Voting System

Distributed Systems - Project 2

Adam Emerson - 1000773509
John Song - 1002306479

Functional Requirements

Creating A Poll

A user can create a new poll with a question and a set of options.

Voting On A Poll

A user can vote for an option on a specific poll.

Viewing All Polls

A user can see a list of all available polls.

Viewing Poll Results

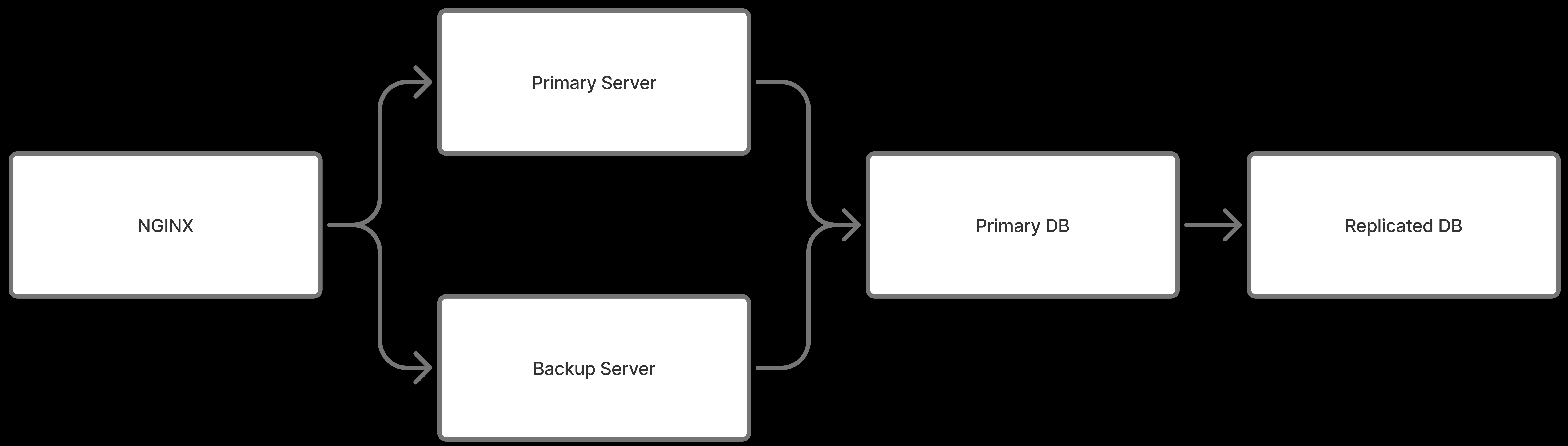
A user can view the current vote counts for a specific poll.

Closing A Poll

Users can close a poll can close it to prevent further voting.

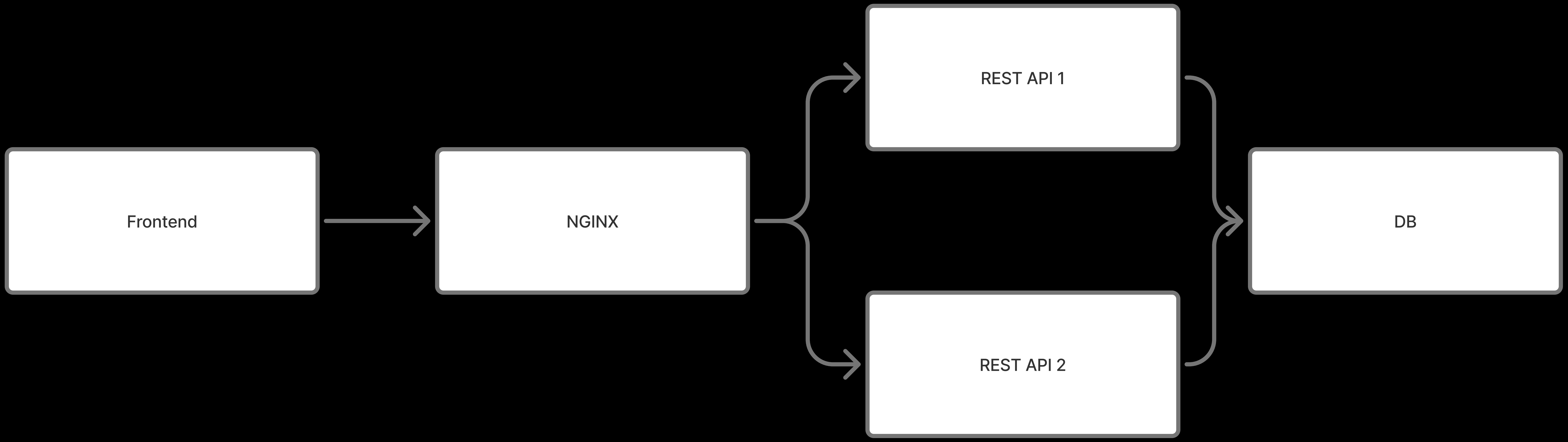
Microservice (GRPC)

A fault-tolerant and scalable polling application designed using a microservice architecture. It distributes the workload across five distinct nodes, ensuring high availability and performance under load.

