Napster-Style P2P System with Replication

Architecture, Fault Tolerance, and Consistency

Vysishtya Karanam (2022102044)
Renu Sree Vyshnavi (2022101035)
Vivek Hruday Kavuri (2022114012)

Distributed Systems - Course Project

System Overview

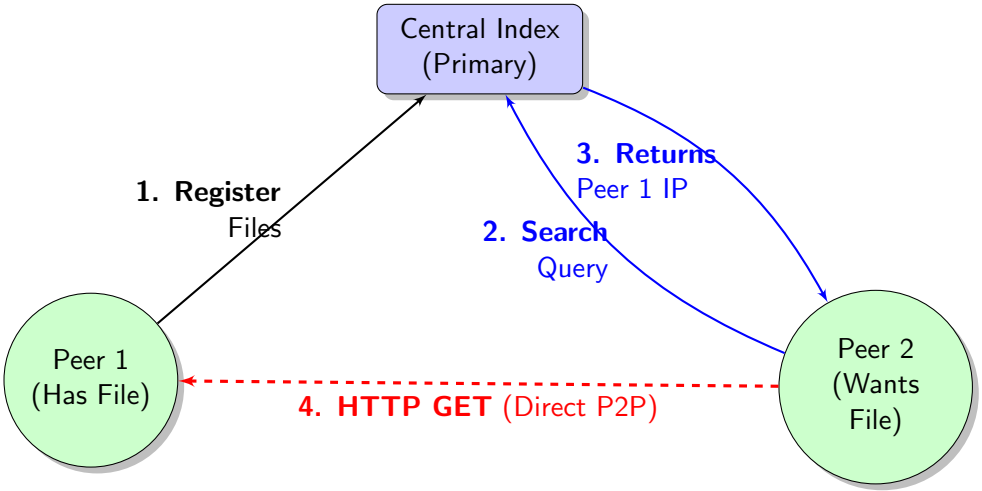
Concept

- A hybrid Peer-to-Peer file sharing system based on the centralized index model (Napster).
- **Language:** Go (Golang) for both Server and Client.
- **Communication:** JSON over HTTP.

Core Architecture

- **Central Index Server:** Stores mapping of `FileName` $\rightarrow$ `[PeerIPs]`.
- **Peer Nodes:**
  - Act as Clients (Search/Register).
  - Act as Servers (Host files for P2P download).
- **Direct Transfer:** File data flows directly between peers, not through the server.

Basic Architecture (Napster Model)



Enhancement I: Fault Tolerance

Problem: The Central Index is a Single Point of Failure (SPoF).

Solution: Primary-Shadow Replication

1. **Shadow Server:** An active standby server.
2. **SyncOps:** Primary asynchronously pushes state changes (‘Register‘, ‘Announce‘, ‘Prune‘) to Shadow via HTTP POST.
3. **Client Failover:**
   - Client configured with list: `[Primary, Shadow]`.
   - On 5xx/Timeout, client automatically retries request on Shadow.
4. **Read-Only Shadow:** Shadow rejects writes (‘403 Forbidden‘) to prevent split-brain issues.

Failover Latency:

9.59 ms
(Measured)

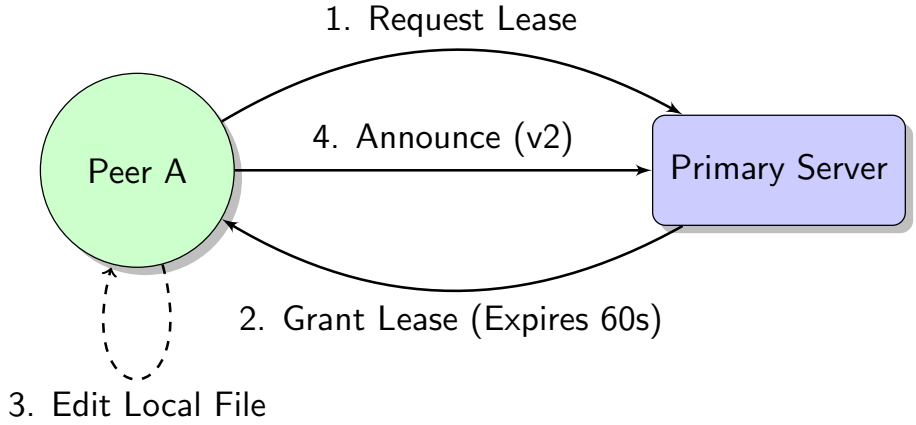
Enhancement II: Strong Consistency

Problem: Race conditions when multiple peers update the same file simultaneously.

Solution: Lease-Based Concurrency Control

- **Strict Workflow:**
  1. Peer requests a **Write Lease** for specific file.
  2. Server grants lease (TTL: 60s) if file is free.
  3. Peer edits file locally and updates version.
  4. Peer calls `/announce`. Server verifies Lease ID before accepting update.

Enhancement II: Strong Consistency



Enhancement III: Automated Data Replication

Problem: If a peer disconnects, its unique files are lost (Churn).

Solution: Active Replication Management

- **Replication Factor (R):** Configurable (Default $R = 2$).
- **Planner Logic:**
  - Server periodically scans file index.
  - If copies $< R$, it identifies "under-replicated" files.
  - Selects available peers and schedules **Replication Tasks**.
- **Piggybacked Delivery:** Tasks are sent to peers inside the response to their periodic **Heartbeat** or **Register** calls.

