http://wso2.com/

WSO2 API Manager 2.1.0 Performance Test Results

Date: 8 September 2017
Version: 1.0.0
Email: support@wso2.com

Revision History

Version	Date	Changes
1.0.0	8 September 2017	Initial Version

Table of Contents

Overview

This document analyzes the results of WSO2 API Manager 2.1.0 performance tests done in the Amazon EC2 environment.

Executive Summary

The performance of WSO2 API Manager was measured using following APIs, which invoke a simple “Netty HTTP Echo Service”. As the name suggests, the Netty service echoes back any request posted to the service.

1. **Echo API**: This is a secured API, which directly invokes the back-end service.
2. **Mediation API**: This is also a secured API, which has a “sequence” as a mediation extension to modify the message.

Tests were done using **100, 200, 300, 1000, and 2000 concurrent users**. Concurrent Users mean that there are multiple users accessing the API Gateway at the same time. Different Message Sizes (Payload) were used for the tests with different back-end service delays. **The message sizes used are 50B, 1KiB, 10KiB, and 100KiB. The back-end delays were 0ms, 30ms, 500ms, and 1s.**

Note: 100KiB message size was not tested with Mediation API. The back-end delay will also be referred as “Sleep Time” in this document.

Two key performance metrics were used to measure the performance of each test.

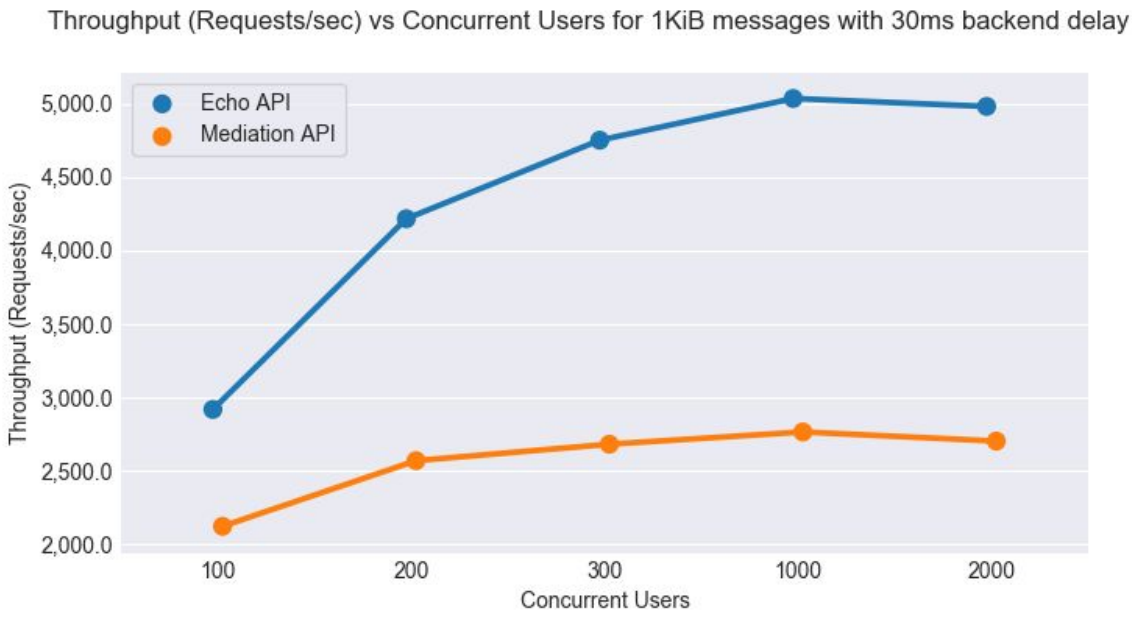
1. **Throughput**: This measures the number of API invocations that the API Manager Gateway server processed during a specific time interval (e.g. per second).
2. **Response Time**: This measures end-to-end processing time for an operation (of invoking an API using HTTPS protocol). The complete distribution of response times were recorded.

The heap size of WSO2 API Manager was increased to 4GB from 2GB, which is the default. Except for increasing the heap size of API Manager, there were no other specific configurations used to optimise the performance of WSO2 API Manager.

With WSO2 API Manager, an average user use ~1KiB messages and most of the back-ends usually responds in ~30ms. Therefore, let’s look at some charts to understand performance test results for above APIs when using 1KiB messages with 30ms backend delay.

The deployment used was “All-in-one API Manager”.

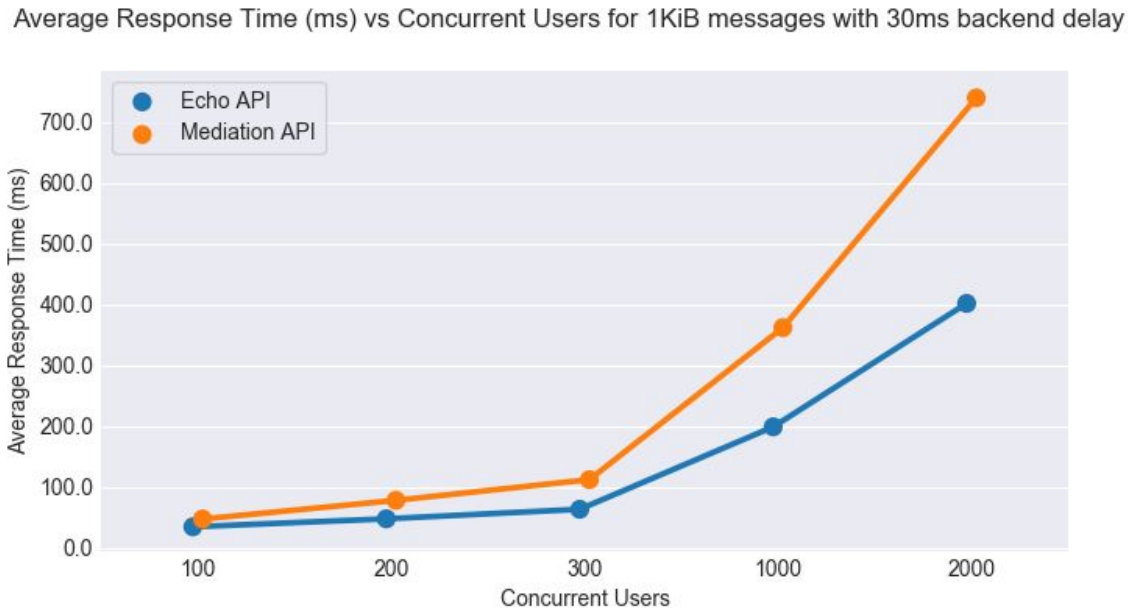
Following figure shows how the Throughput changes for different number of concurrent users.



Key observations:

- More concurrent users mean more requests to the API Manager Gateway. Therefore, the Throughput of the API Manager Gateway increases as the number of concurrent users accessing the APIs increases. The maximum throughput is obtained for 1000 concurrent users for both “Echo API” and “Mediation API” and the throughput degrades after 1000 concurrent users due to resource contentions in the system. The degradation point mainly depends on the hardware resources.
- Echo API throughput is much better than the Mediation API. Main reason is that the Mediation API has a mediation extension, which uses a “Payload Factory” Mediator. This mediation in the sequence does a JSON to JSON message transformation. That means, the Mediation API reads the message (payload) to do the message transformation and it has a considerable overhead than the “Echo API”, which is similar to a “Direct Proxy”. A “Direct Proxy” does not perform any processing on the messages that pass through it.

Following figure shows how the Average Response Time changes for different number of concurrent users.

