ECOFIT: A Framework to Estimate Energy Consumption of Fault Tolerance Protocols for HPC Applications

M. el Mehdi Diouri, Olivier Glück, Laurent Lefèvre
Laboratoire de l'Informatique du Parallélisme
CNRS, ENS Lyon, INRIA, Universite Lyon 1
{*Mehdi.Diouri, Olivier.Gluck, Laurent.Lefevre*}*@ens-lyon.fr*

Franck Cappello
Laboratoire de Recherche en Informatique - NCSA
INRIA and University of Illinois at Urbana-Champaign
cappello@illinois.edu

***Abstract*—Energy consumption and fault tolerance are two interrelated issues to address for designing future exascale systems. Fault tolerance protocols used for checkpointing have different energy consumption depending on parameters like application features, number of processes in the execution and platform characteristics. Currently, the only way to select a protocol for a given execution is to pre-execute the application and monitor the energy consumption of different fault tolerance protocols. This is needed for any variation of the execution setting. To avoid this time and energy consuming process, we propose an energy estimation framework. It relies on an energy calibration of the considered platform and a user description of the execution setting. We evaluate the accuracy of our estimations with real applications running on a real platform with energy consumption monitoring. Results show that our estimations are highly accurate and allow selecting the best fault tolerant protocol without pre-executing the application.**

***Keywords*-Fault tolerance protocols, Checkpoint/Restart; Energy Consumption; Performance; Estimation.**

I. Introduction

Supercomputers are used to run a wide range of scientific applications in many domains like the design of cars and aircrafts, the prediction of severe weather phenomena and seismic waves. In order to meet new scientific challenges and address critical problems with high societal impact, the HPC community has set a new performance objective for the end of the decade: Exascale.

To achieve such performance, an exascale computer will gather several millions of CPU cores running up to a billion of threads. From the current knowledge and observations of existing large systems, it is anticipated that exascale systems will experience various kind of faults many times per day [1]. In addition to the increase of errors and failures, the electrical power consumption is considered as a potentially limiting factor to the future growth in high performance computing (HPC) [2]. For exascale systems, the Defense Advanced Research Projects Agency (DARPA) has set to 20MW the maximum power consumption. If we compare it to the 8.21 MW consumed by the current fastest supercomputer (17.59 PFlops Cray XK7 Titan in the USA), it is clear that we need to improve the flops/watt by a factor of 25 in less than 6 years. However, fault tolerance and energy consumption are interrelated: fault tolerance consumes energy and some energy reduction techniques can increase error and failure rates [3].

Thus, to ensure the transition to the exascale era, we must address both power/energy consumption and fault tolerance. We know from [4] that the power consumption of fault tolerant protocols depends on many execution parameters (the number of processes used, the volume of message exchanged in the application, characteristics of the execution platform...) and may vary if even a single parameter differs. We also know that none of the fault tolerant protocol outperforms the other with respect to fault tolerance. The best protocol depends on the execution configuration.

Currently, in order to evaluate the power consumption of a fault tolerant protocol for any particular execution, the only approach is to pre-execute the application and monitor the energy consumption. This approach is not practical for protocol selection since it does not allow to evaluate power consumption before the execution. To address this problem, this paper proposes an accurate estimator of the power consumption for fault tolerant protocols. This estimator can be used to estimate the power consumption of a particular fault tolerant protocol for a large variety of execution configurations. It can also be used to compare fault tolerant protocols from given execution configurations.

Our estimator considers the three main families of fault tolerance protocols: coordinated checkpointing, uncoordinated checkpointing with message logging and the recent hierarchical protocols. Coordinated checkpointing protocols are currently the most popular fault tolerant protocols used in HPC. Coordination essentially consists in removing inflight messages from the communication network before all processes checkpoint [5]. In practice, the performance overhead of coordination is considered negligible. One open question is how the power consumption of coordinated checkpointing protocols evolves with variations of execution parameters. Another important aspect of coordinated checkpointing is the need to restart globally the execution during the recovery even if a single process fails. In principle, global restart is not needed when processes form totally independent clusters during the execution [6]. However, in practice, HPC

applications do not present such communication patterns. As a consequence, global restart is the common case and it leads to a huge waste of energy because it forces all non failed processes to redo all their computations and communications from the last checkpoint.

