Performance Test Result for Autobahn and Socket.IO

Buse Nur Sabah

March 4, 2024

Contents

1 Introduction

1.1 Autobahn

Autobahn is an open-source WebSocket framework for building real-time web applications and services. It provides both client and server implementations for WebSocket communication in Python. I selected Autobahn because it is widely used in production environments and is known for its performance, reliability, and feature-richness. It's a popular choice for developers who need to implement real-time communication in their web applications.

Autobahn is used with Twisted networking engine or asyncio. Asyncio is a built-in asynchronous I/O framework introduced in Python 3.4 and it provides an event loop, coroutine objects. Twisted is a separate asynchronous networking engine and framework that has been available for Python for much longer. I used asyncio rather than Twisted because asyncio is a suitable choice for modern Python applications. Twisted, while powerful, may be seen as less compatible with newer Python paradigms.

Some key features of Autobahn framework that lead me select it:

Asyncio Integration: Autobahn supports asyncio, which is a modern Python asynchronous I/O framework. Asyncio provides a simpler and more Pythonic way to write asynchronous code compared to Twisted, which has a steeper learning curve and can sometimes be more complex to work with.

Performance: Autobahn aims to provide high performance for WebSocket communication, which is crucial for real-time applications where low latency is essential.

Robustness: Autobahn has been extensively tested and is widely used in production environments, making it a reliable choice for building real-time applications that require WebSocket support.

Feature-Rich: Autobahn offers a wide range of features beyond basic WebSocket support, including support for WebSocket subprotocols, message compression, and advanced error handling mechanisms.

1.2 Socket.IO

Socket.IO is a Python library that enables real-time, bidirectional communication between web clients and servers. I selected Socket.IO because it supports custom events, which bring flexibility and enable the creation of diverse real-time applications tailored to specific use cases.

Socket.IO needs to be integrated with asynchronous web servers to achieve optimal performance and scalability, especially in high-traffic or real-time applications where responsiveness is critical. Asynchronous servers are better suited for handling multiple concurrent connections. I chose Uvicorn as the asynchronous web server because it is designed for high performance and can efficiently handle high traffic loads. Its asynchronous architecture minimizes overhead and maximizes throughput, resulting in lower latency and faster response times.

Some key features of Autobahn framework that lead me select it:

> Automatic reconnection: Socket.IO automatically handles reconnection attempts in case of network disruptions or server failures, ensuring a reliable connection.
>
> Support for multiple transport mechanisms: Socket.IO can use various transport mechanisms, including WebSocket, AJAX long polling, and others, to ensure compatibility with different client environments and network configurations.
>
> Room-based communication: Socket.IO provides the concept of "rooms," allowing clients to join specific rooms to receive targeted messages, which is useful for applications with multiple channels or chat rooms.
>
> Real-time analytics: Socket.IO enables real-time monitoring and analytics by allowing servers to push updates to clients as soon as they occur, without the need for clients to constantly poll for updates.
>
> Middleware support: Socket.IO provides middleware support, allowing you to easily extend and customize the behavior of your Socket.IO server with middleware functions.

2 Tests

The tests were coded based on their purpose with slight variations between them. The tests were conducted by connecting one, two, and three clients to a single server simultaneously. Four different data sizes were used as payloads. In the tables below, the left column represents the number of concurrently connected clients, and the first row represents the size of the sent data. The data size indicates how many stocks were sent, where each stock consists of "open, high, low, close prices, and volume" values. Data was read from files and sent, with file opening and loading operations performed before each transmission to simulate the overhead of data change. A single CPU was used for the server, and another CPU were used for the clients. All operations were performed in the main thread. The operating system used was "Linux 5.15.0-97-generic 107 20.04.1-Ubuntu."

Three different implementations were tested for Autobahn. In the first one, packets were sent to clients without binary serialization and without using asynchronous structures when sending data to clients. In the second one, data packets were transmitted to clients without binary serialization, but asynchronous mechanisms were employed during the data transmission process to clients. In the last one, data packets were delivered to clients utilizing binary serialization, but without applying asynchronous structures during the data transmission to clients. The most efficient result was achieved when binary serialization was applied. For Socket.IO, packets were sent to clients using binary serialization as well.