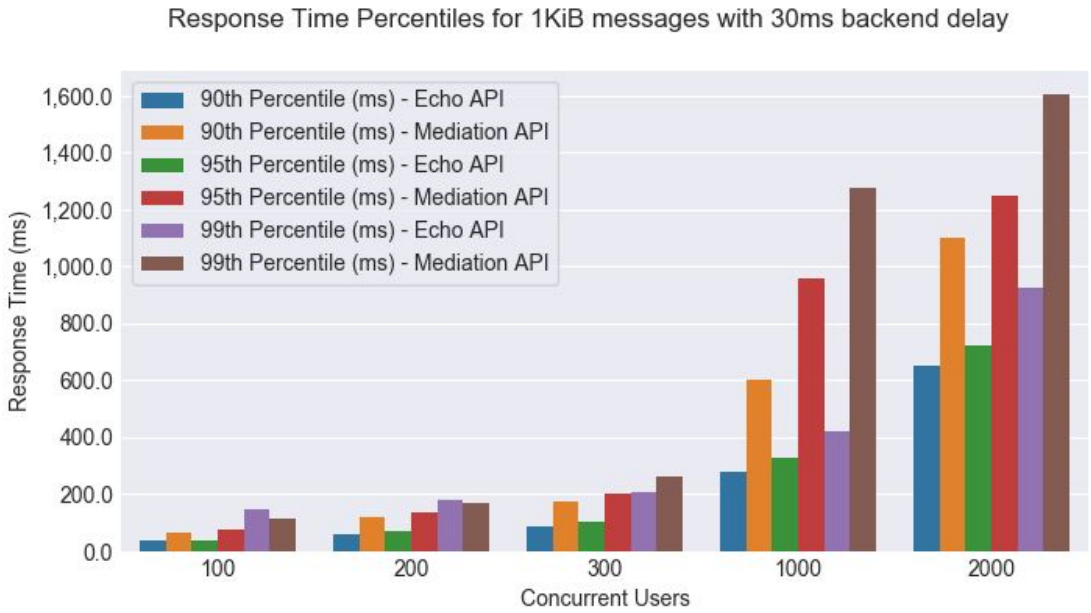


Key observations:

- The Average Response Time increases with the number of concurrent users. Since the number of requests to serve increases with more users, there are more resource contentions. Therefore, the number of concurrent users served by the API Gateway needs to be decided on the required response time limits. For example, in order to keep average response time below 100ms for APIs with 30ms backend delay, the maximum concurrent users accessing the API Gateway should be limited to 300.
- The Mediation API response times are higher than Echo API due to the performance overhead of mediation extension.

Let’s look at the 90th, 95th, and 99th Response Time percentiles. This is useful to measure the percentage of requests exceeded the response time value for a given percentile. A percentile can also tell the percentage of requests completed below the particular response time value.

For example, when there are 100 concurrent users, 90th response time percentile for Echo API is 36ms. This means that 10% of the requests have taken more than 36ms to respond. Similarly, 99th response time percentile for Echo API is 146ms, which means that 99% of the requests have completed within 146ms.



Key observations:

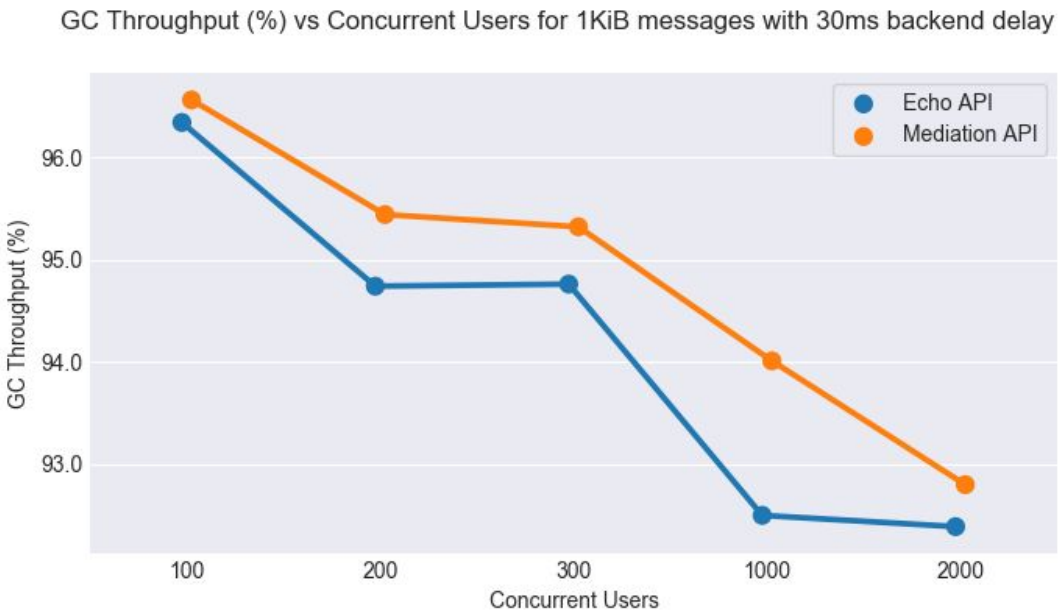
- Mediation API is slower than the Echo API due to the performance overhead of mediation extension.
- Response Times percentiles are less than 300ms up to 300 concurrent users.
- For higher concurrent users, 99th percentile of Mediation API response times goes beyond 1 second, which means that 1% of the API invocations took 1 second to 1.6 seconds.

Note: 1000 to 2000 concurrent users mean a lot and it is not very common. To support more concurrent users with acceptable response times, it is recommended to scale horizontally or vertically. When scaling horizontally, two or more Gateway nodes need to be used with a load balancer. To measure the performance after scaling, another load test must be carried out.

In order to see the memory usage, the Garbage Collection (GC) logs in the API Manager was enabled and the GC log for each performance test was analyzed using the GCViewer.

The GC Throughput was calculated for each test to check whether GC operations are not impacting the performance of the server. The GC Throughput is the time percentage of the application, which was not busy with GC operations. For example, if the application ran for 10 minutes and 30 seconds were taken for GC operations, the GC Throughput is $\left(1 - \frac{30}{10 \times 60}\right) \times 100 = 95\%$. A GC Throughput over 90% is good and that means the allocated heap was enough to handle all concurrent requests, which allocate objects in the memory.

Following chart shows the GC Throughput (%) for different number of concurrent users.



Key observations:

- GC Throughput decreases when the number of concurrent users increase. This means that the time spent for GC is increasing with concurrent users.
- GC Throughput for Mediation API is better than Echo API. The Mediation API processes requests slower than the Echo API, which means that the object allocation rate is also less than the Echo API.

Observations from all results

In Executive Summary, there are key observations for the average user scenario of accessing APIs with 1KiB messages and the back-end service having 30ms delay.

Following are the key observations from the all performance tests done with different message sizes and different backend delays. (See Comparison of results in Appendix for all charts used to derive the pointed mentioned below)

Key observations related to throughput:

- Throughput increases up to a certain limit when the number of concurrent users increase. Mediation API throughput increase rate is much lower than the Echo API.
- Throughput decreases when the message sizes increase. The Mediation API throughput decrease rate is much higher than the Echo API.
- Throughput decreases when the backend sleep time increase. This observation is similar to both APIs. This means that if the backend takes more time, the request processing rate at the API Manager gateway will be less.

Key observations related to response time:

- Average response time increases when the number of concurrent users increase. The increasing rate of average response time for Mediation API is much higher than the Echo API.
- Average response time increases considerably for Mediation API when the message sizes increase due to the message processing. The average response time of Echo API is not increasing as much as the Mediation API.
- Average Response Time increases when the backend sleep time increases. This observation is similar to both APIs.

Key observations related to GC Throughput:

- The GC throughput decreases when the number of concurrent users increase. When there are more concurrent users, the object allocation rate increases.
- The GC throughput increases when the message sizes increases. The request processing rate slows down due to the time taken to process large messages. Therefore, the object allocation rate decreases when the message sizes increases.
- The GC throughput increases when the backend sleep time increases. The object allocation rate will be low when the backend takes more time to respond.