Uncoordinated checkpointing 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. There are several families of message logging protocols: optimistic, pessimistic, causal [7]. However, all message logging protocols need to log all messages sent by all processes during the whole execution. The performance impact is limited [8]. However, they are consuming more energy than coordinated protocols for in free situations. How the energy cost of message logging evolves with the execution parameters is another open question.

A third family of fault tolerant protocols has been proposed recently to address the limitations of coordinated protocols and message logging protocols: hierarchical protocols [9], [10], [11] These protocols combine the advantages of limited message logging in fault free situation and limited restart during recovery. These protocols organize processes of the execution in clusters and log only the inter cluster messages. Intra cluster messages are not logged. There is no global coordinated checkpointing but processes inside every cluster do coordinate before checkpointing. The evolution of the power consumption of hierarchical protocols with the execution parameters in fault free and recovery situation is an open question.

The ECOFIT framework proposed in this paper will help to give answers to these open questions. This paper is structured as follows. Section II describes related works. The design of the energy estimation framework is detailed in Section III. Section IV presents validation results. Section V discusses how to choose the less energy consuming fault tolerance protocol while section VI concludes the paper and presents some future works.

II. Related works

Monitoring energy consumption of large scale HPC systems still deals with precision, frequency and scalability issues. Evaluating power consumption of one node or one process has been extensively explored.

Hardware monitoring systems like Powermon [12] allows a fine grain monitoring of one server by monitoring each of the DC power rails supplying the motherboard at a high frequency level. The Powerpac [13] framework allows a precise energy monitoring of one HPC node. Based on the energy profile and independent measures of one node of an homogeneous cluster, authors emulate measurements on several nodes. On the opposite side, software systems like SoftPower [14] estimate the energy consumption by observing the usage of internal resources. In [15], authors present a way of evaluating application power consumption. They describe a methodology for predicting the power consumption of a computer, depending on performance counters, and then use these counters to estimate the power consumption of each single process.

For large scale precise energy monitoring, in works like [16], [17], authors measure energy consumption of a full site of the Grid5000 experimental testbed [18]. For their energy measurements, they use a dedicated energy-sensing infrastructure available on Grid5000 Lyon site. Authors analyze information on the energy consumed by the nodes and show the correlation between the energy logs collected and the user resource reservation requests.

III. Design of the ECOFIT Framework

We propose the ECOFIT framework which estimates the energy consumption due to fault tolerance protocols. In [19], we describe some basics of this estimator. Our study focuses on the coordinated, uncoordinated, and hierarchical protocols. We identify the following high-level 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 provided in the Berkeley Lab Checkpoint/Restart library (BLCR), and available in the MPICH2 implementation.
- 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, ...);
- 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. When there is no more inflight message, all the processes exchange a synchronization marker.
- Recovery: in case of failure, it consists of restarting the execution of the application from the last checkpoint. In uncoordinated protocols, only crashed processes are restarted. At the opposite, in coordinated protocols, all processes are restarted even if only one process crashes.

A. Basic operations and associated parameters

Estimating the energy consumption of a given high-level operation *op* (checkpointing, message logging, coordination, or recovery) is really complex as it depends on a large set of parameters. In checkpointing, the basic operation is to write the checkpoint on a reliable media storage. For our study, we consider only the HDD since RAM is not reliable. In message logging, the basic operation is to write the message on a given media storage. For our study, we

consider the RAM and the HDD. In coordination, the basic operations are the active polling during the transmission of inflight messages and the synchronization that occurs when there is no more inflight message. In recovery, the basic operations are the restarting and the application re-execution. The restarting consists of reading the last checkpoint from a reliable media storage. The application re-execution consists of running the application from the checkpoint loaded until the instant of failure. In this paper, ECOFIT do not consider the application re-execution energy estimation but it will in the next future.

These basic 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, ECOFIT 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.).

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. In order to take into consideration all the parameters, ECOFIT integrates an automated calibration component. This calibrator is described in III-B.

B. Calibration approach

Energy consumption depends strongly on the hardware used in the execution platform. The goal of the calibration process is to gather energy knowledge of all the identified basic operations according to the hardware used in the supercomputer. At this end, we developed a set of simple benchmarks that extracts the energy consumption ξ_{op} of the basic operations encountered in fault tolerance protocols. The goal of our calibration approach is to make our energy estimations accurate on any supercomputer, regardless of its size (exascale, for instance). Although this knowledge base has a significant size, it needs to be done only occasionally.

