Some Issues to Consider in the Management of Energy Consumption in HPC Systems with Fault Tolerance

Marina Morán[1] [0000−0002−6334−1190], Javier Balladini[1] [0000−0002−9769−7830], Dolores Rexachs[2] [0000−0001−5500−850X], and Enzo Rucci[3] [0000−0001−6736−7358]

[1] Universidad Nacional del Comahue, Neuquén, Argentina
{marina,javier.balladini}@fi.uncoma.edu.ar
[2] Universitat Autònoma de Barcelona, Bellaterra, España
dolores.rexachs@uab.es
[3] Universidad Nacional de la Plata - CIC, La Plata, Argentina
erucci@lidi.info.unlp.edu.ar

Abstract. Inquiring about different ways to reduce energy consumption during the execution of large-scale applications is essential to maintain and increase the enormous computing power achieved in HPC systems. Fault tolerance methods can have an impact on power consumption. In particular, rollback-recovery methods using uncoordinated checkpoints prevent all processes from re-executing in the event of a failure. In this context, it is possible to take actions on the nodes of the processes that do not re-execute to reduce energy consumption. In this work, we describe some issues to consider when we extend the application of energy-saving strategies beyond the nodes that communicate directly with the failed one.

Keywords: Energy consumption · Fault tolerance · Uncoordinated checkpoints · HPC.

1 Introduction

The energy consumption of supercomputers and HPC systems continues to be a central topic of research. Inquiring about different ways to reduce energy consumption during the execution of large-scale applications is central to maintaining and increasing the enormous computing power achieved.

In parallel HPC message-passing applications, it is mandatory to use some fault tolerance method, which ensures the progress and completion of the application. The most widely used method today is rollback recovery, using coordinated checkpoints. With this method, the application stops every certain period, all the processes perform the checkpoint in a coordinated way and then continue executing. When there is a node failure, all nodes must roll back and resume execution from the last checkpoint. Although this method is widely used due to its low complexity, it is also true that it can imply great pressure on the storage system due to the simultaneous access of all the processes. A method that relaxes this limitation is the method known as uncoordinated or semi-coordinated checkpoints. In this method, the processes checkpoint independently, and consistency when recovering from a failure is achieved through some support, such as the message log. This method can reduce concurrent accesses to the file system. In addition, only the processes of the node that has failed must be restarted, which prevents all resources perform duplicate tasks. In previous work [6], we evaluate a series of strategies that can be applied to improve energy efficiency when a failure occurs, considering uncoordinated checkpoints. The strategies use the Advanced Configuration and Power Interface (ACPI), in particular we consider the processor P-State that uses the dynamic voltage and frequency scaling (DVFS) techniques and system sleeping states (system states S1 to S4), that is, system hibernation at the node level. By having a characterization of the energy consumption required to execute the application, as in [5], and its communication pattern, we estimate the execution and waiting times of the processes that do not fail. Then, by using a simulator that we have designed and developed, we can evaluate the use of the strategies.

By analyzing the wave of failure propagation, the number of nodes that receive the application of the strategies can be increased. We call this *cascade analysis* and it seeks to extend the application of strategies beyond the processes/nodes that communicate directly with the node where the failure occurred. The greater the number of nodes that receives some strategy, the greater the energy savings. In this work, we consider some aspects to take into account when analyzing cascade blocks.

The rest of the article is organized as follows: Section 2 presents some related works; Section 3 describes the strategies proposed to achieve energy savings, and Section 4 discusses some aspects to take into account in the cascade analysis. Finally, Section 5 presents the conclusions and future work.

2 Related Work

Some works have studied how to take advantage of the waits of processes that do not roll back when a failure occurs. In [1], the authors seek to improve the efficiency of the computer system by replacing the application when the waits are long enough. In [3], the authors simulate a load of an HPC system to evaluate the energy savings when activating and deactivating nodes according to the computational and power requirements of the cluster. Other works slow down the non-critical path to consume less power without substantially increasing execution time [4,7]. [2] can be considered the most similar proposal to this work, since they propose a localized rollback based on the data flow, and reduce the clock frequency of the waiting processes to the minimum possible. We evaluate other strategies, in addition to changing to the minimum frequency, and we do so both for the computation and waits of the processes that continue to execute.

