

Improving the Efficiency of Replication for Highly Reliable Systems

Masumi Toyoshima, Adel Cherif, and Takuya Katayama

School of Information Science, Japan Advanced Institute of Science and Technology
{adel,masumi,katayama}@jaist.ac.jp

Abstract

Fault Tolerance must be provided to increase system reliability. Combining efficiency with fault tolerance is a difficult task. Fault Tolerance requires the use of redundancy while efficiency requires the elimination of redundancy. Several fault tolerance techniques have been proposed in the literature to manage the redundancy existing in the system in order to provide fault tolerance. These techniques focus mainly on providing fault tolerance. We propose a technique named Active Parallel Replication (APR) for managing the redundancy existing in the system. The APR technique provides the means to achieve fault tolerance and efficiency.

I. INTRODUCTION

Our reliance on computer systems has drastically increased. Today's systems have numerous hardware and software components that interact together. These systems have a high complexity and require more computation resources such as memory and computation time. As a result, the risk of failure also increased. To justify our reliance on these systems, fault tolerance must be provided. Moreover, fault tolerance must be provided efficiently.

Distributed systems and parallel computing offer several advantages for designing and implementing highly reliable systems. Distributed systems allow for the distribution of the computation on different interconnected processing elements, while parallel computing allows for the partitioning of the computation into more fine grained computation elements, that could be computed in parallel.

The proposed Active Parallel Replication approach (APR) uses the redundancy existing in the distributed systems to provide fault tolerance while using parallel computing to provide efficiency [CK98].

Traditional approaches provide fault tolerance by replicating the computation on different groups of processing elements called replicas. In the active replication approach, also called the state machine approach [Sch93], all replicas are active and perform the same computation in the same order and at the same time. Results produced by all replicas are sent to a voter which selects the correct result to be used by all replicas. If an error occurs at a replica, the results produced by other replicas are used to mask this error. In the passive replication approach, also called primary/back-up approach [BMST 93], one replica, called the primary replica, is active while other replicas, called back-ups, are in a stand-by mode. In this approach, checkpoints are periodically established at the primary replica [KT87]. During a checkpoint, the state of the system is saved to stable storage [Cri85] and sent to the back-ups.

In case of failure of the primary, one of the back-ups is activated and computation is restarted from the last checkpoint.

In this abstract we briefly introduce the APR approach and its advantages. The APR approach increases system efficiency by executing independent computation elements, we call modules, in parallel at different replicas, and increases system reliability by validating modules' results after being computed by at least a majority of replicas.

II. THE APR APPROACH

The main goal of the approach is to provide fault tolerance transparently to the application programmer, to reduce the computation cost resulting from replica management and the activities necessary to support fault tolerance such as error detection and recovery, and to use the existing redundancy in the system not only in providing fault tolerance but also to improve system performance. In the proposed APR approach, all replicas are active. The approach lets the computation proceed in a different order at each replica. That is, each replica executes a module possibly different from the one being concurrently executed by other replicas. We call it Active Parallel Computation.

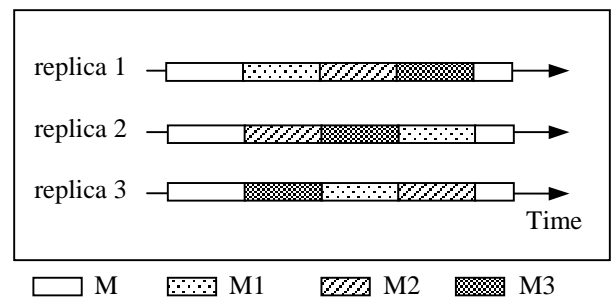


Figure 1. Active Parallel Computation.

Figure 1 shows how computation is expanding as a function of time at all replicas. We assume that there are three replicas and that the computation consists of a main module M that spans in parallel three independent modules M₁, M₂, and M₃. All replicas start executing module M, then the first replica starts computation of module M₁. After completing computation of module M₁, it will compute module M₂ and then module M₃. Concurrently, the second replica starts computation of module M₂, then it will compute module M₃ and finally will perform computation of module M₁. Concurrently to replica 1 and replica 2, the third replica starts the computation of module M₃, then it will compute module M₁ and finally will perform

computation of module M_2 . We say in that case that we have *active parallel computation*.