In our calibrator component, the energy consumption of a node i performing a basic operation op is:

$$\xi^i_{op} = \rho^i_{op} \cdot t^i_{op}.$$

t^i_{op} is the time required to perform op by the node i. ρ^i_{op} is the power consumption of the node i during t^i_{op}. Thus, for each node i, we need to get the power consumption ρ^i_{op}, and the execution time t^i_{op} of each operation. Therefore, our energy calibrator integrates a power calibrator described in III-B1 and an execution time calibrator described in III-B2.

1) Power consumption ρ^i_{op}: In our power calibrator, the power consumption of an operation op is:

$$\rho^i_{op} = \rho^i_{idle} + \Delta\rho^i_{op}$$

ρ^i_{idle} is the power consumption when the node i is idle (i.e. switched on but running only the operating system) and $\Delta\rho^i_{op}$ is its extra power cost due to the basic operation execution. We showed in [4] that ρ^i_{idle} may be different even for identical nodes. Thus, we calibrate ρ^i_{idle} by measuring the power consumption of each node while it is idle. We also showed in [4] that for a given operation, $\Delta\rho^i_{op}$ depends only on the hardware used on the node and is consequently the same on identical nodes. Thus, for each type of nodes, we measure $\Delta\rho^i_{op}$ during each basic operation op.

In order to measure $\Delta\rho^i_{op}$ experimentally, we isolate each basic operation by instrumenting the implementation of each fault tolerance protocol that we consider, and we use OmegaWatt an external power meter [1]. This external power meter provides up to 1 measure each second with a precision of 0.125W. We validated our external power measurements thanks to Powermon 2 [12] which is an internal power meter that provides up to 1000 power measurements per second. We simultaneously performed a set of power measurements with Powermon and OmegaWatt on a same monitored node. We noticed that during a given operation, $\Delta\rho^i_{op}$ is almost constant. We can then consider that one power measurement per second is enough to measure $\Delta\rho^i_{op}$. That is why we make the choice in ECOFIT to measure $\Delta\rho^i_{op}$ externally as it is easier to plug an external power meter that furthermore includes also the extra power consumption due to the fans and to the power supply.

In order to take into account the impact of parallelism, ECOFIT calibrates $\Delta\rho^i_{op}$ by varying the number of cores that perform the same op. Now that we know how to calibrate the power consumption ρ^i_{op} for each operation and for each node of the supercomputer, we need to describe how to calibrate the execution time t^i_{op}.

2) Execution time t^i_{op}: In this section, we describe the execution time model that we consider for each high-level operations of fault tolerance protocol. For each operation op, t^i_{op} depends on different parameters.

Checkpointing, Message logging and Restarting: For a given node i, the time required for checkpointing a volume of data, for logging a message, or for restarting from a checkpoint is:

$$t^i_{op} = t^i_{access} + t^i_{transfer} = t^i_{access} + \frac{V_{data}}{r^i_{transfer}}$$

t^i_{access} is the time needed to access the storage media where the message will be logged, where the checkpoint will be stored, or from where the checkpoint will be loaded. $t^i_{transfer}$ is the time needed to write/read a data on/from a given storage media. $r^i_{transfer}$ is the transmission rate of the storage media.

t^i_{access} and $r^i_{transfer}$ are almost constant when we consider volumes of data of the same order of magnitude.

[1] http://www.omegawatt.fr/gb/index.php

Therefore, to calibrate t^i_{op}, ECOFIT automatically runs a simple benchmark that measures the execution time for different values of V_{data}. In order to take into consideration the eventual contention that may occur on the same storage media, we also perform this calibration for different numbers of processes per node which are logging messages, checkpointing, or restarting at the same time. We perform this calibration process for all the different storage medium (RAM, HDD, SSD, NFS, ...) that are available in the supercomputer.

