

# Dynamic Scalable State Machine Replication

---

Long Hoang Le

long.hoang.le@usi.ch

Eduardo Bezerra

eduardo.bezerra@usi.ch

Fernando Pedone

fernando.pedone@usi.ch

April 12, 2016

University of Lugano, Switzerland

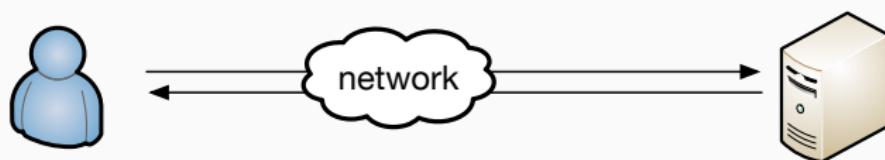
# Outlines

1. Motivation
2. Dynamic Scalable State Machine Replication
3. Implementation
4. Performance Evaluation
5. Conclusion

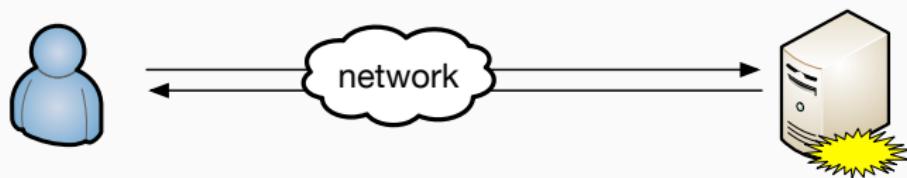
# Motivation

---

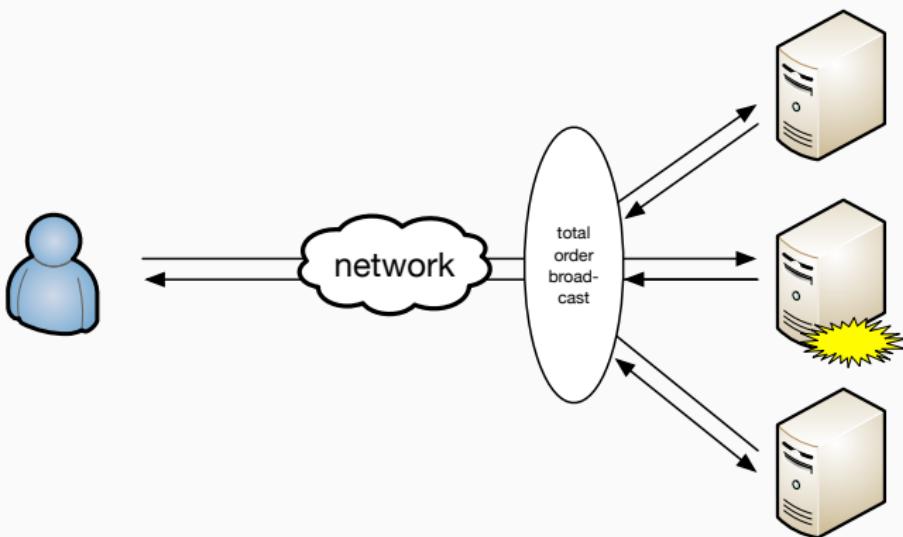
# State Machine Replication



# State Machine Replication



# State Machine Replication



# State Machine Replication

Providing fault-tolerance, strong consistency

- All replicas start in the same initial state
- Apply same set of commands in the same order
- The executions of commands are deterministic
- Proceed through the same set of states

Production systems

- Google Spanner, Windows azure storage, CockroachDB, HydraBase, Chubby

# Scaling State Machine Replication

SMR lacks of scalability:

- Every replica executes all commands.
- Adding servers does not increase the maximum throughput

Different techniques have been developed to deal with these limitations

- Scaling up
- Scaling out

# Scalable State Machine Replication

Applies state partitioning

- Partitions application's state and replicates each partition
- Relies on an atomic multicast primitive

Guarantees strong consistency (ie. linearizability)

- Provides execution atomicity

# Scalable State Machine Replication

Assumes a static workload partitioning

- Relies on static mapping
- Mapping does not need to be minimal

Expensive cost of multi-partition commands

- Relies on static mapping
- Mapping does not need to be minimal

# **Dynamic Scalable State Machine Replication**

---

# System Model

Communication by message passing

- One-to-one communication use reliable
- One-to-many communication relies on atomic multicast
- Message can be lost, reorder, but not corrupted