Appendix

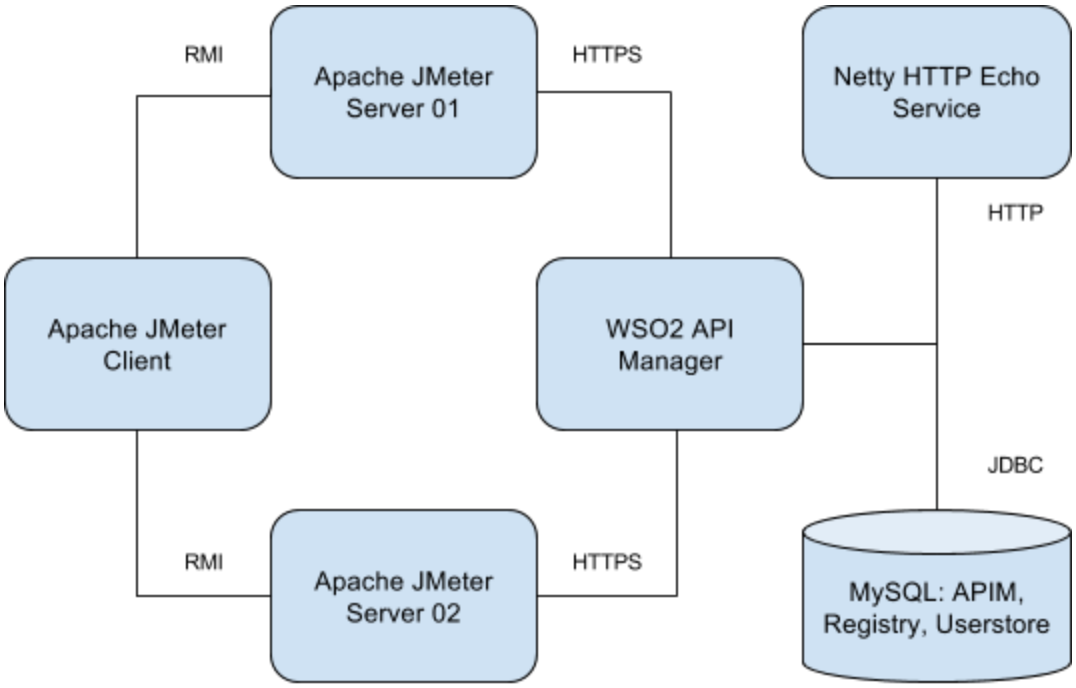
Deployment Details

WSO2 API Manager All-in-one Deployment

In this deployment, the WSO2 API Manager is deployed in one EC2 instance and a RDS instance is used for databases.

The back-end service, which is a simple Netty HTTP Echo Service is deployed in a separate EC2 instance.

There are two JMeter servers with a JMeter client in three EC2 instances. See: JMeter Remote Test. Two JMeter servers are used to simulate high number of concurrent users.



Name	EC2 Instance Type	vCPU	Mem (GiB)
Apache JMeter Client	c3.large	2	3.75
Apache JMeter Server 01	c3.xlarge	4	7.5
Apache JMeter Server 02	c3.xlarge	4	7.5
WSO2 API Manager	c3.xlarge	4	7.5
Netty HTTP Echo Service	c3.xlarge	4	7.5
MySQL	db.m3.medium (RDS)	1	3.75

See following links for more details on Amazon Instance Types
https://aws.amazon.com/ec2/instance-types/
http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Concepts.DBInstanceClass.html

The operating system is Ubuntu 16.04.2 LTS.
WSO2 API Manager version is 2.1.0 and Apache JMeter version is 3.2.
MySQL version in RDS instance was 5.7

Backend Service

The backend service used for tests were developed using Netty. Since there can be up to 2000 users, 2000 threads were used for Netty (to avoid the back-end becoming a bottleneck)

The Netty server also has a parameter to specify the number of seconds to sleep to simulate the delays in the backend service)

Performance Testing Tool

As mentioned before, Apache JMeter was used to run load tests.

In JMeter, the number of concurrent users was specified and following details were taken after each test.

- **# Samples** - The number of requests sent with the given number of concurrent users.
- **Error Count** - How many request errors were recorded.
- **Error %** - Percent of requests with errors
- **Average** - The average response time of a set of results
- **Min** - The shortest time taken for a request
- **Max** - The longest time taken for a request

- **90th Percentile** - 90% of the requests took no more than this time. The remaining samples took at least as long as this
- **95th Percentile** - 95% of the requests took no more than this time. The remaining samples took at least as long as this
- **99th Percentile** - 99% of the requests took no more than this time. The remaining samples took at least as long as this
- **Throughput** - The Throughput is measured in requests per second.
- **Received KB/sec** - The throughput measured in received Kilobytes per second
- **Sent KB/sec** - The throughput measured in sent Kilobytes per second

In addition, to above details, some additional details were recorded for every test.

- **GC Throughput** - Time percentage the application was not busy with GC

GC throughput and other GC related details were obtained from the GC logs produced by the WSO2 API Manager.

Following are the GC flags used:

```
-XX:+PrintGC -XX:+PrintGCDetails -XX:+PrintGCDateStamps
-Xloggc:"$CARBON_HOME/repository/logs/gc.log"
```

Note: The process memory was not considered as Java is working on an already reserved heap area.

Performance Test Scripts

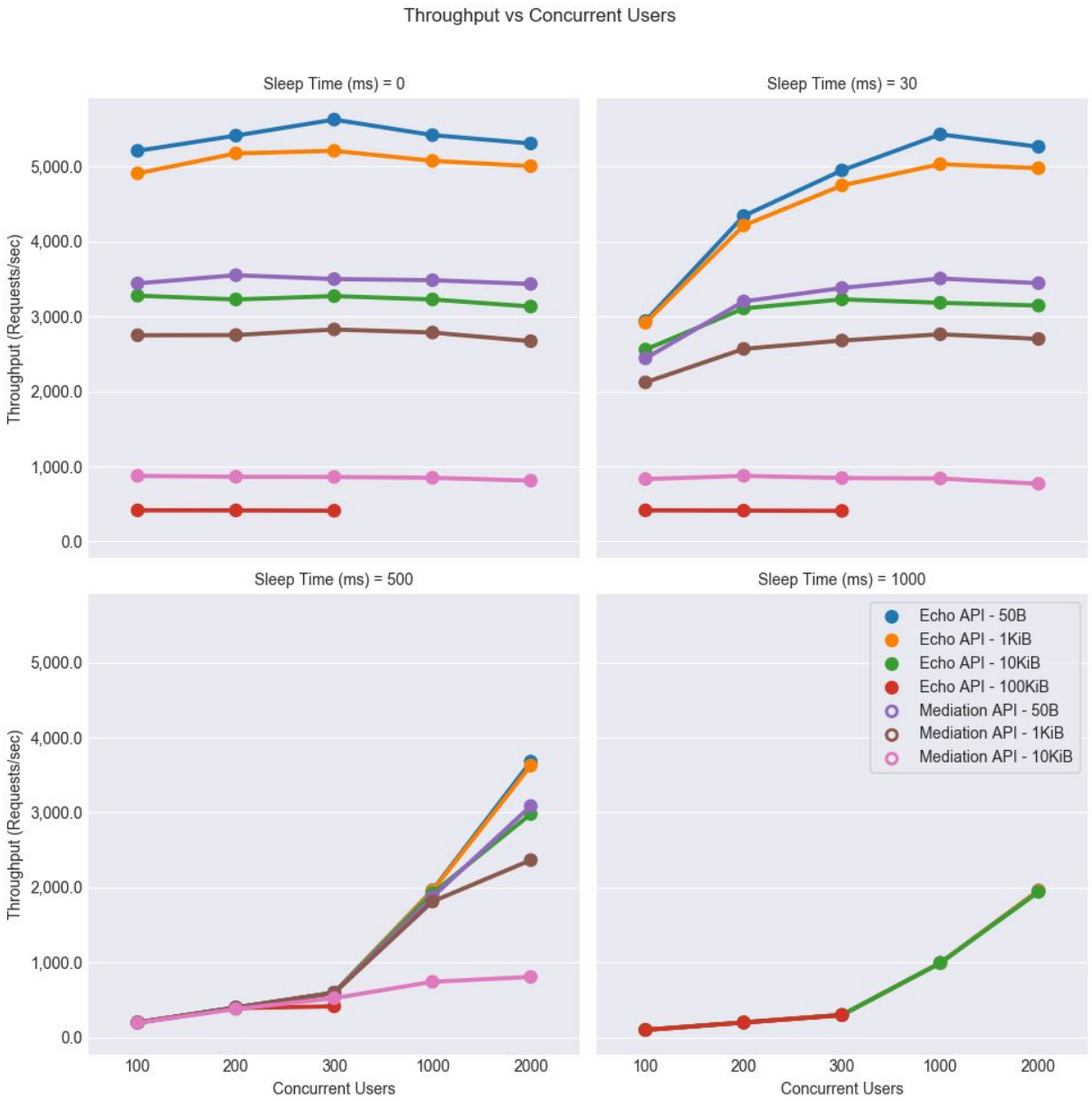
All scripts used to run the performance tests and analyze results are in following repositories.

https://github.com/wso2/performance-common/
https://github.com/wso2/performance-apim/

Comparison of results

Let's compare the results for all test scenarios mentioned in Executive Summary. Refer Observations from all results for more details on the charts.