Coordination: Since checkpointing is considered at the system level in our study, the coordinated checkpointing that we consider requires an extra synchronization between the processes. The time required for a process coordination is:

$$t^i_{op} = t^i_{polling} + t_{synchro} = \frac{V_{data}}{r^i_{transfer}} + t_{synchro}$$

$t_{synchro}$ is the time needed to exchange a marker among all the processes. $t_{synchro}$ depends on the number of processes to synchronize and the number of processes per node. To calibrate $t_{synchro}$, ECOFIT measures the time required to perform a synchronization barrier among processes that are already synchronized meaning $t^i_{polling}$ is equal to zero (the best case). These measurements are performed by varying the number of nodes and the number of processes per node. $t^i_{polling}$ is the time necessary to finish transfers of inflight messages at the coordination time. In other words, $t^i_{polling}$ is equal to the time required to transfer the bulkiest message. Thus, in order to calibrate $t^i_{polling}$, we measure for different message sizes, the mean transfer time of this message. We choose different message sizes V_{data} in the same way as we do for message logging and checkpointing.

These simple relationships used in our energy models are not directly applicable since the energy consumption of fault-tolerant protocols is highly dependent on the hardware and the execution context used. In our approach, we overcome this difficulty by running a complete calibration process, which is the key point of our estimating approach.

C. Estimation methodology

We described previously how we perform the calibration process in ECOFIT. This calibration needs to be done by the administrator each time that a change occurs in the hardware used in the supercomputer. Once this calibration is completed, our energy estimator framework is able to provide estimations of the energy consumption due to fault tolerance protocols.

Figure 1 shows the components of ECOFIT and their interactions. The user provides some information related to the execution context and to the application he would like to run. These information is taken as an input by the estimator component. As an output, the calibrator component provides the calibration data on which ECOFIT relies on to estimate the energy consumption of fault tolerance protocols. In the following subsections, we detail for each protocol phase, the information collected from the user and the corresponding calibration data transmitted to our estimator component. We also detail how the energy consumption is computed thanks to this calibration output and to the information provided by the user.

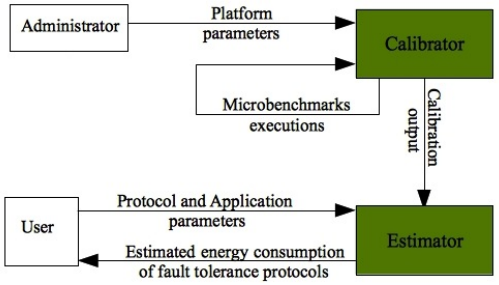

Figure 1. ECOFIT components and interactions

1) Message logging: To estimate the energy consumption of message logging, the estimator component collects from the user the number of processes per node, the total number and size of the messages sent during the application. From this information, the estimator computes the mean volume of data V^{mean}_{data} sent (so logged) by each node. Besides, the estimator collects from the calibrator the logging time $t^i_{logging}$ corresponding to V^{mean}_{data} for each node and according to the number of processes per node.

If V^{mean}_{data} is not a size recorded by the calibrator, the estimator computes the equation that gives $t^i_{logging}$ according to V_{data}, and adjusts the equation using the method of least squares [20]. We denote by $\xi^i_{logging}$ and $\rho^i_{logging}$ the energy and the power consumption of each node i performing a message logging. The estimated energy consumption of message logging is:

$$E_{logging} = \sum_{i=1}^{N} \xi^i_{logging} = \sum_{i=1}^{N} \rho^i_{logging} \cdot t^i_{logging}$$

where N is the total number of nodes, $\rho^i_{logging}$ is an output of our calibrator and $t^i_{logging}$ is computed by our estimator.

2) Checkpointing/Restarting: To estimate the energy consumption of checkpointing or restarting, the estimator component collects from the user the total memory size required by the application to run, the total number of nodes and the number of processes per node. From this information, the estimator computes the mean memory size required by each node. The estimator component collects also the number of checkpoints to perform during the application execution. Besides, the estimator collects from the calibrator the checkpoint and restart times corresponding to the calibrated checkpoint sizes.

Similarly to message logging, the estimator calculates the checkpoint and restart times t^i_{op} corresponding to the mean memory size V^{mean}_{memory} required by each node.

The estimated energy consumption of one checkpointing or one restarting is:

$$E_{op} = \sum_{i=1}^{N} \xi^i_{op} = \sum_{i=1}^{N} \rho^i_{op} \cdot t^i_{op}$$

In the previous equation, op is either "$checkpoint$" or "$restart$". The total estimated energy consumption of checkpointing is obtained by multiplying $E_{checkpoint}$ by the number of checkpoints C.