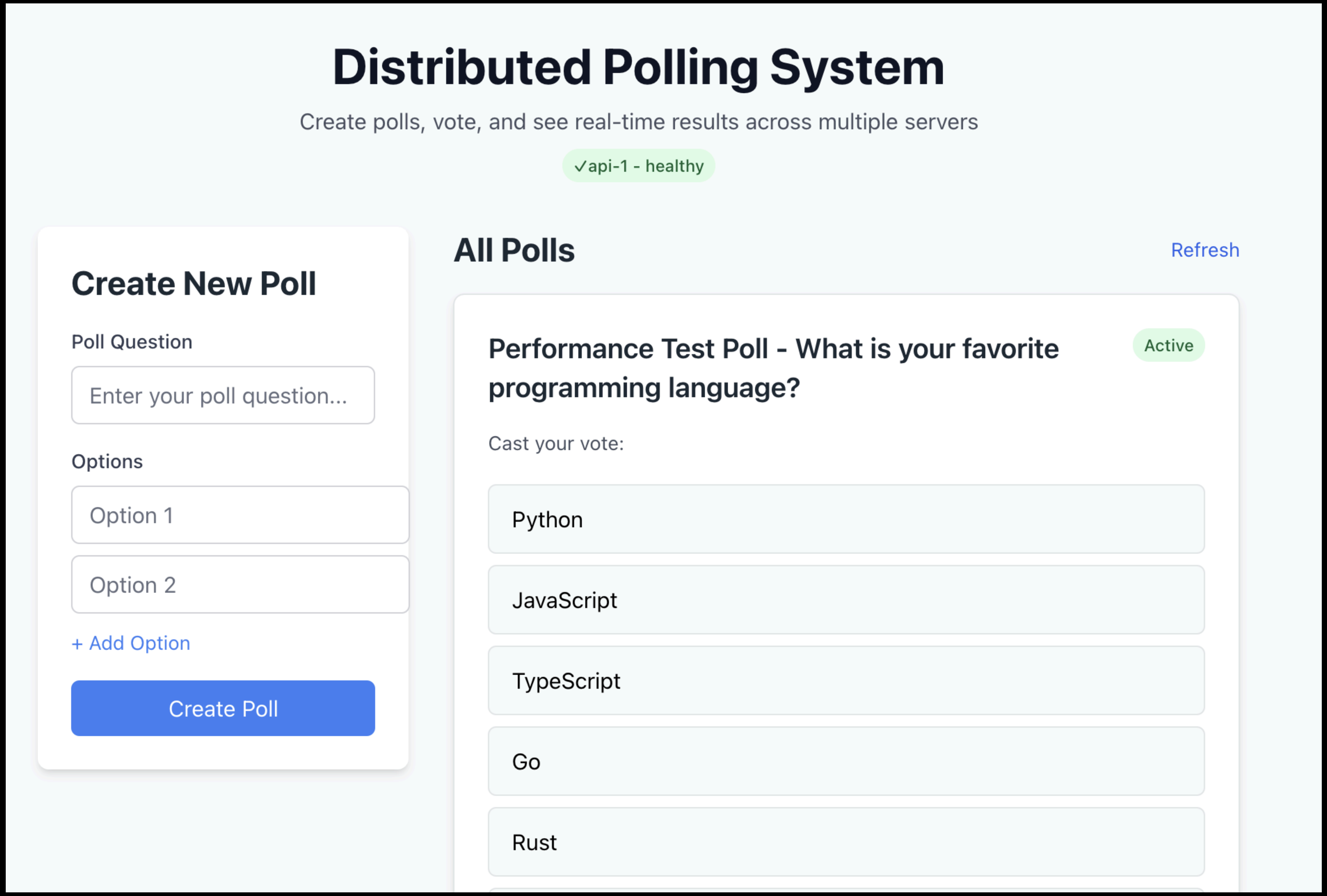


REST (HTTPS)

The REST-based architecture implements a distributed polling system using HTTP communication and a resource-oriented design. The system consists of 5 containerized nodes:



REST Frontend



Distributed Polling System

Create polls, vote, and see real-time results across multiple servers

✓api-1 - healthy

Create New Poll

Poll Question

Enter your poll question...