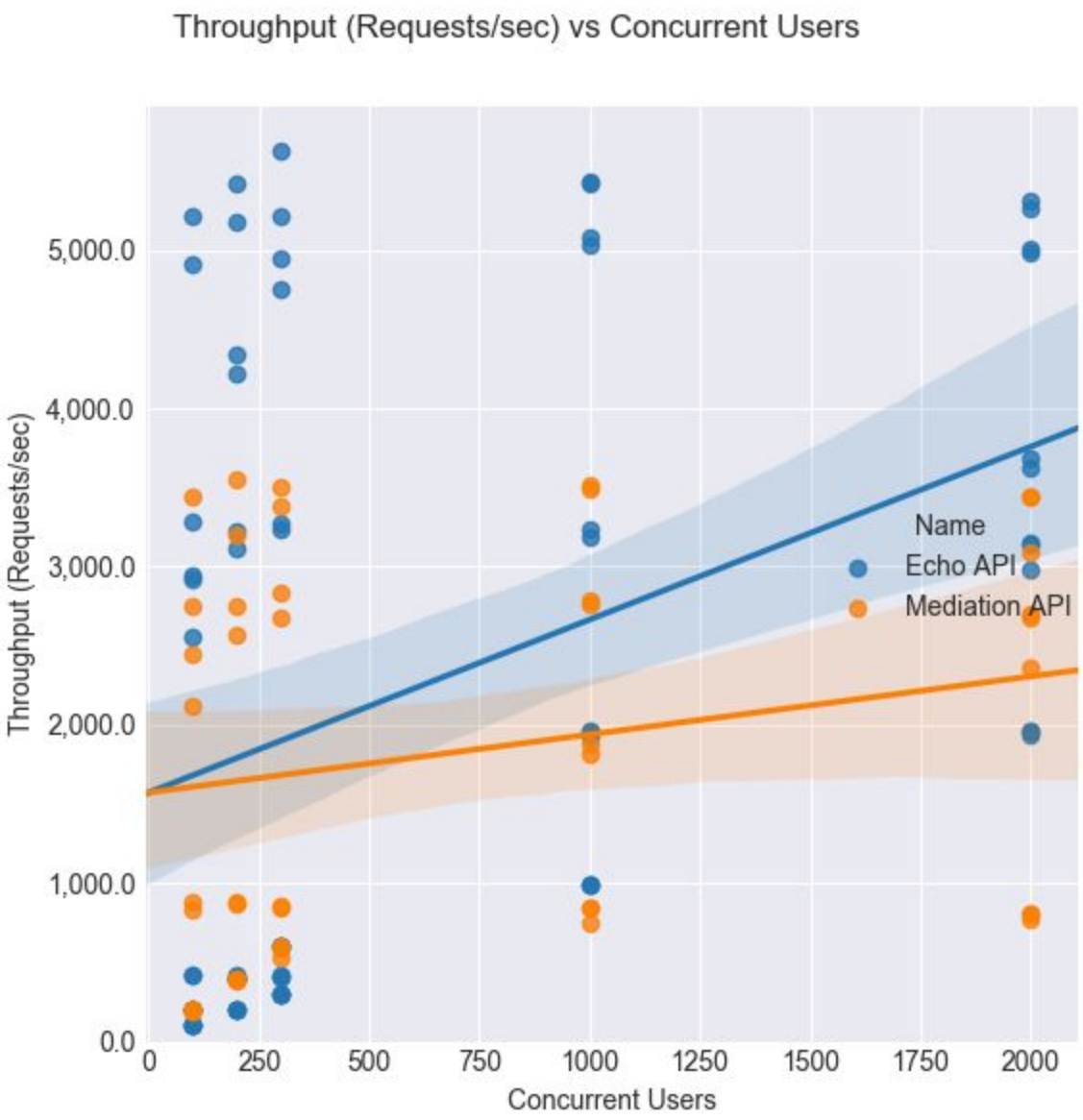
Throughput Comparison



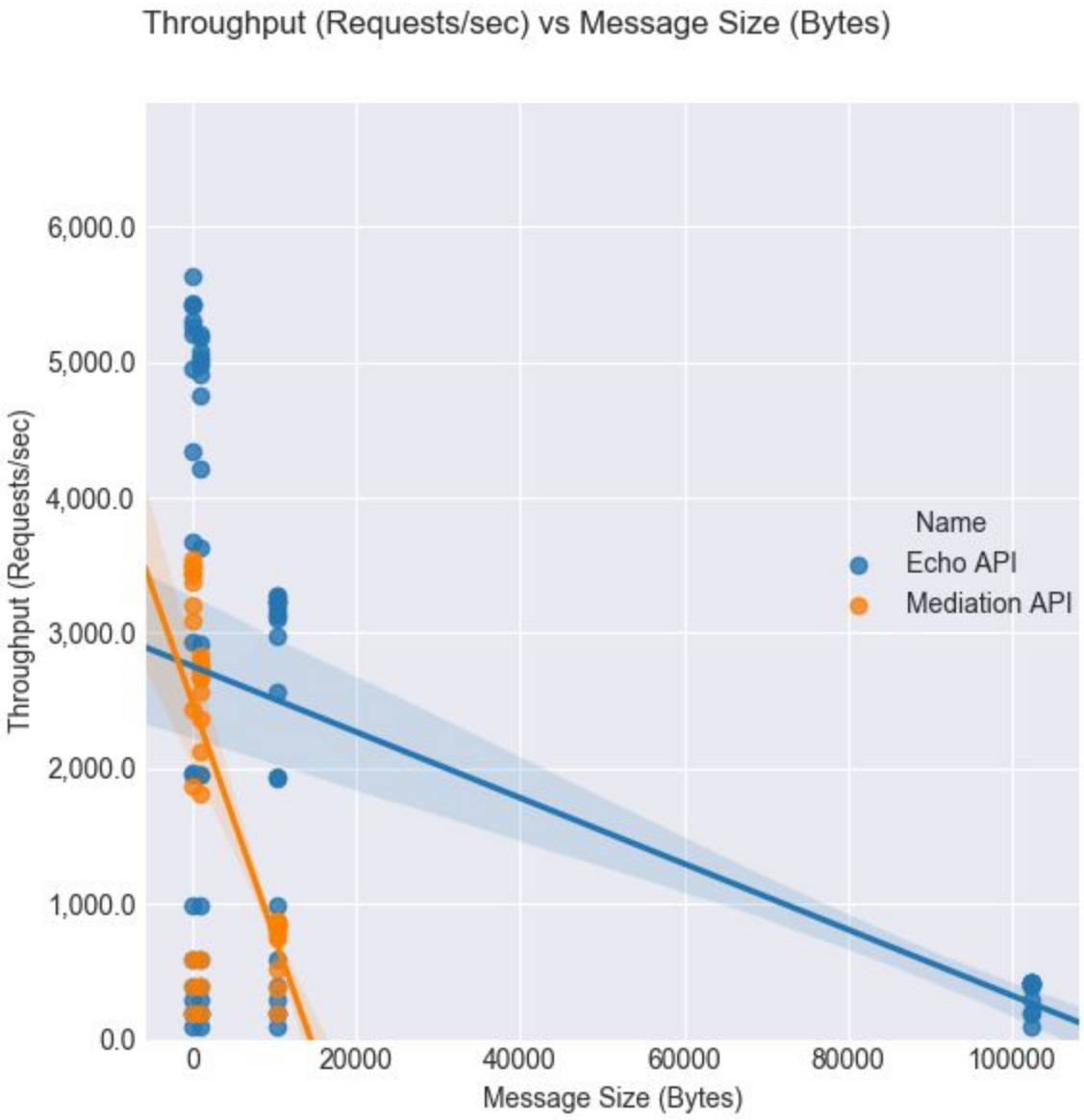
As mentioned in Executive Summary, the Mediation API was not tested with 100KiB message size. The Echo API has some errors with 100KiB message size for 1000 and 2000 concurrent users.

