JMeter Stress Testing

Version 1.0

Prepared by Richard Ashman
ID:1342966

31/05/2019

Table of Contents

Revision History

Name	Date	Reason For Change	Version
Richard Ashman	31/05/2019	Release	V1.0

1. Introduction

1.1. Aims

This report is to document the testing of app.py using Python/Flask/Redis/Docker.
JMeter has been used for stress testing the application.

2. Manual Test Cases

All tests are black-box tests unless otherwise stated.

2.1. Case 1

- curl localhost:42966/isPrime/

Should not store any prime as it is not a recognised request.

2.2. Case 2

- curl localhost:42966/isPrime/1

White-box test
Special edge case in the make isPrime method
Should return ‘1 is not prime” as 1 is not a prime number.

2.3. Case 3&4

- curl localhost:42966/isPrime/2

Edge case as 2 is the first prime number
Should return “2 is prime”

- curl localhost:42966/primesStored

Black-box test
Should return the 2 that should have been stored from case 3

2.4. Case 5&6

- curl localhost:42966/isPrime/4

Should return “4 is not prime” as 4 is not a prime

- curl localhost:42966/primesStored

Should not return the 4 from case 5

2.5. Case 7

- curl localhost:42966/isPrime/5

Should return “5 is prime” as 5 is a prime

- curl localhost:42966/isPrime/4

Should return “4 is not prime” as 4 is not a prime

- curl localhost:42966/isPrime/17

Should return “17 is prime” as 17 is a prime

- curl localhost:42966/primesStored

Should return the 5 and the 17 but not the 4 (should also include a 2 from if test case 3 has been run)

2.6. Case 8

curl localhost:42966/isPrime/2147483647
Should return “2147483647 is prime” as 2147483647 is a prime

- curl localhost:42966/primesStored

The last number returned should be 2147483647. (There might be more numbers if other tests have been run.

3. JMeter Test Cases

3.1. Case 1

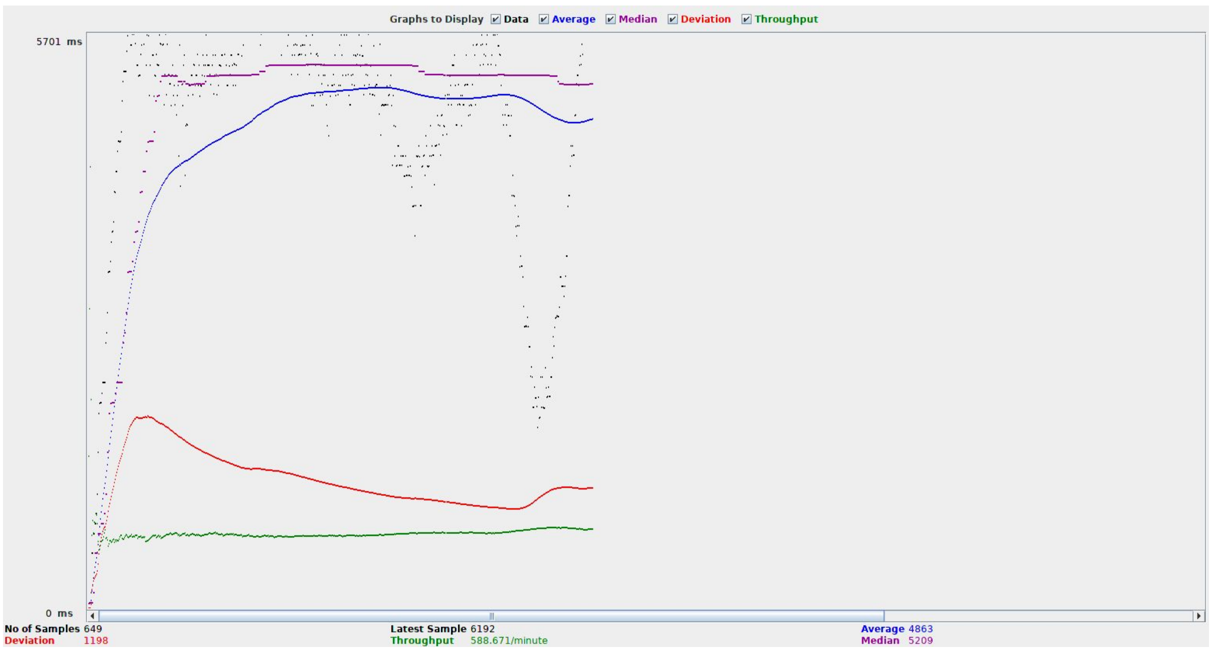
2147483647 is checked to see if it is a prime. This is called using the app's isPrime URL (/isPrime/<num>). This is done with 50 threads for 60 seconds.