3) Coordination: To estimate the energy consumption of coordination, the estimator component uses the mean message size $V_{message}^{mean}$ as the total size of messages divided by the total number of messages. The estimator component also uses the number of checkpoints C, the total number of nodes N and the number of processes per node that are provided for message logging and checkpointing estimations.

From the calibration output, the estimator collects the synchronization time $t_{synchro}$ corresponding to the number of processes per node and the total number of nodes specified by the user. $t_{synchro}$ corresponds to one synchronization among all the processes. The estimated energy consumption $E_{synchro}$ of one synchronization is:

$$E_{synchro} = \sum_{i=1}^{N} \xi_{synchro}^{i} = t_{synchro} \cdot \sum_{i=1}^{N} \rho_{synchro}^{i}$$

Similarly to message logging and to checkpointing, the estimator calculates the message transfer time $t_{polling}^{i}$ corresponding to the mean message size $V_{message}^{mean}$. The estimated energy consumption $E_{polling}$ of one active polling is:

$$E_{polling} = \sum_{i=1}^{N} \xi_{polling}^{i} = \sum_{i=1}^{N} \rho_{polling}^{i} \cdot t_{polling}^{i}$$

The estimator calculates the estimated energy consumption of all coordinations as follows:

$$E_{coordinations} = C \cdot (E_{polling} + E_{synchro})$$

C is the number of checkpoints and so the number of coordinations.

IV. Validation of our Energy Estimating Approach

To apply our energy estimating approach, we calibrate and run real HPC applications on a real homogeneous cluster of the large scale experimental platform Grid5000 [18]. To validate our approach and demonstrate the accuracy of our energy estimations, we run the different fault tolerance protocols during real HPC application executions and compare the measured energy consumption to the one we estimate thanks to our energy estimating framework.

A. Experimental infrastructure

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 power meters from the SME Omegawatt. This energy-sensing infrastructure, which was also used in [16], enables to get the instantaneous consumption in Watts, at each second for each monitored node [17]. 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.

B. Platform calibration

In this section, we calibrate our platform according to the calibration process described in III-B. Since each node has 12 cores, we consider the cases of 1, 4, 8 or 12 processes per node.

1) Power calibration: First, we measure the idle power consumption ρ_{idle}^{i} of each node i. If we were at extreme scale, we would have determined the distribution of the idle power of the nodes and pull estimates for each node from the measured distribution. Then, we calibrate the extra power consumption $\Delta\rho_{op}^{i}$ of all the basic operations found in the fault tolerance protocols (Figure 2). As each node has 12 cores each, we calibrated the extra power cost by assuming that 1, 4, 8 or 12 processes are running the same operation at the same time.

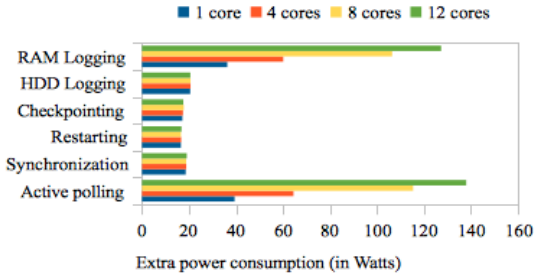

Figure 2. Extra power cost of basic operations per node

We show that the most power consuming operations are the RAM logging and the active polling that occurs during the coordination if processes are not synchronized. These two operations require a more intensive use of the CPU. We also notice that for these two basic operations, the extra power consumption varies with the number of cores per node that perform the same operation. This comes from the fact that more cores are running intensively for these two basic operations.

2) Execution time calibration: To calibrate the execution time, ECOFIT automatically runs the corresponding benchmark for each basic operation. As we have announced previously, this execution time for a same operation and with the same parameters may vary from one node to another even if we consider identical nodes. That is why we calibrate this execution time for each operation and for each node of our cluster. In order to take into account the possible contention, we consider 1, 4, 8 or 12 cores of a same node performing the same operation.

Message logging: We present our calibration results for message logging on HDD in Figure 3. For different message sizes, we measure the logging time for each node of

the cluster. For each message size, we represent in Figure 3, the mean logging time and the standard deviation over all the nodes of our cluster. The standard deviations are invisible since the differences between the logging times all over the nodes are insignificant. We notice that when several cores are logging at the same time, the execution time is higher: simultaneous accesses on HDD create I/O contentions. That's why we need to calibrate our execution time for different numbers of processes per node.

