Future Generation Computer Systems 51 (2015) 7–19

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

On the impact of process replication on executions of large-scale parallel applications with coordinated checkpointing

Henri Casanova [a], Yves Robert [b,c], Frédéric Vivien [b], Dounia Zaidouni [b,*]

[a] University of Hawaii at Manoa, Honolulu, USA
[b] ENS Lyon & INRIA, France
[c] University of Tennessee Knoxville, USA

HIGHLIGHTS

- Process replication combined with checkpoint–rollback–recovery.
- Exact values for the Mean Number of Failures To Interruption and the Mean Time To Interruption for Exponential failure distributions.
- Closed-form expression for the Mean Time To Interruption for Weibull distributions.
- Scenarios where replication is beneficial.

ARTICLE INFO

Article history:
Received 24 March 2014
Received in revised form
21 February 2015
Accepted 5 April 2015
Available online 27 April 2015

Keywords:
Fault-tolerance
Parallel computing
Checkpoint
Rollback–recovery
Process replication

ABSTRACT

Processor failures in post-petascale parallel computing platforms are common occurrences. The traditional fault-tolerance solution, checkpoint–rollback–recovery, severely limits parallel efficiency. One solution is to replicate application processes so that a processor failure does not necessarily imply an application failure. Process replication, combined with checkpoint–rollback–recovery, has been recently advocated. We first derive novel theoretical results for Exponential failure distributions, namely exact values for the Mean Number of Failures To Interruption and the Mean Time To Interruption. We then extend these results to arbitrary failure distributions, obtaining closed-form solutions for Weibull distributions. Finally, we evaluate process replication in simulation using both synthetic and real-world failure traces so as to quantify average application makespan. One interesting result from these experiments is that, when process replication is used, application performance is not sensitive to the checkpointing period, provided that period is within a large neighborhood of the optimal period. More generally, our empirical results make it possible to identify regimes in which process replication is beneficial.

1. Introduction

As plans are made for deploying post-petascale high performance computing (HPC) systems [1,2], solutions need to be developed for ensuring resilience to processor failures. Resilience is particularly critical for applications that enroll large numbers of processors. For such applications, processor failures are projected to be common occurrences [3–5]. For instance, the 45,208-processor Jaguar platform is reported to experience on the order of 1 failure per day [6], and its scale is modest compared to upcoming platforms. Failures occur because not all faults are automatically detected and corrected in current production hardware, due to both technical challenges and high cost. To tolerate failures the standard approach is to use rollback and recovery for resuming application execution from a previously saved fault-free execution state, or *checkpoint*, which is termed checkpoint–rollback–recovery (CRR). Frequent checkpointing leads to higher overhead during fault-free execution, but less frequent checkpointing leads to a larger loss when a failure occurs. A large volume of literature is devoted to CRR, including both theoretical and practical results. The former typically rely on assumptions regarding the probability distributions of times to failure of the processors (e.g., Exponential, Weibull), while the latter rely on simulations driven by failure datasets obtained on real-world platforms.

Even assuming an optimal checkpointing strategy, at large scale processors end up spending as much or even more time saving

* Corresponding author.
E-mail addresses: henric@hawaii.edu (H. Casanova), Yves.Robert@ens-lyon.fr (Y. Robert), Frederic.Vivien@ens-lyon.fr (F. Vivien), Dounia.Zaidouni@ens-lyon.fr (D. Zaidouni).

http://dx.doi.org/10.1016/j.future.2015.04.003
0167-739X/

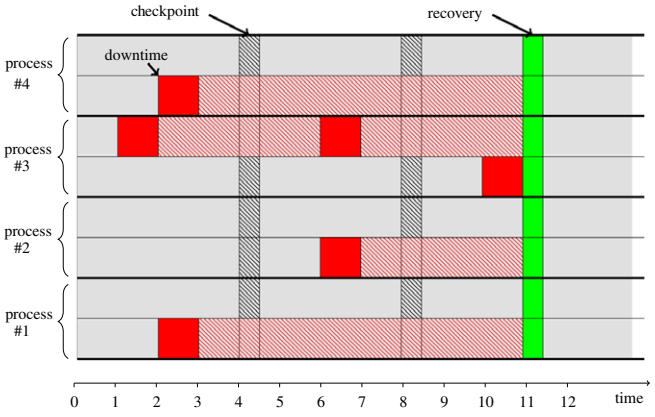

Fig. 1. Gantt chart for a example application execution with process replication on 8 processors (4 logical MPI processes with one replica per process). A gray fill indicates that a processor is computing. A red fill denotes when a process is experiencing a downtime. Red hatches indicate when a processor is not longer participating in the application execution. Black hatches indicates when a processor is involved in saving a checkpoint. A green fill indicates when a processor is involved in a recovery. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

state than computing state, leading to poor parallel efficiency [3–5]. Consequently, additional resilience mechanisms must be used. In this work we focus on *replication*: several processors perform the same computation synchronously, so that a failure of one of these processors does not lead to an application failure. Replication is an age-old fault-tolerance technique, but it has gained traction in the HPC context only relatively recently [5,7,8].