Options

Option 1

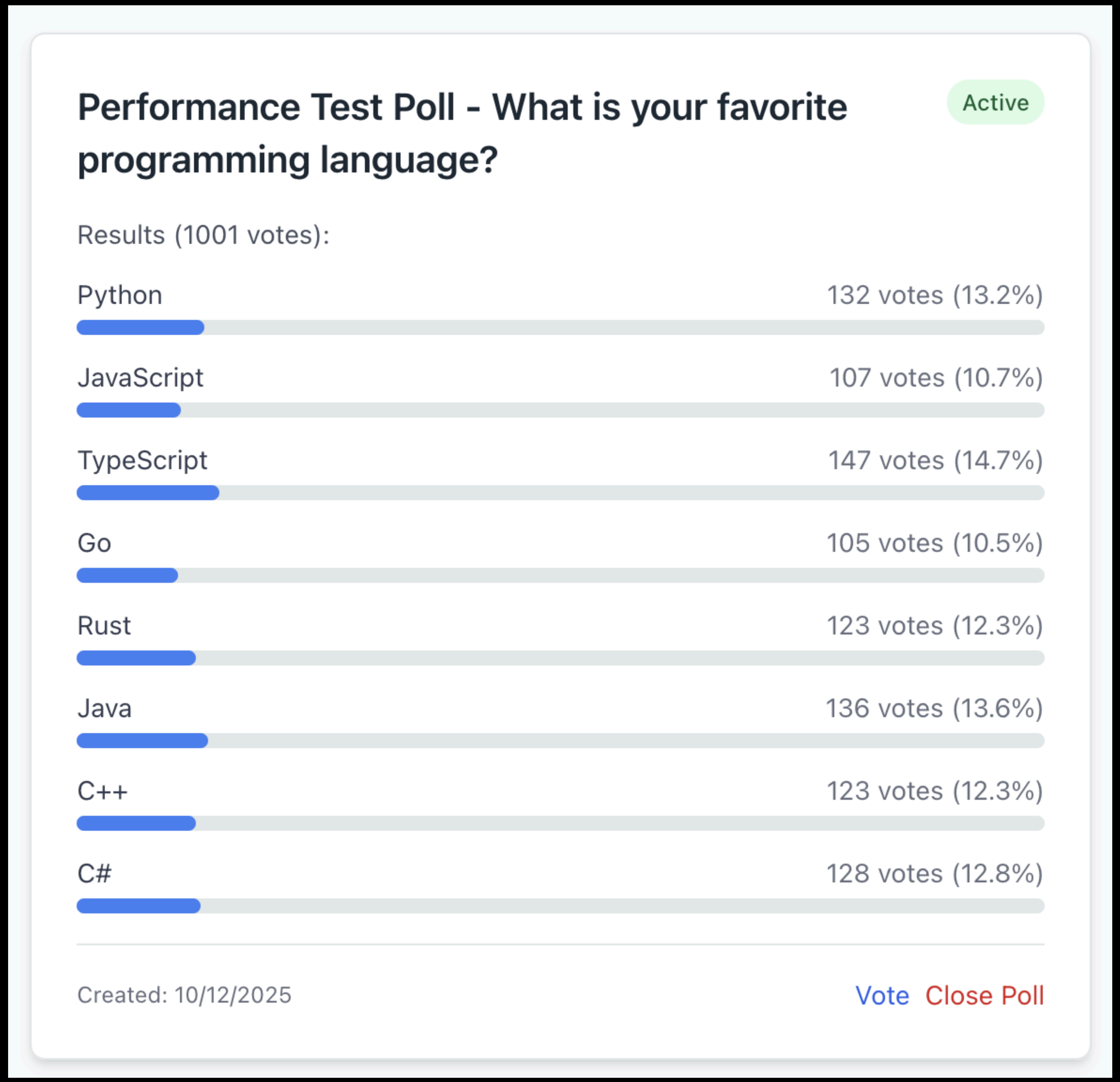
Option 2

All Polls

Performance Test Poll - What is your favorite programming language? Active

Cast your vote:

- Python
- JavaScript
- TypeScript
- Go
- Rust



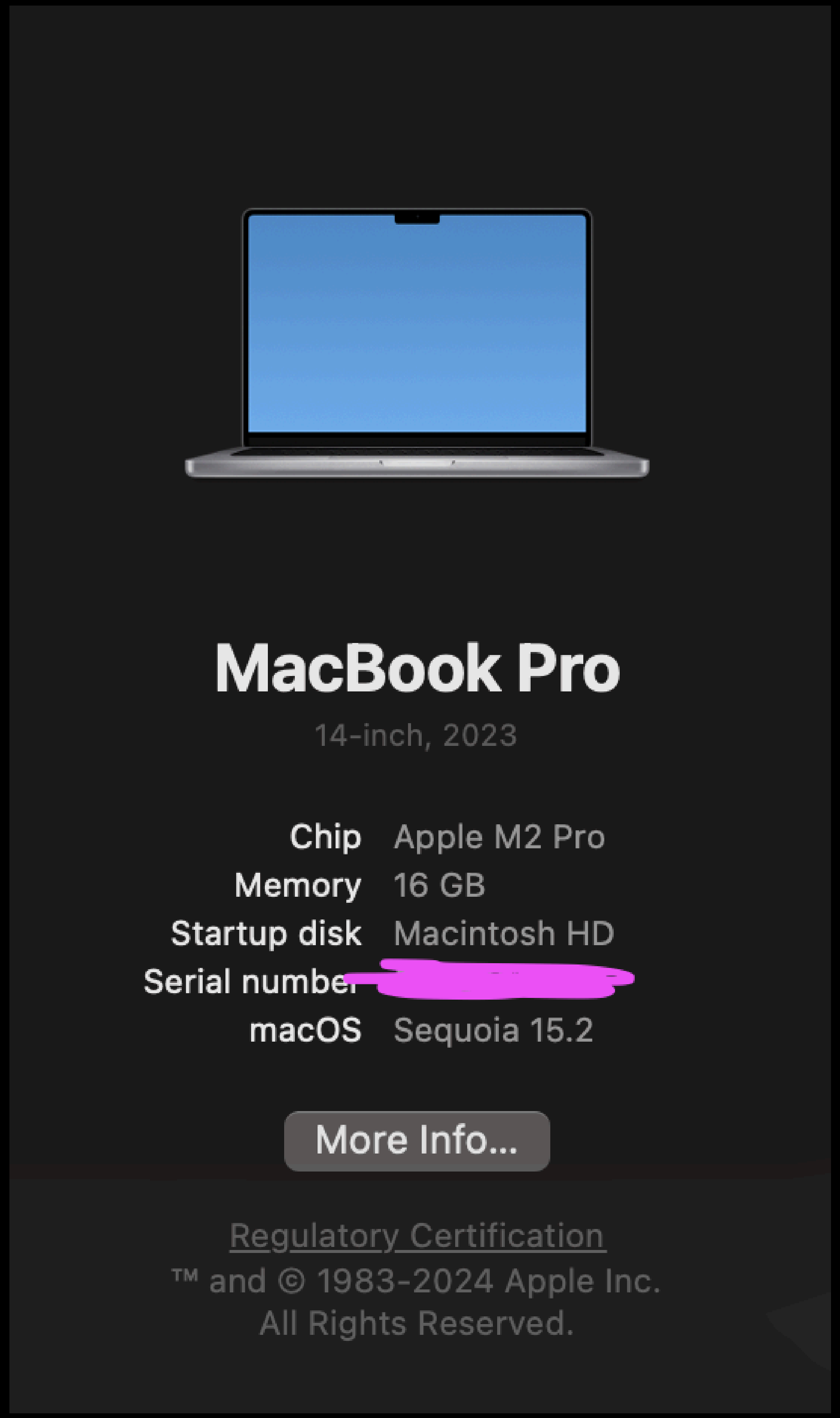
Performance Test Poll - What is your favorite programming language? Active

Results (1001 votes):

Python	132 votes (13.2%)
JavaScript	107 votes (10.7%)
TypeScript	147 votes (14.7%)
Go	105 votes (10.5%)
Rust	123 votes (12.3%)
Java	136 votes (13.6%)
C++	123 votes (12.3%)
C#	128 votes (12.8%)

Created: 10/12/2025

Evaluation



MacBook Pro

14-inch, 2023

Chip	Apple M2 Pro
Memory	16 GB
Startup disk	Macintosh HD
Serial number	[redacted]
macOS	Sequoia 15.2