When we plotted the same figure for message logging on RAM, the only difference that we observed is that logging time does not vary when the number of cores per node is changed. Indeed, there is no contention when several cores do RAM access simultaneously.

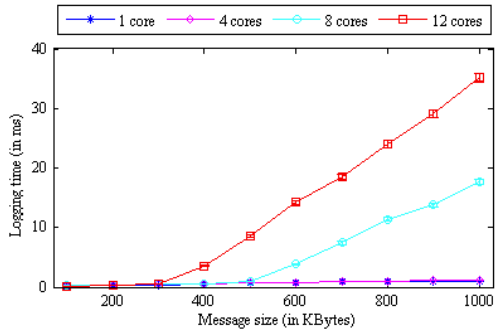

Figure 3. Calibration of the HDD logging time in our cluster

Checkpointing/Restarting: To calibrate time checkpointing on HDD, we consider a variable number of cores per node checkpointing at the same time and we measure the checkpointing time for different checkpoint sizes for each node of the cluster. We represent in Figure 4 the mean checkpointing time and the standard deviation over all the nodes of our cluster.

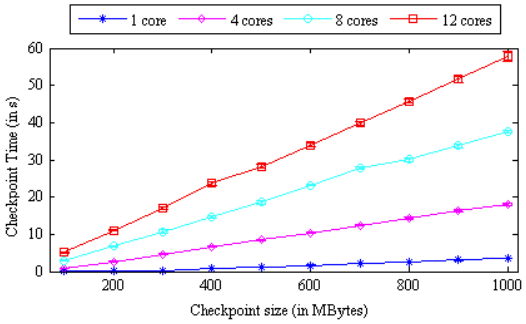

Figure 4. Calibration of the HDD checkpoint time in our cluster

The results lead to the same conclusions that we had previously for HDD logging. When we plot the same figure for restarting from HDD, the only difference is that the restarting time is a little less important than the checkpointing time. Indeed, restarting do read accesses on HDD whereas checkpointing do write accesses.

Coordination: As concerns the coordination, we need to calibrate the synchronization time and the transfer time of a message. As we explained in III, the active polling time is correlated to the transfer time of an inflight message. Figure 5 presents the synchronization time measured by our calibrator. For instance, the point 4 cores / 8 nodes is the time of a synchronization barrier between 32 processes.

First, we show that the synchronization time is slightly higher if we consider more processes per node. Indeed, synchronizing processes that are on the same node requires much less time than processes on distinct nodes (the network transmission rate is much lower than the transmission rate within a same node).

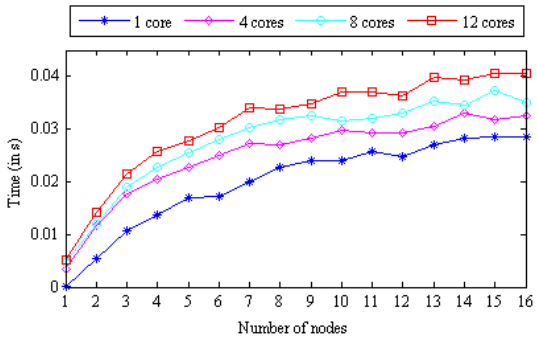

Figure 5. Calibration of the synchronization time in our cluster

Moreover, we notice that the curve shape in Figure 5 is not linear but logarithmic. Hence, in our estimator, we consider the following model for $t_{synchro}$:

$$t_{synchro} = \alpha ln(N) + \beta$$

N is the number of nodes involved in the synchronization. α and β are two constants that depend on the supercomputer features and on the number of processes per node.

We measure $t_{synchro}$ for different number of nodes, N. We determine the corresponding α and β and we adjust the model thanks to the method of least squares [20]. Then, for a given number of nodes, we can estimate $t_{synchro}$.

The last time to calibrate is the transfer time. We measure the time necessary to transfer a message on our platform by varying the message size. The transfer times measured depend linearly on the message size to transfer.