While replication wastes compute resources in fault-free executions, it can alleviate the poor scalability of CRR. With *process replication*, a single instance of an application is executed but each application process is (transparently) replicated. For instance, instead of executing the application with $2n$ distinct processes on a $2n$-processor platform, one executes the application with n *logical* processes so that there are two replicas of each process, each running on a distinct *physical* processor.

We illustrate this approach on an example in Fig. 1 for a platform with 8 processors running an application with 4 MPI processes so that each process has one replica (i.e., 8 processes in total). At time 1, one of the replicas for process #3 experiences a failure and a downtime (of 1 s). Because the other replica is still running, the application does not fail. At time 2, a replica of process #1 and a replica of process #4 each experience a failure, but once again the application can continue running. At time 4, a checkpoint is saved (perhaps as dictated by a periodic checkpointing strategy), which is possible because at least one replica of each process is running. At time 6 two other failures occur but have no impact on the application execution because enough process replicas remain (one of these two failures occurs at a processor that is no longer participating in the application execution). Another checkpoint is saved at time 8. At time 10, a failure and downtime occur for the only remaining replica of process #3. This causes an application failure, and triggers a recovery. At time 11.5, all processors resume from a previous checkpoint, each with 2 replicas. In this example, for simplicity, we do not show failures during checkpointing and/or recovery, but we do allow such failures in this work.

Process replication is sensible because the mean time to failure of a group of two replicas is larger than that of a single processor, meaning that the checkpointing frequency can be lowered thus improving parallel efficiency. In [9] Ferreira et al. have studied process replication, with a practical implementation and some analytical results. In this paper, we focus on the theoretical foundations of process replication, and we make the following novel contributions:

- We derive exact expressions for the *MNFTI* (Mean Number of Failures To Interruption) and the *MTTI* (Mean Time To Interruption) for arbitrary numbers of replicas assuming Exponential failures.
- We extend these results to arbitrary failure distributions, notably obtaining closed-form solutions in the case of Weibull failures.
- We present simulation results, based on both synthetic and real-world failure traces, to compare executions with and without process replication. We find that with process replication application performance is not sensitive to the checkpointing period, provided that period is within a large neighborhood of the optimal period.
- Based on our results, we determine in which conditions the use of process replication is beneficial. We perform a *fair* comparison between the replication and the no-replication cases, i.e., our comparison is not impacted by the (critical) choice of a particular checkpointing period in the no-replication case.

The rest of this paper is organized as follows. Section 2 discusses related work. Section 3 states our assumptions and defines the process replication approach. Section 4 presents the bulk of our theoretical contribution. Section 5 presents our simulation methodology and empirical results obtained in simulation. Finally, Section 6 concludes the paper with a summary of our findings and directions for future work.

2. Related work

Checkpointing policies have been widely studied. In [10], Daly studies periodic checkpointing for Exponential failures, generalizing the well-known bound obtained by Young [11]. In [12] he studies the impact of sub-optimal checkpointing periods. In [13], Venkatesh develops an "optimal" checkpointing policy, based on the popular assumption that optimal checkpointing must be periodic. In [14], Bouguerra et al. *prove* that the optimal checkpointing policy is periodic when checkpointing and recovery overheads are constant, for either Exponential or Weibull failures. But their results rely on the unstated assumption that all processors are rejuvenated after each failure and after each checkpoint. In [15], Bougeret et al. show that this assumption is unreasonable for Weibull failures. They propose optimal solutions for Exponential failures and dynamic programming solutions for Weibull failures. Exponential failures are often assumed due to their convenient memoryless property, i.e., the fact that the time to the next failure does not depend on when the last failure has occurred [16]. But the non-memoryless Weibull distribution is recognized as a more realistic model [17–21]. The work in this paper relates to CRR in the sense that we study a replication mechanism that is complementary to checkpointing.

In spite of all the above advances, the feasibility of pure CRR for large-scale systems has been questioned [3–5]. Replication has long been used as a fault-tolerance mechanism in distributed systems [22]. Using replication together with CRR has been proposed in the context of grid computing [23]. One concern with replication is the induced resource waste. However, given the scalability limitations of pure CRR, replication has recently received more attention in the HPC literature [5,7,8]. The use of redundant MPI processes is advocated in [24] for HPC applications, and in [25] for grid computing with volatile nodes. The work by Ferreira et al. [9] has studied the use of process replication for MPI (Message Passing Interface) applications, using 2 replicas per MPI process. They provide a theoretical analysis of parallel efficiency, an MPI implementation that supports transparent process replication (including failure detection, consistent message ordering among replicas, etc.), and a set of experimental and simulation results. Partial redundancy is studied in [26,27] (in combination with coordinated