

# Coordinated Access to Shared Data Sources for Geo-replicated State Machines

Enes Bilgin<sup>1</sup> and Tolga Ovatman<sup>2</sup> <sup>a</sup>

<sup>1</sup>Siemens R&D Center İstanbul, Turkey

<sup>2</sup>Department of Computer Engineering, Istanbul Technical University, 34469 İstanbul, Turkey

**Keywords:** State Machine Replication, Geographic Locality, Serverless Computing.

**Abstract:** State machine replication techniques are widely used to create fault tolerant services in cloud computing. However, reaching consensus between remote regions may take substantially long time. Many algorithms are proposed to solve this problem regarding the geographic locality of the objects while trading off some consistency properties of the system. Most of these studies consider direct replication of arbitrary parts of the system, but contemporary online services share some unique data sources that may not be replicated. This paper proposes a coordinated method to manage shared data sources between replicas by utilizing geographic locality of the data sources. The proposed algorithm present better performance than using a leader based approach in terms of request processing throughput. We provide the design and implementation of the proposed algorithm in detail and present the throughput performance of the algorithm in a geographically distributed environment.

## 1 INTRODUCTION

Geo-distributed state machine replication (Coelho and Pedone, 2018) enables systems to tolerate failures by maintaining consistent copies of an application's states at different geographical regions. Geographic replication could improve performance by placing the data close to the clients, which reduces latency and it prevents failure in case of outages in different regions. Even if application's states are replicated, some systems may still share some non-replicable data sources which can be local services from different regions or single control units. This also raises new problems in active replication techniques for the cases where replicas are expected to process the requests in the same order, but only one replica is able to update the state of shared data sources.

Geo-replicated state machine studies in the literature focus on reducing latency between geographically distributed replicas while trying to reach consensus. Therefore, it may be assumed that all data sources of systems can be easily replicated. However, there may exist non-replicable third party online services or private database accesses as shared data sources. In our study, we work on a coordinated solution to handle access to non-replicable shared data


sources, regarding their geographical positions, while comprising the read consistency of the system. For the use cases where read consistency is not of primary concern (Eischer et al., 2020) this trade off could be reasonable in improving performance of the system.

We have implemented two different approaches using the Spring State Machine framework<sup>1</sup>. First approach uses a predetermined replica to manage all shared data sources (leader based approach) and the second approach selects different replicas by considering their geographic locations in order to reduce latency (coordinated approach).

In our experiments, we have implemented a geographically distributed state machine replica system on Amazon Web Services. We get better performance results when the system takes the geographical positions of shared data sources into account, especially for the specific requests accessing the data source.

## 2 BACKGROUND AND RELATED WORK

In this section, we provide background on replicated state machines techniques and consensus protocols.

<sup>a</sup>  <https://orcid.org/0000-0001-5918-3145>

<sup>1</sup><https://spring.io/projects/spring-statemachine>

Additionally this section also motivates why coordinated access is required between geo-replicas.

Replication techniques are divided into two categories as active and passive replication. In the passive replication, the primary state machine executes the events and updates states of replicas. However, in active replication, each event is processed by all replicas and ends up in the same state. The active state machine replication uses consensus protocols for ordering requests by assigning them to slots with consecutive sequence numbers which define the execution order. This also means that a replica can execute a request only after it has received and processed all predecessors to ensure that all replicas execute the same set of requests in the same order.

State machine replication (Schneider, 1990), mostly serves to provide fault-tolerant systems. It works by running the same deterministic state machine on multiple devices, called replicas, which process events guaranteeing total order consistency. Consensus is an abstraction where replicas agree on a common value (e.g., the order, next operation to be executed). One way to implement the consensus algorithm is by using a leader based protocol like Raft (Ongaro and Ousterhout, 2014), Zap (Junqueira et al., 2011) and MultiPaxos (Chandra et al., 2007). As a first step the replicas elect a replica as leader which needs votes from a majority quorum of replicas. Then the leader assigns requests to sequence numbers by sending Propose messages to all replicas. Each assignment must be confirmed by a majority quorum of replicas by replying with an Accept message to the leader. The leader's Propose message also implicitly counts as an Accept. The operation is committed by replicas when the leader confirms the majority of votes received.

Most of the replicated systems are distributed across different regions to be not affected by cascaded failures in a single region. However, this causes serious latencies when these consensus algorithms are considered, and also wide area communication incurs some latency overheads. Shortly, reaching an agreement between different regions can take several hundred milliseconds afterwards a request can be processed. In terms of shared data source access, the same rules are applied like geo-replication and still there are some open problems to be solved. Reaching to shared data sources from a close replica instead of a remote leader will reduce latency.

