# DV1619 Assignment 4 Report

Yikwenmein Victor Magheng
yikwenmeinvictor1995@gmail.com,
19950218-T614

2021-08-12

## ABSTRACT

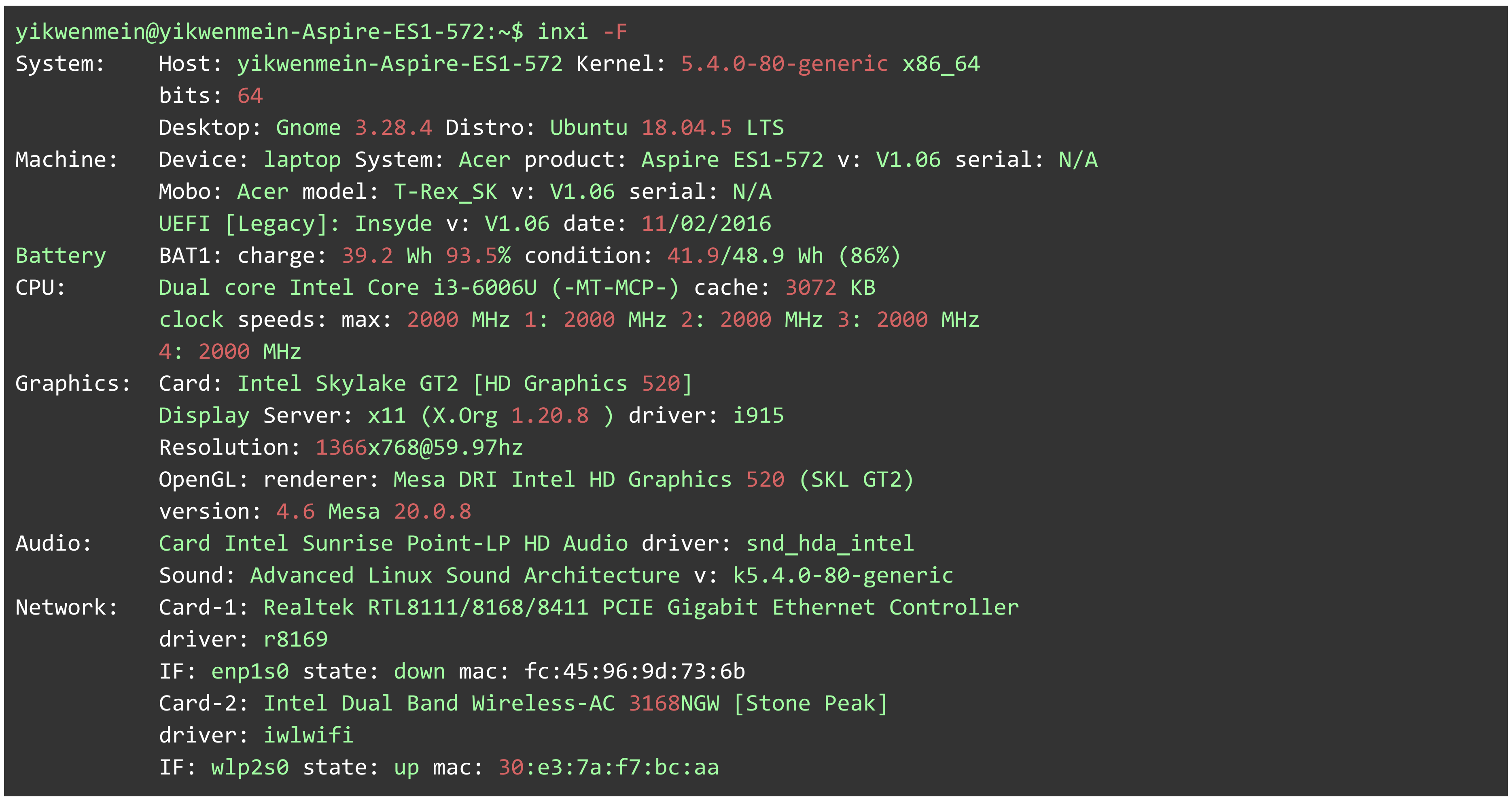
**This report presents an investigation of processes and threads,the trade-offs in Web servers. A Web server needs to support concurrency. The server should service clients in a timely, fair manner to ensure that no client starves because some other client causes the server to hang. Processes and threads are traditional ways to achieve concurrency.To highlight the trade-offs among the approaches, I have two echo servers written in C(forking server and a threading server) and their performance aspects measured using Apache benchmarking tool.**

## 1. INTRODUCTION

This report is part of assignment 4 of the computer communications Course and provides an investigation into performance aspects of a web server . It dives into the performance aspects of a threaded and forked web server. The report presents performance based findings obtained for a threaded and a forked web server. It uses the Apache benchmarking tool to investigate which of the servers (threaded or forked) performs better than the other and under what conditions(platform and hardware properties) they do so

