Performance Comparison Report: Async vs Non-Async gRPC Server

Date: May 5, 2025

Objective: Evaluate and compare the response time performance of two gRPC concurrency methods for the TrendStory application.

Test Setup

Two gRPC server configurations were tested:

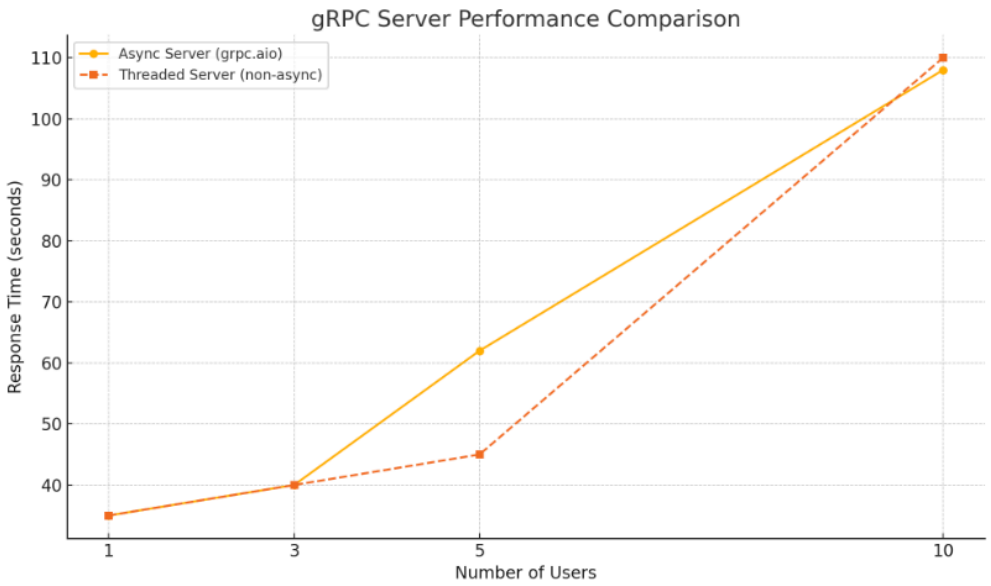
1. Approach 1: grpc.aio-based asynchronous server using async def and await.

2. Approach 2: Traditional concurrent server using grpc.server() with ThreadPoolExecutor (non-async).

Manual load tests were conducted with 1, 3, 5, and 10 simulated users.

Load Test Results (in seconds)

Users	Async Server (grpc.aio)	Threaded Server (non-async)
1	35s	35s (mirrored)
3	40s	40s (mirrored)
5	62s	45s
10	108s	110s



Observations

• Low-concurrency (1–3 users): Both methods performed nearly identically.
• Moderate concurrency (5 users): The non-async version slightly outperformed the async version (62s vs 45s).
• High concurrency (10 users): Both versions scaled similarly, with async being marginally faster (108s vs 110s).
• Async server showed more predictable scaling across different user loads, which may benefit scenarios involving I/O-heavy operations.

Conclusion

Both approaches handle low to moderate load effectively. However:

• The async implementation using grpc.aio offers better scalability and flexibility, especially when working with I/O-bound workloads or in containerized/cloud-native environments.
• For CPU-light or low-traffic apps, the non-async concurrent approach performs comparably and is easier to implement.

Recommendation

Use the async server (grpc.aio) for production to benefit from:
• Better handling of concurrent requests under load
• Improved maintainability for modern Python backends
• Compatibility with async-compatible libraries and I/O tasks