Project Specification: Distributed Online Game Backend

(Multiplayer Checkers)

Reza Almassi
MSc Artificial Intelligence and Data Engineering
University of Pisa

Professor: Alessio Bechini
Distributed Systems and Middleware Technologies
Academic Year 2024/2025

July 7, 2025

1. Introduction

This project aims to design and implement a distributed online game backend for the classic board game, **Checkers**, as part of the Distributed Systems and Middleware Technologies course. The system will showcase essential distributed systems concepts by enabling multiple users to play Checkers online in real time, with the system running across multiple networked nodes.

2. Objectives

- Develop a distributed software platform enabling real-time multiplayer Checkers games.
- Address synchronization and coordination challenges typical in distributed environments.
- Ensure robust and reliable communication between distributed components and clients.
- Demonstrate fault tolerance and dynamic recovery using Erlang for coordination.
- Deploy and present the system operating over multiple physical or virtual nodes.

3. System Architecture

The proposed system comprises the following main components:

1. **Game Client** (Java): Provides a graphical or command-line interface for players to interact, submit moves, and receive real-time updates.
2. **Game Server Nodes** (Java): Manage game logic, handle player sessions, enforce rules, and maintain game state synchronization.
3. **Coordination Service** (Erlang): Implements registry, monitoring, and leader election for game sessions, ensuring high availability and resilience.

4. **Communication Layer**: Facilitates message passing between clients, game servers, and the Erlang coordination service, using TCP sockets, RESTful APIs, or gRPC.

4. Key Distributed Systems Concepts

- **Synchronization**: Maintains consistent turn-taking and enforces valid move sequences.
- **Coordination**: Assigns players to games, manages game sessions, and handles leader election (using Erlang) in the event of server failure.
- **Communication**: Provides reliable, low-latency message passing between distributed nodes and clients.
- **Fault Tolerance**: Detects and recovers from failures using Erlang's OTP features and supervision trees.
- **Scalability**: Supports multiple parallel Checkers games distributed across nodes.

5. Technology Choices

- **Java**: Selected for implementing the game server logic and client interfaces due to its maturity and ecosystem.
- **Erlang/OTP**: Used for the coordination service, leveraging its strengths in distributed concurrency and fault-tolerance.
- **Communication**: Inter-process communication via TCP sockets, RESTful APIs, or gRPC, as best fits the system.
- **Deployment**: System will be deployed across several virtual or physical machines (e.g., VMs or Docker containers).

6. Component Responsibilities and Technology Mapping

Game Client (Java): Implements the user interface for players, supporting both command-line and graphical options. The client is responsible for capturing player moves, displaying game state updates, and interacting with game servers over the network.

Game Server Nodes (Java): Each game server manages active Checkers games, processes player moves, validates game logic, and maintains local game state. Servers handle real-time message passing with clients and regularly synchronize session status with the coordination service.

Coordination Service (Erlang/OTP): The Erlang coordination service provides:

- A central *Game Session Registry*, tracking all active game sessions and their associated servers.

- *Leader Election and Failover*: Detects server failures using heartbeat messages and elects a new game server to recover the session.
- *Monitoring*: Supervises the health and status of game servers, ensuring high availability through Erlang's robust OTP framework.

Inter-Process Communication:

- Game clients communicate with Java game servers via TCP sockets or RESTful APIs for move submission and state updates.
- Java servers communicate with the Erlang coordination service using RESTful HTTP endpoints, ensuring interoperability between Java and Erlang components.

This mapping of functionality ensures clear modularization, leveraging Java's strengths for rapid application development and user interaction, while utilizing Erlang's concurrency and fault-tolerance capabilities for system coordination and resilience.

7. Synchronization, Coordination, and Communication Strategies

- **Turn Management**: Enforces correct turn order and legal move execution using distributed mutual exclusion or logical clocks.
- **Game Session Management**: The Erlang coordination service tracks active games, assigns players, and provides failover capabilities.
- **Leader Election and Recovery**: On detecting server failure, the Erlang service elects a new leader server and reallocates the affected game session.
- **Real-Time Updates**: Clients and servers communicate asynchronously, with game state updates broadcast to all players in a session.

8. Demonstration and Evaluation

- The system will be deployed on at least three distributed nodes.
- The demonstration will cover user join, live gameplay, simulated server failure and recovery, and real-time state synchronization.
- Complete documentation and a user manual will accompany the project submission.

9. Conclusion

This project will demonstrate practical distributed systems concepts and middleware technologies by implementing a resilient, scalable, and interactive online Checkers platform. The design will combine Java and Erlang, justify all technology decisions, and clearly address synchronization, coordination, and communication challenges in distributed systems.