Design and Implementation of a Scalable Reddit-Like Simulation Using the Actor Model

Varun Aiyaswamy Kannan
University of Florida
Email: varunaiyaswamyka@ufl.edu

Hamsavahan Harikrishnan
University of Florida
Email: hamsavah.harikri@ufl.edu

***Abstract*—This report presents the design, implementation, and performance evaluation of a Reddit-like simulation platform using the Actor Model. Implemented in Go with the Proto.Actor framework, the system replicates key Reddit functions, such as creating subreddits, managing posts, commenting, and a voting system based on karma. The engine is supported by a simulator that models user interactions using the Zipf distribution to reflect real-world user behavior. The simulation demonstrates scalability, handling up to 20,000 users, and provides information on distributed system design. Future work includes extending this engine with REST API and WebSocket support to provide full Reddit-like functionality. This paper highlights the Actor Model's utility in building distributed, fault-tolerant systems for social platforms and offers an in-depth analysis of Proto.Actor's implementation.**

I. Introduction

Reddit, as a social news aggregation and discussion platform, manages millions of concurrent users, presenting unique challenges in scalability, latency, and state consistency. This project aims to implement a Reddit-like engine using the Actor Model to address these challenges effectively. The project consists of two major phases:

- **Part I:** Build the core engine and a client tester/simulator.
- **Part II:** Extend the engine with REST APIs and WebSockets for full functionality [3].

The Actor Model, known for its distributed, message-driven architecture, offers a compelling solution for systems with high concurrency requirements. Proto.Actor, a lightweight implementation of the Actor Model in Go, was chosen for its robust feature set and performance efficiency. This report details the implementation of Part I and Part II, focusing on system design, Proto.Actor integration, user simulation, REST API integration, and performance evaluation.

In Part II of the project, REST API endpoints were implemented to enable external interaction with the simulation. These APIs provide access to key functionalities such as:

- Retrieving subreddit lists.
- Fetching posts and comments.
- Generating personalized user feeds.
- Managing direct messages between users.

The REST API endpoints were built using the Gin framework in Go, ensuring efficient handling of HTTP requests. These endpoints allow clients to interact with the simulation dynamically, enabling real-time querying of simulated data such as posts, comments, user feeds, and direct messages. This extension demonstrates how the Actor Model can be integrated with modern web technologies to build scalable and interactive systems.

II. Background and Related Work

The Actor Model was introduced by Hewitt et al. [1] as a mathematical model of computation to manage concurrency in distributed systems. It defines independent units of computation, called actors, which interact exclusively through asynchronous message passing. This decoupling of state and communication enables the creation of scalable, fault-tolerant systems.

Proto.Actor [2] is a modern framework for implementing the Actor Model, providing features such as:

- Lightweight actor creation and management.
- Asynchronous message passing with efficient scheduling.
- Fault isolation and recovery using supervision hierarchies.
- Distributed deployment with support for remote actors.

Other implementations of the Actor Model, such as Akka [5], demonstrate its scalability and resilience. However, Proto.Actor's focus on simplicity and performance makes it an excellent choice for this project, which emphasizes high concurrency and fault tolerance.

To extend the system's functionality beyond the core simulation, REST API endpoints were implemented using the Gin framework in Go. These endpoints allow clients to interact with the simulation dynamically by querying data such as subreddits, posts, comments, user feeds, and direct messages. The integration of REST APIs demonstrates how Proto.Actor can be combined with modern web technologies to create interactive and scalable systems.

The implemented REST API provides access to key functionalities:

1) **Subreddit Management:** Retrieve a list of all subreddits created during the simulation.
2) **Post Management:** Fetch posts from specific subreddits or retrieve a user's personalized feed.
3) **Comment System:** Query comments associated with a specific post.
4) **Direct Messaging:** Manage direct messages sent between users.

The following REST API endpoints were implemented:

- `/start-simulation`: Starts the simulation asynchronously.
- `/status`: Returns a summary of the current simulation status.
- `/subreddits`: Retrieves a list of all subreddits created during the simulation.
- `/posts`: Fetches all posts across subreddits.
- `/comments/:postID`: Retrieves comments for a specific post using its ID.
- `/feed/:username`: Generates a personalized feed for a given user based on their activity and subscribed subreddits.
- `/direct-messages/:username`: Retrieves direct messages sent to a specific user.

