Multicore Programming

Project Erlang

Key-value store

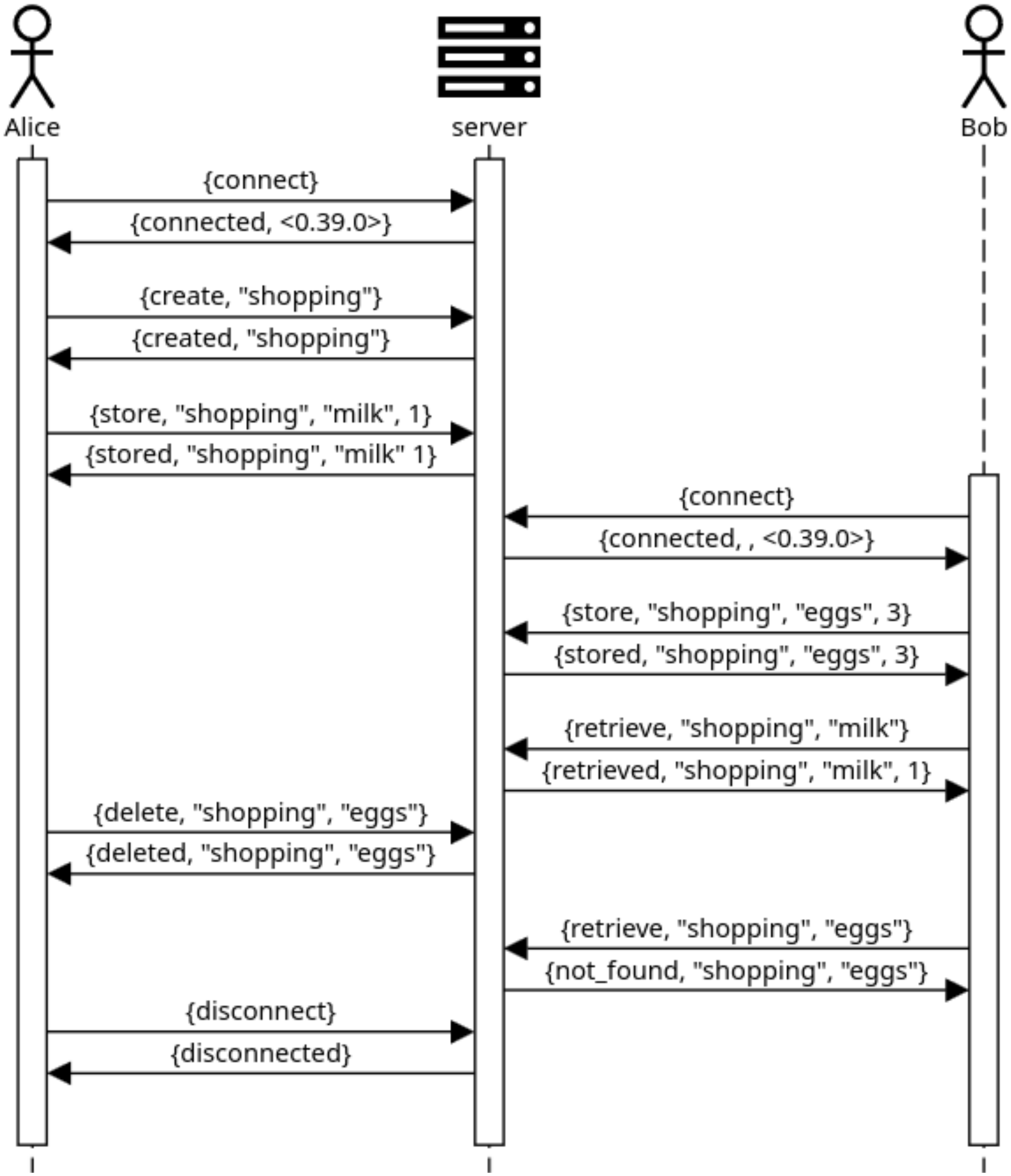
Key-value store

A key-value store is a database in which you associate a key with a value.

