Efficient Write Operations in Event Sourcing with Replication

Tiago Rolo* tiago.rolo@uc.pt

Nuno Preguiça** nmp@fct.unl.pt

Filipe Araujo* filipius@uc.pt

* University of Coimbra, CISUC, DEI
** Nova University of Lisbon, LINCS, FCT

Bertinoro, Italy, 24-10-2024

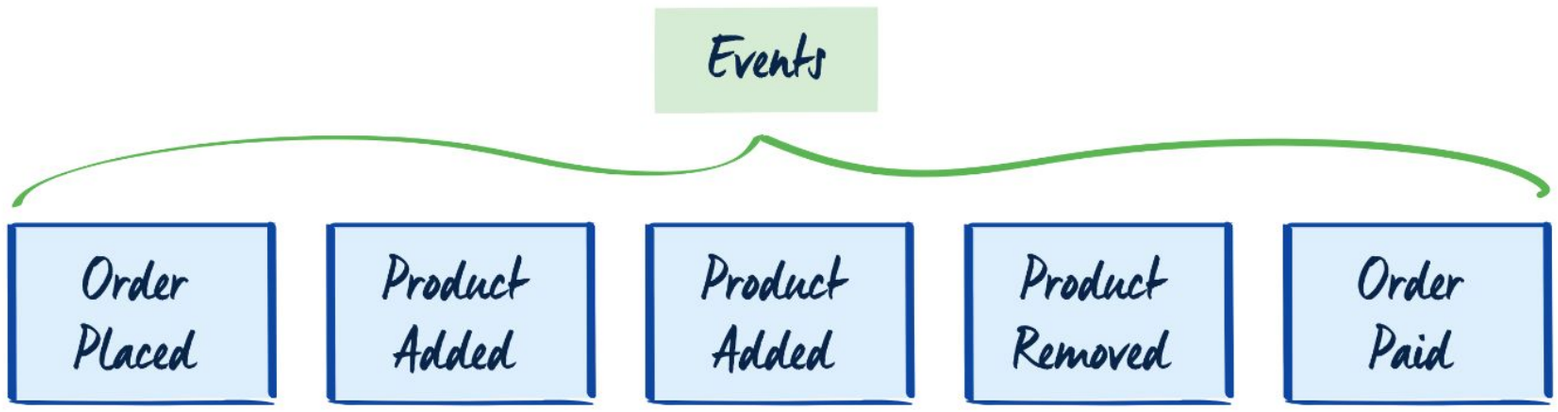
Outline

1. Introduction & Background
2. Server Replication
3. Experiments and Results
4. Conclusion & Future Work

Introduction & Background

Event Sourcing

- Architectural Pattern
- Events represent past actions
- Stores all past actions as events
- Immutable sequence as a single source of truth
- Particularly efficient in event-driven architectures
- One Event Log leads to single point of failure



DDD, CQRS, Saga

- Domain-Driven Design:
  - Software development approach that emphasizes collaboration between domain experts and developers
  - Aggregate, domain events, commands

DDD, CQRS, Saga

- Domain-Driven Design:
  - Software development approach that emphasizes collaboration between domain experts and developers
  - Aggregate, domain events, commands
- CQRS:
  - Split write and read concerns into two models
  - Each model can be its own database

DDD, CQRS, Saga

- Domain-Driven Design:
  - Software development approach that emphasizes collaboration between domain experts and developers
  - Aggregate, domain events, commands
- CQRS:
  - Split write and read concerns into two models
  - Each model can be its own database
- Saga Pattern:
  - Orchestrates actions across services and aggregates
  - Guaranteed to complete execution
  - If failure occurs, use compensation action

Event Sourcing with Replication

- Master/Slave Replication Type
- Axon Framework + MongoDB
- The clients farther away from the master node will not benefit from efficient writes
- Independent and Efficient write operations are not exploited