Crash-recovery failure model

- No Byzantine behavior

System is partially synchronous

# General idea

## Dynamically changing the partitioning

- DS-SMR was designed to exploit workload locality
- Involved variables are moved to a single partition
- Command is executed against this partition
- Variable mapping managed by an Oracle partition

Distinguishes five types of commands

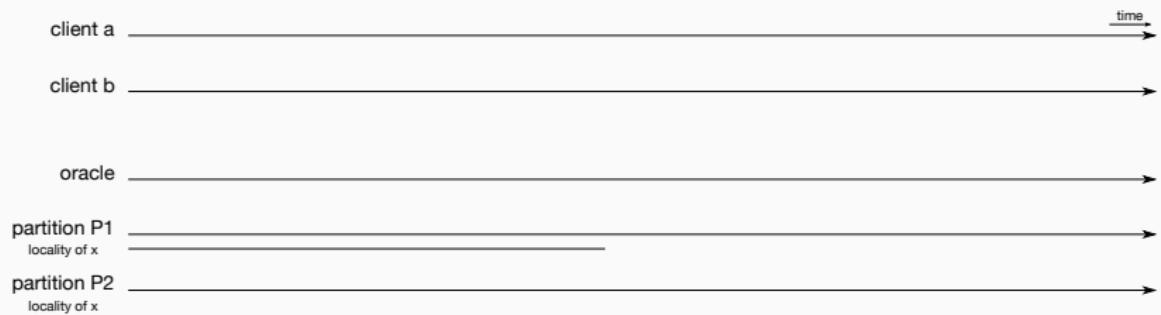
- $\text{access}(w)$ ,  $\text{create}(v)$ ,  $\text{delete}(v)$ ,  $\text{move}(v, Ps, Pd)$ ,  $\text{consult}(C)$

Clients consult the oracle to know variable's location

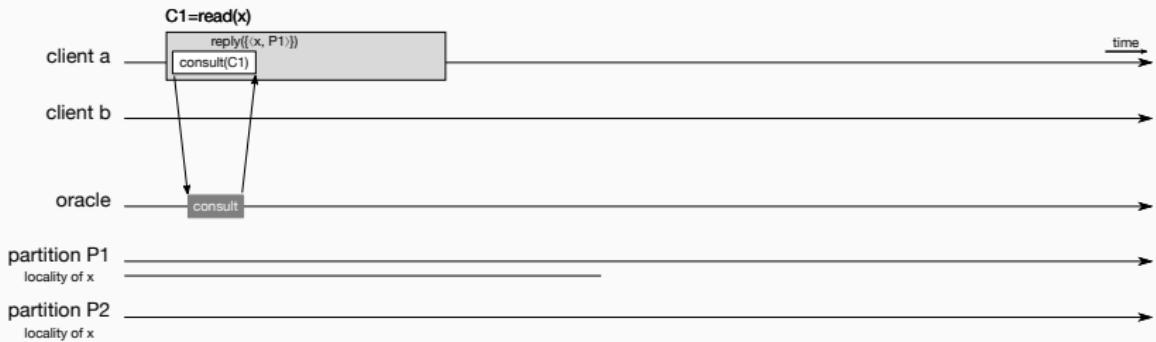
Clients *retry* if command execution is failed

Clients retry command using S-SMRs execution atomicity after a number of retries.

