RADIC based Fault Tolerance System with Dynamic Resource Controller*

Jorge Villamayor[1 [0000−0002−1729−037X]], Dolores Rexachs[1 [0000−0001−5500−850X]], and Emilio Luque[1 [0000−0002−2884−3232]]

CAOS - Computer Architecture and Operating Systems
Universidad Autónoma de Barcelona
Barcelona, Spain
{jorgeluis.villamayor,dolores.rexachs,emilio.luque}@uab.cat

Abstract. The continuously growing High-Performance Computing requirements increments the number of components and at the same time failure probabilities. Long-running parallel applications are directly affected by this phenomena, disrupting its executions on failure occurrences. MPI, a well-known standard for parallel applications follows a fail-stop semantic, requiring the application owners restart the whole execution when hard failures appear losing time and computation data. Fault Tolerance (FT) techniques approach this issue by providing high availability to the users' applications execution, though adding significant resource and time costs. In this paper, we present a Fault Tolerance Manager (FTM) framework based on RADIC architecture, which provides FT protection to parallel applications implemented with MPI, in order to successfully complete executions despite failures. The solution is implemented in the application-layer following the uncoordinated and semi-coordinated rollback recovery protocols. It uses a sender-based message logger to store exchanged messages between the application processes; and checkpoints only the processes data required to restart them in case of failures. The solution uses the concepts of ULFM for failure detection and recovery. Furthermore, a dynamic resource controller is added to the proposal, which monitors the message logger buffers and performs actions to maintain an acceptable level of protection. Experimental validation verifies the FTM functionality using two private clusters infrastructures.

Keywords: High-Performance Computing · Fault Tolerance · Application Layer FT · Sender-Based Message Logging.

1 Introduction

The constantly increasing scale of High Performance Computing (HPC) platforms increments the frequency of failures in clusters and cloud environments

* This research has been supported by the MICINN/MINECO Spain under contracts TIN2014-53172-P and TIN2017-84875-P.

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93713-7_58

[1]. In contemporary HPC systems the Mean Time Between Failures (MTBF) is in range of hours, depending on the maturity and age of installation [4]. Users employing this kind of systems to run their parallel and distributed applications are directly affected. Usually, parallel and distributed applications are implemented using a Message Passing Interface (MPI), which by default follows fail-stop semantics and lacks failure mitigation, meaning the loss of user's computation time and data when failure occurs.

Fault Tolerance (FT) solutions are necessary to ensure high availability to parallel applications execution and minimize the failure impact [3]. Rollback-Recovery protocols are widely used within FT to protect application executions. The methods consist of snapshots created from the parallel execution and stored as checkpoints. In case of failures an application can recover using the last taken checkpoint, though most of the time the recovery process is not automatic, and may require human intervention. A well-known issue is that most FT solutions comes with added overhead during failure-free executions, and also with high resource consumption.

Coordinated, semi-coordinated and uncoordinated are some of the most used rollback-recovery protocols [2]. The coordinated protocol synchronizes the application processes to create a consistent state. For applications with large amount of processes, the coordination may present a source of overhead. Furthermore, when failures appear, all application processes must rollback to the last checkpoint causing waste of computation work [2]. In order to avoid this problem, the semi-coordinated and uncoordinated protocols make use of a message logger facility that allows the recovery of only affected processes when failures appear. However, the logger have to store each interchanged message during the application execution, meaning a significant source of resource usage.

In this work, a Fault Tolerance Manager (FTM) for HPC application users is presented. FTM offers automatic and transparent mechanisms to recover applications in case of failures, meaning users do not need to perform any action when failures appear. The solution uses semi-coordinated and uncoordinated rollback-recovery protocols following RADIC architecture. FTM combines application-layer checkpoints with a sender-based message logger using the concepts of ULFM for detection and recovery purposes. Furthermore, a Dynamic Resource Controller is added, it performs the monitoring of main memory usage for the logger facility, allowing to detect when its usage is reaching a limit. With this information, it invokes automatic checkpoints, in an optimistic manner, which allows freeing memory buffers used for the message logger avoiding the slowdown or stall of the application execution.

The content of this paper is organized as follows: Section 2 presents the design of FTM in the application-layer with the dynamic resource controller. In Section 3, experimental evaluation is shown, which contains the FTM functionality validation and the dynamic resource controller verification with the NAS CG benchmark application. Finally the conclusions and future work are stated in Section 5.