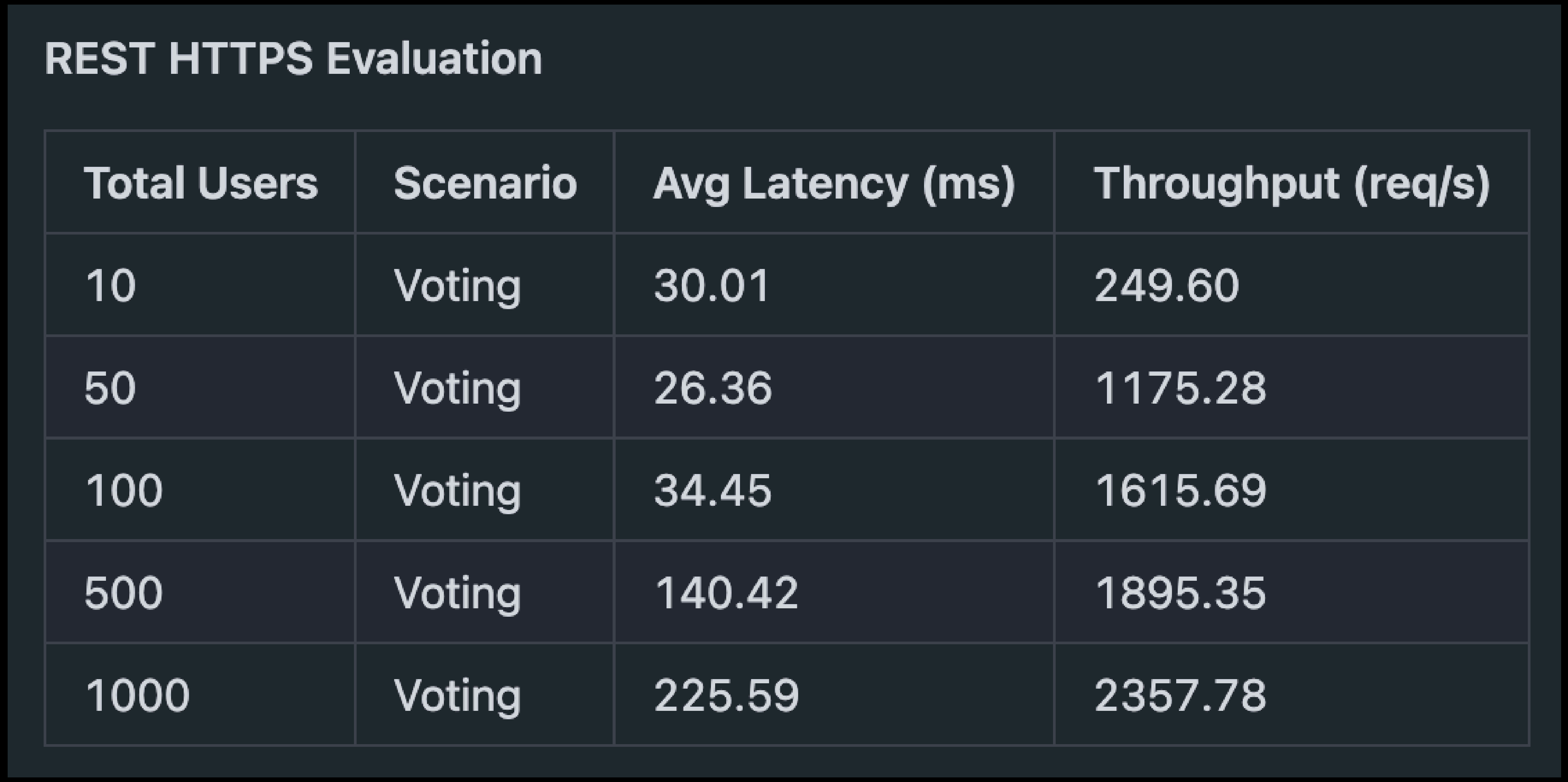
Regulatory Certification

™ and © 1983-2024 Apple Inc.
All Rights Reserved.

In order to evaluate our two architectures we simulated high-loads for both “read” and “write” activities.

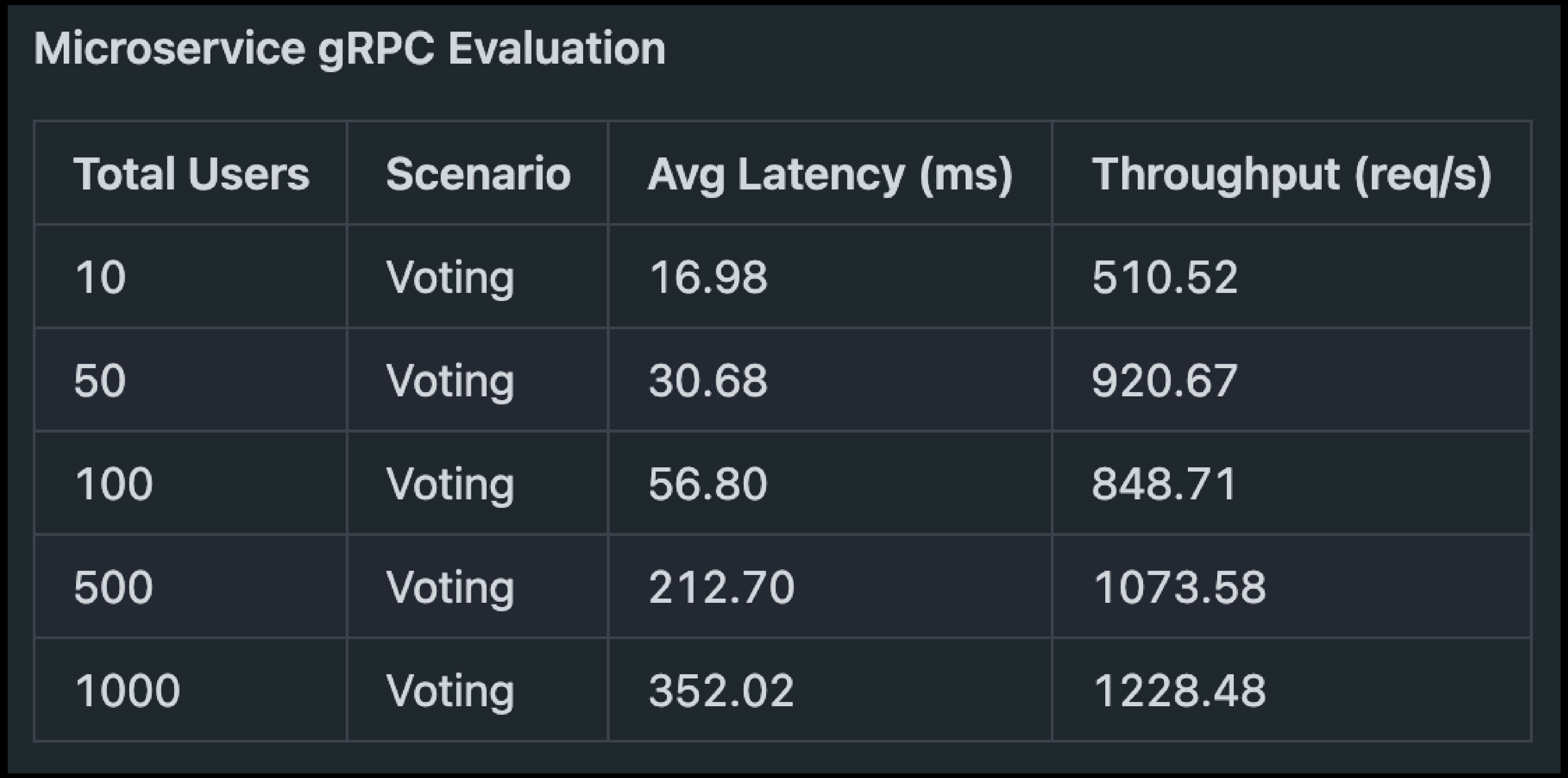
All tests were conducted on a **Macbook Pro with the M2 Pro chip and 16gb of ram.**

