Performance Testing Project – Flask + Locust

1. Project Goals

The project is conducted to simulate and analyze system performance based on Flask, Locust, and monitoring script.

The goals of the project are:

- Compare CPU-bound and I/O-bound workload behavior
- Measure response time, throughput, and error rate under increasing load
- Observe system resource utilization (CPU, memory)
- Identify bottlenecks and evaluate potential improvements

2. System and Project Design

1. Flask Application

The application consists of three endpoints:

- /health : simple health check
- /io : simulate blocking operation using time.sleep()
- /cpu : performs heavy computational loop

I/O stands for Input/Output and refers to the communication between an information processing system (such as a computer) and the outside world, including human users, peripheral devices, or other computers

I/O bound is a term used to describe a program where the speed of execution is limited by the time taken to perform input/output operations rather than the processing speed of the CPU.

2. Locust User Flow

Each virtual user performs:

- 50% /health
- 30% /io

- 20% /cpu

Wait time between requests are 0.1 - 0.5 seconds

3. Test Scenarios

Each test was excused for 3 minutes under increasing concurrent load.

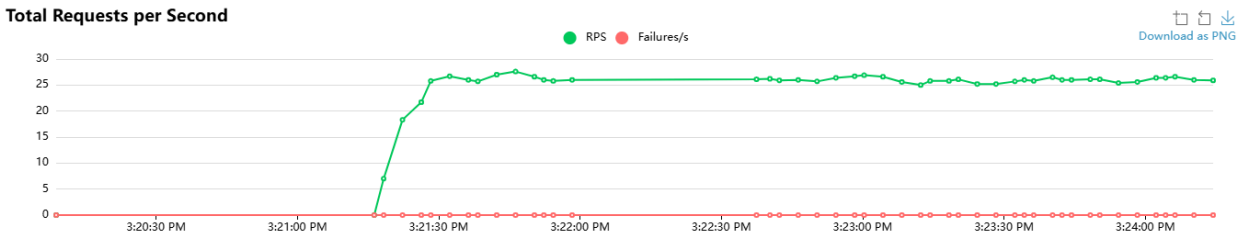
Scenario	Concurrent Users	Ramp up	Duration
Test 1	10 users	2	3 min
Test 2	50 users	5	3 min
Test 3	100 users	10	3 min