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93713-7_58

2 FTM with Dynamic Resources Controller

This section describes the design of the Fault Tolerance Manager (FTM) to provide high availability to user's applications. The traditional stack for HPC execution environment is composed of several failure prompt layers. FTM architecture isolates the user's application layer from failures. It suits the execution environment with a handler controller, which deals with failures recovering the user's application execution when failures appear. The architecture components are depicted in Fig. 1.

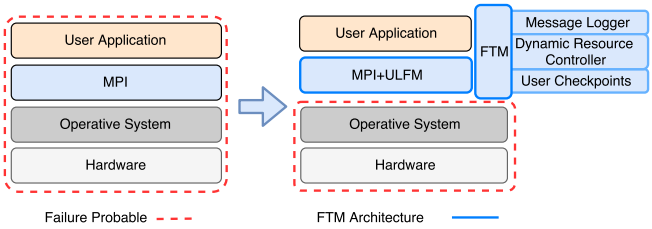


Fig. 1. FTM architecture.

The Fault Tolerance Manager uses application-layer checkpoints combined with a sender-based message logger, following the uncoordinated and semi-coordinated rollback-recovery protocols, to protect the application during failure-free executions. In order to monitor and manage the FTM resource usage for FT protection, a dynamic resource controller is also added.

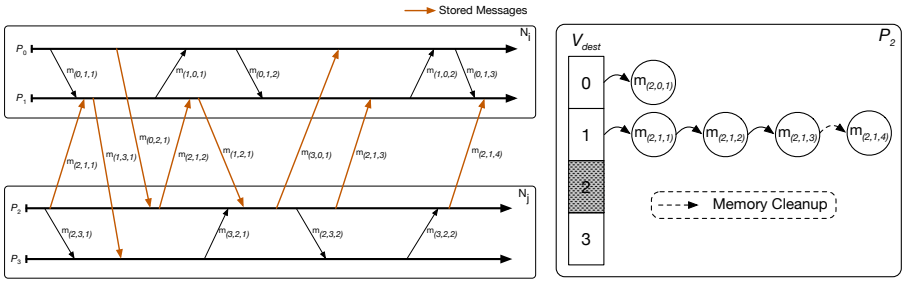
Checkpointing operation is initiated by the application, hence modifications in the application's source are needed, although the application algorithm remains intact. The checkpoint operations are inserted during natural synchronization of the application processes. The checkpoints store structures containing only necessary information to restore execution in case of failures, avoiding the need to store SO particular information. The checkpoint files are used when the application is recovered from a failure.

Exchanged application's messages are stored into the message logger facility, in order to replay them to the processes affected by failures. After the processes are restarted, they directly consume messages from the logger. For the uncoordinated approach, all exchanged messages between the application processes are stored. The semi-coordinated approach stores only exchanged messages between the application processes that are in distinct nodes, as shown in Fig. 2.

When failures appear, a mechanism of detection is needed to start the recovery procedure. In this work, ULFM is used to detect failures. FTM implements an error handler, which is invoked by the ULFM detection mechanism to recovery the application execution.

The failure detection, reconfiguration and recovery procedures are implemented as a handler in FTM. To illustrate the procedures, one process per node

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93713-7_58



(a) Semi-coordinated stored messages (red arrows). (b) Messages storage structure.

Fig. 2. Sender-Based Message Logger.

is used, with the uncoordinated protocol and a pessimistic sender-based message logger, shown in Fig. 3. It depicts that P_3 fails, and P_2 is the first process which detects the failure, causing the revocation of the global communicator using MPI_Comm_revoke. After the revocation, all remaining processes are notified and they shrink the global communicator, taking out the failed processes using the MPI_Comm_shrink call. Finally, the remaining processes spawn the communicator using dynamically launched processes of the application.

After the reconfiguration, the affected processes load the checkpoint and jump to the correct execution line in order to continue the application execution. The messages are consumed from the message logger to finish the re-execution. Meanwhile, non-failed processes continue their execution.