Problem Statement

- Independent write operations in Event Sourcing

Problem Statement

- Independent write operations in Event Sourcing
- Order of events across replicas:
  - A replicated log might get inconsistent due to latency

Problem Statement

- Independent write operations in Event Sourcing
- Order of events across replicas:
  - A replicated log might get inconsistent due to latency
- Application Resource Exhaustion:
  - Replicas might consume the same last resource

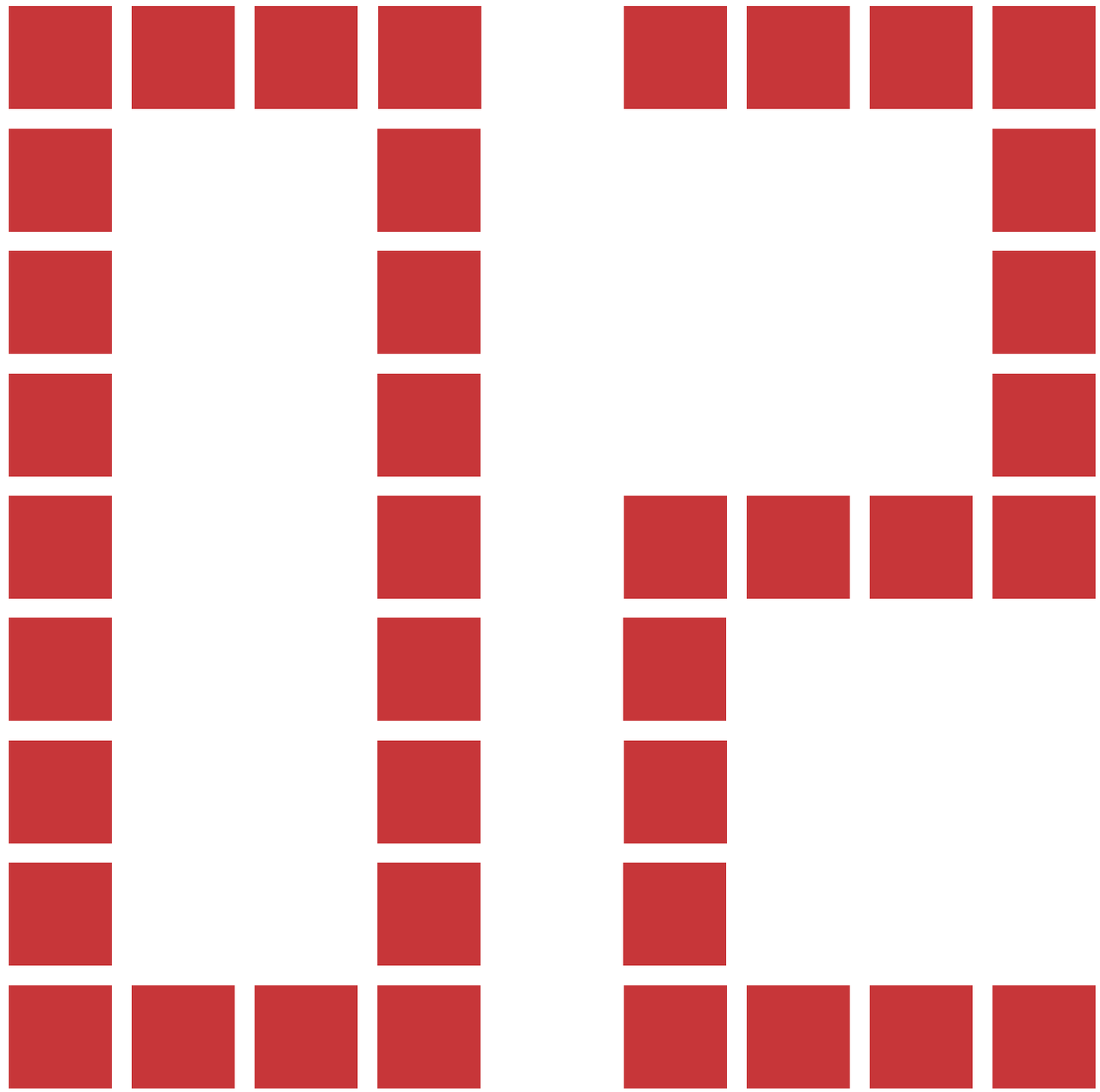
Problem Statement

- Independent write operations in Event Sourcing
- Order of events across replicas:
  - A replicated log might get inconsistent due to latency
- Application Resource Exhaustion:
  - Replicas might consume the same last resource
- Stored Logical Clock:
  - Axon uses logical clocks to order events
  - Might overlap with an already existing event

Related Works

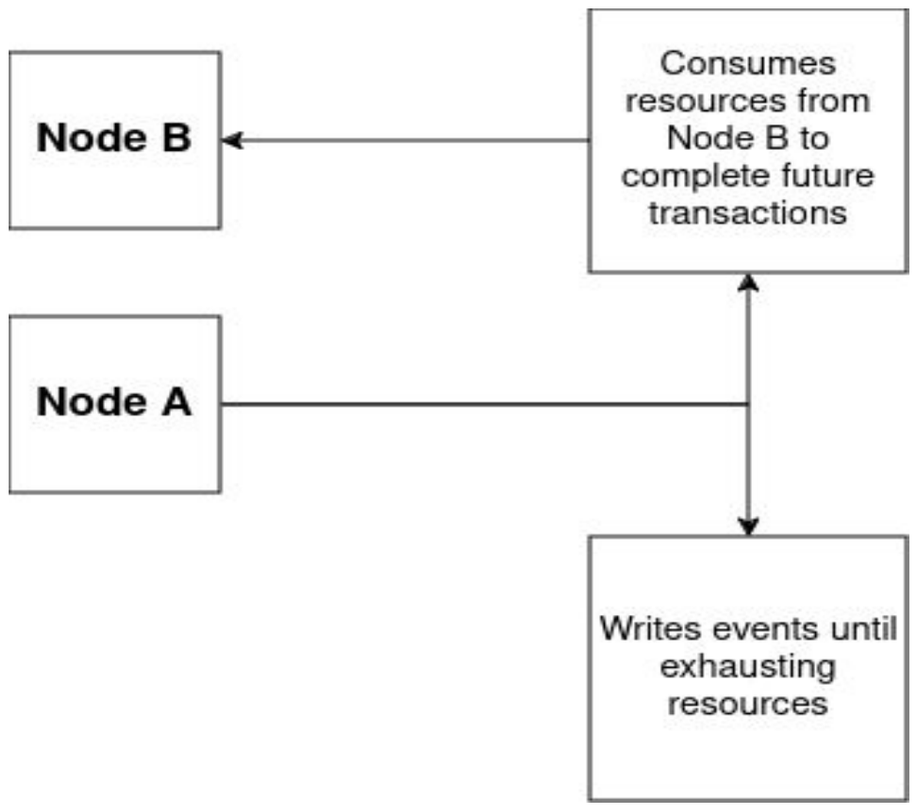
- Limón et al [1]:
  - Built a framework that utilizes the Saga pattern with a multi-agent system, to coordinate distributed transactions between multiple microservice
- O'Neil [2]:
  - Escrow mechanism that increases the amount of transactions that could be executed simultaneously
- Balegas et al [3]:
  - Escrow transactional method using invariant numbers as an escrow to enforce eventual consistency in a geo-replicated system
- Leite [4]:
  - Data replication using the Axon Framework on top of MongoDB Replica Sets