In GeoPaxos (Coelho and Pedone, 2018), authors are trying to solve the coordination among geographically distributed replicas. Instead of providing total order consistency, their method proposes a partial order for execution of operations. Besides, they are also

benefiting from the idea that objects are located at sites where they are most likely to be accessed. In our approach, we also take advantage of the idea to keep a single replica for shared data sources.

In Mencius (Mao et al., 2008), researchers try to find a better approach to implement Paxos (Lamport et al., 2001) for geographically distributed state machines. Mencius is a multi-leader version of Paxos and tries to eliminate the single entry-point requirement in Paxos. It splits the leader role across all replicas by statically partitioning data and assigning a part to each replica site. A client sends its request to the nearest leader to avoid the first wide area communication step. In our approach, we do not require a leader role for shared data access since we affiliate a single replica with a designated shared data source.

In Gemini (Vogels, 2009), authors created a solution to reduce latency across different regions by relaxing consistency of some operations. Mainly, they categorize operations as red (strong consistency) and blue (weak consistency). Also, they provide a method to increase the number of potential blue operations. Experiments show that benefiting from eventual consistency (DeCandia et al., 2007) for some operations, significantly improves the performance of geo-replicated systems. In our study, we designate weak consistency for all of the operations that are performed on reading from shared data sources.

In Weave (Eischer et al., 2020), authors tried to minimize high response times of write requests which occurred in case of totally-ordered geo-distributed systems. To address this problem they present Weave (Paxos based) that relies on replica groups in multiple geographic regions to efficiently assign stable sequence numbers to incoming requests. Weave's architecture enables the protocol to perform guaranteed writes without involving wide-area communication.

As recognized, these studies focus on improving reaching consensus about order of operations regarding geographic locations of requests; our approach, alternatively, approaches the problem by initially considering (and increasing) the locality of operations performed on shared data sources before ordering the operations performed.

### 3 PROPOSED APPROACH

The proposed approach, presented in the pseudo-code in Algorithm 1, utilizes geographic locality to remove the induced latency during a replica's access to a distant data source by delegating access operations to a geographically closer replica. The closer replica will perform all of the update and write operations

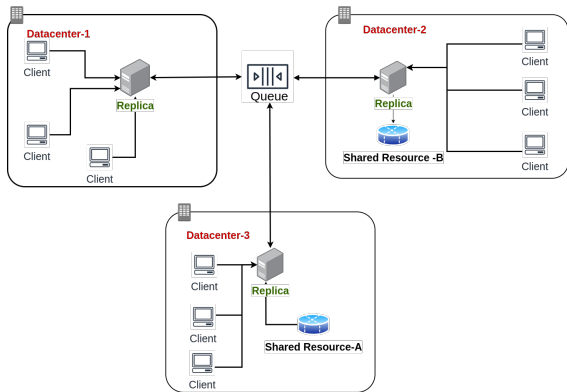


Figure 1: A deployment of the system in three regions.

on the data source and the other replicas will proceed without experiencing a huge delay when accessing the data on a remote region.

The replicas that are authorized to access the shared data sources are determined with configuration parameters of the system. These configuration parameters also contain possible successors of these authorized replicas in order to replace them in case of stop-failures.

When implementing the coordination algorithm we use a distributed queue to avoid missing some operations in case of a failure in the replica(s) in the shared data source’s site. The distributed queue keeps a copy of the received requests in each replica so that they can lazily perform the write operations on the data source when no replicas exist because of a failure or any other reason.

A drawback of this algorithm is losing read consistency of the system, since the shared data source’s coordinator may perform the operations on shared data later than the system. In our implementation we use a consensus protocol relying on majority quorums which may slow down the writing operations. As mentioned earlier, our approach will be suitable mostly for the systems where write consistency is of prior importance.