Following charts show what happens to the server throughput when considering all results.

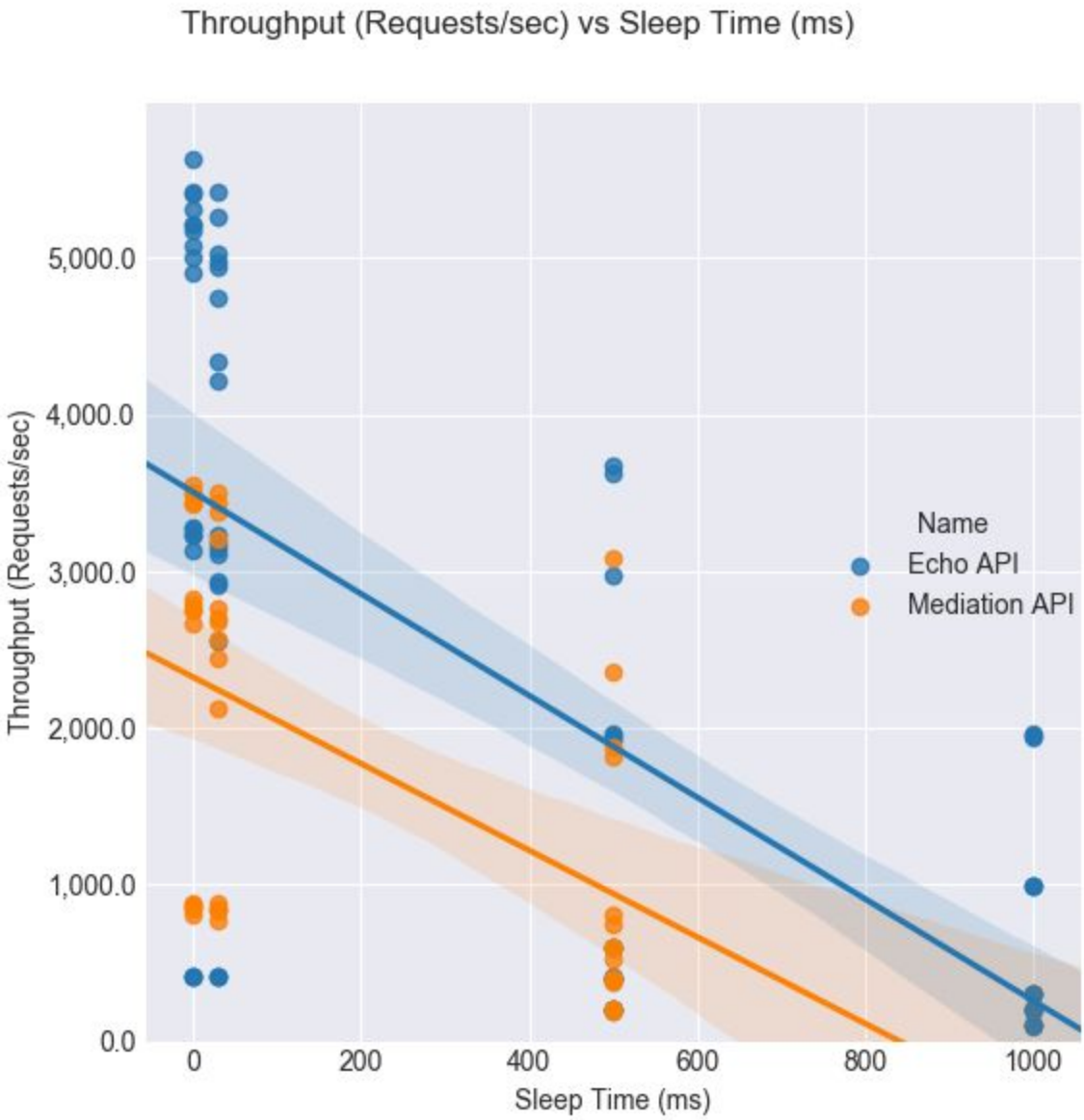
Throughput (Requests/sec) vs Concurrent Users



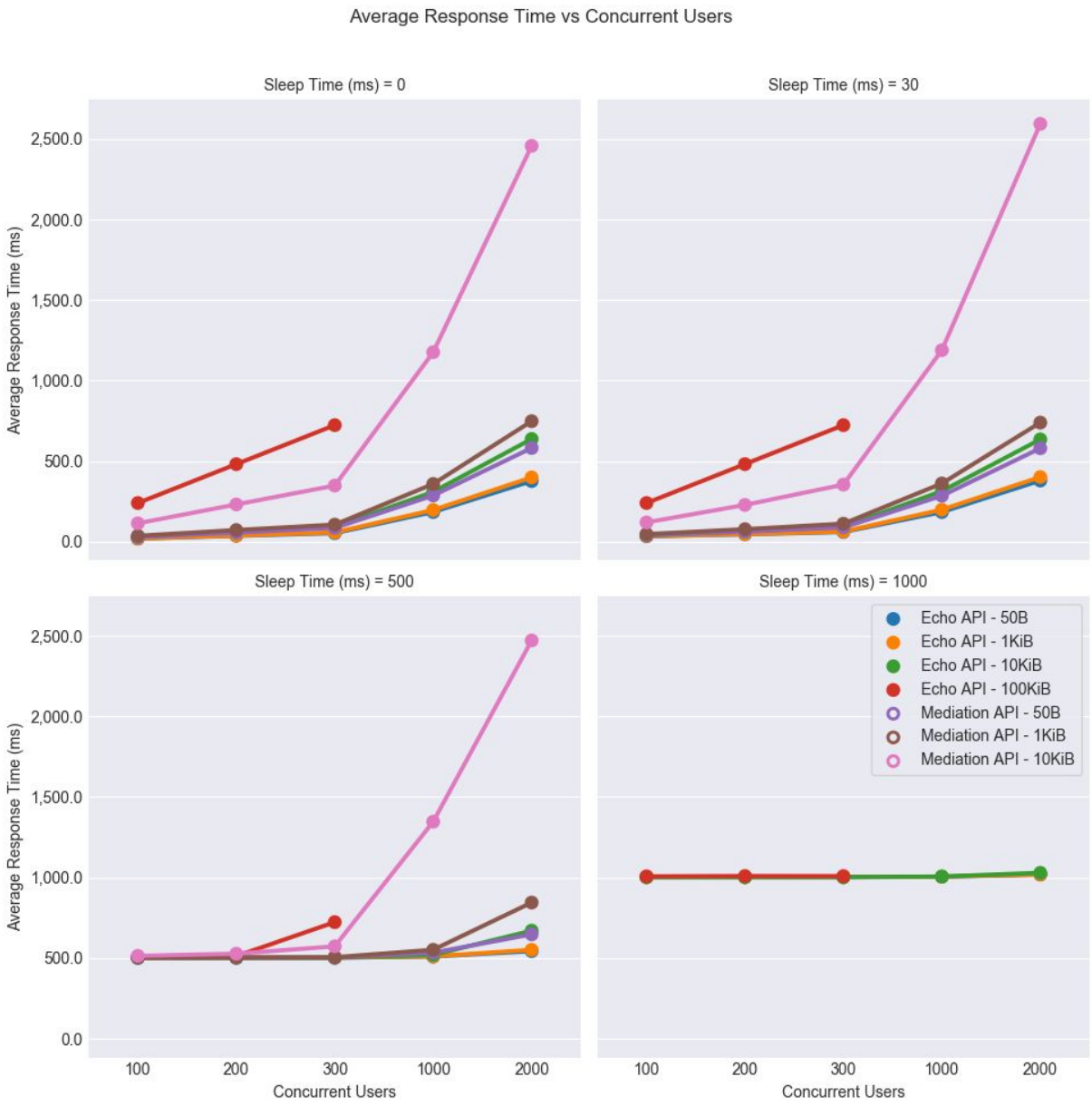
Throughput (Requests/sec) vs Message Size (Bytes)