2.1 Throughput

The pink column represents the number of packets sent per second from the server, while the lavender column indicates the number of packets received per second by the per client. The server and clients were run for 2 minutes, and the average values at the end of this period were obtained.

	1		2		10		100	
1	2551	2551	2434	2434	2220	2220	1024	331
2	2344	2344	2289	2289	2125	1444	898	162
3	2158	2158	2122	2122	2035	963	830	106

Table 1: autobahn throughput with no binary serialization, sync

	1		2		10		100	
1	2445	2445	2383	2382	2028	2028	940	323
2	2113	2113	2030	2023	1824	1418	835	160
3	1869	1869	1800	1800	1676	942	762	105

Table 2: autobahn throughput with no binary serialization, async

	1		2		10		100	
1	2612	2612	2598	2598	2479	2479	1915	1915
2	2460	2460	2383	2383	2318	2318	1798	1798
3	2269	2269	2264	2264	2176	2176	1713	1713

Table 3: autobahn throughput with binary serialization

	1		2		10		100	
1	1613	1613	1608	1608	1560	1560	1296	1296
2	1282	1282	1275	1275	1242	1242	X	X
3	X	X	X	X	X	X	X	X

Table 4: Socket.IO throughput with binary serialization

In Socket.IO, disconnection problems occurred consistently when three clients were connected, and when two clients were connected during high-traffic periods. These issues were accompanied by the appearance of 'TCP Zero Window' and 'TCP Window Full'. Despite the absence of packet loss in healthy conditions, the throughput observed was not as high as that achieved with Autobahn.

2.2 Latency

Timestamps were appended to the packets before transmission from the server. The server and client were run for 2 minutes. The differences in timestamps upon packet arrival at the client were accumulated, and at the end of the duration, the average latency per packet was computed in seconds.

	1	2	10	100
1	0.0002074	0.0002444	0.0005237	45.2
2	0.0002525	0.0003001	21.51	50.75
3	0.0002795	0.0004706	32.5	52.31

Table 5: autobahn latency with no binary serialization, sync

	1	2	10	100
1	0.0002196	0.0002802	0.0005415	42.94
2	0.0003199	0.0003896	20.23	53.09
3	0.0004113	0.0004653	32.73	58.71

Table 6: autobahn latency with no binary serialization, async

	1	2	10	100
1	0.0001093	0.0001136	0.000128	0.0003008
2	0.0001591	0.0001626	0.0001957	0.0004954
3	0.000212	0.0002336	0.0003032	0.0005769

Table 7: autobahn latency with binary serialization

	1	2	10	100
1	0.001145	0.001162	0.001194	0.001451
2	0.001675	0.001711	0.003126	X
3	X	X	X	X

Table 8: Socket.IO latency with binary serialization

In cases where Socket.IO operates without experiencing packet loss, the average latency per packet is higher than that observed in the autobahn tests.

2.3 CPU Utilization

The server and clients were run for 3 minutes, and the instantaneous data was obtained at the end of each minute. The CPU usage on the server side is at 100% in all cases for both Autobahn and Socket.IO. However, the CPU usage per client varies. Having 50% for 2 clients and 33.3% for 3 clients indicates the upper limit as they share a single CPU. Thus, there is a CPU bottleneck in these scenarios.

	1	2	10	100
1	30	43.5	86.5	100
2	31.5	37	50	50
3	27.5	33.3	33.3	33.3

Table 9: autobahn cpu utilization with no binary serialization, sync

	1	2	10	100
1	31	39	80.5	100
2	28	33.3	50	50
3	24	28	33.3	33.3

Table 10: autobahn cpu utilization with no binary serialization, async

	1	2	10	100
1	21.5	22.5	24.5	42.5
2	25	25.5	26	35.5
3	22	22	23.5	29

Table 11: autobahn cpu utilization with binary serialization

	1	2	10	100
1	60	60	60	66
2	48.5	48.5	50	50
3	33.3	33.3	33.3	33.3

Table 12: Socket.IO cpu utilization with binary serialization

In Autobahn, when binary serialization is applied, there is no CPU bottleneck on the client side. However, in Socket.IO, CPU usage remains at 100% in most cases.