Server Replication



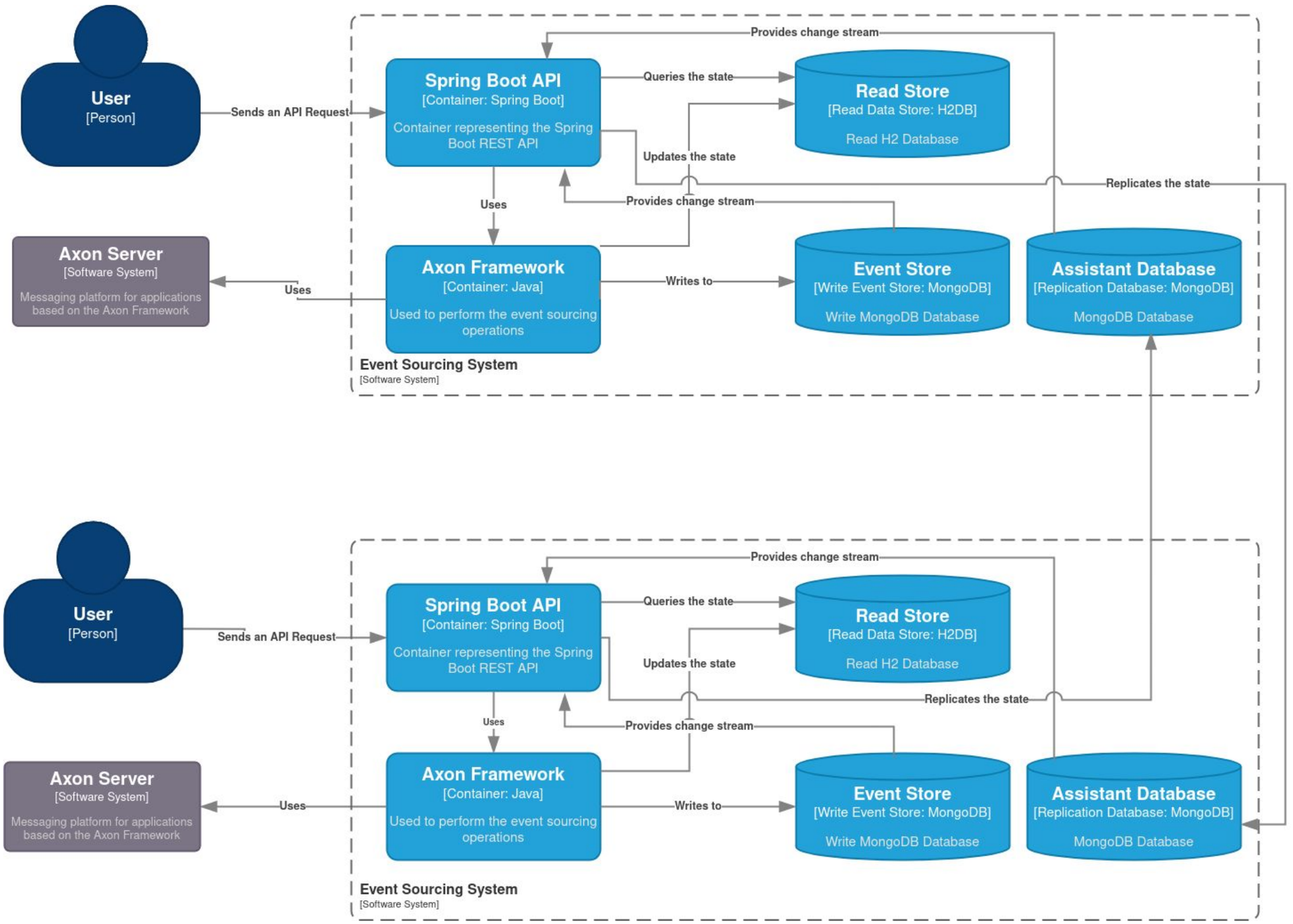
Proposed Approach

- We aim to replicate the event log independently, improving dependability
- Partition the resources across the two nodes with a transferring rights mechanism
- Additional Database as a callback
- Callback orders the events using Logical Clocks
- Sagas orchestrate transactions