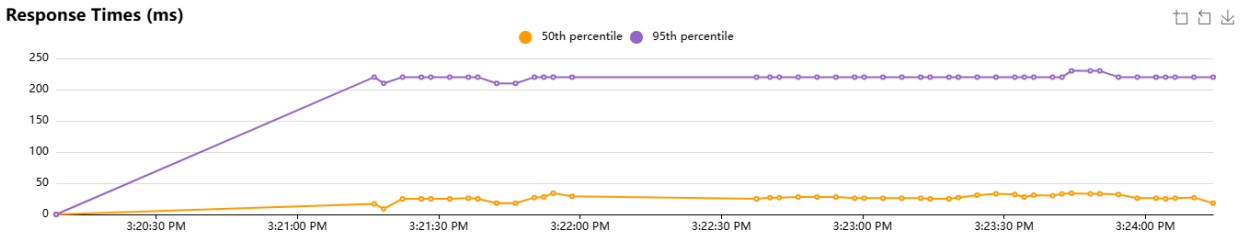
4. Test Results

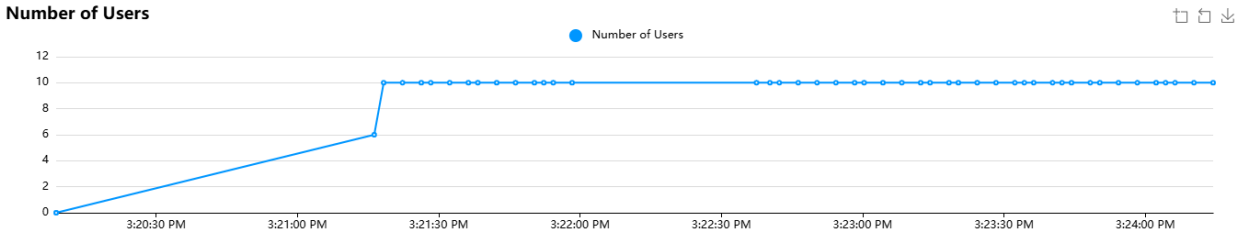
1. 10 Users

Type	Name	# Requests	# Fails	Median (ms)	95%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	/cpu	926	0	34	62	80	36.85	24	137	33	5.2	0
GET	/health	2349	0	5	30	43	8.35	1	78	21	13.1	0
GET	/io	1398	0	210	230	260	211.13	202	272	24	7.6	0
	Aggregated	4673	0	26	220	240	74.66	1	272	24.28	25.9	0

- Avg Response Time: 74.66 ms
- P95 Response Time: 220 ms
- Requests per Second (RPS): 25.9
- Failure: 0
- CPU Usage: 54.4%
- Memory Usage: 8.3 %







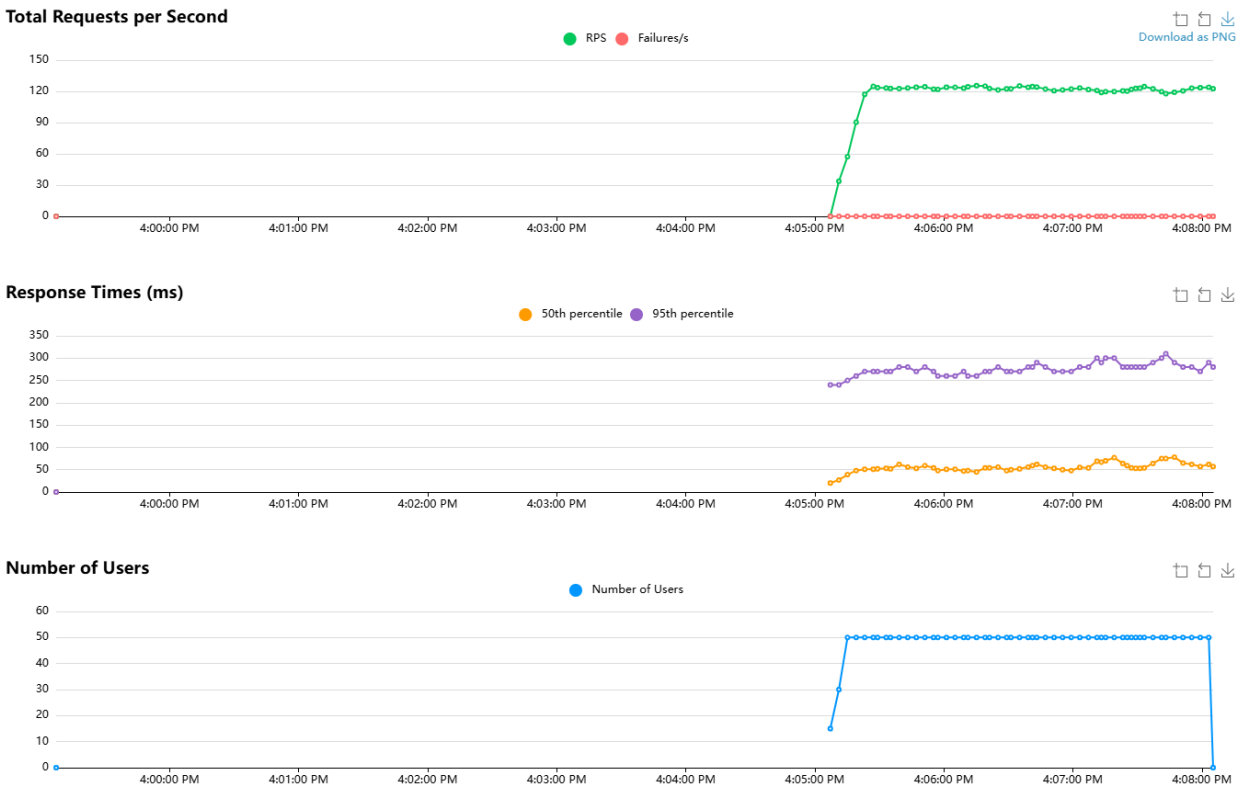
Under mixed workload (10 users, ramp-up 2/sec, 180s duration), the aggregated average latency was 74.66 ms with no failures. However, endpoint-level analysis revealed that the `/io` endpoint exhibited significantly higher latency (Avg 211 ms, P95 230 ms), indicating I/O-bound behavior dominating tail latency

