and L. V. Kalé. Hiding checkpoint overhead in HPC applications with a semi-blocking algorithm. In *Proc. 2012 IEEE Int. Conf. Cluster Computing*. IEEE Computer Society, 2012.

[36] A.-C. Orgerie, L. Lefevre, and J.-P. Gelas. Save Watts in your Grid: Green Strategies for Energy-Aware Framework in Large Scale Distributed Systems. In *ICPADS 2008 : The 14th IEEE International Conference on Parallel and Distributed Systems, Melbourne, Australia*, December 2008.

[37] P. Alonso and M. F. Dolz and R. Mayo and E S. Quintana-Ortí. Energy-efficient execution of dense linear algebra algorithms on multi-core processors. *Cluster Computing*, May 2012.

[38] P. Alonso, M. F. Dolz, F. D. Igual, R. Mayo and E. S. Quintana-Ortí. DVFS-control techniques for dense linear algebra operations on multi-core processors. *Computer Science - R&D*, 27(4):289–298, 2012.

[39] E. Pinheiro, R. Bianchini, E. V. Carrera, and T. Heath. Load balancing and unbalancing for power and performance in cluster-based systems. In *IN WORKSHOP ON COMPILERS AND OPERATING SYSTEMS FOR LOW POWER*, 2001.

[40] R. Rajachandrasekar, A. Moody, K. Mohror, and D. K. D. Panda. A 1 PB/s file system to checkpoint three million MPI tasks. In *Proceedings of the 22nd international symposium on High-performance parallel and distributed computing*, HPDC '13, pages 143–154, New York, NY, USA, 2013. ACM.

[41] C. Rao, H. Toutenburg, A. Fieger, C. Heumann, T. Nittner, and S. Scheid. Linear models: Least squares and alternatives. *Springer Series in Statistics*, 1999.

[42] 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`.

[43] J. Shalf, S. Dosanjh, and J. Morrison. Exascale computing technology challenges. In *VECPAR'10, the 9th Int. Conf. High Performance Computing for Computational Science*, LNCS 6449, pages 1–25. Springer-Verlag, 2011.

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

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

[46] G. Zheng, X. Ni, and L. V. Kalé. A scalable double in-memory checkpoint and restart scheme towards exascale. In *Dependable Systems and Networks Workshops (DSN-W)*, 2012.

[47] G. Zheng, L. Shi, and L. V. Kalé. FTC-Charm++: an in-memory checkpoint-based fault tolerant runtime for Charm++ and MPI. In *Proc. 2004 IEEE Int. Conf. Cluster Computing*. IEEE Computer Society, 2004.