## 4 EXPERIMENTAL ENVIRONMENT

Figure 1 presents the overall architecture of the experimental environment we have implemented. In our experiments we have used a fully replicated setup in terms of application states and we distribute state machine instances in different regions. Active state machine replication is implemented by using Spring State Machine framework which uses Zookeeper in order to implement replication among replicas. Basi-

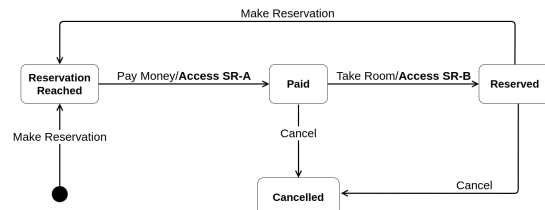


Figure 2: Experimental room reservation state machine.

cally, Zookeeper (Hunt et al., 2010) works as an API which guarantees some consistency semantics such as linearizability (Herlihy and Wing, 1990) for write operations and eventual consistency (Vogels, 2009) for read operations. Under the hood, Zookeeper also uses a consensus algorithm, called Zap. This way, we provide all of the operations conducted on Zookeeper to be linearizable.

We also placed non-replicable data sources on different regions, in proximity of specific state machine instances. By definition, updating or writing these data sources must be conducted only by a single replica (coordinator).

Figure 2 presents the state machine used in our experiments, representing the application logic of a hypothetical hotel reservation application. In this application, hotel reservation is performed in two steps, once the room to be reserved is determined and locked. In the first step payment is performed for the room to be reserved and afterwards reservation is finalized and written to database. Databases for the monetary transactions room reservation statuses are distinct and kept in single locations where a state machine replica for each is running in the designated site.

Data sources (A and B) are placed in geographically distinct locations and accessed during two transitions of the state machine. We choose to define requests of the system as events in order to be compatible with our design. Clients send requests to the system from different regions. Data sources are accessed in transitions of state machine. In Figure 2, each transition that performs a write on shared data source (such as in Figure 1) is labeled with “SR-A” and “SR-B” corresponding to “Shared Resource A” and “Shared Resource B” respectively.

In our test scenarios we generate bursty write requests from each simulated client with uniform random distribution for a possible event that may be received in a certain state. For the cases where the transaction is cancelled due to the receiving event or some other failures we execute rollback operations if necessary for the failing request.

Algorithm 1: Sample Event Processing System.

```

procedure PROCESS(event)
  if event contains writing a shared data source then
    if Replica is leader then
      add operation to queue
    else if Replica is coordinator for the shared data source then
      Pop the operation from queue
      Apply the operation on shared data source
    else
      Skip the operation on data source
  Move on to a new state
    
```

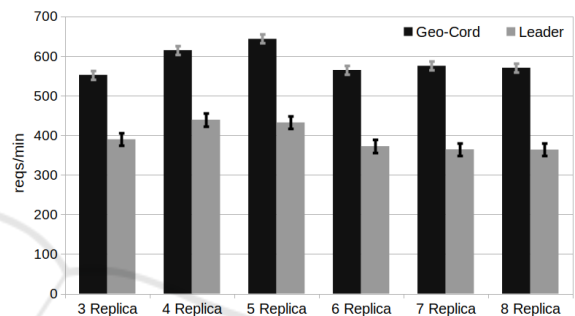
## 5 EXPERIMENTS

In this section we evaluate the proposed algorithm using the room reservation state machine replicas. We run the leader based approach and proposed coordinated algorithm on a cloud environment and compare the results of using a conventional leader based approach and assigning coordinator replicas to shared data sources. Implementation of the system that has been used in the experiments can be accessed for the author’s online repository<sup>2</sup>.

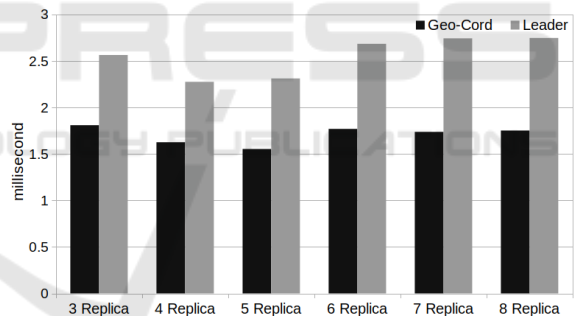
In our experiments, we have implemented a baseline method, where just the leader of the replicated cluster will access the shared data even if the source’s locations is remote, causing latency problems. We have compared this baseline method with the proposed approach in terms of provided throughput and latency. Throughput analyses are conducted on the leader replica since the requests arrived on other replicas are forwarded the leader replica by Zookeeper architecture to be ordered properly. Clients submit their request synchronously in a closed loop. We increase the number of clients until the system is saturated and no increase in throughput is possible. Clients connect to servers through remote procedure calls and each server has a local buffer store for incoming requests and a separate thread to manage this buffer.