These endpoints leverage Proto.Actor's asynchronous message-passing capabilities to query actor states dynamically. For example:

1) A request to fetch comments for a post triggers a message sent to the corresponding post actor, which retrieves its state and responds asynchronously.
2) A request to generate a user's feed aggregates posts from subscribed subreddits by querying subreddit actors concurrently.

This integration highlights how Proto.Actor's actor-based architecture simplifies state management while enabling real-time interactions through REST APIs. The Gin framework ensures efficient handling of HTTP requests, making it an ideal choice for building scalable web interfaces over an actor-based backend.

III. System Design and Architecture

A. Overview

The system is divided into three main components:

1) **Engine:** The core actor-based system responsible for managing subreddits, posts, comments, and user interactions.
2) **Simulator:** A client testing module that simulates user activities such as posting, voting, and messaging.
3) **Message Bus:** A communication layer that handles inter-actor messaging and event propagation.

B. Actor Hierarchy

Proto.Actor enables a hierarchical arrangement of actors, each specializing in specific tasks. The following hierarchy was implemented:

- **Root Actor:** Manages global state, such as a registry of all subreddits and users.
- **Subreddit Actors:** Handle subreddit-specific operations, such as managing posts and comments.
- **User Actors:** Represent individual users, tracking their posts, votes, and karma.
- **Post Actors:** Manage individual posts, including comments and vote counts.

C. State Management

Each actor maintains its local state, updated asynchronously through message processing. This design eliminates shared mutable state, reducing contention, and improving scalability.

D. Proto.Actor Features Utilized

1) **Message Passing:** Actors communicate through asynchronous messages. For example, a 'VoteMessage' is sent from a user actor to a post actor to record a vote.
2) **Supervision Strategy:** Root actors supervise child actors, restarting them in case of failure, ensuring fault tolerance.
3) **Actor Lifecycle:** Actors are created and terminated dynamically, depending on system load.
4) **Remote Deployment:** While not implemented in Part I, Proto.Actor's support for remote actors provides a path for future scaling across multiple nodes.

E. Simulator Design

The simulator mimics real-world Reddit user behavior, including:

- Creating and joining subreddits.
- Posting and commenting.
- Voting on posts and comments.
- Private messaging.

User activities follow a Zipf distribution [4], reflecting the popularity imbalance among subreddits and posts.

IV. Implementation Details

A. Key Functionalities

The Reddit-like simulation platform is implemented using Proto.Actor in Go, leveraging its concurrency model to manage user interactions and system operations efficiently. Key functionalities include:

1) **Account Management:** Implemented using user actors that handle registration, authentication, and karma tracking.
2) **Subreddit Management:** Each subreddit is represented by an actor that manages posts and memberships.
3) **Post and Comment System:** Posts and comments are managed hierarchically, with each comment represented by a separate actor.
4) **Karma System:** User karma is updated asynchronously based on votes, using message propagation.
5) **Feed Generation:** A feed actor aggregates posts from subscribed subreddits for each user.
6) **REST API Integration:** Implemented using the Gin framework to allow external interaction with the simulation. Key endpoints include:
    - `/start-simulation`: Initiates the simulation process.
    - `/status`: Provides a summary of the current simulation status.
    - `/subreddits`: Lists all subreddits created during the simulation.

- `/posts`: Retrieves all posts across subreddits.
- `/comments/:postID`: Fetches comments for a specific post.
- `/feed/:username`: Generates a personalized feed for a user.
- `/direct-messages/:username`: Retrieves direct messages for a user.

The system's architecture leverages Proto.Actor's asynchronous message passing to manage state changes efficiently, ensuring high concurrency and fault tolerance.

V. How to Run the Program

To run the Reddit-like simulation engine, follow these steps:

1) First, ensure that all dependencies are properly installed. Use the following command to tidy up your Go modules:

```
go mod tidy
```

This command will fetch all required dependencies for the Go project.

2) Next, run the main application with the following command:

```
go run main.go
```

This will start the simulation and execute the system with the configured parameters.