2. 50 Users

Type	Name	# Requests	# Fails	Median (ms)	95%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	/cpu	4323	0	53	130	170	60.18	23	245	33	25.3	0
GET	/health	10696	0	27	99	140	33.62	1	232	21	62.3	0
GET	/io	6519	0	240	310	350	244.34	202	441	24	35	0
	Aggregated	21538	0	56	280	320	102.73	1	441	24.32	122.6	0

- Avg Response Time: 102.73 ms
- P95 Response Time: 280 ms
- RPS: 122.6
- Failure: 0

- CPU Usage: 54.4 %
- Memory Usage: 13 %

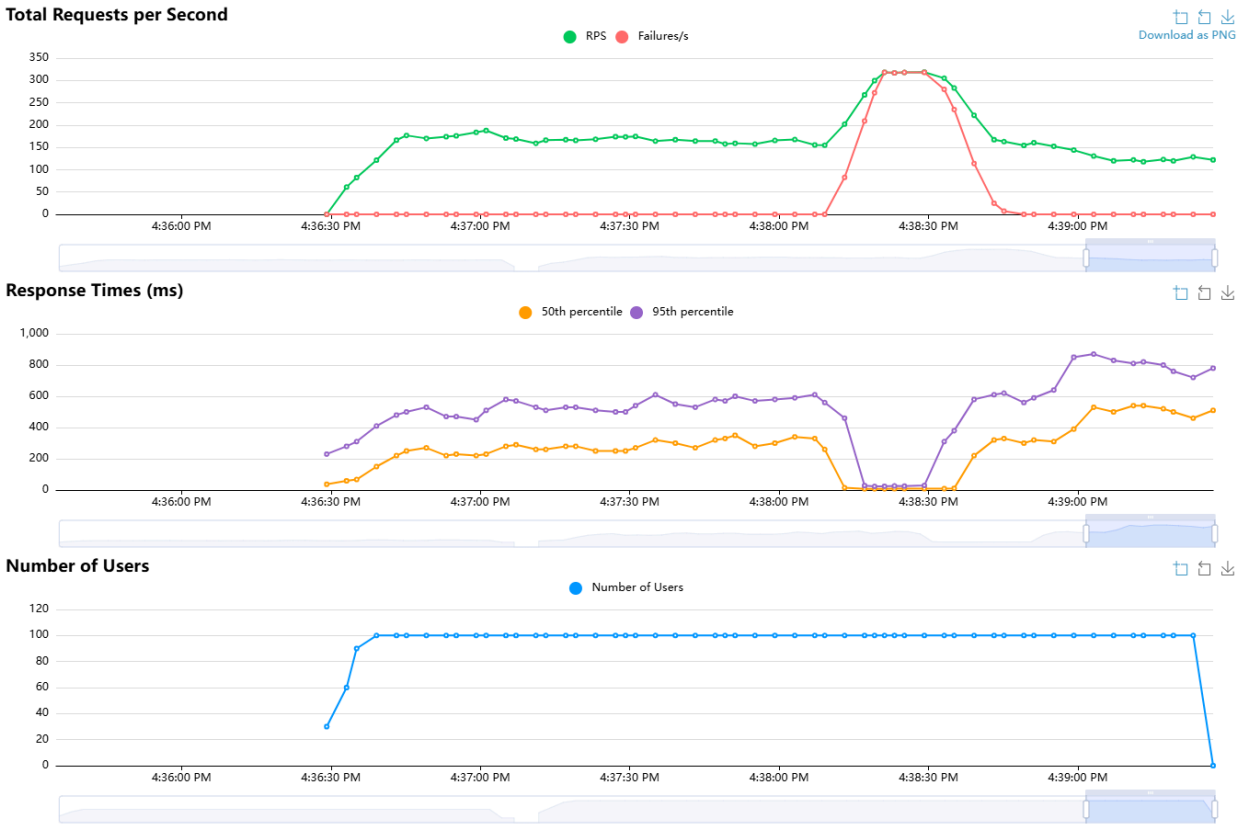


At 50 concurrent users, the system demonstrated near-linear throughput scaling (122 RPS vs 26 RPS at 10 users). While average latency increased moderately (102 ms), no failures were observed. The /io endpoint continued to dominate tail latency, suggesting I/O-bound behavior as the primary contributor to P95 degradation under load.

3. 100 Users

Type	Name	# Requests	# Fails	Median (ms)	95%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	/cpu	6300	1435	220	540	670	225.13	1	929	25.48	26.9	0
GET	/health	15568	3492	190	500	620	199.81	1	937	16.29	60.2	0
GET	/io	9422	2119	410	720	850	370.25	1	1114	18.6	34.9	0
	Aggregated	31290	7046	240	610	770	256.23	1	1114	18.84	122	0

- Average Response Time: 256.23 ms
- P95 Response Time: 610 ms
- Requests per Second (RPS): 122
- Failures: 7046
- CPU: 54.4%
- Memory: 18%



As shown in the graph, the test showed failures between 4:38:15 and 4:38:45 (timeline). The Total Requests per Second reached over 300, but the failures (red line) also spiked exactly wrong with it.

Regarding Throughput, the PRS dropped to about 120, lower than the 170 it had before the crash. The system is likely able to handle 100 users at roughly 170 PRS for a short burst, but it is highly unstable and hit a bottleneck that caused an error.

Once the failure stops, the system does not return to its previous health. Also, the response time for 95th went high up to 800ms and more.

5. Result Analysis

1. Observation Analysis