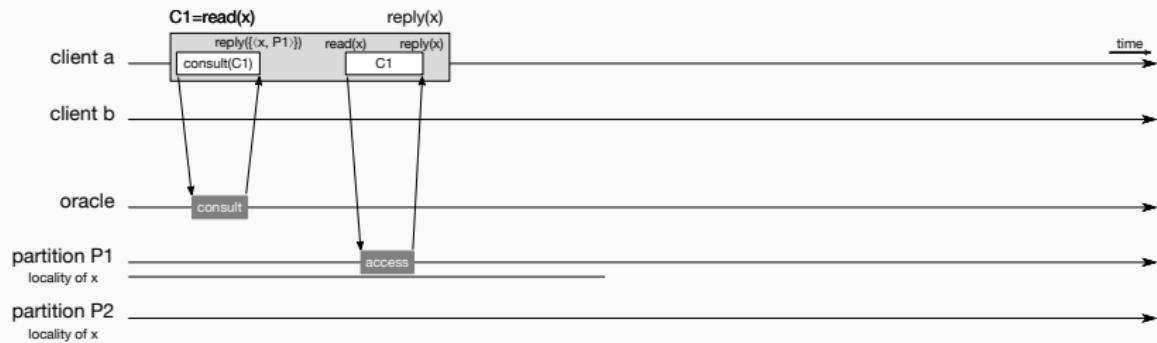
# General idea



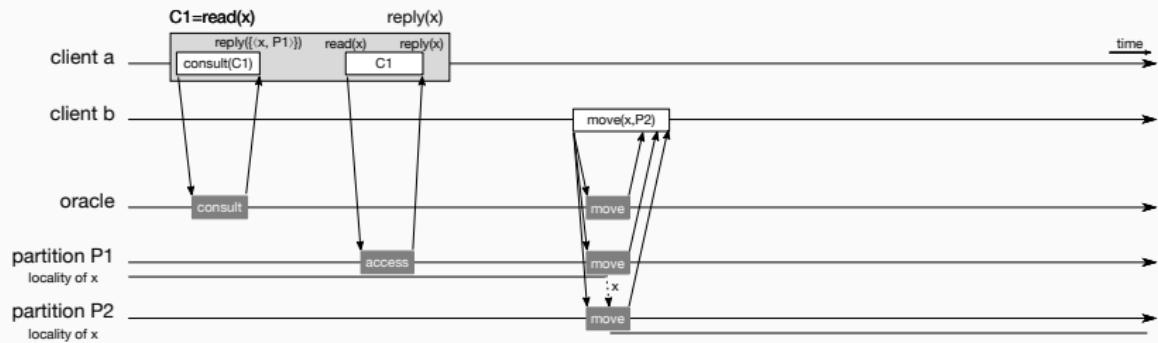
# General idea



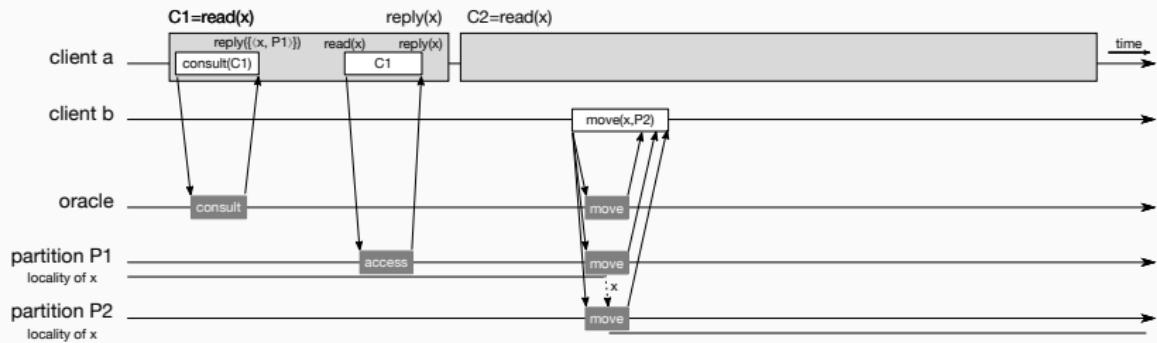
# General idea



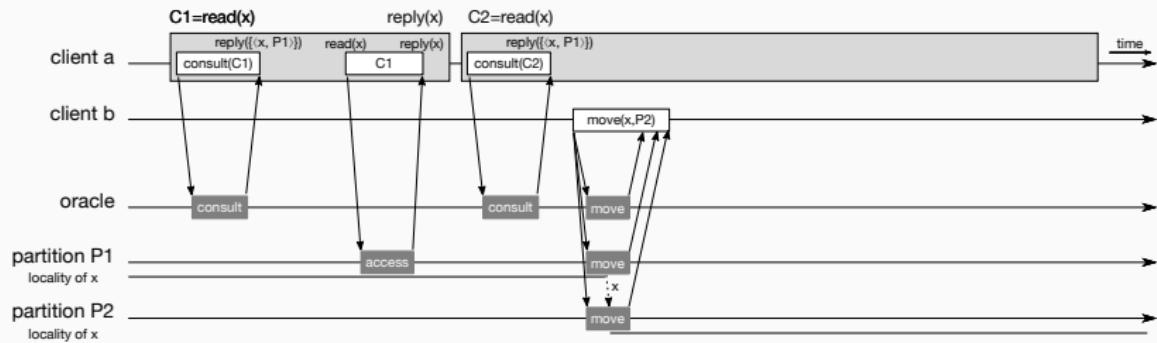
# General idea



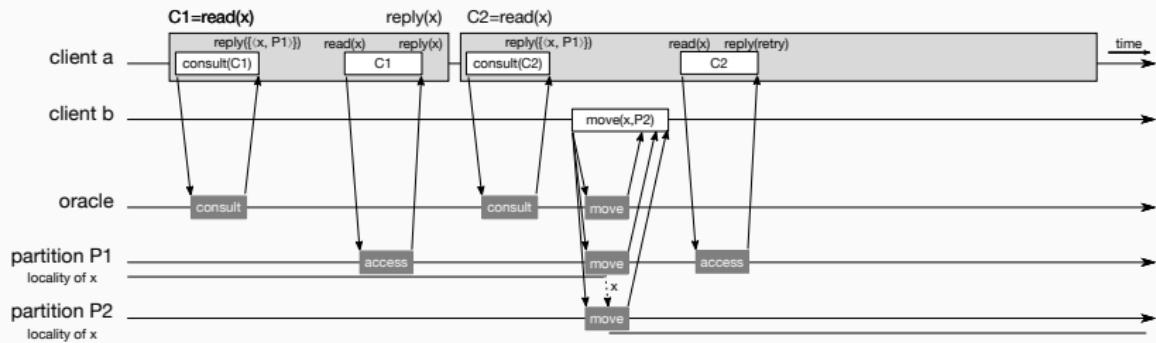
# General idea



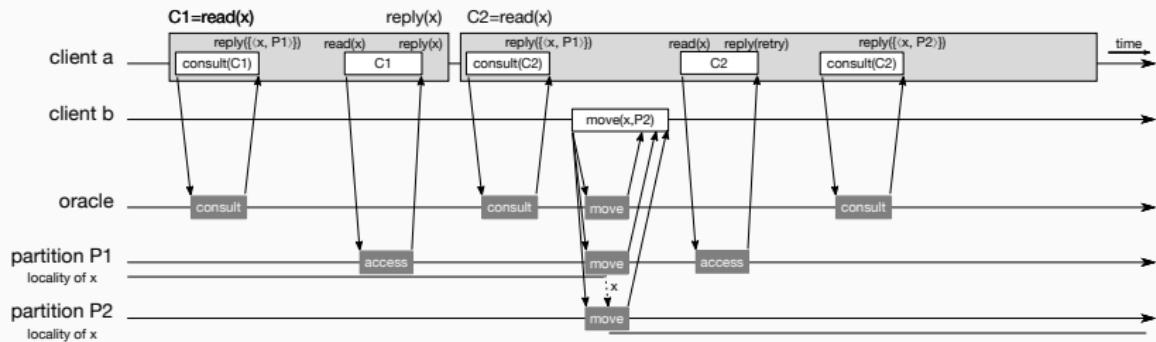
# General idea



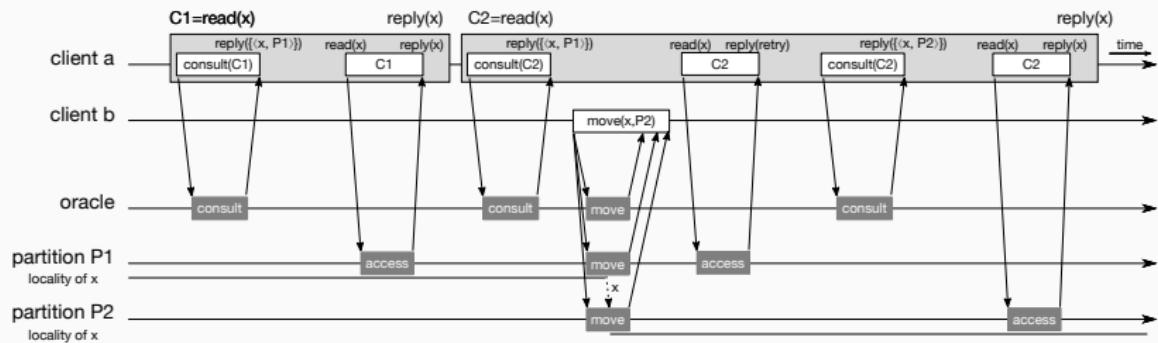
# General idea



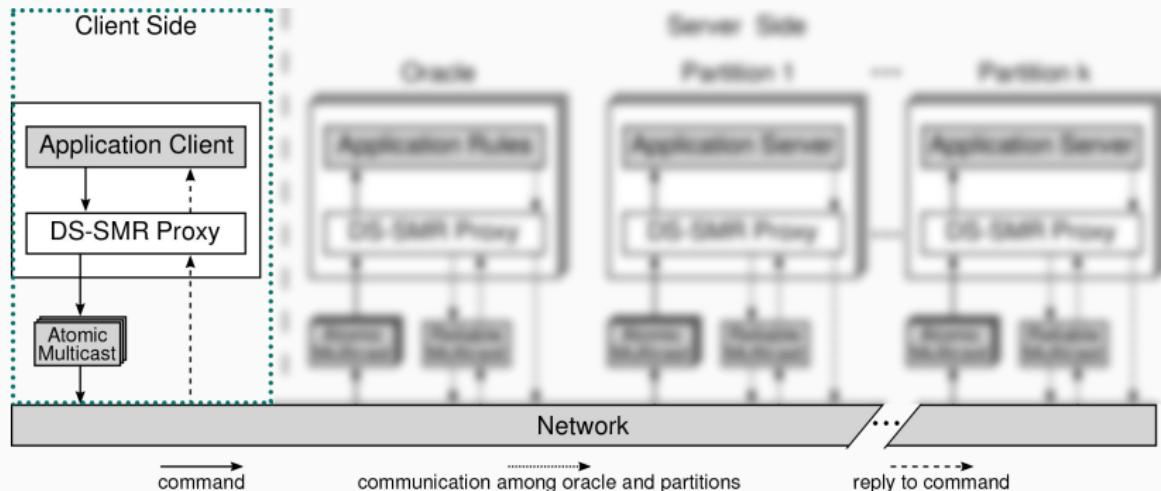
