Implementation and evaluation of Collision-fast Atomic Broadcast protocols

Rodrigo Q. Saramago, Lasaro Camargos
Federal University of Uberlândia, Brazil
Email: rod@comp.ufu.br, lasaro@ufu.br

***Abstract*—State Machine Replication, a common technique for achieving fault tolerance, can be implemented by Atomic Broadcast primitives. Atomic Broadcast is usually implemented by solving infinitely many instances of the well-known consensus problem. This approach has the disadvantage of forcing messages broadcast concurrently, not decided in some instance, to be re-proposed in new instances, delaying execution. Collision-fast algorithms, which deliver many messages within two steps in good runs, exist, but either make assumptions that may be too restrictive; [1] requires a certain degree of clock synchronization among nodes; [2] does deal efficiently with failures; and [3] has not been experimentally evaluated. In this we propose an architecture to implement [3] a service, exploring the parallelism in today's machines, and also to compare it too the other two protocols, solving the issue mentioned above.**

I. Introduction

Atomic Broadcast is commonly used in implementation of state machine replication and frequently solved using infinite instances of consensus problem. Examples of such an approach are the Chubby Lock Service [4] and Zookeeper [5].

Although there are several Atomic Broadcast algorithms, they cannot guarantee optimal latency of two communication steps to all processes, due to concurrent proposals leading to collisions. Collisions, in turn, lead re-proposals, which lead to more collisions and accrue to longer delays.

There are algorithms that have latency of two communication steps to all processes, but only under certain conditions, and are impractical in most situations, as the proposed in [6], which requires that all but one of the Collision-fast proposer, those whose proposals may be decided in two steps, to be also an acceptor and still have the problem that only one value, among the proposed ones, may be decided per Consensus instance. [1] and [2] are collision-fast, but the first assumes synchronized clocks and the latter requires a restart to overcome node failures.

The other known collision-fast algorithm [3], does not suffer from the same drawbacks, but because it has not been implemented and experimentally evaluated, it will probably not have a great impact in academic and industrial communities. In this work, we propose to remove such a limitation. Since State Machine Replication is a widely used method to achieve redundancy and increased service availability, the existence of efficient Atomic Broadcast algorithms is paramount to a whole range of applications; in this work we will verify if [3] is such an algorithm.

In summary, the contributions of this work in progress is twofold. First, we propose a real world implementation of the [3] algorithm. Second, we compare performance against [1] and [2].

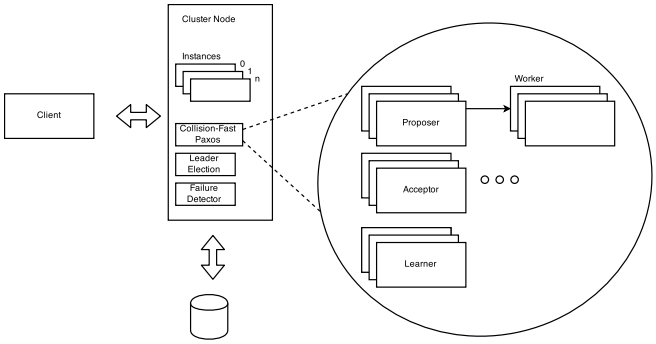

Fig. 1. CFABCast member

II. Contribution

The first major contribution of this work in progress is an implementation of CFABcast [3] using an architecture that is close to the abstract algorithm specification, giving more confidence on the correctness of the implementation, while also being efficient, exploring recent developments in programming languages for better parallelism. We do so by using the Scala programming language and the Akka toolkit.

Agents of the protocol (*proposer*, *acceptor* and *learner*) is exchange messages asynchronously to solve M-Consensus. In our implementation, such agents are impersonated by Akka actors, lightweight processes that may be created by the thousands per node. Upon receiving a message, the agent will delegate to other actors, workers, the task of processing and responding. Hence, each agent may process messages belonging to different instances in parallel. This is shown in the right-hand side of Figure 1.

The solution of each M-Consensus instance is a small sequence of broadcast commands, and a deterministic aggregation of such solutions compose a totally ordered sequence of commands in the Atomic Broadcast layer of our implementation.

Moreover, our implementation uses Akka cluster to dynamically control the cluster membership and protocol agents, allowing the reconfiguration with the exclusion of faulty *proposers* and *learners*. *Acceptors* are replaced using the same technique as in regular (multi)Paxos.

As our implementation is modular, new features can be easily added to the cluster node, like different atomic broadcast

(e.g., for evaluation purposes) and leader election protocols or failure detection mechanism. A node can play the role of one or more agents of the protocol, and each agent creates Akka actors as needed to meet clients requests, as show in .

Our second contribution is an experimental performance evaluation of the implemented protocol and comparison with main related works. The method may be used to evaluate similar group communication tools and is inspired by other evaluations in the literature.