3 Energy-saving Strategies

A series of strategies are defined to be applied to the surviving nodes after the failure, in a parallel message-passing application, running in a homogeneous cluster, which uses the uncoordinated checkpoint fault tolerance method. The failures considered are permanent node failures which in environments using MPI are fail-stop. Fig. 1 shows two processes, P1 and P2, running on different nodes. The green area means regular execution of the application; the red area means that the process is blocked for communication; the blue area means re-execution caused by the failure; while yellow squares indicate that the process is performing checkpoints. When the node where P1 is running fails (indicated by the yellow star), P2 will be affected by the failure because it will have to wait for P1 for a proportional time to its recovery one (recovery time is indicated in blue in the figure). At this time, the strategies to be applied to the computing and waiting phase (indicated in the figure) of surviving process P2 are evaluated.

The strategies are: (1) frequency change for the computational phase, (2) frequency change for the waiting phase, and (3) sleeping for the waiting phase. When the selected strategy is to change the clock frequency, it is applied to all cores of the node.

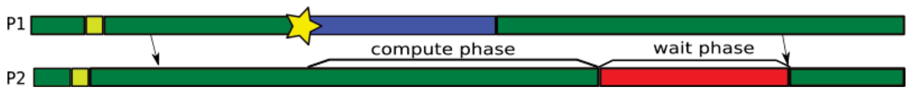

Fig. 1. Computing and waiting phases

4 Cascading Analysis Considerations

In the past, we have developed an event-based simulator to assess the different scenarios that involve the application of the strategies [6]. For the selection of the strategy to apply to each node, it is necessary to know the time until processes of the node get blocked (computing phase) and the duration of that blocking (waiting phase). The simulator results show us that, in general, the duration of the waiting phase defines the action to be applied to each node. If the waiting phase lasts long enough to put the node to sleep, this will be the preferred option, since the savings obtained largely exceed the benefits of the rest. With "long enough to sleep the node" we mean that the wait has a duration greater than the sum of the time it takes for the node to sleep and to wake up plus a configurable fraction of time.

Sleeping a node affects all processes on the node. This is why it is necessary to identify which process will be the one that defines the strategy. In applications with a single process (and multiple threads), there is a single option. In tightly coupled applications, all processes on the node will be blocked nearly at the same time, and the selected strategy can be applied without a problem. In applications where there are substantial differences in the computing phases of processes that shares the same node, sleeping the node could affect the execution of the other processes.

To which processes is it appropriate to apply the strategy? A first approach is to apply the strategies to the processes that are blocked by communication with the failed process. But it is possible to increase energy savings if we apply the strategies to a larger set of processes that are also affected by the failure, even if they do not communicate directly with the failed process. We then consider analyzing the processes that gets blocked with processes that are already blocked due to the failure. We call this *cascade blocking*. Including this aspect obviously increases the complexity of the algorithm that estimates the computing and waiting phases. Besides, we call *parent process* to the one responsible for blocking the child process. When we analyze processes that are blocked in cascade, some situations may arise:

– The child process communicates several times with its parent before reaching the communication that will block him due to the failure. We define *depth* as the number of subsequent communications to analyze when looking for one that gets blocked due to the failure. The depth can be calculated by looking at the communication pattern between each pair of processes and choosing the maximum amount of communications found (there will never be more communications than that). It would also be possible to define the depth by process, and apply the corresponding one in each case. Here again, we need to check the duration of the recovery. If the recovery time of the failed processes will be greater than a certain threshold, we can ensure that the nodes will sleep. In this case it would not be necessary to consider the depth or make an estimate of the moment when the blocking will occur, since we can mark the action that the node should take, and when that block arrives apply it.
– The parent process has been slowed down (it received the application of a strategy that was to decrease the clock frequency in the computation phase) but the child has not, or it has been slowed down in a different way. This will affect the communication time between the processes, probably generating unexpected waiting pahses.
– If the application uses non-blocking MPI operations, it could happen that a process starts a non-blocking send and goes to sleep (by applying the strategy) before data is transmitted. In this case, when the receiving process issues the matching receive, the communication cannot be completed. But as the node is hibernating, the second process will not be able to complete the communication and will block before expected. A possible solution to this is to wake up the node. In this way, the buffers with the messages become available.

Fig. 2 shows a scenario of 8 processes that communicate in a pipeline manner (each process sends data to the next one) in order to observe the effect of incorporating the cascade analysis. Fig. (a) shows the case where cascade are not supported, and Fig. (b) shows the case where it is used (depth = 2). The red blocks are waiting times when the node is awake, and the gray blocks are waiting times when the node sleeping. In case (a), the strategies are applied to a single node, while in case (b) the same strategies are applied to the 7 "surviving" nodes. In this way, energy savings increase from 32,500 J to almost 207,000 J. This represents more than six times the number of joules saved in a similar period of time (almost 4 minutes). According to the energy model defined in [6], energy saving is defined as the difference between the joules consumed with and without the application of the strategies. Table 1 shows the results, including cascade with depth = 1. Note that with this depth, only one node is added to the analysis.