C. Validation of the estimation framework

In this section, we want to compare the energy consumption obtained by our estimator once the calibration done (but before running the application) to the real energy consumption measured by our energy sensors during the application execution. We consider 4 HPC applications: CM1 with a resolution of 2400x2400x40 [2] and 3 NAS [3] in Class D (SP, BT, and EP) running over 144 processes (i.e. 12 nodes with 12 cores per node). For each application, we measure the total energy consumption of one application execution with and without the high-level operations

[2]Cloud Model 1: http://www.mmm.ucar.edu/people/bryan/cm1/
[3]NAS: http://www.nas.nasa.gov/publications/npb.html

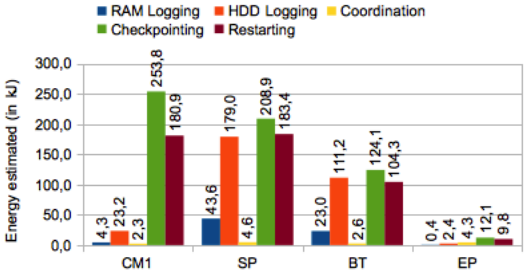

Figure 6. Estimated energy consumption (in kJ) of high-level operations

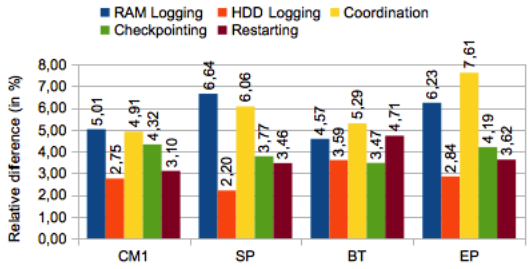

Figure 7. Relative difference (in %) between the estimated and the measured energy consumption

activated in the fault tolerance protocols. To do that, we have instrumented the code of fault tolerance protocols in MPICH2. Thus, we obtain the energy consumption of each high-level operation. Each energy measurement is done 30 times and we compute the average value. For checkpointing measurements, we consider a checkpoint interval of 120 seconds. For recovery measurements, processes are restarted just after the checkpoint ends. This allows us to measure the energy consumption of restarting only (i.e. recovery without application re-execution).

In Figure 6, we plot the estimated energy consumption computed by our estimator for each high-level operation and for each HPC application considered. In Figure 7, we plot the relative differences (in percentage) between the estimated and the measured energy consumption. Figure 7 shows that our energy estimations are accurate: the relative difference between the estimated and the measured energy consumptions are low. The worst estimation is 7.6 % for coordination with EP. These results allow us to give answers to some of the open questions announced at the introduction.

Figure 6 shows that energy consumption of the high-level 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 high-level 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. Lastly, to estimate the energy consumption of hierarchical checkpointing, we need the user to provide our estimator the composition of each cluster (i.e the list of processes in each cluster).

V. Choose the less energy consuming fault tolerance protocol

The results we presented in section IV allow us to address the following question: how ECOFIT can help to select the less energy consuming fault tolerance protocol ?

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.

Figure 6 shows that from one application to another the less energy consuming protocol is not always the same. Indeed, the less energy consuming protocol for BT, SP and CM1 is the coordinated protocol whereas it is the uncoordinated one for EP. 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 since the volume of data to log for these applications is relatively important and leads to a higher energy consumption. Thus, by providing such energy estimations before executing the HPC application, we can select the best fault tolerant protocol in terms of energy consumption.

The energy consumption is one of the main criteria for the choice of fault-tolerant protocols but it is not the only one. Depending on the user need, we may establish a multi-criteria decision that takes into account the energy consumption, the performance and the level of resilience of the fault-tolerant protocols.

VI. Conclusion

This paper presents a framework that estimates the energy consumption of fault tolerance protocols. In our study, we consider the three families of fault tolerance protocols:

coordinated, uncoordinated and hierarchical protocols. To provide accurate estimations, ECOFIT relies on an energy calibration of the execution platform and a user description of the execution settings. Thanks to our approach based on a calibration process, this framework can be used in any energy monitored supercomputer. We showed in this paper that the energy estimations provided by ECOFIT are accurate. Indeed for the applications we considered in our experiments, the relative difference between these energy estimations and the measured energy consumption are equal to 4.9 % in average and do not exceed 7.6 %.

By estimating the energy consumption of fault tolerance protocols, ECOFIT allows selecting the best fault tolerant protocol in terms of energy consumption without pre-executing the application. A direct application of our energy estimating framework is the energy consumption optimization of fault tolerance.

As a future work, we will investigate energy efficient solutions for fault tolerance protocols. We also plan to extend ECOFIT to more protocols that are needed at extreme-scale such as data management protocols.

