Distributed Music Queue System

Matthew Moran
1001900489

Introduction & Requirements

Functional Requirements

- Add/remove tracks from a shared queue
- Vote tracks up/down (reordering)
- Synchronized state across all nodes
- Retrieve track metadata
- View playback history

Non-Functional Requirements

- Scalability (5+ nodes)
- Fault tolerance (Hopefully)
- Low latency, high throughput
- Easy portable deployment (Docker Compose)
- Automated testing

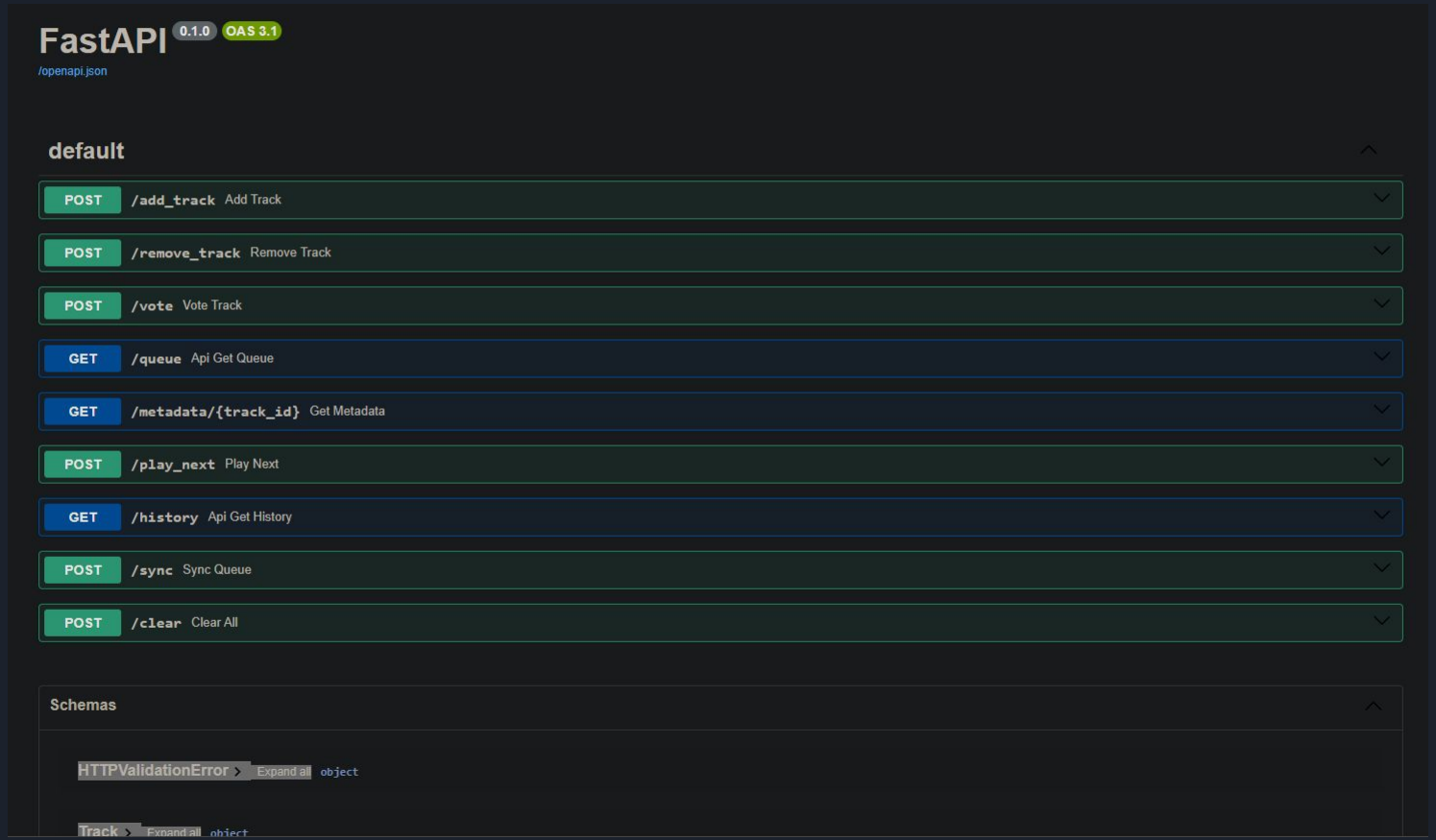
System Designs Overview

Two Architectures:

- **REST/HTTP (Layered/Resource-Based)**
  - FastAPI nodes, Nginx load balancer, Redis backend
  - Communication: HTTP/JSON
  - Stateless, resource-based, with built in web app access
- **gRPC Microservices**
  - Python gRPC nodes, Nginx load balancer, Redis backend
  - Communication: gRPC protocol
  - High performance, binary protocol, strong typing

Both use Redis as a shared backend for a distributed state and use Nginx to load balance.