Voting (Write Simulation)



REST HTTPS Evaluation

Total Users	Scenario	Avg Latency (ms)	Throughput (req/s)
10	Voting	30.01	249.60
50	Voting	26.36	1175.28
100	Voting	34.45	1615.69
500	Voting	140.42	1895.35
1000	Voting	225.59	2357.78



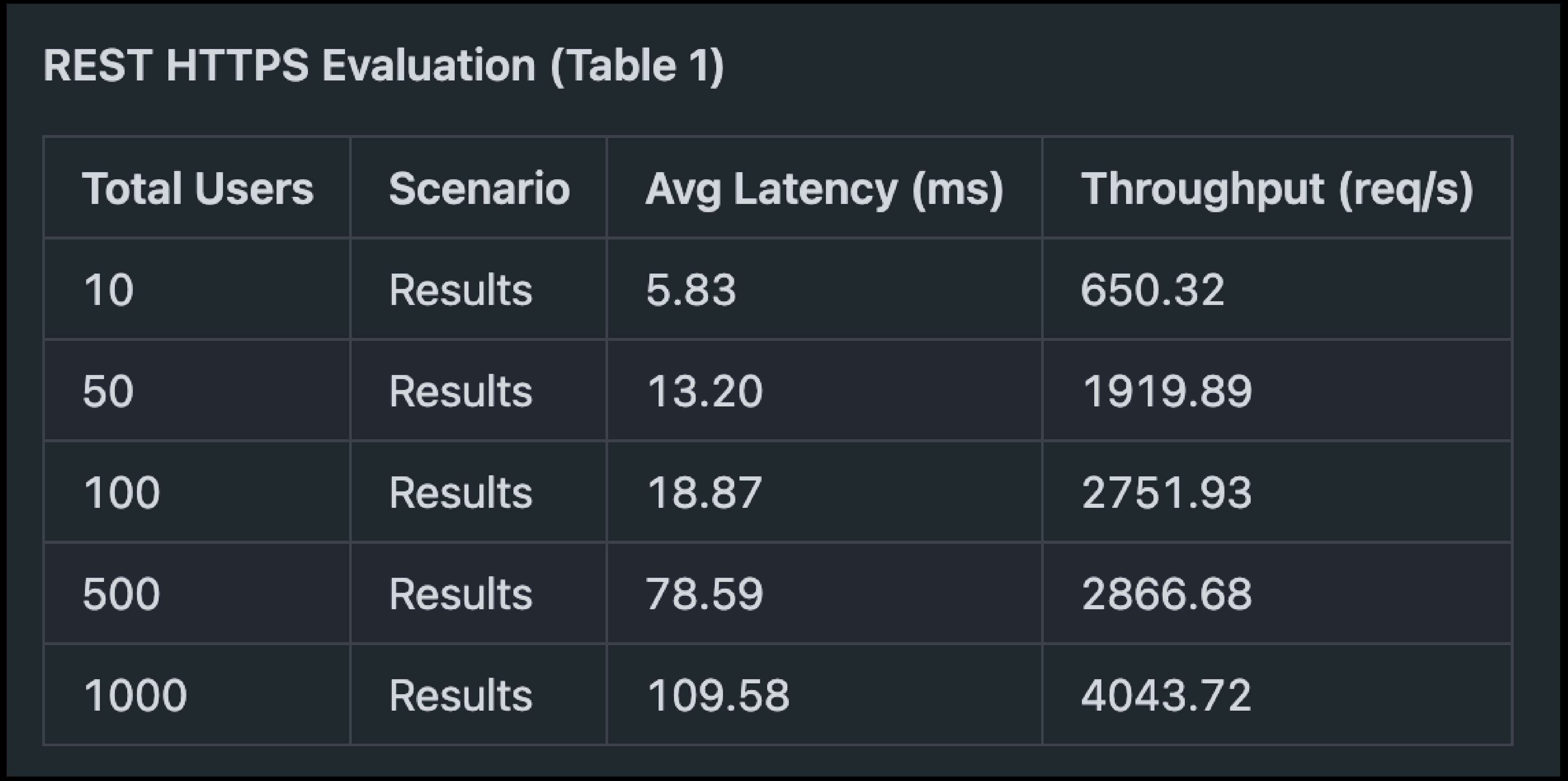
Microservice gRPC Evaluation

Total Users	Scenario	Avg Latency (ms)	Throughput (req/s)
10	Voting	16.98	510.52
50	Voting	30.68	920.67
100	Voting	56.80	848.71
500	Voting	212.70	1073.58
1000	Voting	352.02	1228.48

For write we **multiple users casting votes in a poll** at the same time.