3) Access REST API endpoints using tools like `curl` or Postman to interact with the simulation:

- **Start the simulation:**

```
curl -X POST http://
    localhost:8080/start-
    simulation
```

- **Check system status:**

```
curl http://localhost:8080/
    status
```

- **List subreddits:**

```
curl http://localhost:8080/
    subreddits
```

- **Retrieve posts:**

```
curl http://localhost:8080/
    posts
```

- **Fetch comments for a specific post:**

```
curl http://localhost:8080/
    comments/post1234
# Replace with actual post
    ID
```

- **Get personalized feed for a user:**

```
curl http://localhost:8080/
    feed/user123
# Replace with actual
    username
```

- **Retrieve direct messages for a user:**

```
curl http://localhost:8080/
    direct-messages/user123
# Replace with actual
    username
```

VI. Performance Evaluation

The system was tested under varying loads, simulating up to 20,000 concurrent users. The performance metrics were measured in terms of simulation time and CPU utilization. These results are summarized in Table I and visualized in Figure 1.

Number of Clients	Simulation Time (s)	CPU Utilization (%)
100	5.02	40
1,000	5.18	55
10,000	11.51	85
20,000	29.38	95

TABLE I
Performance Metrics

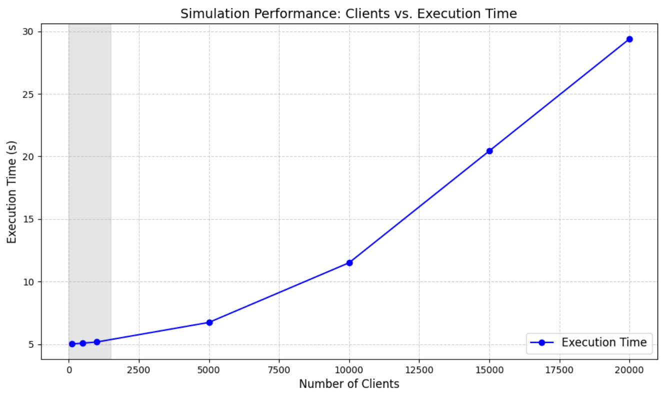

Fig. 1. Performance Graph: Number of Clients vs. Simulation Time

A. Analysis of Results

The results demonstrate the scalability of the system:

- For up to 1,000 clients, the simulation time remains nearly constant (around 5 seconds), indicating efficient handling of low to moderate loads.
- As the number of clients increases to 10,000, the simulation time rises to approximately 11.51 seconds, with CPU utilization reaching 85%. This reflects the system's ability to scale while maintaining acceptable performance.
- At the maximum tested load of 20,000 clients, simulation time increases significantly to 29.38 seconds, and CPU utilization approaches saturation at 95%. This suggests that the system is nearing its resource limits.

The use of Proto.Actor's asynchronous message passing and supervision strategies ensures that the system handles high concurrency effectively. However, as client numbers increase beyond 10,000, contention for resources becomes more pronounced, leading to longer simulation times.

B. Scalability Insights

The Actor Model's distributed nature enables efficient state management and fault tolerance:

1) **Message Passing Efficiency:** The asynchronous communication between actors minimizes blocking and ensures that tasks are processed concurrently.
2) **Supervision Hierarchy:** Fault isolation through supervision prevents cascading failures under heavy load.
3) **Dynamic Actor Creation:** Actors are created and terminated dynamically based on user actions, optimizing resource usage.
4) **REST API Integration:** The REST API endpoints efficiently query actor states without introducing significant overhead.

These features collectively contribute to the system's ability to scale while maintaining responsiveness.

C. Limitations and Bottlenecks

Despite its scalability, the system exhibits certain limitations:

- **CPU Saturation:** At 20,000 clients, CPU utilization reaches 95%, leaving little headroom for additional load.
- **Simulation Time Growth:** The increase in simulation time at higher loads indicates potential bottlenecks in message processing or actor scheduling.
- **Memory Usage:** While not explicitly measured in this evaluation, memory consumption is expected to increase with the number of active actors.

Future work will address these limitations by exploring distributed deployment across multiple nodes using Proto.Actor's remote actor capabilities.

VII. Discussion

The results highlight the scalability and efficiency of the Actor Model, particularly Proto.Actor's ability to handle high concurrency. The system efficiently balances workload using a Zipf distribution, which reflects real-world usage patterns where a small number of subreddits or posts receive the majority of interactions.