ACKNOWLEDGMENT

Some experiments of this article were performed on the Grid5000 platform, an initiative from the French Ministry of Research through the ACI GRID incentive action, INRIA, CNRS and RENATER and other contributing partners. This research is partially supported by the INRIA-Illinois Joint laboratory on Petascale Computing.

REFERENCES

[1] F. Cappello, A. Geist, B. Gropp, L. Kale, B. Kramer, and M. Snir, "Toward exascale resilience," *Int. J. High Perform. Comput. Appl.*, vol. 23, no. 4, pp. 374–388.

[2] H. D. Simon, "Is HPC Going Green ?" ScientificComputing.com, May/June 2008.

[3] 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*, vol. 34, no. 3, pp. 375–408, 2002.

[4] M. Diouri, O. Gluck, L. Lefevre, and F. Cappello, "Energy considerations in checkpointing and fault tolerance protocols," in 2^{nd} *Workshop on Fault-Tolerance for HPC at Extreme Scale (FTXS), co-located with DSN.*, Boston, USA, jun 2012.

[5] R. H. B. Netzer and J. Xu, "Necessary and sufficient conditions for consistent global snapshots." *IEEE Transactions on Parallel and Distributed Systems*, vol. 6, no. 2, pp. 165–169, 1995.

[6] R. Koo and S. Toueg, "Checkpointing and rollback-recovery for distributed systems." in *FJCC*. IEEE Computer Society, 1986, pp. 1150–1158.

[7] L. Alvisi and K. Marzullo, "Message logging: Pessimistic, optimistic, causal, and optimal," *IEEE Transactions on Software Engineering*, vol. 24, no. 2, pp. 149–159, 1998.

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

[9] A. Guermouche, T. Ropars, M. Snir, and F. Cappello, "HydEE: Failure Containment without Event Logging for Large Scale Send-Deterministic MPI Applications," in *Proceedings of IEEE IPDPS 2012*. IEEE CS Press, to appear.

[10] C. L. M. Esteban Meneses and L. V. Kalé, "Team-based message logging: Preliminary results," in *3rd Workshop on Resiliency in High Performance Computing (Resilience) in Clusters, Clouds, and Grids (CCGRID 2010).*, May 2010.

[11] A. Bouteiller, T. Hérault, G. Bosilca, and J. J. Dongarra, "Correlated set coordination in fault tolerant message logging protocols," in *17th The International Conference on Parallel Processing (Euro-Par)*, ser. Lecture Notes in Computer Science, vol. 6853. Springer, September 2011, pp. 51–64.

[12] D. Bedard, M. Y. Lim, R. Fowler, and A. Porterfield, "PowerMon: Fine-grained and integrated power monitoring for commodity computer systems," in *Proceedings Southeastcon 2010*. Charlotte, NC: IEEE, Mar. 2010.

[13] R. Ge, X. Feng, S. Song, H.-C. Chang, D. Li, and K. W. Cameron, "Powerpack: Energy profiling and analysis of high-performance systems and applications," *IEEE Transactions on Parallel and Distributed Systems*, vol. 99, pp. 658–671, 2009.

[14] M. Y. Lim, A. Porterfield, and R. Fowler, "SoftPower: Fine-Grain Power Estimations Using Performance Counters," in *The ACM International Symposium on High Performance Distributed Computing (HPDC)*. Chicago: ACM, Jul. 2010.

[15] G. Da Costa and H. Hlavacs, "Methodology of measurement for energy consumption of applications," in *The IEEE Energy Efficient Grids, Clouds and Clusters Workshop (E2GC2), co-located with GRID, Brussels, Belgium*, October 2010.

[16] 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)*, Hangzhou, China, Dec. 2010, pp. 61–68.

[17] 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.

[18] F. Cappello et al, "Grid'5000: A large scale, reconfigurable, controlable and monitorable grid platform," in *6th IEEE/ACM International Workshop on Grid Computing, Grid'2005*, Seattle, Washington, USA, Nov. 2005.

[19] M. Diouri, O. Gluck, L. Lefevre, and F. Cappello, "Towards an energy estimator of fault tolerance protocols," in 18^{th} *ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming PPoPP (poster paper)*, Shenzhen, China, Feb 2013 (to appear).

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