DISTRIBUTED CONNECT FOUR

Professor : Andrea Omicini

Subtitle: A Fault-Tolerant, Highly Available Architecture

Presenter: Arvin Lajani arvin.lajani@studio.unibo.it

University of Bologna

Architectural Foundations & The CAP Theorem

- **The CAP Choice (A/P Focus with CA Balance):** The architecture favors High Availability (A) and Partition Tolerance (P) during infrastructure disruptions to preserve active match states across abrupt software crashes.
- **The Consistency Model:** *Immediate Primary Validation:* Moves are processed synchronously by the authoritative leader's engine, eliminating illegal or conflicting board states.
- *Eventual Backup Convergence:* The Primary node streams transactional logs to the warm standby. While replication introduces a microscopic network transmission gap, the system guarantees **Eventual Consistency** once the log queue flushes.

Project Topology & Modular Separation

- **Structural Decoupling:** The project is modularized into independent packages to ensure strict separation of concerns:
- game_logic/: Pure, stateless mathematical grid engine.
- game_service/: Authoritative primary server and standby replication daemons.
- client/: PyQt6 presentation layers and asynchronous network worker threads.
- common/: Shared JSON message schemas and serialization protocols.
- **Visual Concept:** A clean system architecture diagram mapping the client-to-server data pathways.

The Core Coordination Services

- **The Matchmaking Service:** A stateless gateway operating as the initial entry point. It handles client handshakes, manages an asynchronous FIFO waiting queue, pairs users, generates a unique game_id, and issues connection configuration redirects.
- **The Authoritative Primary Node:** Manages active multi-tenant game containers, enforces validation rules, and broadcasts state deltas.
- **The Warm Standby Backup Node:** Listens to live transaction streams, mirroring memory matrices in real time, poised for automated election.

Domain Logic vs. Infrastructure Management

- **The Rulebook (ConnectFour):** Pure mathematical domain logic. It is completely blind to networking, player IDs, or timers. It strictly governs 6x7 grid constraints, piece gravity, and line checks.
- **The Referee (GameManager):** The operational orchestrator. It manages multi-tenancy (mapping multiple concurrent matches via game_id), tracks player identities, and coordinates asynchronous life cycles.
- **The $O(1)$ Optimization:** Instead of performing expensive full-board matrix traversals ($O(N)$), the engine executes an **Incremental Pivot Scan** shooting outward along 4 algebraic vector axes from the last dropped token coordinates to detect wins instantly.

Protocol Specifications Communications Bridge

- **Dual-Protocol Implementation:**
- *HTTP / REST (Port 8000):* Utilized strictly for short-lived, stateless transactional bootstrapping at the Matchmaking Gate.
- *Asynchronous WebSockets (Ports 8866 / 8766):* Long-lived, full-duplex TCP transport pipes allowing low-overhead bi-directional communication streams for active games.
- **JSON Transaction Serialization:** Every network event (move intents, state updates, heartbeats, migration targets) is strictly marshaled into predictable, human-readable JSON payloads defined in the common/ package.

Threaded Concurrency & The Graphical Thin Client

- **Presentation Isolation:** The PyQt6 client holds zero game state authority or rule validation mechanics to prevent local client-side memory tampering.
- **QThread Worker Integration:** Network polling loops are offloaded from the main GUI thread onto an asynchronous background worker.
- **Reactive Qt Signals:** When the background worker receives a BOARD_UPDATE JSON packet, it passes the new matrix up to the UI thread using thread-safe PyQt slots, preventing presentation lag or application freezes.

State Machine Replication & Automated Failover

- **State Machine Replication (SMR):** The primary server streams transaction logs to the backup server sequentially. The backup processes these logs directly to keep its internal matrices synchronized.
- **The Failure Detection Loop:** The backup node monitors the primary via a continuous heartbeat validation cycle.
- **The Failover Window:** If the primary crashes, the backup triggers a promotion sequence within a strict **10-second timeout window**, binding to the target port, taking over leadership, and prompting clients to transparently migrate their active sockets without resetting the match.

System Deployment & Dependency Sequence

- **Deterministic Boot Dependency:** Because the distributed transport layers rely on established endpoints, components must be initialized in a strict operational order:
- *Backup Daemon:* Launched first to open its replication listener port.
- *Primary Server:* Initiates second, establishing its sync bridge to the standby node.
- *Matchmaking Gateway:* Spins up its stateless web service to begin routing traffic.
- *PyQt6 Clients:* Executed by users to begin the automated pairing and redirect cycle.

Empirical Validation & Destructive Testing

- **The Chaos Test Scenario:** To validate the high-availability targets, a destructive test was executed by manually killing the primary process mid-game via a keyboard interrupt (Ctrl+C).
- **Validation Performance Benchmarks:**
- *Detection Latency:* The backup successfully registered the lost heartbeat and initiated promotion within the designed threshold.
- *State Consistency:* 100% of the volatile game matrices were preserved due to the real-time SMR log pipeline.
- *Reconnection Windows:* Client worker loops achieved transparent socket redirection and resumed active gameplay in approximately 1.2 seconds following leader promotion.

Architect's Self-Evaluation

- **Architectural Decoupling and Structural Isolation:** Isolating the entryway matchmaking gateway from the core operational servers contains process vulnerabilities, ensuring that high-volume matchmaking bursts cannot inject latency into ongoing matches.
- **Fault Resilience and Transparent Self-Healing:** The combination of SMR transaction logs and heartbeat detection ensures high availability, recovering state perfectly in volatile memory without forcing connection drops.
- **Asynchronous Concurrency Model:** Leveraging asyncio event loops ensures extreme resource efficiency, executing concurrent matches without encountering heavy thread locks, race conditions, or performance degradation.

Conclusion

- **Successful Separation of Concerns:** Completely isolated the mathematical game rules (ConnectFour) from the network transport layers (WebSockets), making the system highly modular and easy to unit test.
- **Guaranteed High Availability:** Implemented a real-time heartbeat loop and a State Machine Replication (SMR) pipeline to ensure the backup node always maintains a bit-perfect copy of the match state in memory.
- **Transparent Fault Tolerance:** Empirical crash testing proved that if the Primary server unexpectedly terminates mid-game, the Backup node executes an unassisted promotion and fully restores the active session within seconds.
- **Seamless User Experience:** Handled server failures gracefully by automatically migrating client sockets in the background, allowing players to finish their match without losing a single move or resetting the application.