# General idea



# General idea



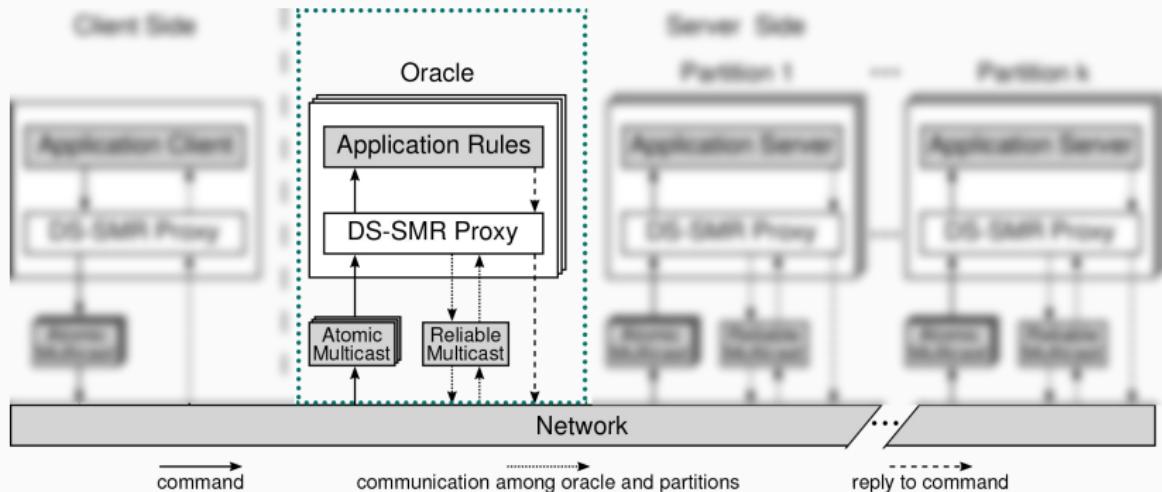
# Architecture



## Client

- Application Client
- DS-SMR Proxy Client

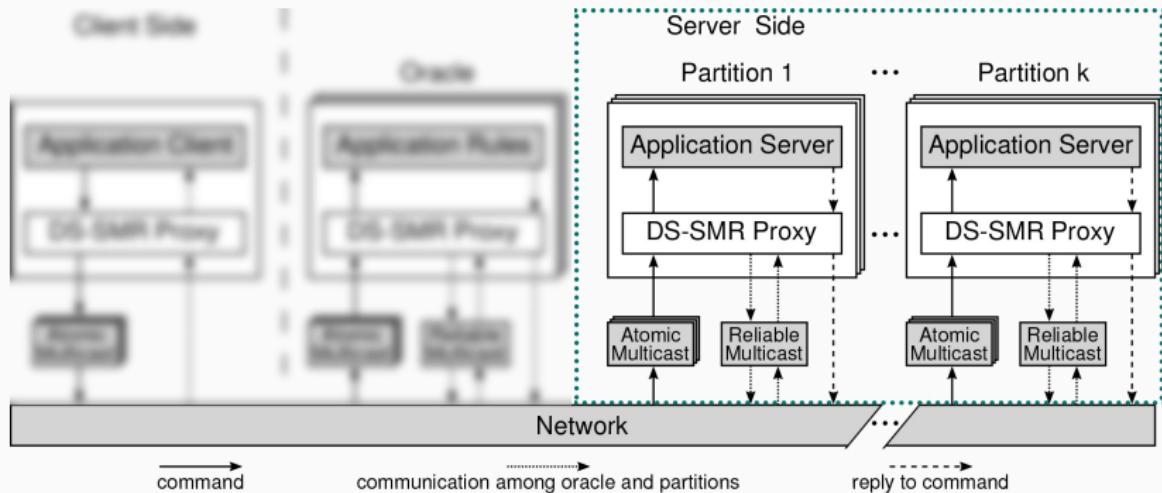
# Architecture



## Oracle

- Application Oracle
- DS-SMR Proxy Oracle

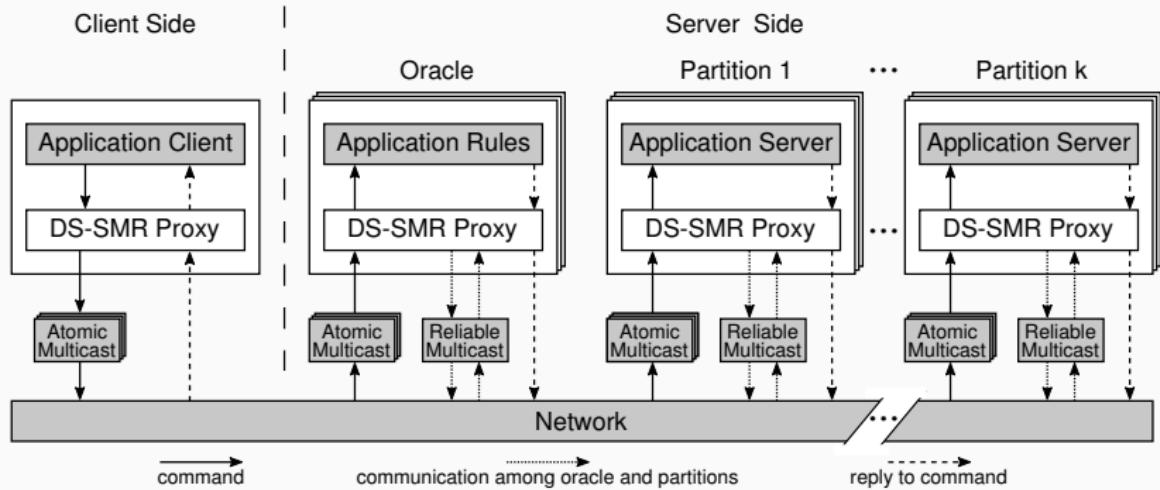
# Architecture



## Server

- Application Server
- DS-SMR Proxy Server

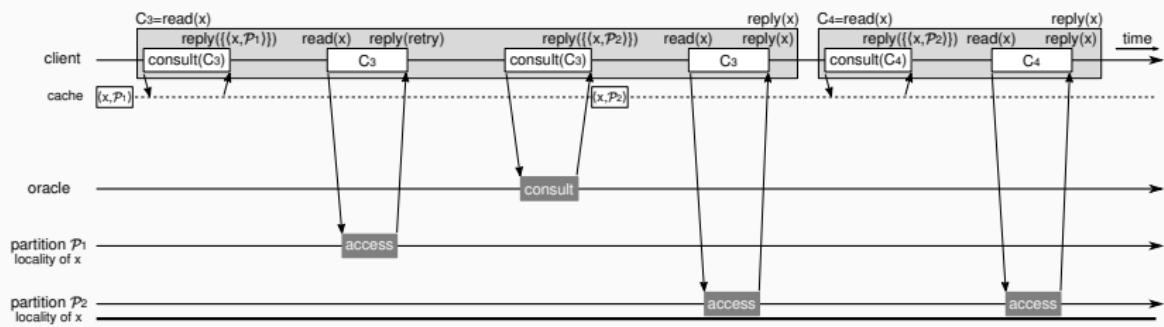
# Architecture



# Performance optimizations

- Caching