Replicas are deployed in different regions of AWS (US, UK, Asia, Brazil and Sydney) in docker containers. We use t3.medium virtual machines of Amazon EC2 platform to host our replicas. Our virtual machines contain 4 vCPUs, 4GB RAM and runs Ubuntu 20.04.2 LTS. When replicas are deployed in 5 different regions, clients are also created in different machines in US, UK, Asia, Sydney and Turkey. Shared databases are located in two different location in Asia and the US which pushes the leader based approach to reach the data sources other than its local region.

<sup>2</sup><https://gitlab.com/enesbilgin61/sharedresourcecoordinator>



(a) Throughput comparison



(b) Latency comparison

Figure 3: Throughput comparison of the proposed coordinated approach with leader based approach.

Figure 3 shows the throughput and latency results when different number of replicas are spread through five different geographical regions over AWS. In our experiments we have used end-to-end latency by measuring the average of each client’s request time span from the beginning of sending the request until receiving the final response of the request (payment done, room reserved, etc.). For the throughput of the system, we have counted the number of requests responded by our system and present the rate of responded requests per minute. Since our clients are issuing write requests (trigger state changes) in a bursty fashion we have measured throughput from the per-

spective of state machine replica system as a whole.

Since, proposed coordinated approach can eliminate much of the communication latency when reaching the shared data it provides better throughput as the number of replicas reach number of regions. When the number of replicas exceed the number of available regions throughput decreases but still taking advantage of geographic locality by applying the coordinated approach provides better throughput.

As the number of replicas increases, the distance between consensus voting machines begins to increase, causing an increasing communication delay due to the distance.

For six, seven and eight replicas in Figure 3, the replica count requires more votes for quorum that increases the necessary number of messages exchanged between replicas. Compared to five replica case, since the advantage obtained by exploiting the geographic locality is weakened (eight replicas are spread to five regions as well) throughput naturally decreases.

## 6 CONCLUSION AND FUTURE WORK

In this study, we proposed a coordinated way for replicated state machines in order to access shared data sources based on their geographical locations. This approach is designed for event processing applications (consume write requests) which are replicated and share some common data. The results indicate that the coordinated approach outperforms the leader based approach in cloud environment by compromising read consistency of the system.

## ACKNOWLEDGEMENTS

This study is supported by the scientific and technological research council of Turkey (TUBITAK), within the project numbered 118E887.

## REFERENCES

- Chandra, T. D., Griesemer, R., and Redstone, J. (2007). Paxos made live: an engineering perspective. In *Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing*, pages 398–407.
- Coelho, P. and Pedone, F. (2018). Geographic state machine replication. In *2018 IEEE 37th Symposium on Reliable Distributed Systems (SRDS)*, pages 221–230. IEEE.
- DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Voshall, P., and Vogels, W. (2007). Dynamo: Amazon’s highly available key-value store. *ACM SIGOPS operating systems review*, 41(6):205–220.
- Eischer, M., Straßner, B., and Distler, T. (2020). Low-latency geo-replicated state machines with guaranteed writes. In *Proceedings of the 7th Workshop on Principles and Practice of Consistency for Distributed Data*, pages 1–9.
- Herlihy, M. P. and Wing, J. M. (1990). Linearizability: A correctness condition for concurrent objects. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 12(3):463–492.
- Hunt, P., Konar, M., Junqueira, F. P., and Reed, B. (2010). Zookeeper: Wait-free coordination for internet-scale systems. In *USENIX annual technical conference*, volume 8.
- Junqueira, F. P., Reed, B. C., and Serafini, M. (2011). Zab: High-performance broadcast for primary-backup systems. In *2011 IEEE/IFIP 41st International Conference on Dependable Systems & Networks (DSN)*, pages 245–256. IEEE.
- Lamport, L. et al. (2001). Paxos made simple. *ACM Sigact News*, 32(4):18–25.
- Mao, Y., Junqueira, F. P., and Marzullo, K. (2008). Mencius: Building efficient replicated state machines for wans. In *Proceedings of the 8th USENIX Conference on Operating Systems Design and Implementation, OSDI’08*, page 369–384, USA. USENIX Association.
- Ongaro, D. and Ousterhout, J. (2014). In search of an understandable consensus algorithm. In *2014 {USENIX} Annual Technical Conference ({USENIX}{ATC} 14)*, pages 305–319.
- Schneider, F. B. (1990). Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys (CSUR)*, 22(4):299–319.
- Vogels, W. (2009). Eventually consistent. *Communications of the ACM*, 52(1):40–44.