Throughput (Requests/sec) vs Sleep Time (ms)

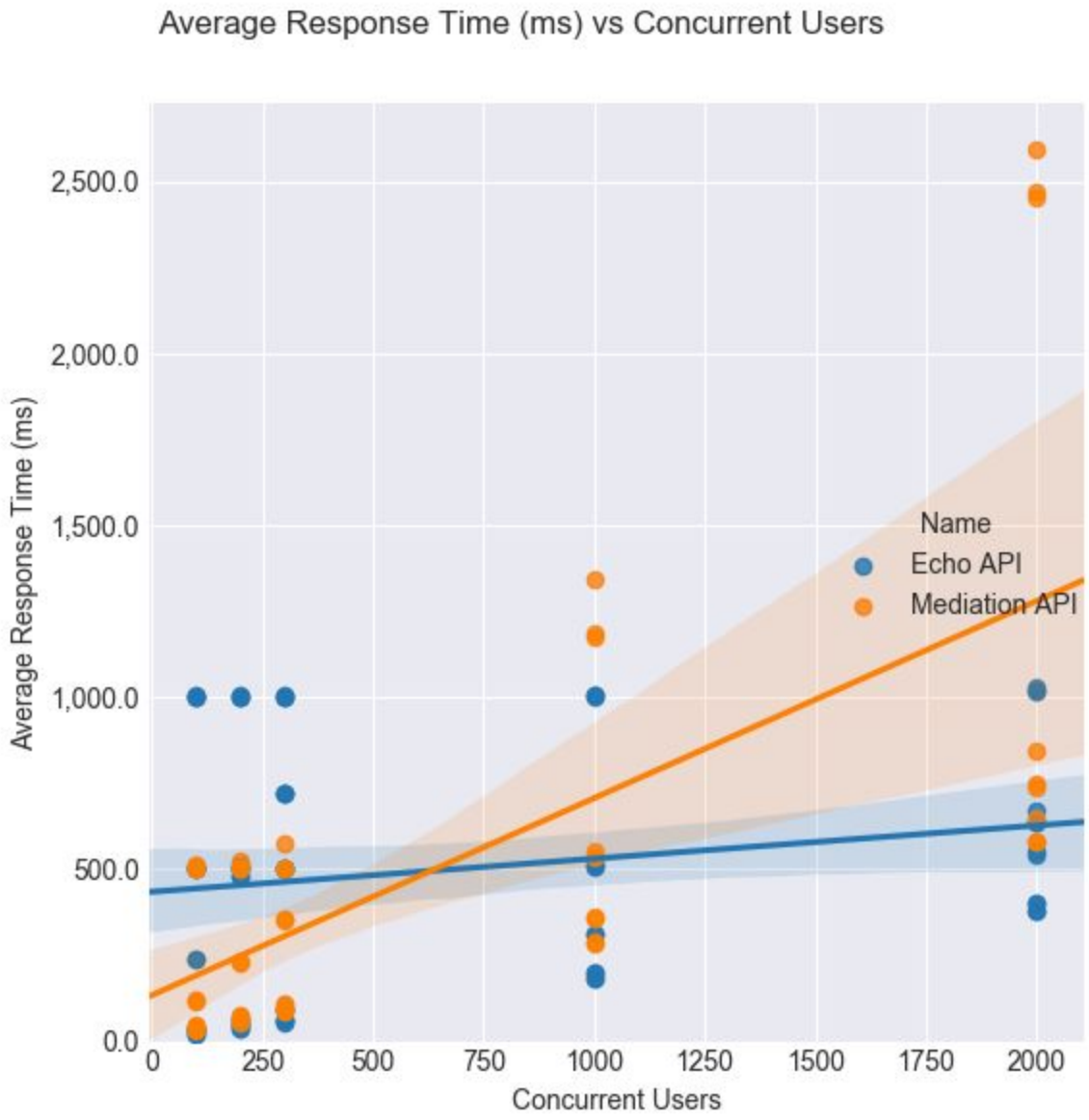


Average Response Time Comparison

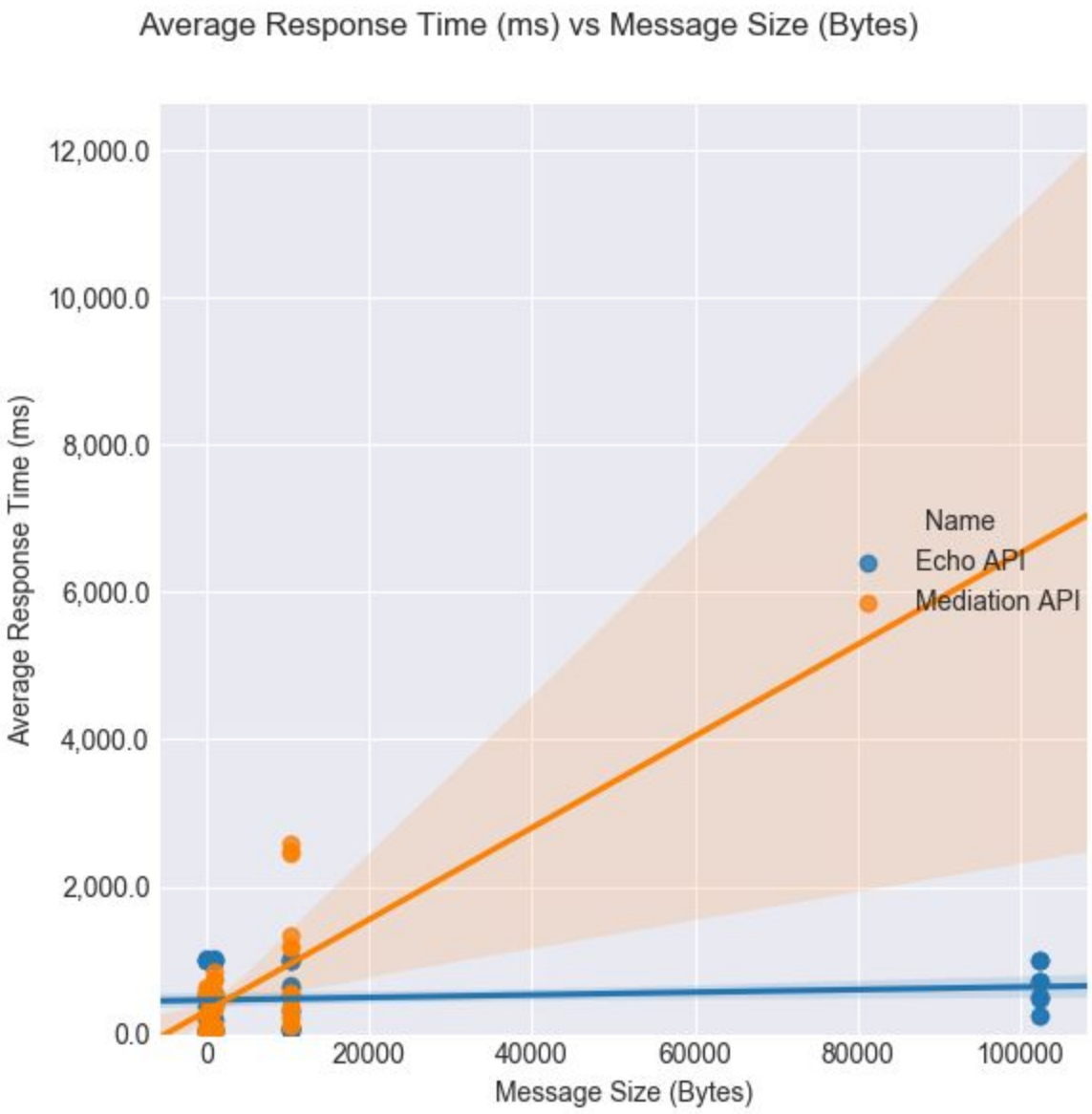


Following charts show what happens to the average response time when considering all results.

