Towards an Energy Estimator for Fault Tolerance Protocols

Mohammed el Mehdi Diouri
Olivier Glück Laurent Lefèvre
INRIA Avalon Team
Laboratoire de l'Informatique du Parallélisme
CNRS, ENS Lyon, INRIA, Université 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

Checkpointing protocols have different energy consumption depending on parameters like application features and platform characteristics. To select a protocol for a given execution, we propose an energy estimator that relies on an energy calibration of the considered platform and a user description of the execution settings.

Categories and Subject Descriptors D.4.5 [*Operating Systems*]: Reliability - Checkpoint/restart, Fault-tolerance.

Keywords Fault tolerance protocols; Checkpointing; Energy Consumption; Estimation.

1. Introduction

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[1] proposes an estimator of the energy consumed by 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 study focuses on the fault free execution of coordinated, uncoordinated, and hierarchical [3] protocols. Checkpointing 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 uncoordinated protocols, message logging consists in saving on each sender process the messages sent on a given storage media. In coordinated protocols, a coordination 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.

Our energy estimation relies on an energy calibration of the considered platform and a user description of the execution settings. Section 2 presents the calibration approach while Section 3 presents the estimation methodology. Section 4 concludes the paper and presents some future works.

[1] This research is partially supported by the INRIA-Illinois Joint Laboratory on Petascale Computing.

PPoPP'13, February 23–27 2013, Shenzhen, China.
ACM 978-1-4503-1922-5/13/02.

2. Calibration approach

Estimating the energy consumption of a given high-level operation *op* (message logging, coordination, or checkpointing) is really complex as it depends on a large set of parameters. In message logging, the basic operation is to write the message on a given media storage. In checkpointing, the basic operation is to write the checkpoint on a reliable media storage. 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. These operations are associated to parameters that depend not only on the protocols (checkpointing interval, checkpointing storage destination, etc.) but also on the application features (volume of messages exchanged between processes, etc.), and on the hardware used (number of nodes, network technology, etc.).

In order to estimate accurately the energy consumption of a high-level operation, our estimator needs to take into consideration all the parameters. Our framework integrates an automated calibration. The goal of this calibration is to gather energy knowledge of all the identified operations according to the hardware used in the platform. At this end, we developed a set of simple benchmarks that extracts for each node i, the energy consumption e^i_{op} for each operation op encountered in fault tolerance protocols: $e^i_{op} = p^i_{op} \cdot t^i_{op}$ where t^i_{op} is the time required to perform op and p^i_{op} the power consumption during t^i_{op}.

2.1 Power consumption p_{op}

For a node i, the power consumption of an operation op is:

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

p^i_{idle} is the power consumption when the node i is idle and Δp^i_{op} is the extra power cost due to op. In [1], we showed that p^i_{idle} may be different even for identical nodes. Thus, we gather p^i_{idle} by measuring the power consumption of each node while it is idle. We also showed in [1] that for a given operation, Δp^i_{op} is the same on identical nodes, and that Δp^i_{op} depends only on the hardware used on the node. In order to take into account the impact of parallelism, Δp^i_{op} is calibrated by varying the number of cores that perform the same op. In order to measure Δp^i_{op} experimentally, we isolate each basic operation by instrumenting the implementation of each fault tolerance protocol that we consider, and by using an external power meter for gathering the power measurements.

2.2 Execution time t_{op}

For each operation op, t^i_{op} depends on different parameters. For each node i, t^i_{op} is calibrated by considering different numbers of

processes per node. The time required for checkpointing a volume of data or for logging a message is:

$$t^i_{ckpt/logging} = t^i_{access} + t^i_{transfer} = t^i_{access} + \frac{V_{data}}{R^i_{transfer}}$$

t^i_{access} is the time needed to access the storage media where the checkpoint will be stored or where the message will be logged. $t^i_{transfer}$ is the time needed to write a data on a given storage media. To calibrate $t^i_{ckpt/logging}$, our framework automatically runs a simple benchmark that measures the execution time for different values of V_{data}. This calibration process is performed for the different storage medium (RAM, HDD, etc.) available in the platform. For instance, Figure 1 presents calibration results for checkpointing on HDD for a cluster composed of 64 nodes of 2 CPU cores each. For different data sizes, we measure the checkpointing time for each node of the considered cluster, by considering one or both CPU cores. For each data size, we represent in Figure 1, the mean checkpointing time and the standard deviation over all the nodes.

