

Asymmetric Byzantine Fault Tolerant State Machine Replication

Ramakrishna Kotla
The University of Texas at Austin

1 Introduction

We present a BFT (Byzantine Fault Tolerant) state machine replication technique that provides better scalability (performance scales with the number of replicas), better reliability (reduces correlated failures among the replicas), fault isolation (by partitioning state machine across replica groups), and strong consistency semantics, for applications with parallelism like file systems. Traditional state machine replication technique [4, 2] requires that all the replicas execute all the requests in the same total order [5] or partial order [3] to ensure consistency of correct replicas, and thus suffers from the drawbacks that the throughput of the system does not scale with the number of nodes in the system and has higher risk of correlated failures as all the replicas execute all the requests.

2 Architecture

Asymmetric replication technique does not completely replicate the application state across all the replicas as in traditional state machine replication (symmetric) but instead distributes the state across all the replicas. We separate execution from agreement [6] in our system. Clients send requests to a central agreement cluster consisting of $3f + 1$ replicas, which runs a Byzantine agreement protocol [2] to order the requests, and they do not process the requests. Agreement cluster forwards the ordered requests to the execution cluster consisting of multiple replica groups each of which consists of $2f + 1$ replicas which process the requests. Application state machine is partitioned into a set of constituent state machines each of which is served by a different replica group. Requests can modify state in multiple replica groups to ensure consistency. For example, in a replicated NFS file system, reads and writes to two files can be served by different replica groups in parallel, and create/delete modifies the state of multiple replica groups (one serving the file and the other serving the directory requests). Replicas in agreement cluster and replica groups can share physical machines. By carefully mapping replica groups (state machine partitions) to the physical machines, we can improve concurrency in the system that can be exploited to improve performance. By separating agreement from execution, and having a single agreement cluster we ensure strong consistency and also amortize the overhead of agreement protocol by batching requests that access different replica groups.

3 Properties

Our system has following properties :

- **Scalability:** As the number of machines increase, we can partition the state machine into more partitions which map to increased replica groups. As the requests to different replica groups can be executed in parallel, performance scales with number of machines in the system.
- **Reliability:** Reliability of the system is improved as the state is distributed across different replicas reducing correlated failures caused by bugs exposed by executing certain set of requests.
- **Fault Isolation:** In our system, if a bug is exposed by executing a request it is limited a particular replica group or set of replica groups where as the rest of the system can continue to execute correctly.
- **Consistency:** Our system provides strong consistency by having a single agreement stage, which ensures that requests are ordered and forwarded to all the relevant replica groups.

4 Limitation:

Centralized agreement cluster reduces protocol overhead and provides better consistency semantics but can become the bottleneck under high load although they just order the requests and do not perform any application processing. We do not think this architecture to be suitable for applications that can have very high load but should perform well for applications with moderate load like NFS serving a site with a few hundreds of users.

5 Prototype:

We are building a system prototype called ABASE (Asymmetric BASE), by modifying BASE system [4], and implement BFT NFS file system to evaluate performance and fault tolerance of the system.

6 Related Work:

Farsite [1] partitions the application state into disjoint partitions with a separate agreement cluster ordering the requests for each partition. As the requests that go to different agreement clusters are not ordered, it provides weak consistency semantics compared to strong consistency provided by our system that may be required for certain applications.

References

- [1] A. Adya, W. Bolosky, M. Castro, R. Chaiken, G. Cermak, J. Douceur, J. Howell, J. Lorch, M. Theimer, and R. Wattenhofer. Farsite: Federated, available, and reliable storage for an incompletely trusted environment. In *5th Symp on Operating Systems Design and Impl.*, 2002.
- [2] M. Castro and B. Liskov. Practical byzantine fault tolerance. In *3rd Symp. on Operating Systems Design and Impl.*, Feb. 1999.

- [3] R. Kotla and M. Dahlin. High throughput byzantine fault tolerance. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN 2004), DCC Symposium*, June 2004.
- [4] R. Rodrigues, M. Castro, and B. Liskov. Base: Using abstraction to improve fault tolerance. In *18th ACM Symposium on Operating Systems Principles*, Oct. 2001.
- [5] F. Schneider. Implementing Fault-tolerant Services Using the State Machine Approach: A tutorial. *Computing Surveys*, 22(3):299–319, Sept. 1990.
- [6] J. Yin, J.-P. Martin, A. Venkataramani, L. Alvisi, and M. Dahlin. Separating agreement from execution for byzantine fault tolerant services. In *19th ACM Symposium on Operating Systems Principles*, Oct 2003.