Bulletin Board Consistency Project

CSCI 565: Distributed Computing Systems
Fall 2024
Christian Moser

October 30, 2024

1. Design Document

System Architecture

The Bulletin Board System (BBS) is designed as a distributed application that allows clients to post, reply, and read articles maintained across replicated servers. The system enforces consistency using three primary models:

- **Sequential Consistency**: Ensures a globally consistent ordering of updates visible to all clients.
- **Quorum Consistency**: Relies on read/write quorums to coordinate updates across multiple servers.
- **Read-your-Write Consistency**: Ensures a client sees its updates, even after switching servers.

Core Components

Coordinator: The coordinator assigns unique IDs for articles and manages global ordering. All servers interact with the coordinator to obtain IDs and confirm operations, ensuring consistent article numbering.

Servers: Each server can handle client operations and enforce one of the three consistency models. Servers maintain a local copy of the bulletin board data and synchronize with peers as needed.

Clients: The client interface enables users to connect to a server, post new articles, reply to existing ones, and read articles in a globally ordered view.

API Design

The API for communication between clients and servers supports the following commands:

- `POST <message>`: Adds a new article to the BBS.
- `READ`: Retrieves a list of all articles.
- `REPLY <article_id> <message>`: Adds a reply to an existing article.

Consistency Mechanisms

Each consistency model is implemented as follows:

- **Sequential Consistency**: Implements a primary-backup protocol, where one server (the coordinator) manages all write operations.
- **Quorum Consistency**: A quorum protocol requires NR servers for reads and NW servers for writes, meeting the constraints $NR+NW > N$ and $NW > N/2$.
- **Read-your-Write Consistency**: A client performing a write on one server is guaranteed to see the update when switching to any other server.

2. Instructions for Using the Code

Running Each Component

Server: Start each server with a unique port. Example command:

```
cargo run --bin bbs_server <port>
```

The server will start on the specified port, establishing connections with peers as defined in the configuration file.

Client: Connect a client to any server by specifying the server's address and port:

```
cargo run --bin bbs_client --server <server_address>
```

Using the System

Client Commands:

- `POST <message>`: Adds a new article to the bulletin board.
- `READ`: Displays a list of all articles with IDs, titles, and indentation for replies.
- `REPLY <article_id> <message>`: Posts a reply to an existing article.

Configuration

The system configuration, stored in `config.toml`, specifies server addresses, consistency type, and quorum sizes (NR, NW). This file allows fine-tuning of system parameters without modifying code.

3. Client and Server Features

Client Features

- **Make Posts**: Clients can create new articles by sending a `POST` command to any server. The server assigns a unique ID and coordinates with the quorum to ensure consistency.
- **List Articles**: The `READ` command displays all articles, showing replies indented under their parent articles.
- **Choose Article to Read**: Clients can select an article by ID to read the full contents.
- **Reply**: The `REPLY` command adds a response to an existing article, linked by the parent's ID.
- **Connect**: Clients can connect to any server to interact with the BBS. Each connection is authenticated and handled independently.

Server Features

- **Ordering and Numbering**: The coordinator assigns unique, sequential IDs to each article, ensuring global ordering across servers.
- **Sequential Consistency**: Sequential consistency is enforced by routing all updates through the primary coordinator server, ensuring all clients see the same article order.
- **Quorum Consistency**: Quorum consistency relies on dynamically chosen read and write quorums. The coordinator directs each request to meet quorum constraints, with $NR + NW > N$ and $NW > N/2$.
- **Quorum Synch Function**: Synchronizes updates across all replicas in the background to maintain consistency within the quorum.
- **Read-Your-Write Consistency**: A client's writes are visible even after switching servers by enforcing replication of recent updates on the newly connected server.

- **Leader Election**: If the coordinator fails, a new coordinator is elected from the available servers. The leader election protocol is triggered on failure detection.

4. Quality of Source Code

The source code is structured into modules for readability and ease of maintenance:

- `server.rs`: Contains server logic, handling requests and propagating updates across replicas.
- `client.rs`: Handles client-side interaction, user commands, and server connections.
- `models.rs`: Defines data models such as `Article` and `ArticleList` used across the system.
- `websocket.rs`: Manages WebSocket connections for client-server communication, implementing message handlers and broadcast functions.

In-line comments and clear function names provide easy understanding of the codebase, and the use of Rust's type system enhances reliability.