Technologies

Architecture



Experiments & Results

Experimental Setup

- Built a generic Port Management application
- Deployed in two AWS regions: us-east-2 and eu-west-3
- Each region represents a replica
- Deployed a similar setup for MongoDB Replica Set as a baseline
- To automate the tests with a custom tool
- Each test ran through 300 requests

Measurements

- Local Consistency:
  - Time between the event being written to the Event Store and H2 Database
- Replication Time:
  - Time between the event being request locally and written to the remote Event Store
- Remote Consistency:
  - Time between the event being requested locally and show up in the peer’s H2 Database

Experimental Results

	Local Consistency (ms)	*Replication Time (ms)*	*Remote Consistency Time (ms)*
Most Favorable Case Replication	58.1 ± 19.4 ms	1230.9 ± 17.7 ms	1236.4 ± 18.1 ms
Least Favorable Case Replication	1889.5 ± 15.5 ms	3081.9 ± 29 ms	3089.2 ± 29.1 ms
MongoDB Replica Set on the Primary	297.4 ± 12.2 ms	202.7 ± 7.6 ms	848.5 ± 60.7 ms
MongoDB Replica Set on the Secondary	408.3 ± 10.7 ms	204.4 ± 31.9 ms	716.8 ± 295.2 ms

Experimental Results

	Local Consistency (ms)	*Replication Time (ms)*	*Remote Consistency Time (ms)*
Most Favorable Case Replication	58.1 ± 19.4 ms	1230.9 ± 17.7 ms	1236.4 ± 18.1 ms
Least Favorable Case Replication	1889.5 ± 15.5 ms	3081.9 ± 29 ms	3089.2 ± 29.1 ms
MongoDB Replica Set on the Primary	297.4 ± 12.2 ms	202.7 ± 7.6 ms	848.5 ± 60.7 ms
MongoDB Replica Set on the Secondary	408.3 ± 10.7 ms	204.4 ± 31.9 ms	716.8 ± 295.2 ms

Experimental Results

	Local Consistency (ms)	*Replication Time (ms)*	*Remote Consistency Time (ms)*
Most Favorable Case Replication	58.1 ± 19.4 ms	1230.9 ± 17.7 ms	1236.4 ± 18.1 ms
Least Favorable Case Replication	1889.5 ± 15.5 ms	3081.9 ± 29 ms	3089.2 ± 29.1 ms
MongoDB Replica Set on the Primary	297.4 ± 12.2 ms	202.7 ± 7.6 ms	848.5 ± 60.7 ms
MongoDB Replica Set on the Secondary	408.3 ± 10.7 ms	204.4 ± 31.9 ms	716.8 ± 295.2 ms

Experimental Results

	Local Consistency (ms)	*Replication Time (ms)*	*Remote Consistency Time (ms)*
Most Favorable Case Replication	58.1 ± 19.4 ms	1230.9 ± 17.7 ms	1236.4 ± 18.1 ms
Least Favorable Case Replication	1889.5 ± 15.5 ms	3081.9 ± 29 ms	3089.2 ± 29.1 ms
MongoDB Replica Set on the Primary	297.4 ± 12.2 ms	202.7 ± 7.6 ms	848.5 ± 60.7 ms
MongoDB Replica Set on the Secondary	408.3 ± 10.7 ms	204.4 ± 31.9 ms	716.8 ± 295.2 ms

Experimental Results

	Local Consistency (ms)	*Replication Time (ms)*	*Remote Consistency Time (ms)*
Most Favorable Case Replication	58.1 ± 19.4 ms	1230.9 ± 17.7 ms	1236.4 ± 18.1 ms
Least Favorable Case Replication	1889.5 ± 15.5 ms	3081.9 ± 29 ms	3089.2 ± 29.1 ms
MongoDB Replica Set on the Primary	297.4 ± 12.2 ms	202.7 ± 7.6 ms	848.5 ± 60.7 ms
MongoDB Replica Set on the Secondary	408.3 ± 10.7 ms	204.4 ± 31.9 ms	716.8 ± 295.2 ms

