Shopping List Manager

SDLE Project Presentation

Diogo Viana – up202108803
Gonçalo Martins – up202108707
Maria Sofia Minnemann - up20200734

Index

Overview

Objective:

In this project, we designed a distributed system to manage shopping lists, combining offline persistence with cloud synchronization.

Key Features

- **Persistent Local Storage**: Saves shopping lists locally for offline access.
- **Cloud Synchronization**: Updates are shared through cloud storage.
- **Collaborative Real-Time Editing**: Multiple users can edit lists simultaneously without conflicts.

Technologies used

- **Python**:
  - Used to implement the system.
- **Conflict-free Replicated Data Types (CRDTs):**
  - Enables real-time collaboration with conflict free updates.
- **Dynamo-inspired Architecture**:
  - Uses consistent hashing and data replication for scalability and fault tolerance.
  - Implements a gossip protocol to check node is alive.
- **ZeroMQ**:
  - Manages communication between clients, proxy and nodes.
  - Dealer-worker model for load balancing .
  - REQ-REP pattern for request handling, data replication and gossip protocol.
- **Compression:**
  - Data is sent after being compressed to ZIP format using zlib and jsonpickle.
- **JSON Local Storage**:
  - Stores shopping lists offline for persistence and offline access.

Conflict Resolution

Pros of CRDTs

- Concurrency Handling
- Automatic Conflict Resolution
- Fault Tolerance
- Scalability
- Robustness

Cons of CRDTs

- Complexity
- Resource Usage
- Latency

Implementation

- pn_counter :
  - Tracks items quantities
  - Ensures conflict-free adjustments
- or_set:
  - Resolves add-remove conflicts for managing list IDS and items
- or_map:
  - Manages shopping lists with or_set and pn_counter
  - Tracks items in the add_map, removed_map and acquired_map
- shopping_list
  - Acts as the core interface to manage shopping lists using CRDTs, managing items, combining the or_map, or_set and pn_counter crdts

Local Storage:

Ensure data persistence on the user's device and supports offline functionality.

Each shopping list is composed by:

- add_map: stores all items
- removed_map: tracks removed items
- acquired_map: tracks acquired items

Shopping lists are stored locally in a JSON file and are managed by CRDTs


```json
{
  "ce7e23a8-c98c-44f0-902f-d7cb336960cc": {
    "add_map": {
      "7f5ca9bb-e0eb-4d17-bc15-df70b67623a7": {
        "name": "pear",
        "pn_counter": {
          "positive": 0,
          "negative": 0
        },
        "acquired": false
      },
      "b293f8e5-c65d-4d15-b628-fe3cdd767281": {
        "name": "banana",
        "pn_counter": {
          "positive": 1,
          "negative": 0
        },
        "acquired": true
      },
      "531aaa44-68aa-49cc-a3f2-0251b4edd716": {
        "name": "apple",
        "pn_counter": {
          "positive": 5,
          "negative": 0
        },
        "acquired": false
      }
    },
    "removed_map": {
      "7f5ca9bb-e0eb-4d17-bc15-df70b67623a7": {
        "name": "pear",
        "pn_counter": {
          "positive": 0,
          "negative": 0
        },
        "acquired": false
      }
    },
    "acquired_map": {
      "b293f8e5-c65d-4d15-b628-fe3cdd767281": {
        "name": "banana",
        "pn_counter": {
          "positive": 1,
          "negative": 0
        },
        "acquired": true
      }
    }
  }
}
```

Cloud Side

Inspired by Dynamo

Implements distributed storage for shopping lists across multiple servers. Dynamo-inspired features:

- Consistent Hashing
- Data Replication
- Gossip Protocol
- Conflict Resolutions
  - Uses CRDTs for conflict-free merging of lists during concurrent updates