This exact test was run twice. The first time this test was run the error count skyrocketed to 90% then spent the next minute slowly dropping.
Throughput stayed around 500/minute
looking into it showed that the app (redis) did a save of data.
This caused errors and displayed the following

```
redis_1  | 1:M 28 May 2019 22:40:41.070 * 100 changes in 300 seconds. Saving...
redis_1  | 1:M 28 May 2019 22:40:41.158 * Background saving started by pid 12
redis_1  | 12:C 28 May 2019 22:40:41.224 * DB saved on disk
redis_1  | 12:C 28 May 2019 22:40:41.225 * RDB: 0 MB of memory used by copy-on-write
redis_1  | 1:M 28 May 2019 22:40:41.259 * Background saving terminated with success
```

The second time the test was run produced no errors with the following info:

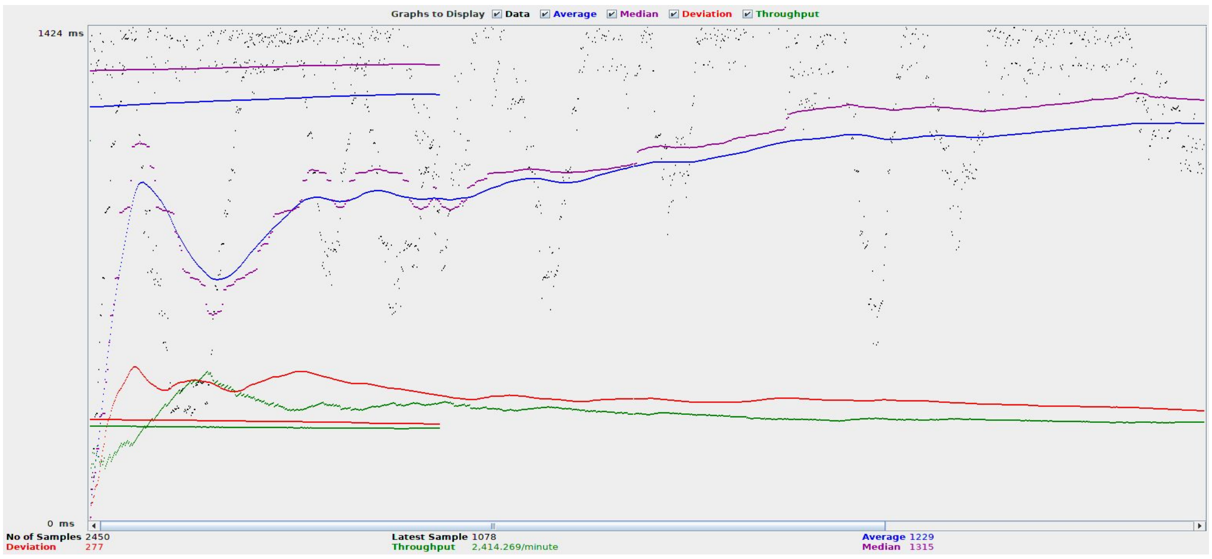
Samples	Average	Median	90%	Min	Max	Error%	Throughput (%)	KB/sec
649	4863	5209	5902	53	6799	0.0	9.8111	1.6767



No of Samples 649
Deviation 1198
Latest Sample 6192
Throughput 588.671/minute
Average 4863
Median 5209

This was run again using 0.5 cpus (rather than 0.1) with the following results.