2.4 Memory Utilization

The pink column represents the utilization of memory in gigabytes by server, and the lavender column indicates the the utilization of memory in gigabytes by per client. The server and clients were run for 3 minutes, and the instantaneous data was obtained at the end of each minute.

	1		2		10		100	
1	0.26	0.26	0.26	0.26	0.26	0.26	5.1	0.26
	0.26	0.26	0.26	0.26	0.26	0.26	9.8	0.26
	0.26	0.26	0.26	0.26	0.26	0.26	14.67	0.26
2	0.26	0.26	0.26	0.26	1.12	0.26	10.52	0.26
	0.26	0.26	0.26	0.26	2.05	0.26	20.88	0.26
	0.26	0.26	0.26	0.26	3.1	0.26	30.96	0.26
3	0.26	0.26	0.26	0.26	2.24	0.26	15.06	0.26
	0.26	0.26	0.26	0.26	4.37	0.26	29.35	0.26
	0.26	0.26	0.26	0.26	6.46	0.26	43.8	0.26

Table 13: autobahn memory utilization with no binary serialization, sync

	1		2		10		100	
1	0.26	0.26	0.26	0.26	0.26	0.26	4.53	0.26
	0.26	0.26	0.26	0.26	0.26	0.26	8.85	0.26
	0.26	0.26	0.26	0.26	0.26	0.26	13.2	0.26
2	0.26	0.26	0.26	0.26	0.68	0.26	9.87	0.26
	0.26	0.26	0.26	0.26	1.44	0.26	19.4	0.26
	0.26	0.26	0.26	0.26	1.93	0.26	28.56	0.26
3	0.26	0.26	0.26	0.26	1.8	0.26	15.06	0.26
	0.26	0.26	0.26	0.26	3.3	0.26	28	0.26
	0.26	0.26	0.26	0.26	4.83	0.26	41.5	0.26

Table 14: autobahn memory utilization with no binary serialization, async

	1		2		10		100	
1	0.27	0.29	0.27	0.3	0.27	0.29	0.27	0.29
2	0.27	0.29	0.27	0.29	0.27	0.28	0.27	0.29
3	0.27	0.28	0.27	0.29	0.27	0.28	0.27	0.29

Table 15: autobahn memory utilization with binary serialization

	1		2		10		100	
1	0.35	0.33	0.35	0.33	0.35	0.33	0.35	0.33
	0.35	0.33	0.35	0.33	0.35	0.33	0.35	0.33
	0.35	0.33	0.35	0.33	0.35	0.33	0.35	0.33
2	0.35	0.33	0.35	0.33	0.35	0.33	0.77	0.35
	0.35	0.33	0.35	0.33	0.35	0.33	1.17	0.35
	0.35	0.33	0.35	0.33	0.35	0.33	1.46	0.35
3	0.35	0.33	0.35	0.33	0.35	0.33	1.67	0.33
	0.35	0.33	0.35	0.33	0.35	0.33	2.07	0.33
	0.35	0.33	0.35	0.33	0.45	0.33	2.9	0.33

Table 16: Socket.IO memory utilization with binary serialization

In Autobahn, when binary serialization is not applied, an increase in packet size results in a proportional rise in memory usage on the server side as time passes. This indicates that prolonged operation may lead to a memory bottleneck. However, when binary serialization is used, there is a stable memory usage. In Socket.IO, despite binary serialization, there are slight increases in memory usage over time when transmitting large data sizes. This can also lead to memory bottleneck with prolonged operation. Furthermore, in all cases where three clients are connected in Socket.IO, and in cases where two clients are connected but high traffic is being transmitted, communication fails due to disconnect problems. Therefore, the results of 10-second intervals are listed for these cases.

2.5 Network

According to socket statistics, there was a buffer problem for Socket.IO when disconnections occurred. An increase in the Send-Q on the server side was observed over time. While Autobahn could smoothly handle the same amount of traffic, Socket.IO's buffer was filling up.