You will need to create:

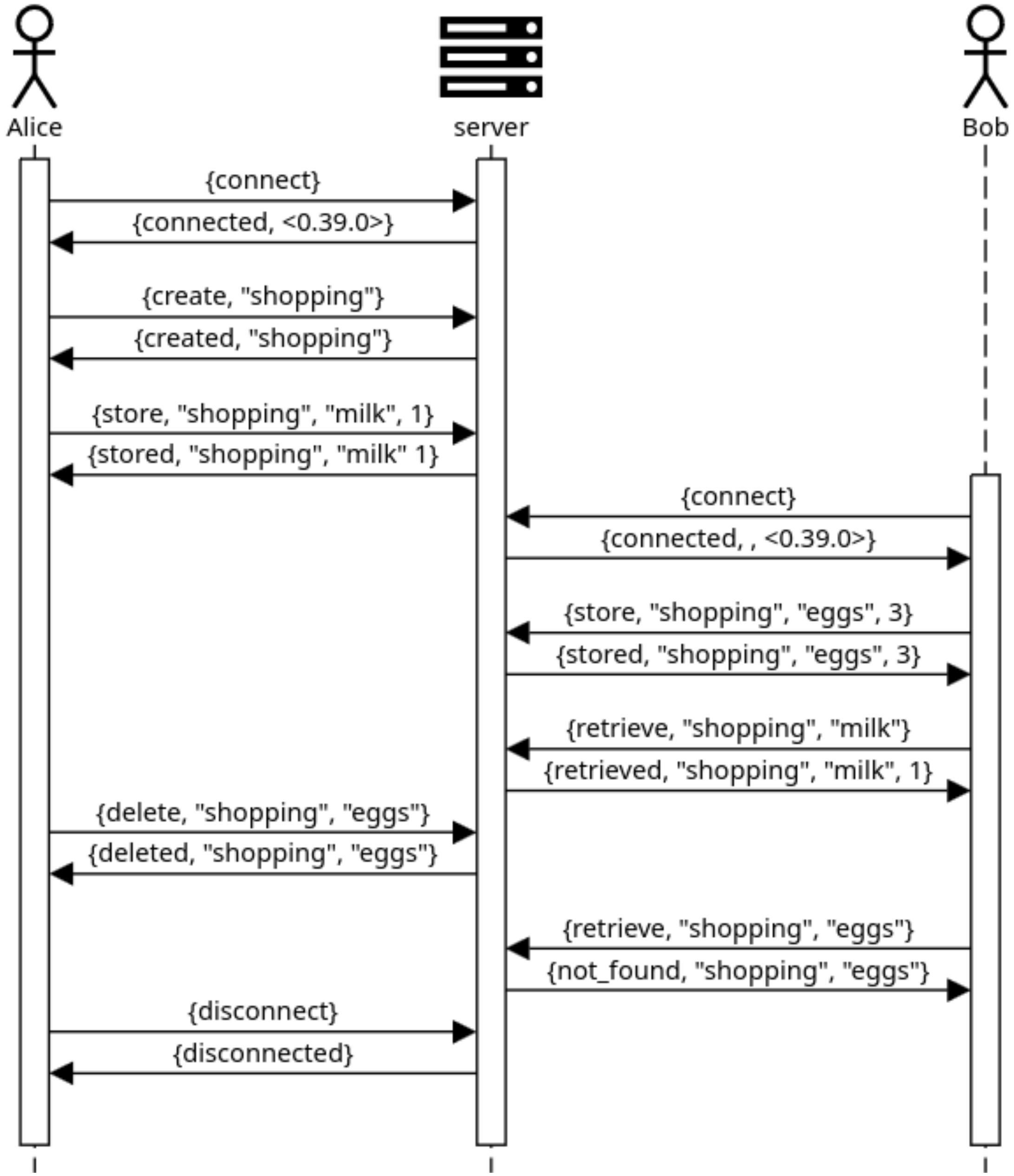
- Implementation
- Evaluation
- Report

Example session


Alice
server
Bob
{connect}
{connected, <0.39.0>}
{create, "shopping"}
{created, "shopping"}
{store, "shopping", "milk", 1}
{stored, "shopping", "milk" 1}
{connect}
{connected, , <0.39.0>}
{store, "shopping", "eggs", 3}
{stored, "shopping", "eggs", 3}
{retrieve, "shopping", "milk"}
{retrieved, "shopping", "milk", 1}
{delete, "shopping", "eggs"}
{deleted, "shopping", "eggs"}
{retrieve, "shopping", "eggs"}
{not_found, "shopping", "eggs"}
{disconnect}
{disconnected}

Key-value pairs are stored in buckets.