Samples	Average	Median	90%	Min	Max	Error%	Throughput (%)	KB/sec
2450	1229	1315	1487	6	1788	0.0	40.2378	6.8765



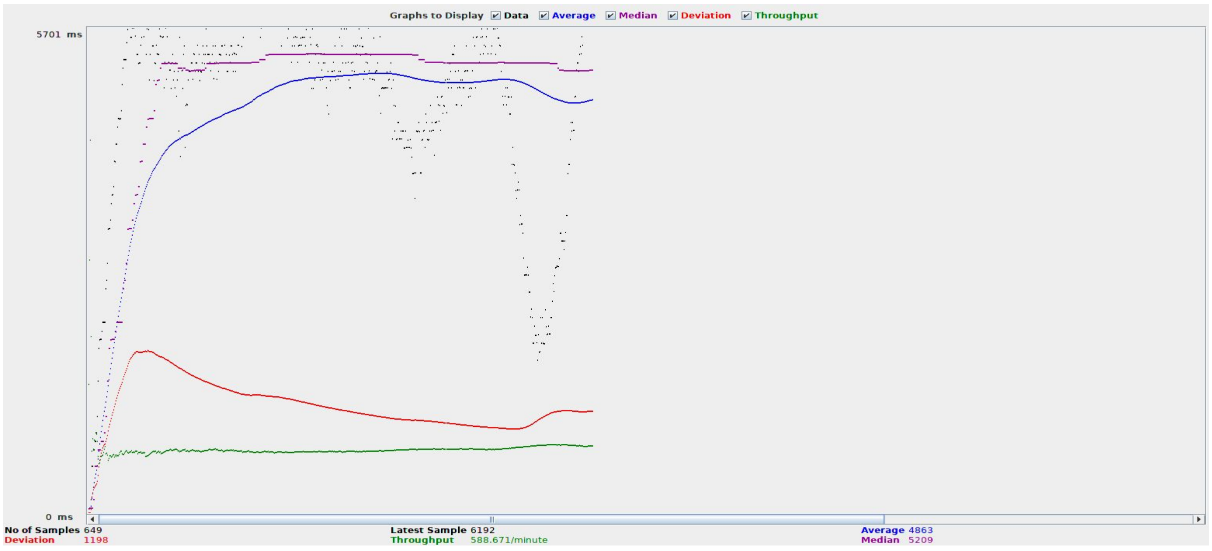
No of Samples 2450
Deviation 277
Latest Sample 1078
Throughput 2,414.269/minute
Average 1229
Median 1315

3.2. Case 2

isPrime is called for the numbers 1-100. Then primesStored is called for 60 seconds. Each of the 50 threads calls isPrime twice with the next number between 1-100 then all threads continue with the primesStored calls until the timer reaches 0.

This test was run first with a Ramp-Up Period (in seconds) of 0 and a Startup delay (seconds) of 0. The results were the following:

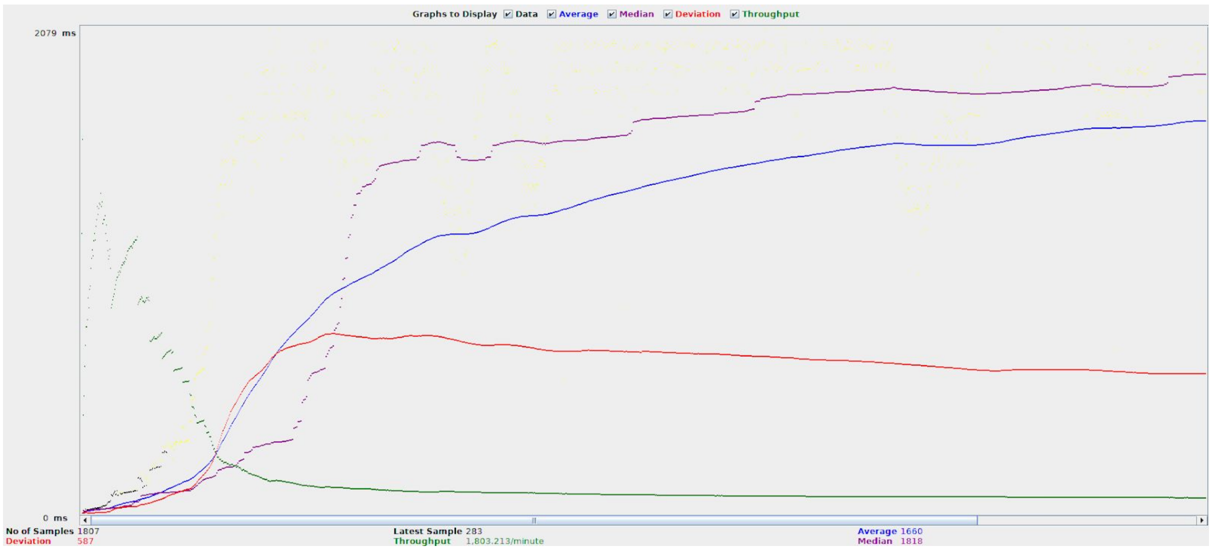
Samples	Average	Median	90%	Min	Max	Error%	Throughput (%)	KB/sec
426	7174	7400	9206	2793	11300	0.0	6.3304	1.4525



No of Samples 649
Deviation 1198
Latest Sample 6192
Throughput 588.671/minute
Average 4863
Median 5209

This test was run second time with a Ramp-Up Period (in seconds) of 0 and a Startup delay (seconds) of 0. This time the cpus was changed from 0.1 to 0.5. It is clear to see when the first part (1-100) changed into the call to primesStpred. Using more of the cpu to stress test the system also started to cause errors. The results were the following:

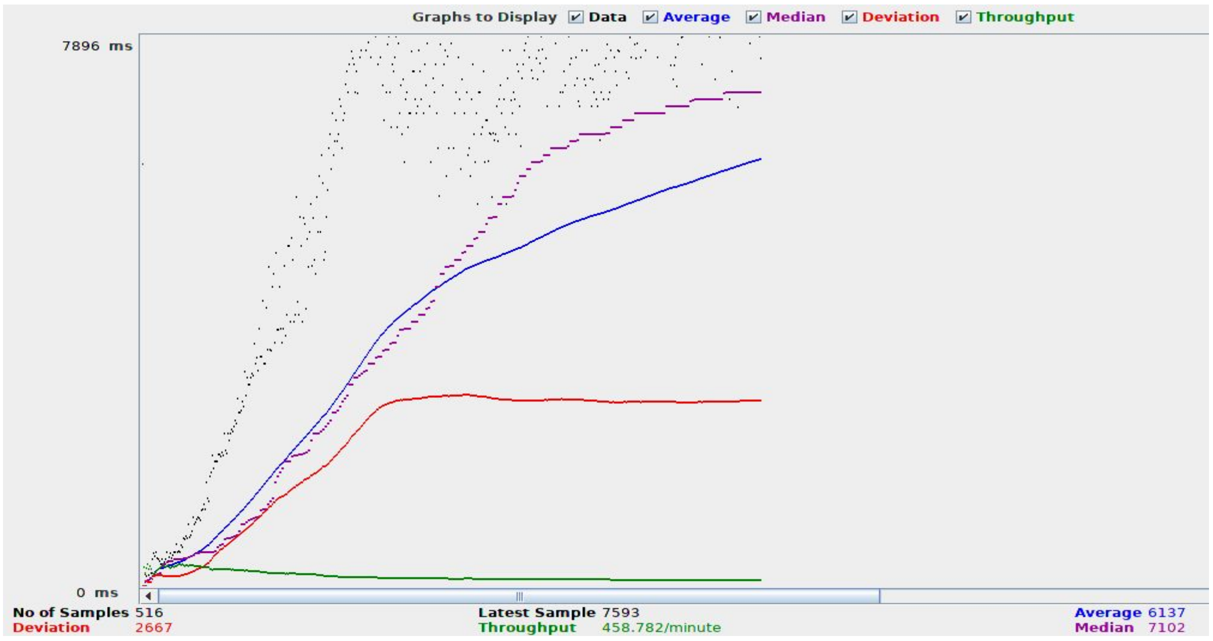
Samples	Average	Median	90%	Min	Max	Error%	Throughput (%)	KB/sec
1707	1752	1877	2190	91	3407	1.0	28.4073	12.4559