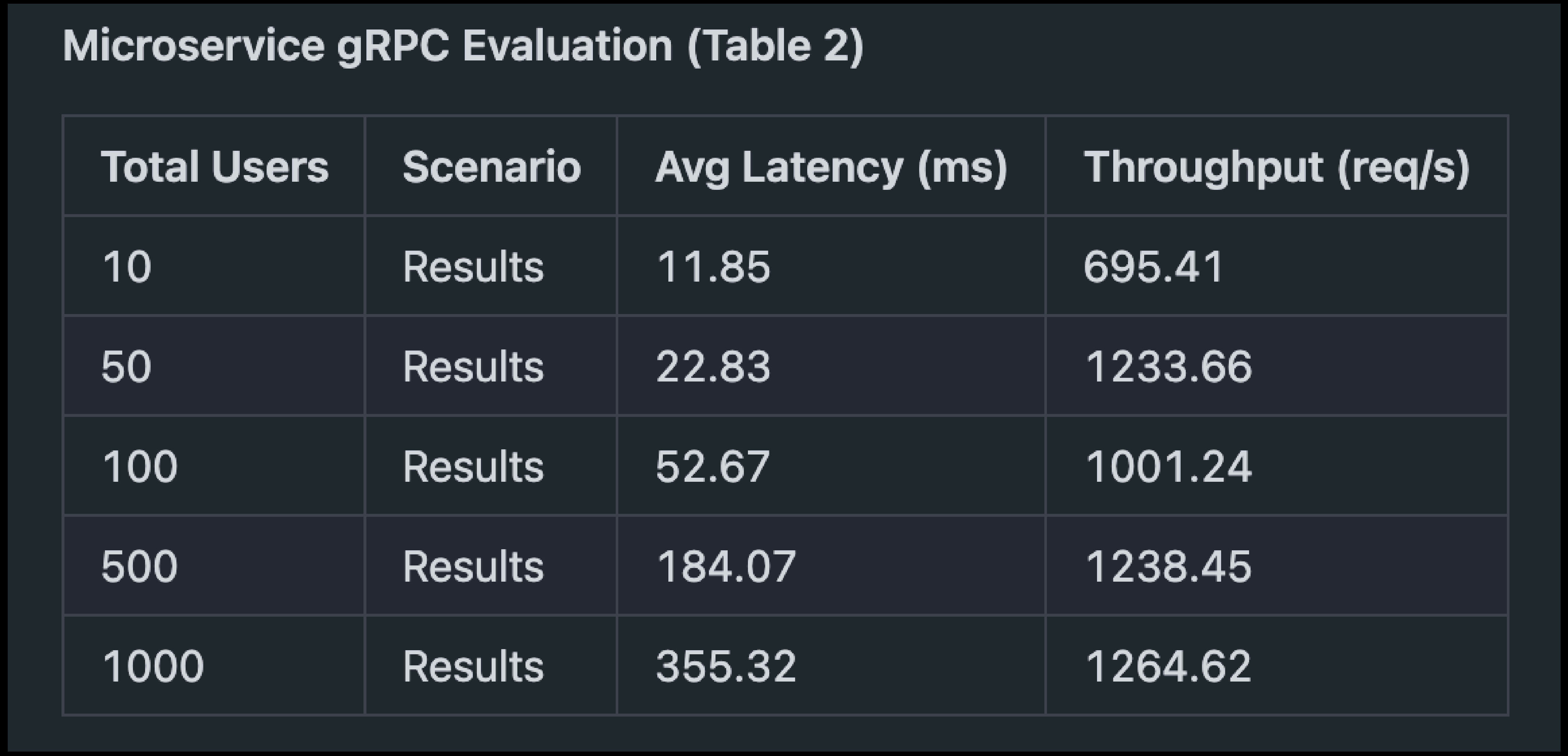
Concurrent requests were simulated using a python script across both systems at increments of 10, 50, 100, 500, and 1000 users.