Merge happens in the node before replication

Pros

- Scalability
- Fault Tolerance
- Availability
- Efficiency

Cons

- Complexity
- Node rebalancing
- Consistency
- Replication overhead

Cloud Side

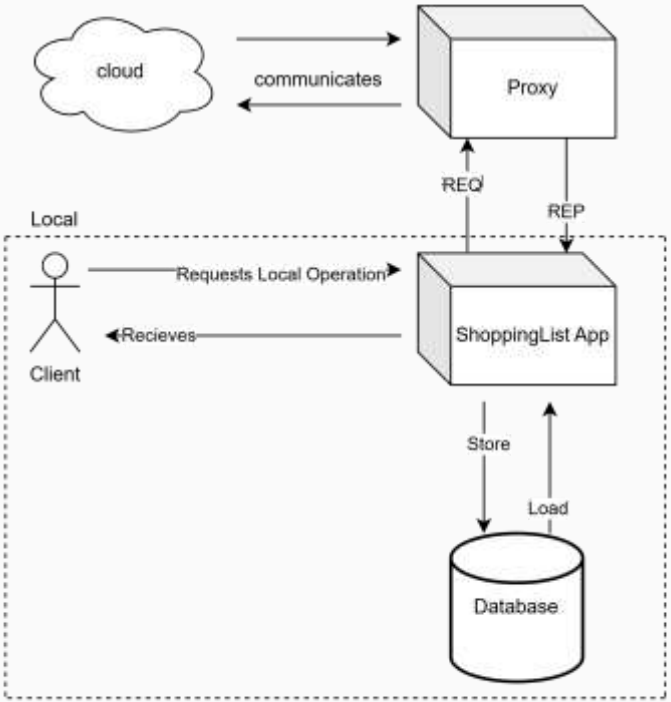
Data Distribution

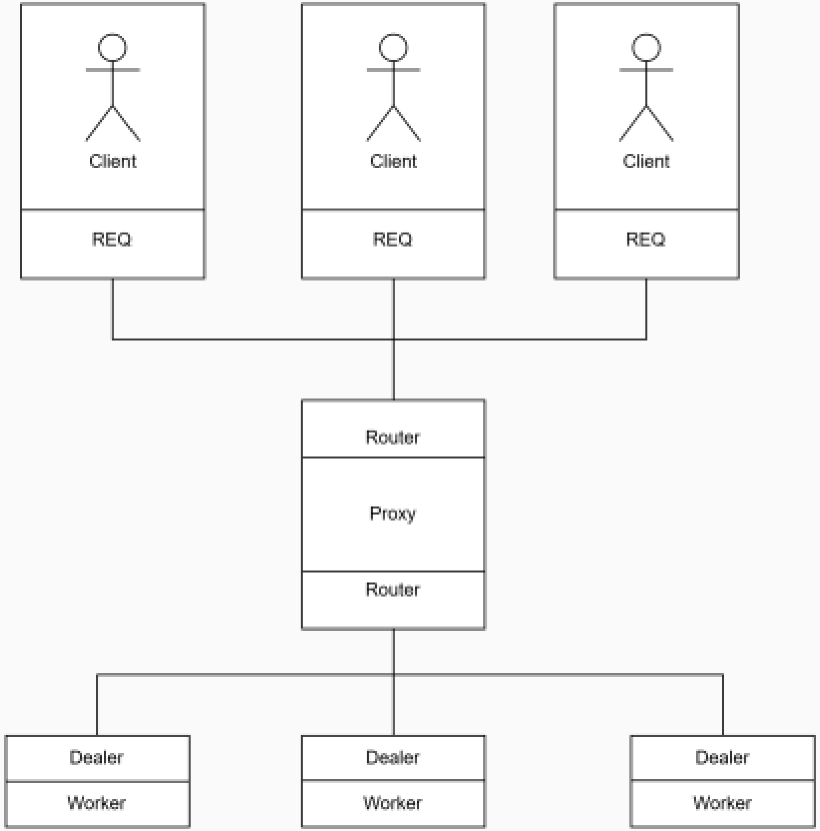
- Hash ring implementation:
  - Maps lists IDs to specific hash ranges (mapping them to nodes)
  - Ensures even distribution of data across servers
- Replication(Req/Rep)
  - Ensures fault tolerance by replicating data to 2 backup nodes.
  - Ensures data availability if main node fails.
- Gossip protocol(Req/Rep)
  - Checks node states and updates the hash ring if any state changes.

Implementation

1. Client sends request to the proxy
2. Proxy:
   I. Implements a Dealer/Worker Model (workers act as independent dealer, distributing and processing requests)
   II. Manages communication between clients and servers using ZeroMQ
   III. Uses a hash ring to determine the responsible node for a given list
3. Nodes:
   I. Each node stores different shopping lists:
      - Updates list by merging local and incoming changes with CRDT
      - Replicates the updated list to 2 nodes using REQ/REP

Architecture





User Interface

```
Welcome!

How would you like to proceed?

[1] - Create new shopping list
[2] - Edit existing shopping list
[Q] - Quit

Please choose one of the choices displayed above:
```

```
Which of the following operations would you like to perform?

[1] - Add product to cart
[2] - Remove product from cart
[3] - Edit quantity of product in cart
[4] - Purchase product in cart
[5] - Check cart
[6] - Delete list
[Q] - Quit

Please choose one of the options displayed above:
```

The app runs in the terminal

Simple, text-based interactions for managing shopping lists

Available options:

- Create a Shopping List
- Edit a shopping list
- Manage item quantities
- Mark item as acquired
- Remove an item from the shopping list
- View Shopping list
- Delete shopping list

Discussion

Challenges

- **Consistency vs Availability**: Ensuring eventual consistency while maintaining system availability
- **Node failure**: Handling failures without data loss or service interruption.
- **Replication overhead**: Managing the cost of data duplication across nodes.
- **Data balancing**: Efficiently distributing data through nodes.

Solutions

- **CRDT integration**: Guarantees conflict-free merging and eventual consistency.
- **Replication strategy**: Replicates data to two nodes for fault tolerance and availability.
- **Compression**: Reduces data size to minimize network overhead.
- **Consistent hashing**: Limits data movement by evenly distributing data across nodes.

Conclusions

- Sucessfully implemented a distributed, scalable shopping list system.
- Used CRDTs for conflict-free updates and Dynamo-inspired architecture for data distribution and replication.
- Used ZeroMQ for efficient communication and load balancing.

Future Work

Performance optimization: improve replication and gossip efficiency for large-scale systems and improvements in node/proxy/client communication to fully avoid service blockage.