No of Samples 1807
Deviation 587
Latest Sample 283
Throughput 1,803.213/minute
Average 1660
Median 1818

This test was run then run for a second time but with a Ramp-Up Period (in seconds) of 2 and a Startup delay (seconds) of 2. The results were the following:

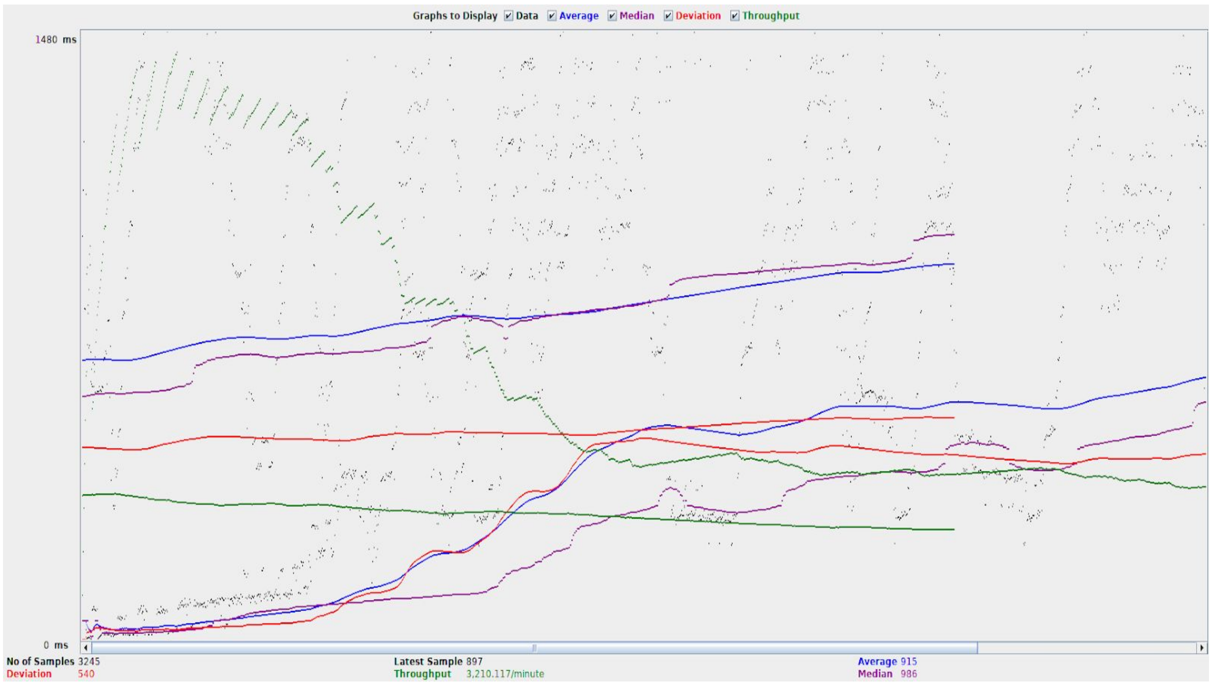
Samples	Average	Median	90%	Min	Max	Error%	Throughput (%)	KB/sec
416	7180	7500	8796	396	10196	0.0	6.1813	1.3533



No of Samples 516
Deviation 2667
Latest Sample 7593
Throughput 458.782/minute
Average 6137
Median 7102