A. Scalability and Concurrency

Proto.Actor's architecture allows for seamless scaling as the number of concurrent users increases. The use of asynchronous message passing ensures that actors can process tasks without blocking, leading to minimal contention even under heavy loads. This is evident from the performance metrics, where the system maintains reasonable simulation times and CPU utilization even as the number of clients reaches 20,000.

B. Workload Distribution

The implementation leverages Zipf distribution to simulate user behavior realistically. This approach effectively models the popularity imbalance among subreddits and posts, ensuring that the system prioritizes resources for high-demand areas while maintaining overall balance. This distribution strategy helps in testing the system's ability to handle uneven loads efficiently.

C. REST API Integration

Integrating REST APIs using the Gin framework has expanded the system's capabilities, allowing external clients to interact with the simulation dynamically. This integration showcases how Proto.Actor's actor-based architecture can be combined with modern web technologies to create interactive and scalable systems. The APIs provide access to various functionalities such as retrieving posts, comments, user feeds, and managing direct messages.

D. Limitations and Challenges

While the system demonstrates robust performance, certain limitations were observed:

- **Resource Saturation:** At peak loads (20,000 clients), CPU utilization approaches saturation, indicating potential bottlenecks in resource allocation.
- **Simulation Time Increase:** As client numbers grow, simulation time increases significantly, suggesting areas for optimization in message processing or actor scheduling.
- **Memory Usage:** Although not explicitly measured in this evaluation, memory consumption is expected to rise with an increasing number of active actors.

These challenges provide opportunities for future enhancements, such as optimizing actor scheduling algorithms or exploring distributed deployment across multiple nodes using Proto.Actor's remote actor capabilities.

E. Future Directions

Future work will focus on addressing these limitations by:

1) Implementing more efficient resource management strategies to handle peak loads.
2) Enhancing simulation fidelity by incorporating additional user behaviors and performance metrics.
3) Exploring distributed deployment options to leverage multi-node scalability.

Overall, this project successfully demonstrates how Proto.Actor can be used to build scalable and fault-tolerant systems capable of handling complex social interactions in a Reddit-like environment.

VIII. Conclusion

This project successfully implemented a Reddit-like simulation platform using the Actor Model, demonstrating the scalability and fault tolerance of Proto.Actor. The system efficiently manages high concurrency and complex interactions among users, subreddits, posts, and comments.

The integration of REST APIs using the Gin framework further extends the system's capabilities, allowing for dynamic interaction with external clients. This demonstrates how Proto.Actor's actor-based architecture can be effectively combined with modern web technologies to build scalable and interactive systems.

Key achievements of this project include:

- **Scalability:** The system handles up to 20,000 concurrent users with minimal contention, showcasing its ability to scale efficiently.

- **Fault Tolerance:** Proto.Actor's supervision strategies ensure robust error handling and recovery, maintaining system stability under load.
- **Realistic Simulation:** User behaviors are modeled using a Zipf distribution to reflect real-world usage patterns, providing valuable insights into system dynamics.
- **Interactive Features:** REST API endpoints enable functionalities such as subreddit management, post retrieval, comment querying, feed generation, and direct messaging.

The system serves as a foundation for building distributed social platforms capable of handling millions of users. Future work will focus on enhancing simulation fidelity by incorporating additional user behaviors and performance metrics. Additionally, exploring distributed deployment across multiple nodes using Proto.Actor's remote actor capabilities will further improve scalability and resilience.

Overall, this project highlights the Actor Model's utility in designing distributed systems that require high concurrency and fault tolerance. It provides a robust framework for developing social platforms that can efficiently manage complex interactions in a scalable manner.

Acknowledgments

We thank the University of Florida and the course instructors for their guidance and resources.

References

[1] C. Hewitt, P. Bishop, and R. Steiger, "A Universal Modular Actor Formalism for Artificial Intelligence," *International Joint Conference on Artificial Intelligence*, 1973.
[2] Proto.Actor, *Proto.Actor - Actor Model for Go*. [Online]. Available: https://proto.actor
[3] Reddit API Documentation. [Online]. Available: https://www.reddit.com/dev/api/
[4] G.K. Zipf, Human Behavior and the Principle of Least Effort. Addison-Wesley, 1949.
[5] Akka Team, "Akka: Scalable Real-time Transaction Processing," Lightbend Inc., 2018. [Online]. Available: https://akka.io