Poll Results (Read Simulation)



REST HTTPS Evaluation (Table 1)

Total Users	Scenario	Avg Latency (ms)	Throughput (req/s)
10	Results	5.83	650.32
50	Results	13.20	1919.89
100	Results	18.87	2751.93
500	Results	78.59	2866.68
1000	Results	109.58	4043.72



Microservice gRPC Evaluation (Table 2)

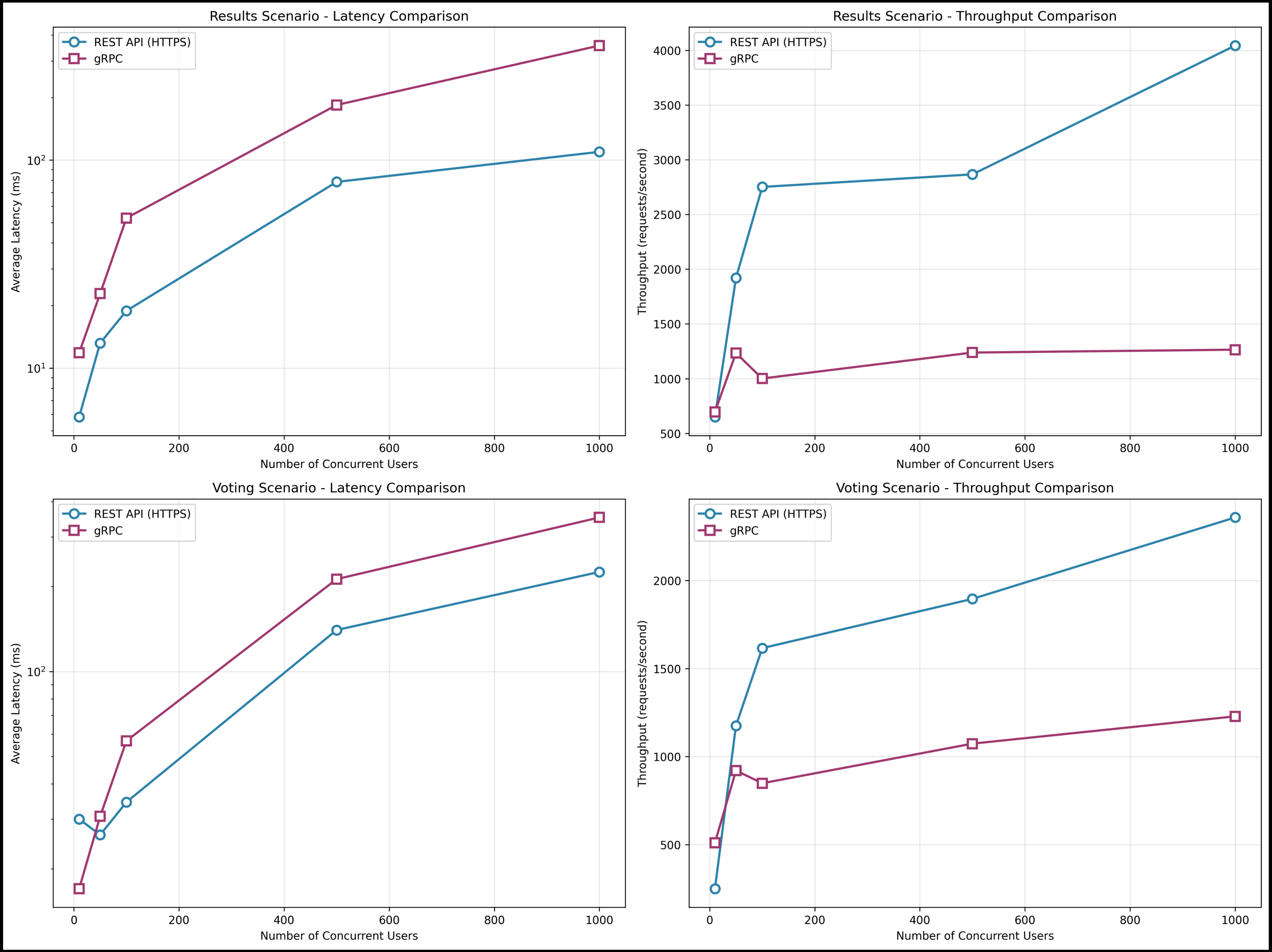
Total Users	Scenario	Avg Latency (ms)	Throughput (req/s)
10	Results	11.85	695.41
50	Results	22.83	1233.66
100	Results	52.67	1001.24
500	Results	184.07	1238.45
1000	Results	355.32	1264.62

For read we simulated **multiple users accessing poll results** at the same time.

Concurrent requests were simulated using a python script across both systems at increments of 10, 50, 100, 500, and 1000 users.

Comparison

Stress tests on the read and write performance of both architectures show a consistent advantage for the HTTPS-based REST system, which outperforms the gRPC implementation in nearly all metrics except for latency at very low user counts (Fig. 1).



Analysis

Unexpected Results

This outcome is counterintuitive, as gRPC typically offers superior efficiency due to its binary protocol and multiplexed streams.

Max Workers Limit

Our gRPC stack was configured with a max_worker limit of 10, creating a thread bottleneck

Database Version Mismatch

We used two different version of postgres across our architectures, which may lead to some performance discrepancies.

Database Replication

The gRPC stack makes use of database replication, which while providing additional security, comes with increased overhead.

Questions?