Interactive CLI: Features & Commands

Overview

- **Persistent Shell:** Peers run a continuous interactive session (via `-cmd serve`).
- **Efficiency:** Allows executing multiple operations (Search → Get → Update) without restarting the client process.

Available Commands

list	Displays all files currently in the local shared directory.
search	Queries the Central Index to find active peers hosting a file.
get	Initiates a P2P download (HTTP GET) and automatically announces ownership to the server.
update	Triggers the atomic consistency workflow: *Lease* → *Edit* → *Announce*.

Interactive CLI: Workflow Example

Scenario: A user searches for a document, downloads it, and then updates it.

Step 1: Search
User inputs: **search doc.txt**
↪ System returns: *Found 1 match: doc.txt (v1) hosted by Peer A.*

Step 2: Download
User inputs: **get doc.txt**
↪ System actions: Connects to Peer A, downloads file, and announces new ownership to Primary Server.

Step 3: Update (Consistency Check)
User inputs: **update doc.txt**
↪ System actions:

1. Requests **Write Lease** from Server (60s timer).
2. Opens local text editor.
3. On save, announces **Version 2** to Server.

Evaluation: Core System Performance

We benchmarked the system using a custom Python evaluation suite.

Metric	Description	Result
Replication Convergence	Time for $R = 2$ to be satisfied	14.53 ms
Write Latency	Total time (Lease + Announce)	8.80 ms
Failover Recovery	Time to switch to Shadow on crash	9.59 ms
Search Latency	In-memory map lookup (Load Test)	28.94 ms

Table: Core Metrics (Source: `eval_log.txt`)

Observation: Core operations are extremely fast (< 30ms) due to in-memory indexing and efficient Go concurrency.

Evaluation: Scalability Stress Test

Setup:

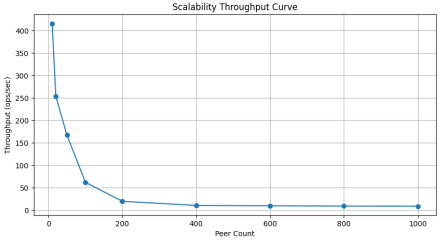
- Tested with 10 to 500 concurrent simulated peers.
- Measured throughput (Ops/Sec) and Latency.

Results (from logs):

- **10 Peers:** 233 ops/sec, 6ms latency.
- **100 Peers:** 12 ops/sec, 156ms latency.

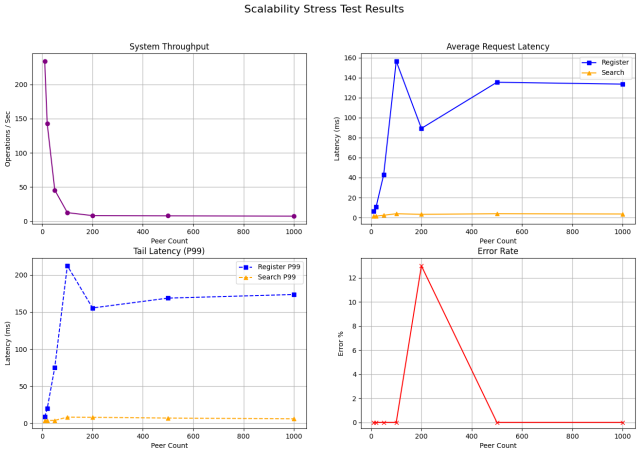
Bottleneck Analysis:

- Throughput follows a logarithmic decay.
- **Cause:** Centralized Mutex locking on the Index Map during high-frequency `/register` bursts.



Throughput vs Peer Count

Evaluation: Detailed Metrics (Visual)



Four-quadrant breakdown of system performance under increasing load (10-500 peers).

Evaluation: Metrics Analysis

Quadrant 1: Throughput (Top-Left)

- Shows a classic decay curve.
- **Cause:** Contention on the central Server Mutex lock during write-heavy operations (Register).

Quadrant 2: Average Latency (Top-Right)

- **Search (Orange):** Remains constant (< 5ms) regardless of load. Read-locks are non-blocking.
- **Register (Blue):** Latency spikes significantly after 50 peers, identifying the write-path bottleneck.

Quadrant 3: P99 Latency (Bottom-Left)

- Follows the average trend.
- Indicates predictable degradation rather than random system stalls.

Quadrant 4: Error Rate (Bottom-Right)

- **0% Error Rate** maintained.
- The system slows down but **never crashes**, proving the robustness of the client retry logic.

Demo Video

Click Here to Watch Video

(Demonstrating: Server Failover, File Replication, and Lease Enforcement)

Conclusion

Summary

- Built a robust P2P system that addresses the classic Napster weaknesses.
- **Availability:** Achieved via Shadow Master (Failover < 10ms).
- **Reliability:** Automated replication ensures data survives peer churn.
- **Consistency:** Lease mechanism prevents write conflicts.

Future Improvements

- **Sharding:** Distribute index across multiple primaries to fix throughput decay.
- **Persistence:** Use SQLite/LevelDB instead of in-memory maps.
- **NAT Traversal:** Implement UDP hole punching for real-world deployment.

Thank You!

Project 31
Distributed Systems