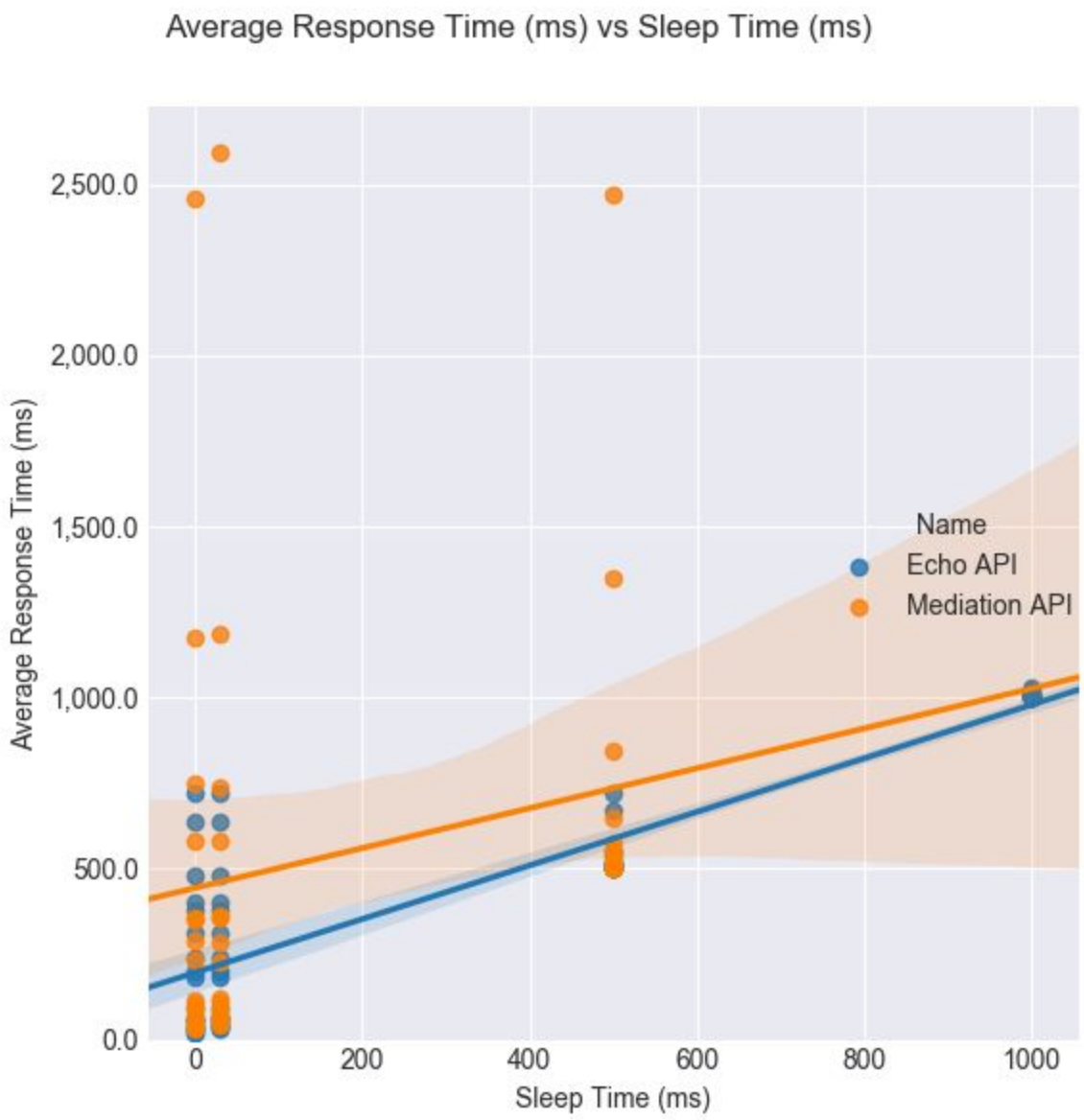
Average Response Time (ms) vs Concurrent Users



Average Response Time (ms) vs Message Size (Bytes)

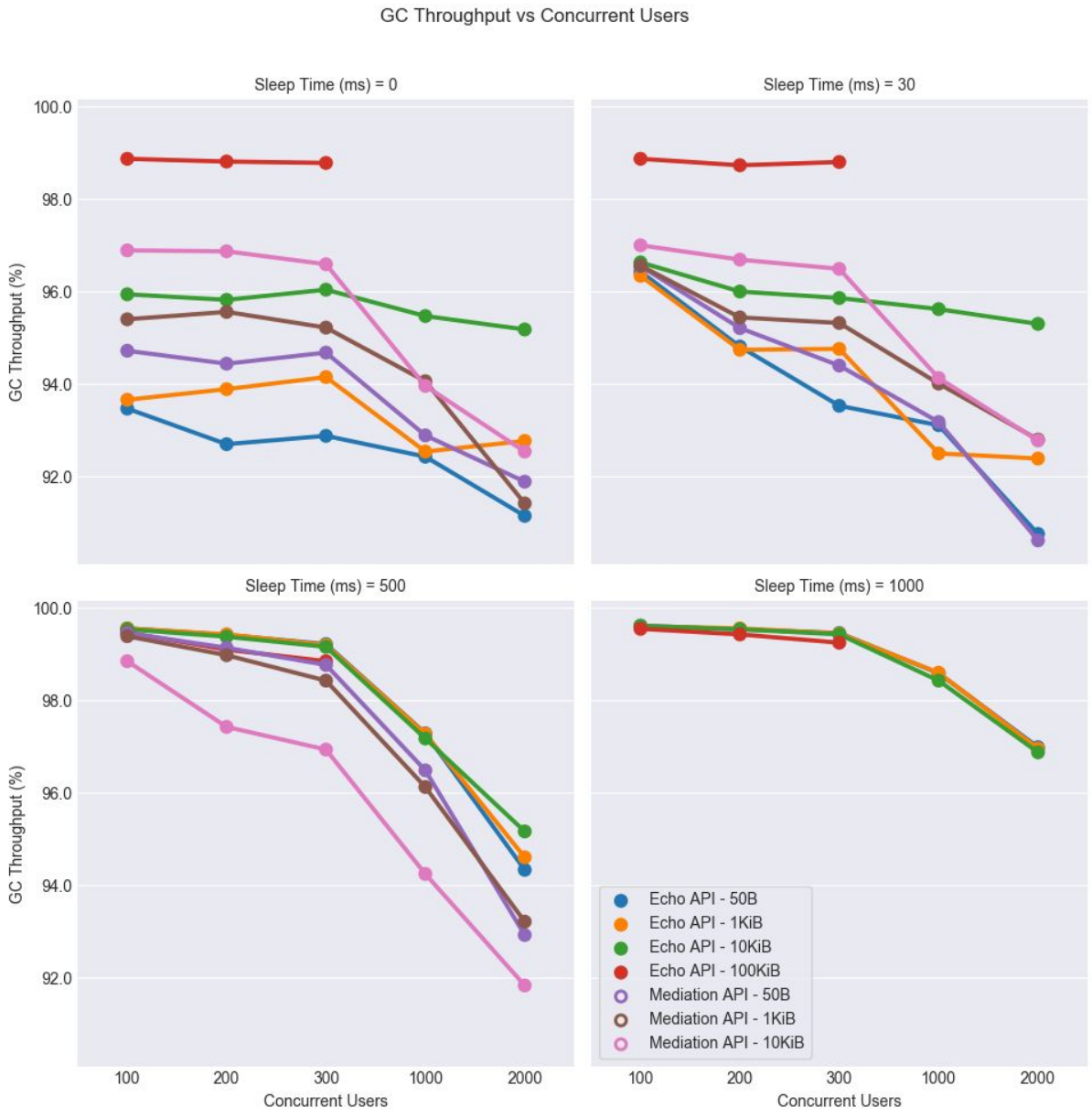


Average Response Time (ms) vs Sleep Time (ms)