2.1 Dynamic Resources Controller

As previously seen, the FT protection requires resources and it comes with overhead for the user's applications. The protection of the application execution stores both, checkpoints and messages of the application processes. The uncoordinated and semi-coordinated protocols avoid the restart of all the application processes when a failure occurs. Although, they require a logger facility to replay

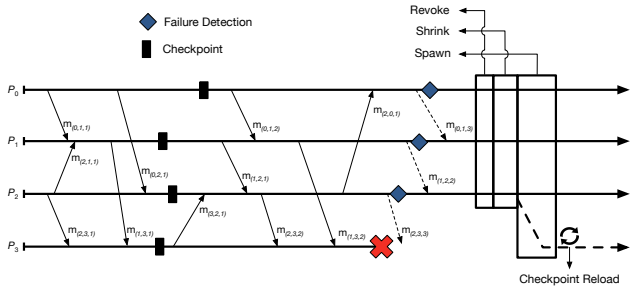
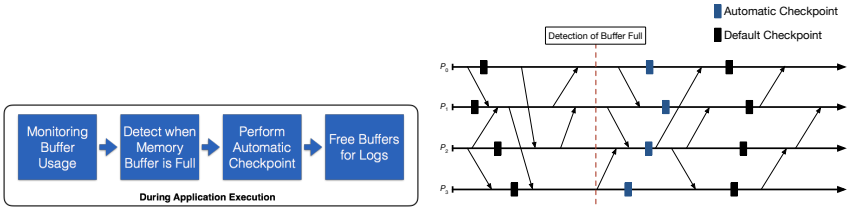


Fig. 3. Failure detection, reconfiguration and recovery procedures.

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93713-7_58



(a) Process schema. (b) Functionality example.

Fig. 4. Dynamic Resources Controller

messages to restored processes. To reduce failure-free overhead, the logger uses main memory to store processes messages. The main memory often provides high speed access compared to local hard disk or a centralized storage. However, the main memory is a limited resource, which is shared between the application processes and the FT components. The usage of FT resources is application dependent, meaning different communication pattern, size and quantity of messages, directly impacts on FT resource usage. The free available memory can rapidly ran out due to FT protection tasks. The impact of running out of main memory can result in the application execution stall.

In order to avoid the free memory ran out due to FTM protection task, the dynamic resources controller is introduced with FTM. It works on each node of the application execution. This controller constantly monitors the memory buffers usage for message logging and detects when they are reaching the limit available, triggering automatically a checkpoint invocation, storing the state of the application and freeing the memory buffers used for logging purposes, providing the application FT protection without interfering its memory usage (Fig. 4(a)). The functionality is shown in Fig. 4(b). By default the application has checkpoint moments, which are chose by the application users, though the memory may ran out meaning the lost in terms of performance. The controller detects it and automatically invokes a checkpoint creation, allowing to free the memory usage by the logging facility, therefore avoiding the application execution stall due to the lack of main memory.

3 Experimental Results

This section presents experimental results obtained applying FTM to provide Fault Tolerance in the application-layer. The results show its automatic functionality and validate the dynamic resources controller in real execution environments and injecting failures.

The application used for the experiments is the NAS CG Benchmark. It is an implementation of the Conjugate Gradient method included in the NAS benchmark suite. The experiments are performed using Class C and D which have 150000 and 1500000 rows respectively. Two clusters were used: AOCLSB-FT, built with 2 quad-core Intel 2.66GHz and 16GB RAM; and AOCLSB-L

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93713-7_58

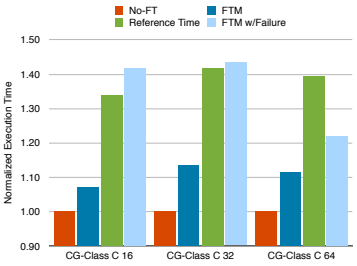


Fig. 5. CG-Class C with FTM in a failure injection scenario in AOCLSB-FT cluster.

suited with 8 AMD Opteron (x8) and 252GB RAM, both clusters with 30GB HDD of local disk, and a NFS shared volume of 715GB.

