Chapter 9
Fault Tolerance Techniques for Distributed, Parallel Applications

Camille Coti
Université Paris 13, France

ABSTRACT

This chapter gives an overview of techniques used to tolerate failures in high-performance distributed applications. We describe basic replication techniques, automatic rollback recovery and application-based fault tolerance. We present the challenges raised specifically by distributed, high performance computing and the performance overhead the fault tolerance mechanisms are likely to cost. Last, we give an example of a fault-tolerant algorithm that exploits specific properties of a recent algorithm.

INTRODUCTION

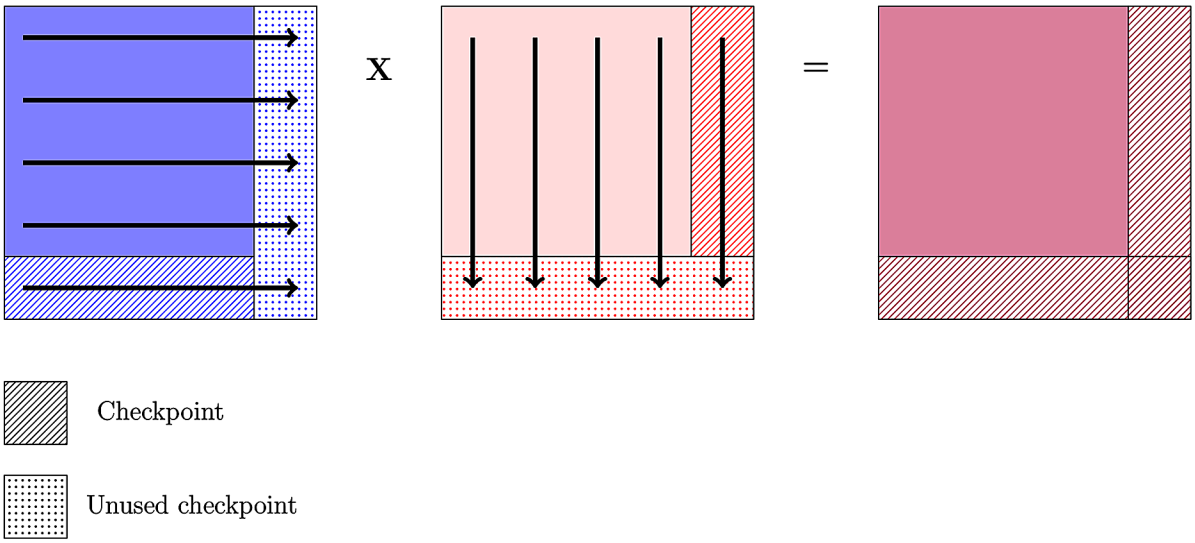
Current systems used for high performance computing feature growing numbers of components. Even with the most reliable components that can be produced by the industry, the larger the system is, the higher the failure probability is (Reed, Lu, & Mendes, 2006).

If failures are independent from each other, the law of total probability can be applied to compute the Mean Time Between Failures (MTBF) of a system. In this case, the mean time between failures is equal to the average time between failures. If the system is made of N components, each of which having a MTBF denoted $MTBF_i$ for component i, the global $MTBF$ of the system can be given by Equation 1:

$$MTBF_{total} = \left(\sum_{i=0}^{n-1} \frac{1}{MTBF_i} \right)^{-1} \quad (1)$$

DOI: 10.4018/978-1-5225-0287-6.ch009

Figure 1. Mean time between failures for systems that use components that have different individual MTBFs



By using Equation 1, one can compute the MTBF of a system with respect to its size for various individual MTBFs. Some of them are represented by Figure 1. We can see that, the MTBF being a hyperbolic function of the size of the system, it drops to only a few hours for large systems even with very reliable components. For instance, a system made components with an individual MTBF of 100,000 hours (which is very reliable) will have a global MTBF of 10 hours if it is made of 10,000 components, and only one hour if it is made of 100,000 components. As a comparison, in the 10 fastest machines of the June 2015 Top 500 list[1], all the machines feature more than 100,000 cores and 5 machines feature more than 500,000 cores.

As a consequence, failures are unexceptional events at large-scale and must be handled when they occur during a computation.

Originally, the default behavior of distributed run-time environment such as MPI (Message Passing Interface Forum (2004)) consisted in terminating the surviving processes and ending the application. Therefore, a computation that runs for longer than the MTBF of the system is unlikely to complete and all the computation is lost.

This observation motivates the need for systems that can handle and tolerate failures and make completion possible in spite of the volatile nature of the resources they are running on. On the other hand, in the context of high performance computing, the overhead induced by the fault-tolerance mechanisms must be as small as possible in order to maintain this performance goal.

Fault tolerance can be handled at two levels. It can be implemented in the distributed middleware that supports the execution of a distributed environment, making it transparent for the application. This is called *system-level* fault tolerance. Fault tolerance can also be implemented by the application itself. In this case the application support system must provide mechanisms to implement fault tolerance, but the way the computation survives beyond failures is actually implemented by the application itself. This is called *application-level* fault tolerance.