Example session

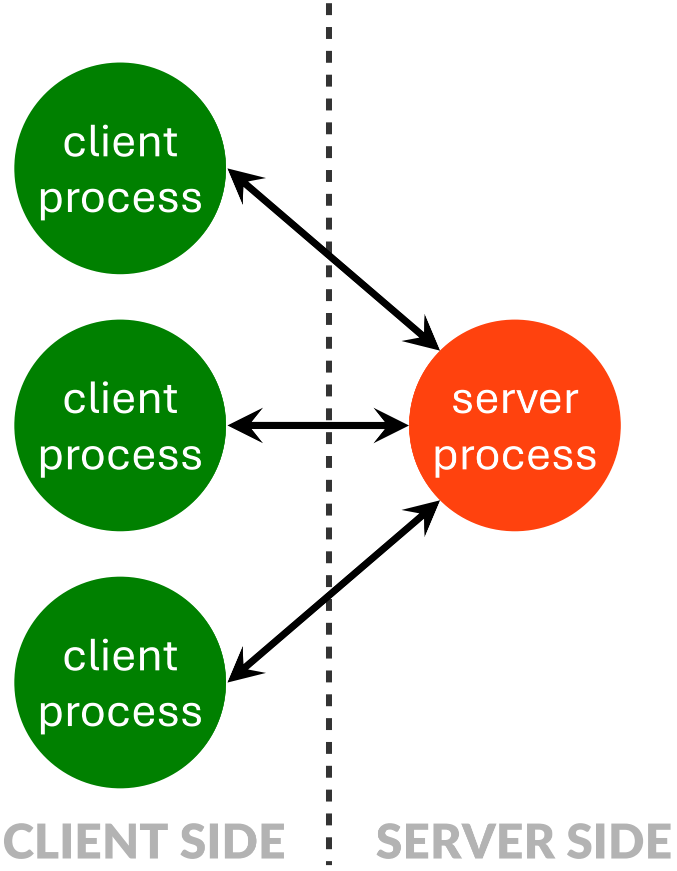

Alice
server
Bob
{connect}
{connected, <0.39.0>}
{create, "shopping"}
{created, "shopping"}
{store, "shopping", "milk", 1}
{stored, "shopping", "milk" 1}
{connect}
{connected, , <0.39.0>}
{store, "shopping", "eggs", 3}
{stored, "shopping", "eggs", 3}
{retrieve, "shopping", "milk"}
{retrieved, "shopping", "milk", 1}
{delete, "shopping", "eggs"}
{deleted, "shopping", "eggs"}
{retrieve, "shopping", "eggs"}
{not_found, "shopping", "eggs"}
{disconnect}
{disconnected}

Operations:

- connect
- disconnect
- create
- store
- retrieve
  → can return not_found

Example implementation

An example implementation is given with one server.
You need to extend this to have multiple server processes.



- 1 process / client → don't change, for benchmarking
- 1 server process → you'll need to change this

Goal: increase scalability

Your goal is to extend the implementation to improve scalability/performance, by using multiple processes.

Possible directions:

- **Sharding** (splitting data into subsets, divided over different processes)
  → Static vs. dynamic distribution?
- **Replication** (copying of data over processes)
  → Strong vs. eventual consistency?
- **Load balancing** incoming requests from clients
  → Static vs. dynamic?
- **Caching**
  → How to handle invalidation?
- … You are free to explore other directions or combine the above techniques.

Evaluation

Benchmarks: generate many client processes that each do a number of operations (store, retrieve).

You can measure metrics like latency & throughput of operations, in variety of situations.

You should do three experiments:

- Speed-up for increasing number of threads
- "Best-case" scenario
- A scenario you can choose freely
  (ideas: change types of requests, change R/W ratio, change # buckets, change # keys/bucket...)

We provide a basic benchmark in Erlang and a matplotlib script in Python to process the results.

Evaluation: hardware

Run experiments on:

- your machine or machine in computer room (requirement: ≥ 4 cores)
- “Firefly”: 64-core server at lab (we will arrange remote access)

Report

Table of contents in assignment sheet:

Implementation

- Describe architecture (incl. diagrams)
- How do you ensure scalability?

Evaluation

- Describe set-up, metrics
- Include plots of results
- Explain what you see and why you see it

Insight questions

- Hypothetical extensions

Details

Deadline: Thursday 10th of April, 23:59
Submit ZIP with implementation & report on Canvas

Project defense in June
⅓ of your final grade
Per day too late, -2 points
We check for plagiarism

Details on Canvas