  - Each client proxy has a local cache
  - Client consults local cache for determine variable's location
  - Client retry command and update invalid cache
- Under the assumption of locality, most consults queries will be accurately resolved by the clients cache

# Performance optimizations



## Implementation

---

Makes the implementation of services based on DS-SMR as easy as possible.

Provides proxy layers

Allows application designers to override default behaviors.

- The PROObject class
- The StateMachine class
- The OracleStateMachine class

Social network application similar to Twitter

State partitioning is based on users interest.

Support commands:

- post
- getTimeline
- follow, unfollow

## **Performance Evaluation**

---

## Environment setup and configuration parameters

Running *Chirper* under different loads and partitionings.

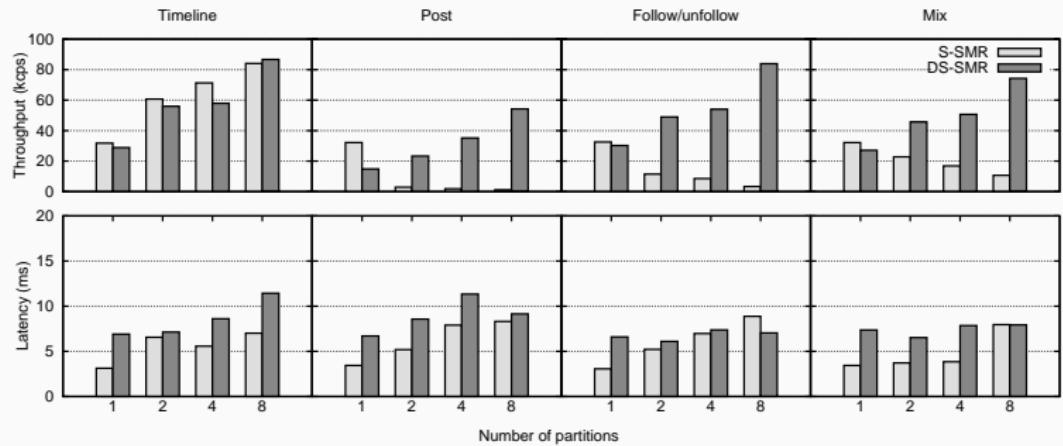
Partitionings: 2 replicas and 3 acceptors for each partition

Experiment on 2, 4, 8 partitions

Workloads:

- Get Timeline
- Post
- Follow, Unfollow
- Mix

# Results



## Conclusion

---

# Summary

- Introduces Dynamic Scalable State Machine Replication (DS-SMR)
- Details Eyrie, a Java library to simplify the design of services based on DS-SMR
- Describes Chirper to demonstrate how Eyrie can be used.
- Presents a detailed experimental evaluation of Chirper to compare the performance of the two replication techniques.

**Questions?**