Table 1. Selected actions and energy savings

	Compute phase		Wait phase		Sum of the phases			
Node	Action	T (m)	Action	T (m)	T (m)	Save (J)	Save Rate (J/s)	Save (%)
No cascade								
1	No action	4.84	sleep	3.67	8.51	32,502.3	147.77	38.35
Cascade deep 1								
1	No action	4.84	sleep	3.67	8.51	32,502.3	147.77	38.35
2	No action	5.00	sleep	3.68	8.68	32,617.8	147.79	37.74
Cascade deep 2								
1	No action	4.84	sleep	3.67	8.51	32,502.3	147.77	38.35
2	No action	5.00	sleep	3.68	8.68	32,617.8	147.79	37.74
3	No action	5.17	sleep	3.69	8.86	32,764.1	147.82	37.12
4	No action	5.35	sleep	3.70	9.05	32,787.2	147.82	36.39
5	No action	5.52	sleep	3.70	9.22	32,841.1	147.83	35.77
6	No action	5.68	sleep	3.71	9.39	32,925.8	147.85	35.19
7	2.1 GHz	9.22	1.2 GHz	0.35	9.57	10,348.97	18.02	11.08

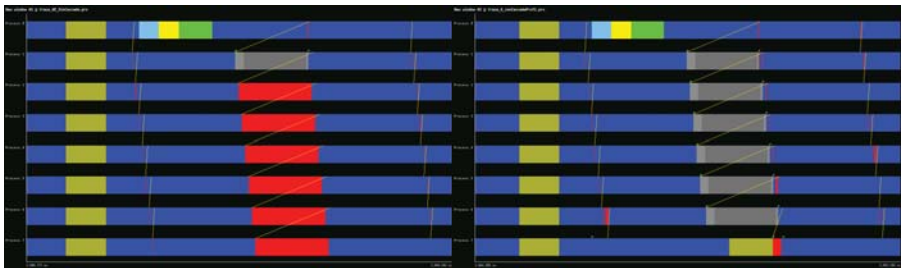

Fig. 2. Application of strategies: (a) Without cascade (b) With cascade

5 Conclusions and Future Work

Energy-saving opportunities exist in a rollback recovery scheme where only some processes must go back and re-execute. If the re-execution time is long and the possibility of sleeping the nodes gets enabled, the energy savings can be large. Depending on the communications pattern, the processes will be blocked sooner or later due to the failure. Extending the application of strategies beyond the nodes that communicate directly with the failed one can improve energy savings, as shown in this work. At the same time, determining which processes block and when they do so, are not easy tasks in applications with loosely coupled and non-homogeneous communication patterns.

Future work includes the analysis of new experimentation that considers non-blocking communications and cascade analysis, and also the implementation of a proof of concept in a cluster.

References

1. Bouteiller, A., Cappello, F., Dongarra, J., Guermouche, A., Hérault, T., Robert, Y.: Multi-criteria checkpointing strategies: Response-time versus resource utilization. In: European Conference on Parallel Processing. pp. 420–431. Springer (2013)
2. Dichev, K., Cameron, K., Nikolopoulos, D.S.: Energy-efficient localised rollback via data flow analysis and frequency scaling. In: Proceedings of the 25th European MPI Users' Group Meeting. pp. 1–11 (2018)
3. Dolz, M.F., Fernández, J.C., Iserte, S., Mayo, R., Quintana-Ortí, E.S.: A simulator to assess energy saving strategies and policies in hpc workloads. In: ACM SIGOPS Operating Systems Review. vol. 46, pp. 2–9. ACM New York, NY, USA (2012)
4. Hajiamini, S., Shirazi, B., Crandall, A., Ghasemzadeh, H.: A dynamic programming framework for DVFS-based energy-efficiency in multicore systems. IEEE Transactions on Sustainable Computing **5**(1), 1–12 (2019)
5. Morán, M., Balladini, J., Rexachs, D., Luque, E.: Prediction of energy consumption by checkpoint/restart in HPC. IEEE Access **7**, 71791–71803 (2019)
6. Morán, M., Balladini, J., Rexachs, D., Rucci, E.: Towards management of energy consumption in hpc systems with fault tolerance. In: 2020 IEEE Congreso Bienal de Argentina (ARGENCON). pp. 1–8. IEEE (2020)
7. Rountree, B., Lowenthal, D.K., De Supinski, B.R., Schulz, M., Freeh, V.W., Bletsch, T.: Adagio: making dvs practical for complex hpc applications. In: Proceedings of the 23rd international conference on Supercomputing. pp. 460–469 (2009)