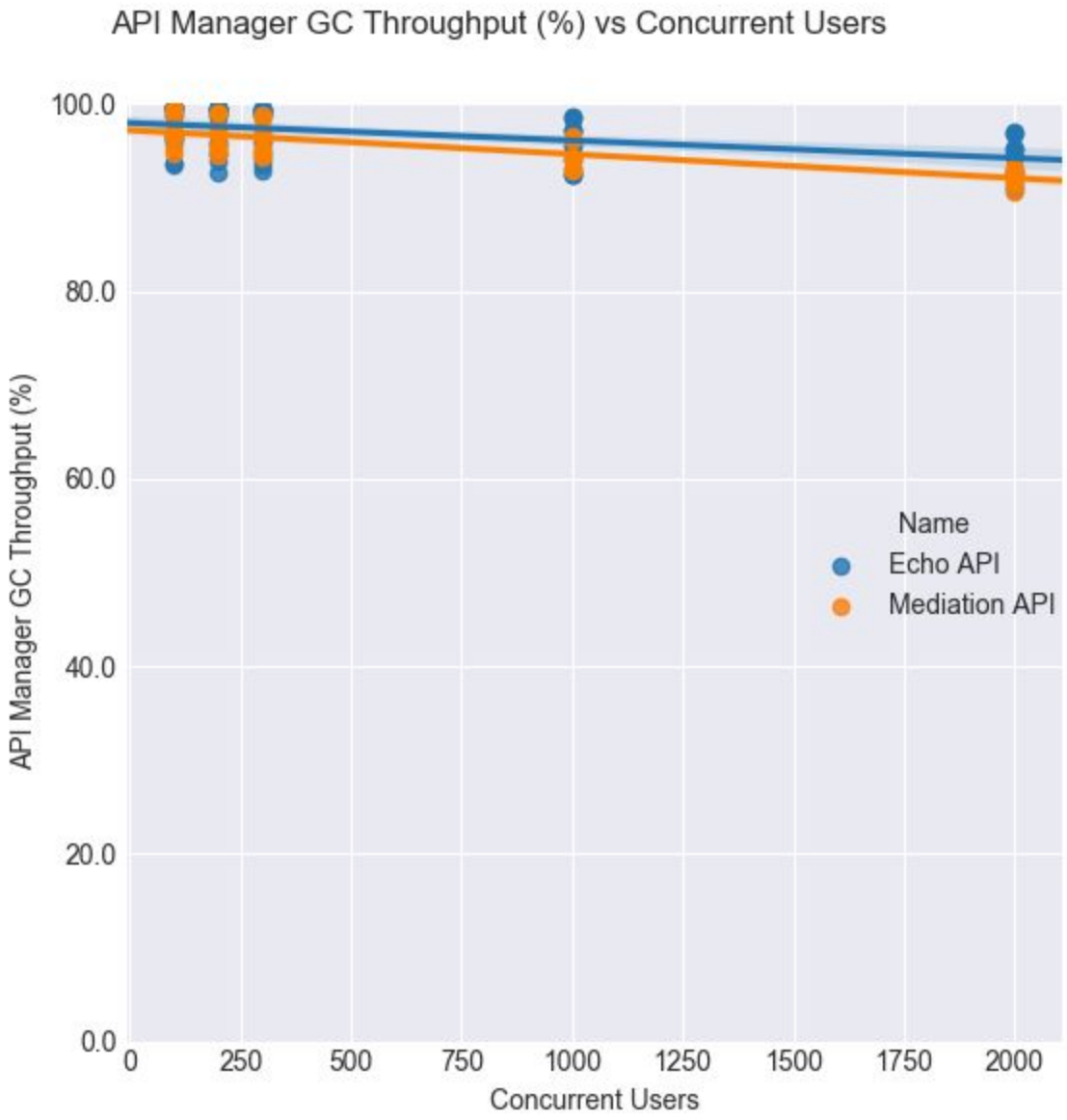
GC Throughput Comparison

Following chart

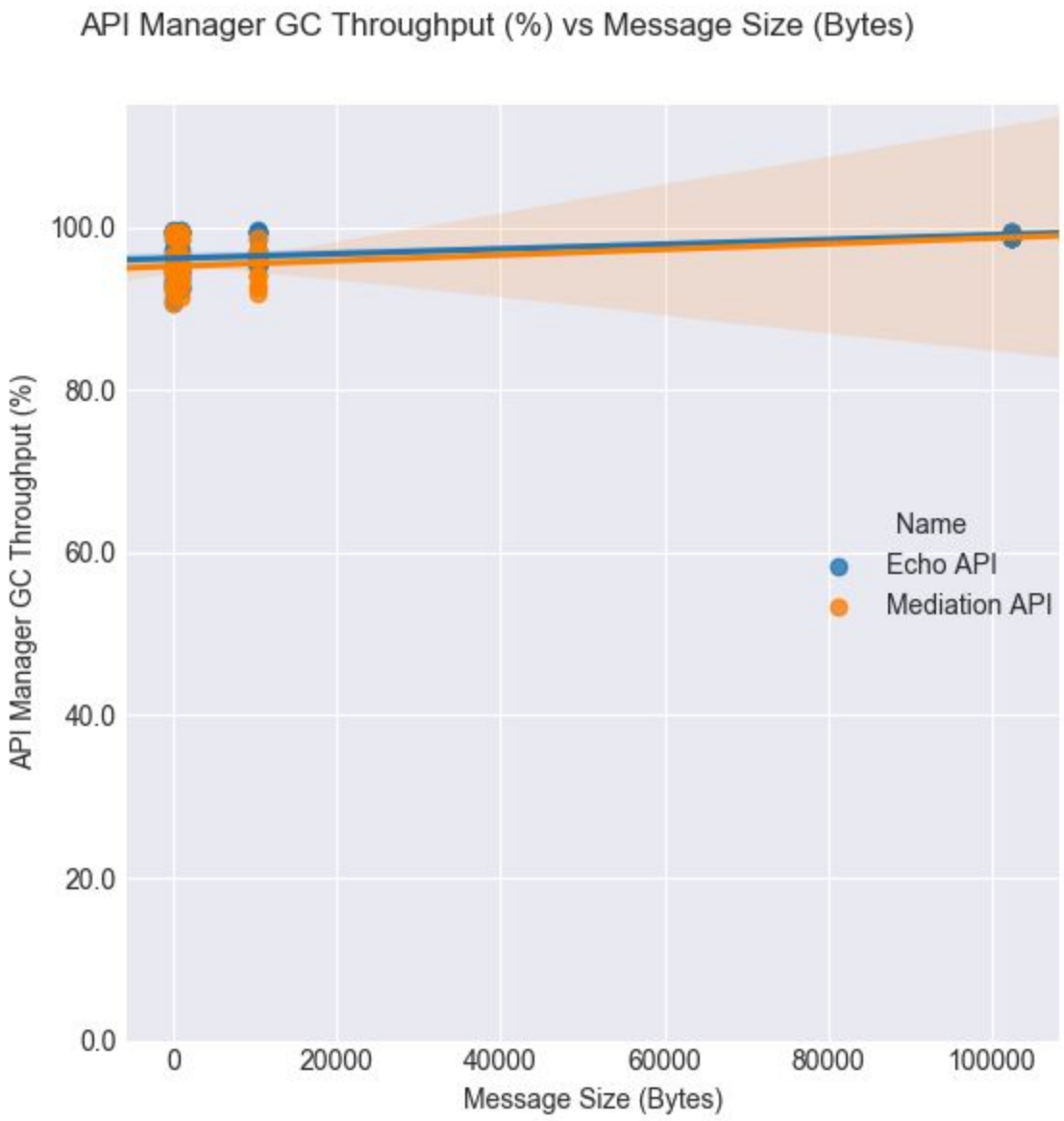


Following charts show the GC throughput behavior when considering all results.

API Manager GC Throughput (%) vs Concurrent Users



API Manager GC Throughput (%) vs Message Size (Bytes)



API Manager GC Throughput (%) vs Sleep Time (ms)