To evaluate the algorithms we will implement a replicated key-value store service, like in [1], and place agents both with a single network/data-center and in different data-center. For each of such scenarios we will evaluate the following metrics: delivery latency and throughput of broadcast messages; memory and CPU utilization on each of the agents throughout the experiment; effect of failure of one or more processes and quorums sizes in the previous metrics; effect of reconfiguring an agent by switching its status as collision-fast proposer and regular one.

III. Related works

As far as we know there are two protocols in the literature that have similarities to CFABcast, proposed in [7]: Mencius and Clock-RSM. It is up to this work to demonstrate the advantages and disadvantages of the CFABcast with respect to such protocols.

A. Mencius

Mencius [2] is a protocol for state machine replication based on a modification of the Paxos protocol that solves a simplified version of the Consensus problem. In the absence of failures, Mencius and CFABcast have similar communication patterns. However, if one of the coordinators fail in Mencius, the others should continuously propose no-op in its regard in the subsequent consensus instance. In the CFABcast, the set of collision-fast proposers is defined per round of the protocol. Hence, if a collision-fast proposer fails, all that is needed is a new round of the M-Consensus protocol, that fills the place of consensus in the heart of CFABcast.

Because the Collision-Fast Paxos protocol can remain collision-fast after reconfigurations of failures, unlike Mencius that needs to be restarted or continued exchanging control messages to treat faulty processes, it is understood that the latter is not actually Collision-fast [3].

B. Clock-RSM

Clock-RSM [1] is also a protocol for state machine replication that provides low latency and makes use of weakly synchronized physical clocks (using a time synchronization protocol, such as NTP) in each replica to fully sort the commands. The communication pattern of Clock-RSM is similar to the CFABcast if all proposers propose with constant frequency. Unlike the CFABcast protocol, where the reconfiguration of future instances is performed by the coordinator on every new round, without the need to solve Consensus, in the Clock-RSM, the reconfiguration is done by solving a Consensus instance explicitly, which implies in leader election and no guarantee of termination.

C. Paxos for System Builders

In [8] is presented a specification of the Paxos algorithm to solve Atomic Broadcast that runs multiple instances of the consensus protocol in parallel. The main focus of [8] is completely specify Paxos algorithm in a practical way, as is intended with this work, but using CFpaxos algorithm to solve M-Consensus problem.

IV. Conclusion

At the end of this study we expect to have a clear idea of how the Collision-Fast Atomic Broadcast algorithms in the literature perform in a typical distributed information system scenario. Combined with theoretical analysis of selected protocols, this information may be used by the scientific and industrial communities in choosing the most appropriate Atomic Broadcast protocol for their applications. In addition, the result of this work will be made available as free software, which will enable its use and improvement by the community. Besides these practical contributions, we have also made algorithmic ones, such as defining a byzantine fault tolerant version of the protocol, but these are out of the scope of this paper

Acknowledgment

The authors would like to thank CAPES, CNPq and Fapemig for supporting the execution of this work.

References

[1] J. Du, D. Sciascia, S. Elnikety, W. Zwaenepoel, and F. Pedone, "Clock-RSM: Low-latency inter-datacenter state machine replication using loosely synchronized physical clocks," in *44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2014, Atlanta, GA, USA, June 23-26, 2014*, 2014, pp. 343–354. [Online]. Available: http://dx.doi.org/10.1109/DSN.2014.42

[2] Y. Mao, F. P. Junqueira, and K. Marzullo, "Mencius: Building efficient replicated state machines for wans," in *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'08. Berkeley, CA, USA: USENIX Association, 2008, pp. 369–384. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855741.1855767

[3] R. Schmidt, L. Camargos, and F. Pedone, "Collision-fast atomic broadcast," in *Proceedings of the 2014 IEEE 28th International Conference on Advanced Information Networking and Applications*, ser. AINA '14. Washington, DC, USA: IEEE Computer Society, 2014, pp. 1065–1072.

[4] M. Burrows, "The chubby lock service for loosely-coupled distributed systems," 2006. [Online]. Available: http://research.google.com/archive/chubby.html

[5] F. P. Junqueira, B. C. Reed, and M. Serafini, "Zab: High-performance broadcast for primary-backup systems," in *Proceedings of the 2011 IEEE/IFIP 41st International Conference on Dependable Systems&Networks*, ser. DSN '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 245–256. [Online]. Available: http://dx.doi.org/10.1109/DSN.2011.5958223

[6] L. Lamport, "Lower bounds for asynchronous consensus," *Distributed Computing*, vol. 19, no. 2, pp. 104–125, 2006. [Online]. Available: http://dx.doi.org/10.1007/s00446-006-0155-x

[7] R. Schmidt, L. Camargos, and F. Pedone, "On collision-fast atomic broadcast," Tech. Rep., 2007. [Online]. Available: http://infoscience.epfl.ch/getfile.py?recid=100857

[8] J. Kirsch and Y. Amir, "Paxos for system builders: An overview," in *Proceedings of the 2Nd Workshop on Large-Scale Distributed Systems and Middleware*, ser. LADIS '08. New York, NY, USA: ACM, 2008, pp. 3:1–3:6. [Online]. Available: http://doi.acm.org/10.1145/1529974.1529979