Experimental Results

	Local Consistency (ms)	*Replication Time (ms)*	*Remote Consistency Time (ms)*
Most Favorable Case Replication	58.1 ± 19.4 ms	1230.9 ± 17.7 ms	1236.4 ± 18.1 ms
Least Favorable Case Replication	1889.5 ± 15.5 ms	3081.9 ± 29 ms	3089.2 ± 29.1 ms
MongoDB Replica Set on the Primary	297.4 ± 12.2 ms	202.7 ± 7.6 ms	848.5 ± 60.7 ms
MongoDB Replica Set on the Secondary	408.3 ± 10.7 ms	204.4 ± 31.9 ms	716.8 ± 295.2 ms

Experimental Results

	Local Consistency (ms)	*Replication Time (ms)*	*Remote Consistency Time (ms)*
Most Favorable Case Replication	58.1 ± 19.4 ms	1230.9 ± 17.7 ms	1236.4 ± 18.1 ms
Least Favorable Case Replication	1889.5 ± 15.5 ms	3081.9 ± 29 ms	3089.2 ± 29.1 ms
MongoDB Replica Set on the Primary	297.4 ± 12.2 ms	202.7 ± 7.6 ms	848.5 ± 60.7 ms
MongoDB Replica Set on the Secondary	408.3 ± 10.7 ms	204.4 ± 31.9 ms	716.8 ± 295.2 ms

Experimental Results

	Local Consistency (ms)	*Replication Time (ms)*	*Remote Consistency Time (ms)*
Most Favorable Case Replication	58.1 ± 19.4 ms	1230.9 ± 17.7 ms	1236.4 ± 18.1 ms
Least Favorable Case Replication	1889.5 ± 15.5 ms	3081.9 ± 29 ms	3089.2 ± 29.1 ms
MongoDB Replica Set on the Primary	297.4 ± 12.2 ms	202.7 ± 7.6 ms	848.5 ± 60.7 ms
MongoDB Replica Set on the Secondary	408.3 ± 10.7 ms	204.4 ± 31.9 ms	716.8 ± 295.2 ms

If the number of cases the “happy path” occurs is 86.93% or better our “local consistency times” improve those of the Replica Set

Conclusion & Future Work

Conclusion & Future Work

- The local operations were in favour of the implemented mechanism
- The Replica Set approach is better than the worst case
- The method loses to the Replica Set in replication
- Consistency times are closer

Conclusion & Future Work

- The local operations were in favour of the implemented mechanism
- The Replica Set approach is better than the worst case
- The method loses to the Replica Set in replication
- Consistency times are closer
- Improve the replication performance using an event-driven approach
- Taking out the additional steps required to ensure consistency will increase performance

References

- [1] Xavier Limón, Alejandro Guerra-Hernández, Angel J Sánchez-García, and Juan Carlos Peréz Arriaga. SagaMAS: a software framework for distributed transactions in the microservice architecture. In 2018 6th International Conference in Software Engineering Research and Innovation (CONISOFT), pages 50–58. IEEE, 2018.
- [2] Patrick E O'Neil. The escrow transactional method. ACM Transactions on Database Systems (TODS), 11(4):405–430, 1986.
- [3] Valter Balegas, Diogo Serra, Sérgio Duarte, Carla Ferreira, Marc Shapiro, Rodrigo Rodrigues, and Nuno Preguiça. Extending eventually consistent cloud databases for enforcing numeric invariants. In 2015 IEEE 34th Symposium on Reliable Distributed Systems (SRDS), pages 31–36. IEEE, 2015.
- [4] Gustavo Miguel Martins Leite. Creation of a Replicated Event Sourcing Application. Master's thesis, University of Coimbra, Coimbra, Portugal, 2023.