CSCI 5673: Performance Report - Programming Assignment 2

Done by - Sahil Shroff Tanish Praveen Nagrani

Evaluation Configuration

We evaluated the system using the tools/bench_part_2.py benchmarking script, which simulates concurrent buyer and seller sessions.

In Programming Assignment 2, the architecture was redesigned to incorporate modern distributed system communication protocols:

- Client ↔ Frontend: REST (FastAPI over HTTP)
- Frontend ↔ Backend Databases: gRPC (Protocol Buffers over HTTP/2)
- Financial Transactions: SOAP (WSDL-based service using Spyne)
- Frontend components remain stateless
- Each component (Seller Server, Buyer Server, Customer DB, Product DB) deployed as separate processes on distinct VMs.

VM Hardware Configuration (VDI) -

- CPU: 4 CPUs
- Memory: 8 GB
- Disk: 64 GB (Thick Provision Eager Zeroed)
- Hypervisor: ESXi 8.0 U2
- Network: Virtual adapter connected
- OS: Linux (Cloud VM)

APIs Used for Simulation

- Buyer → AddItemToCart()
- Seller → DisplayItemsForSale()

Each scenario consisted of:

- 10 independent runs
- Each client performing 1000 API calls per run
- Average response time computed across all calls
- Throughput computed as total operations per run divided by total execution time

Scenario 1: 1 Seller, 1 Buyer

- Average Response Time: 209.47 ms
- Throughput: 9.47 ops/sec

Explanation

This scenario represents the baseline performance under minimal concurrency (2 total clients).

Compared to PA1

- PA1 throughput: 462.5 ops/sec
- PA2 throughput: 9.47 ops/sec

The significant increase in response time (from ~5 ms in PA1 to ~209 ms in PA2) is primarily due to:

- REST overhead (HTTP parsing and JSON serialization)
- gRPC inter-service communication
- SOAP financial transaction service
- Additional network hops between services
- Virtualized cloud deployment overhead

Unlike PA1, which used direct TCP socket communication with a local PostgreSQL instance, PA2 introduces multiple protocol layers and distributed components, significantly increasing per-request processing cost.

Scenario 2: 10 Sellers, 10 Buyers

- Average Response Time: 195.87 ms
- Throughput: 16.25 ops/sec

Explanation

With 20 concurrent clients, the system achieves its highest observed throughput. Compared to Scenario 1, throughput increases because the servers and downstream services (gRPC DB calls, network stack, VM CPU) are better utilized: while one request is waiting on network/IO (HTTP parsing, gRPC round trip, DB query), other threads can make progress. This increases overall completed operations per second.

At the same time, the response time remains around ~200 ms because the system is near efficient saturation but not overloaded. There is limited queueing: requests are largely handled promptly, and concurrency primarily hides latency rather than creating long wait lines.

Scenario 3: 100 Sellers, 100 Buyers

- Average Response Time: 659.35 ms
- Throughput: 13.43 ops/sec

Explanation

At 200 concurrent clients, the system becomes over-subscribed. The key observation is:

- Throughput drops from 16.25 → 12.43 ops/sec
- Latency increases from ~196 ms → ~660 ms

This indicates the system has passed its optimal concurrency point and is now limited by contention and queueing overhead rather than compute/IO capacity.

Comparison between PA1 and PA2

Metric	PA1 (Sockets, Localhost)	PA2 (REST + gRPC + SOAP, Distributed VMs)
Scenario 1 (1 Seller, 1 Buyer) Avg Response Time	Buyer 5.99 ms, Seller 3.75 ms	209.47 ms
Scenario 1 (1 Seller, 1 Buyer) Throughput	462.5 ops/sec	9.47 ops/sec
Scenario 2 (10 Sellers, 10 Buyers) Avg Response Time	Buyer 5.4 ms, Seller 5.0 ms	195.87 ms
Scenario 2 (10 Sellers, 10 Buyers) Throughput	207.5 ops/sec	16.25 ops/sec

Scenario 3 (100 Sellers, 100 Buyers) Avg Response Time	Buyer 6.1 ms, Seller 4.1 ms	859.35 ms
Scenario 3 (100 Sellers, 100 Buyers) Throughput	427.7 ops/sec	12.43 ops/sec
Protocol Layers	TCP sockets only	HTTP (REST) + HTTP/2 (gRPC) + XML (SOAP/WSDL)
Deployment	Single machine (localhost)	Multiple VMs (distributed components)
Architecture	Monolithic-ish processes over TCP	Multi-tier / microservice-style (frontend + DB services + financial SOAP service)

Key Observations

1. Latency Increase
    - PA2 introduces multiple serialization layers (JSON, Protobuf, XML/SOAP).
    - Each API call involves several inter-service network hops.
    - VM virtualization introduces additional network latency.
2. Throughput Decrease
    - PA1 benefited from minimal protocol overhead.
    - PA2's layered architecture increases per-request CPU cost.
    - Python GIL restricts effective parallelism under high concurrency.