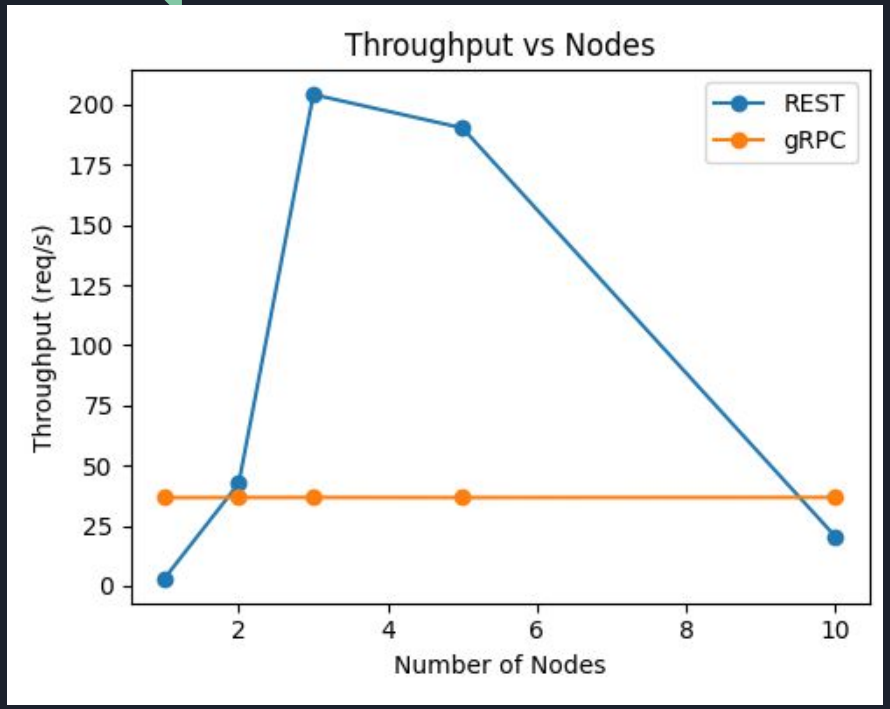
REST Web Interface

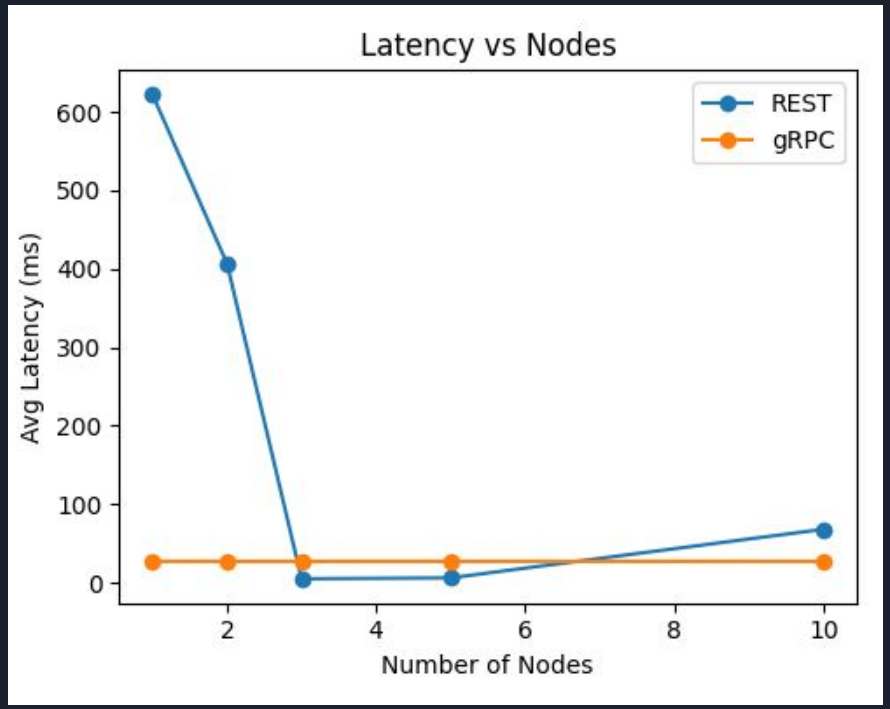


Evaluation Methodology

- **Automated Benchmarking Script**
  - Used increasing number of nodes (1, 2, 3, 5, 10)
  - 200 requests per trial across 20 threads with 3 trials per configuration
  - Measured latency and throughput
- **Test Automation**
  - All tests can be run through Docker containers
  - Uses health checks and readiness probes

Experimental Results





Results Analysis

- **REST/HTTP:**
  - Throughput and latency vary greatly with node count
  - Seemingly higher node counts don't always increase throughput and/or decrease latency. May be a mistake in implementation or evaluation
  - Sensitive to proper load balancing and resources
- **gRPC**
  - Consistent throughput and latency at all scales. May be a mistake.
  - Nginx was required to be able to load balance

AI Tools

- **ChatGPT**
  - Used to brainstorm the project idea, functional requirements, and architectures
- **Copilot**
  - Used for code generation, gave good templates but was pretty bad at large sweeping changes and error fixing.
  - Was quite good at simple error fixing or fixing my dumb mistakes
  - Was also decent at helping write the report and ReadMe

Both were helpful in accelerating doing this project under a crunch, especially for a single person

Conclusion

- REST was far more user friendly and simple to use but seemed much more affected by scaling and load
- gRPC was much more difficult to implement and test for me
- Docker was difficult to get to work but once it did it made things much easier
- AI was incredibly helpful because I had never done anything like this and acted in place of a teammate for me

Q&A