While performing active parallel computation, replicas communicate in order to synchronize and to detect and mask or recover from failures. After a replica completes computation of a module it sends its computed results to other replicas. Replicas store the received results. When a module is computed by a majority of replicas, its results are validated using voting. If a majority is found, results are validated. Replicas that did not yet perform the computation of the validated module will use the validated results without performing the module's computation. In doing so, the approach provides fault tolerance while increasing system efficiency.

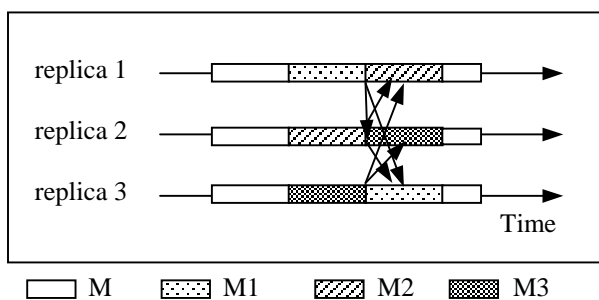


Figure 2. Active Parallel Replication.

Figure 2 shows that using the APR approach, and in the case where no failures occur, the overall computation time is reduced. Replica 1 for example did not have to compute module M_3 since this module was already computed by replica 2 and replica 3 and its results validated.

In case of failure, replicas that completed computation of the module where an error was detected continue normal computation using the results computed locally while awaiting other replicas that did not complete the faulty module's execution to send their results. When a majority is found, faulty replicas are identified. Faulty replicas recover by using the newly validated results and discard all computation of modules having dependencies with the faulty module. If no majority can be found, all replicas discard the module's computation and the computation with modules having dependencies with the faulty module. All replicas restart the computation of faulty modules after performing a system diagnosis in order to identify the origin of the error and system reconfiguration in order to remove faulty components.

III. IMPLEMENTATION

We are implementing the APR approach on a distributed system that consists of a number of nodes connected by a communication network. Each node consists of one or more processors connected by an interconnection network. There is no shared memory available between processors. Processors at each node have

access to a stable storage. A stable storage at one node cannot be accessed by processors at other nodes. Processes communicate solely by means of messages exchanged through the communication or the interconnection network.

When an application is first launched, it is replicated on a number of nodes in the distributed system. Each node where an instance of the replicated application is running is called a replica. The number of replicas in the system depends on the degree of parallelism existing in the application, and the number of faults that must be tolerated by the system.

Each replica in the system is assigned a system-wide unique identifier. The replica where the application was initiated is assigned the lowest replica-id and is called the primary replica. Processors at each replica are also assigned a system wide unique identifier. At each replica, one of the processors is designated as the leader. The leader processor manages replica coordination and cooperation. It handles all communication among replicas and schedules modules for execution at processors constituting the replica. Other processors at the replica are called application processors and only perform the computation of application modules.

IV. CONCLUSION

In this abstract, we presented the Active Parallel Replication approach. We are now implementing this approach using the Ocaml programming language. We are using a group communication layer offering atomic multicast and group membership services to the APR layer. The APR approach provides fault tolerance transparently to the application programmer and to the user. The approach implements fault tolerance while improving system throughput especially for highly parallel applications.

REFERENCES

- [BMST 93] N. Budhiraja, K. Marzullo, F.B. Schneider, and S. Toueg. *The Primary-Backup approach*. In Sape Mullender, editor, *Distributed Systems*, pp:199-216, ACM Press, 1993.
- [CK98] A. Cherif and T. Katayama. *Replica Management for Fault Tolerant Systems*. *IEEE Micro*, Vol.18, No.5, pp:54-65, 1998.
- [Cri85] F. Cristian. *A rigorous approach to fault-tolerant programming*. *IEEE Transactions on Software Engineering*, SE-11, No.1, 1985.
- [KT87] R. Koo and S. Toueg. *Checkpointing and rollback-recovery for distributed systems*. *IEEE Transactions on Software Engineering*, Vol.13, No.1, pp:23-31, 1987.
- [Sch93] F.B. Schneider. *Replication Management Using the State-Machine Approach*. In Sape Mullender, editor, *Distributed Systems*, pp:169-197, ACM Press, 1993