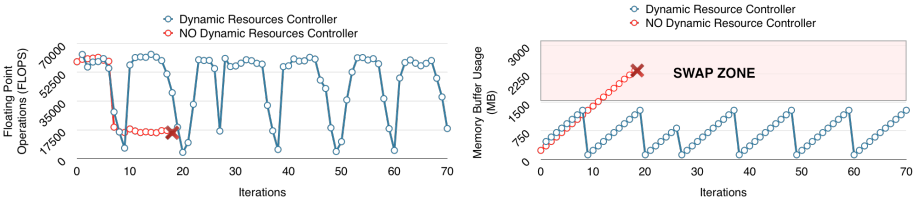
The performance of FTM is tested applying it to the CG application. During the experiments a failure is injected to one node of the cluster at 75% of the application execution. To analyze the benefits of applying the solution, a Reference Time is calculated, which represents the scenario where a failure is injected at 75% of the application execution. As the application checkpoints at 50% of its execution, 25% of the execution is lost. This means a total execution time of around 125% in case of failures and without applying FTM. This time compared to the measured time of executing the application with FTM protection experimenting a failure (FTM w/Failure). FTM protection is setup to take checkpoints at 50% of the application execution.

Figure 5, shows the results of the execution time normalized to the execution without Fault Tolerance (No-FT). The experiments were done using 16, 32 and 64 processes with the CG Class C. It is possible to observe that having FTM protection save user time approximately 13% compared to the Reference Time when a failure appears for the 64 processes application.

An evaluation of the dynamic resource controller is also performed, executing CG Class D application in the AOCLSB-FT cluster, which has less resources compared to the AOCLSB-L, and configured to take one checkpoint at 50%. Two scenarios were evaluated, with and without the dynamic controller. Figure 6(a) shows both executions starting with a similar throughput, though when the execution without the dynamic controller starts using SWAP memory zone of the system, the throughput drastically drops, making the whole application crash. Meanwhile, the execution with the dynamic controller, optimistically invoke the checkpoints, that after their completion, release the memory buffers used for the logger facility, allowing the continuous execution of the application.

It is important to remark that the dynamic controller does not interfere in the user-defined checkpoint events, letting users the control of the checkpoint moments, though it may perform optimistic checkpoints, to free resources for the application. The solution allows the application to continue the execution, though it may come with larger overhead, due to the automatic checkpoints invocation. Figure 6(b) shows how the memory is managed during the application execution in contrast to the execution without the management.

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93713-7_58



(a) Application process throughput. (b) Memory usage monitoring.

Fig. 6. CG-Class D execution in the AOCLSB-FT cluster.

4 Conclusion and Future Work

This work contributes providing a novel Fault Tolerance Manager in the application-layer, allowing users to define only the necessary protection information for their applications. Furthermore, the work suits FTM with a dynamic resource controller, which monitor FT resource usage and perform actions when the usage reach boundaries where it may affect the application execution.

Experiments show the automatic functionality of the FTM applied to the NAS CG, a well-known benchmark application. Results show up to 13% of execution time benefits by applying the solution. During the experiments, throughput measurements of the application processes shows the operation of the dynamic controller, which performing optimistic checkpoints, allows the application maintain its throughput, keeping the FT protection in case of failures. Future work aims to evaluate the FTM with the dynamic resource controller for real applications in different execution environments, such as cloud and containers.

References

1. Cappello, F., Geist, A., Gropp, W., Kale, S., Kramer, B., Snir, M.: Toward Exascale Resilience: 2014 update. Supercomputing Frontiers and Innovations **1**(1), 5–28 (sep 2014). https://doi.org/10.14529/jsfi140101, http://superfri.org/superfri/article/view/14
2. Castro-León, M., Meyer, H., Rexachs, D., Luque, E.: Fault tolerance at system level based on RADIC architecture. Journal of Parallel and Distributed Computing **86**, 98–111 (8 2015). https://doi.org/10.1016/j.jpdc.2015.08.005, http://www.sciencedirect.com/science/article/pii/S0743731515001434
3. Egwutuoha, I.P., Levy, D., Selic, B., Chen, S.: A survey of fault tolerance mechanisms and checkpoint/restart implementations for high performance computing systems. The Journal of Supercomputing **65**(3), 1302–1326 (2 2013). https://doi.org/10.1007/s11227-013-0884-0, http://link.springer.com/10.1007/s11227-013-0884-0
4. Wang, C., Vazhkudai, S., Ma, X., Mueller, F.: Transparent Fault Tolerance for Job Input Data in HPC Environments (2014), http://optout.csc.ncsu.edu/ mueller/ftp/pub/mueller/papers/springer14.pdf

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93713-7_58