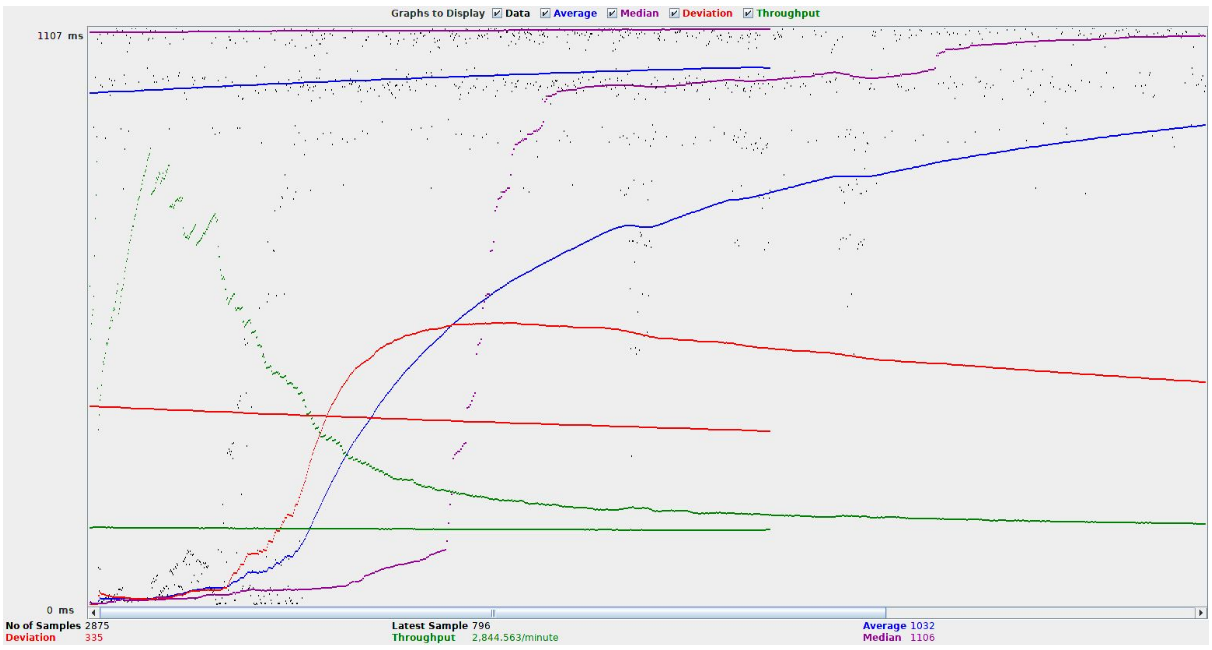
This test was run for a third time with the same 2 second delay but also using 0.5 cpus rather than 0.1 cpus.

Samples	Average	Median	90%	Min	Max	Error%	Throughput (%)	KB/sec
3145	939	999	1614	4	2274	0.0	51.9036	11.3303



```
No of Samples 3245        Latest Sample 897                 Average 915
Deviation     540         Throughput    3,210.117/minute    Median  986
```

This test was run for a last time with the same 2 second ramp up period and using 0.9 cpus.



```
No of Samples 2875        Latest Sample 796                 Average 1032
Deviation     335         Throughput    2,844.563/minute    Median  1106
```

Samples	Average	Median	90%	Min	Max	Error%	Throughput (%)	KB/sec
1623	1843	1995	2406	108	2795	0.0	26.4586	7.6880

4. Discussion

There was a small issue with some errors when the cpus was ramped up to 0.5 cpus rather than 0.1 cpus but this later this did not show again even when the cpus were changed again. This was likely when the web service got increased without the redis service being increased. This would have caused an imbalance in the throughput that each service could handle. Potentially having the web service firing requests at the redis service when redis did not have the resources to complete the request.

The first large set of errors was caused by Redis saving but this did not always show up leading me to believe some changes in the redis configuration could have mitigated some of the delays caused.

This is a good result because having even when limiting the web app between 0.9 cpus and 0.1, JMeter still did not make many errors. A lot of the time JMeter got 0 errors.

5. Links

GitHub: https://github.com/RichardAshman/COMPX341-Assignment4