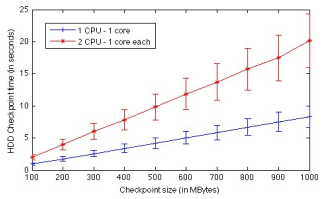

Figure 1. Calibration of the HDD Checkpointing time

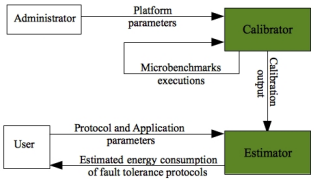

Figure 2. Energy estimator framework components

First, the curve shapes consolidate our linear approach to calibrate checkpointing. Then, the significant differences between the checkpointing times of nodes from a same cluster demonstrate why we need to calibrate execution times for all the nodes.

Since checkpointing is considered at the system level, the coordinated checkpointing requires an extra synchronization between the processes. The time required for a process coordination is:

$$t^i_{coordination} = t^i_{synchro} + t^i_{polling} = t^i_{synchro} + \frac{V_{data}}{R^i_{transfer}}$$

$t_{synchro}$ is the time needed to exchange a marker among all the processes. It is calibrated by measuring 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). $t^i_{polling}$ is the time necessary to finish transfers of inflight messages. It is calibrated by measuring for different message size values, the mean transfer time of this message. $R^i_{transfer}$ is the transfer rate of the network infrastructure used.

3. Estimation methodology

Once this calibration is completed, the energy framework can estimate the energy consumption of fault tolerance protocols. The user provides information related to the execution context and to the application he wants to run. This information is taken as an input by the calibrator. As an output, the calibrator provides the calibration data on which the framework relies on to estimate the energy consumed by fault tolerance protocols. Figure 2 shows the components of our framework and their interactions.

To estimate the energy consumed by checkpointing, the estimator 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 V^{mean}_{mem} required by each node. The estimator collects also the number of checkpoints to perform during the application execution. Besides, it collects from the calibrator the checkpoint times corresponding to the calibrated checkpoint sizes. The estimator calculates the checkpoint times t^i_{ckpt} corresponding to V^{mean}_{mem}. If V^{mean}_{mem} is not a size recorded by the calibrator, the estimator computes the equation that gives t^i_{ckpt} according to V^{mean}_{mem}, and adjusts the equation using the method of least squares.

To estimate the energy consumed by message logging, the estimator 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, it computes the mean volume of data V^{mean}_{data} sent (so logged) by each node. Similarly to checkpointing, it 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.

To estimate the energy consumed by coordination, the estimator uses the mean message size $V^{mean}_{message}$ as the total size of messages divided by the total number of messages. It 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, it 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. Similarly to checkpointing, the estimator calculates the message transfer time $t^i_{polling}$ corresponding to the mean message size $V^{mean}_{message}$.

The estimated energy of one basic operation *op* (checkpointing, logging, polling or synchronization) is:

$$E_{op} = \sum_{i=1}^{N} e^i_{op} = \sum_{i=1}^{N} p^i_{op} \cdot t^i_{op}$$

The total estimated energy consumption of checkpointing is obtained by multiplying E_{ckpt} by the number of checkpoints C. The estimator calculates the estimated energy of all coordinations as follows: $E_{coordinations} = C \cdot (E_{polling} + E_{synchro})$.

To estimate the energy consumed by hierarchical checkpointing, the estimator collects from the user the composition of each cluster (i.e the list of processes in each cluster).

4. Conclusion

This paper presents a framework that estimates the energy consumption of three families of fault tolerance protocols: coordinated, uncoordinated and hierarchical protocols. To provide accurate estimations, it relies on an energy calibration of the considered 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 monitored platform.

To obtain a coherent global state, checkpointing is combined with message logging in uncoordinated protocols and with coordination in coordinated protocols. Therefore, to compare the energy consumed by coordinated and uncoordinated protocols, we compare the energy cost of coordinations to message logging. By providing energy estimations before pre-executing the HPC application, we can select the less energy consuming fault tolerant protocol. As a future work, we plan to extend our framework to more services needed at extreme-scale [2] such as data management protocols.

References

[1] MM. Diouri et al. Energy considerations in checkpointing and fault tolerance protocols. In *FTXS 2012*, Boston, MA, USA, June 2012.

[2] MM. Diouri, O. Glück and L. Lefèvre. Towards a novel smart and energy-aware service-oriented manager for extreme-scale applications. In *PFGC 2012*, San Jose, CA, USA, June 2012.

[3] T. Ropars et al. On the use of cluster-based partial message logging to improve fault tolerance for MPI HPC applications. In *Euro-Par 2011*, Bordeaux, France, 2011.