In the test with 100 users, there was a significant changes from the previous test with 50 users.

First, Requests per Second (RPS) did not increase.

- 50 users → 122 RPS
- 100 users → 122 RPS

The result incidates that the system can only process ~122 requests/sec. Adding more users does not increase capacity. It only increases waiting time.

As a result,

- Throughput stayed the same. Throughput represents the number of tasks a system can process within a specific timeframe.
- Latency exploded
- Failures appeared
- Max response time jumped over 1 second

The system reached its maximum processing capacity around 50 users.

2. Failure Analysis

The image shows the detailed error message from Locust testing page.

# Failures	Method	Name	Message
1435	GET	/cpu	OSError(10048, '[WinError 10048] Only one usage of each socket address (protocol/network address/port) is normally permitted.')
3492	GET	/health	OSError(10048, '[WinError 10048] Only one usage of each socket address (protocol/network address/port) is normally permitted.')
2119	GET	/io	OSError(10048, '[WinError 10048] Only one usage of each socket address (protocol/network address/port) is normally permitted.')

[Error Message]

“ OSError(10048, '[WinError 10048] Only one usage of each socket address (protocol/network address/port) is normally permitted.') “

The error was mainly caused by Windows networking limits, not by application failures, CPU saturation, or Flask crash. It indicates that Windows ran out of available ports.

At 100 concurrent users, failures occurred due to Windows socket exhaustion (WinError 10048), indicating the system hit the OS networking limit at 100 users. So, Locust could not open new sockets and returned OSError 10048, showing up as test failures.

That also explains why RPS stayed at 122. Locust could not create more connections and the system limit prevented higher throughput. That is, 100 Users testing was limited by windows networking stack.