## 2. EXPERIMENTAL SETUP

Ubuntu 18.04.5 was used for all experiments. All hardware details of this  Distro and other system properties can be seen on the screenshot below. As for the network setup, the experiments were all run on localhost. Both client and server were run from the same machine(Ubuntu 18.04.5 ) with the properties seen below.



```
yikwenmein@yikwenmein-Aspire-ES1-572:~$ inxi -F
System:    Host: yikwenmein-Aspire-ES1-572 Kernel: 5.4.0-80-generic x86_64
           bits: 64
           Desktop: Gnome 3.28.4 Distro: Ubuntu 18.04.5 LTS
Machine:   Device: laptop System: Acer product: Aspire ES1-572 v: V1.06 serial: N/A
           Mobo: Acer model: T-Rex_SK v: V1.06 serial: N/A
           UEFI [Legacy]: Insyde v: V1.06 date: 11/02/2016
Battery    BAT1: charge: 39.2 Wh 93.5% condition: 41.9/48.9 Wh (86%)
CPU:       Dual core Intel Core i3-6006U (-MT-MCP-) cache: 3072 KB
           clock speeds: max: 2000 MHz 1: 2000 MHz 2: 2000 MHz 3: 2000 MHz
           4: 2000 MHz
Graphics:  Card: Intel Skylake GT2 [HD Graphics 520]
           Display Server: x11 (X.Org 1.20.8 ) driver: i915
           Resolution: 1366x768@59.97hz
           OpenGL: renderer: Mesa DRI Intel HD Graphics 520 (SKL GT2)
           version: 4.6 Mesa 20.0.8
Audio:     Card Intel Sunrise Point-LP HD Audio driver: snd_hda_intel
           Sound: Advanced Linux Sound Architecture v: k5.4.0-80-generic
Network:   Card-1: Realtek RTL8111/8168/8411 PCIE Gigabit Ethernet Controller
           driver: r8169
           IF: enp1s0 state: down mac: fc:45:96:9d:73:6b
           Card-2: Intel Dual Band Wireless-AC 3168NGW [Stone Peak]
           driver: iwlwifi
           IF: wlp2s0 state: up mac: 30:e3:7a:f7:bc:aa
```

```
Drives:    HDD Total Size: 500.1GB (24.7% used)
           ID-1: /dev/sda model: WDC_WD5000LPCX size: 500.1GB
Partition: ID-1: / size: 30G used: 15G (51%) fs: ext4 dev: /dev/sda5
           ID-2: /home size: 170G used: 41G (25%) fs: ext4 dev: /dev/sda4
           ID-3: swap-1 size: 68.15GB used: 2.19GB (3%)
           fs: swap dev: /dev/sda6
RAID:      No RAID devices: /proc/mdstat, md_mod kernel module present
Sensors:   System Temperatures: cpu: 36.0C mobo: N/A
           Fan Speeds (in rpm): cpu: N/A
Info:      Processes: 297 Uptime: 14:02 Memory: 2859.6/3804.4MB
           Client: Shell (bash) inxi: 2.3.56
```

## 3. RUNNING TEST

To test the two servers, two files big.txt(10Kb) and small.txt (1Kb) were generated. For example `base64 /dev/urandom | head -c 10000 > big.txt`

### 3.1 Testing with ab -n 10000 -c 10 http://local8host:port/filename

To test the servers, Apache Benchmark tool was used. For example, testing the threaded server with the tool against the big.txt file, we execute the threaded server in one terminal and in another, we run `ab -n 10000 -c 20 http://127.0.0.1:7799/big.txt` to get statistics for the table beneath. For example

```
yikwenmein@yikwenmein-Aspire-ES1-572:~$  ab -n 10000 -c 10 http://127.0.0.1:7799/big.txt
This is ApacheBench, Version 2.3 <$Revision: 1807734 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking 127.0.0.1 (be patient)
Completed 1000 requests
Completed 2000 requests
Completed 3000 requests
Completed 4000 requests
Completed 5000 requests
Completed 6000 requests
Completed 7000 requests
Completed 8000 requests
Completed 9000 requests
Completed 10000 requests
Finished 10000 requests


Server Software:        Threaded
Server Hostname:        127.0.0.1
Server Port:            7799

Document Path:          /big.txt
Document Length:        10000 bytes

Concurrency Level:      10
Time taken for tests:   2.320 seconds
Complete requests:      10000
Failed requests:        0
Total transferred:      101270000 bytes
HTML transferred:       100000000 bytes
Requests per second:    4310.87 [#/sec] (mean)
Time per request:       2.320 [ms] (mean)
Time per request:       0.232 [ms] (mean, across all concurrent requests)
Transfer rate:          42633.01 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    0   0.1      0       5
Processing:     0    2   0.6      2      11
```

```
Waiting:        0    2   0.6      2       11
Total:          1    2   0.6      2       11

Percentage of the requests served within a certain time (ms)
  50%      2
  66%      2
  75%      2
  80%      2
  90%      3
  95%      3
  98%      4
  99%      5
 100%     11 (longest request)
```

### 3.1.1 Results from `ab -n 10000 -c 10 http://local8host:port/filename`

| | Threaded | | | | | | | Forked | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **Mean[ms]** | | **Statistics[ms]** | | | | | **Mean[ms]** | | **Statistics[ms]** | | | | |
| | *mean1* | *mean2* | *min* | *mean* | *std* | *median* | *max* | *mean1* | *mean2* | *min* | *mean* | *std* | *median* | *max* |
| **Small** | 2.146 | 0.215 | 1 | 2 | 0.4 | 2 | 7 | 2.341 | 0.234 | 0 | 2 | 1.5 | 2 | 21 |
| **Big** | 2.312 | 0.231 | 1 | 2 | 0.5 | 2 | 11 | 2.460 | 0.246 | 0 | 2 | 1.5 | 2 | 18 |

***Table 1***: Results you gotten for the first set of experiments (small,big vs threaded, forked)

***Table Key:***
**mean1**: Time per request: [ms] (mean)
**mean2**: Time per request:[ms] (mean, across all concurrent requests).

The table shows latency metrics(Time per request) and aggregated connection times total.
For the latency metric(**Time per request**), ab provides two variations on this metric, and both depend on the number of responses that ab has finished processing (**done**), as well as the value of the metric **time taken for tests (timetaken)**. Both multiply their results by 1,000 to get a number in milliseconds.

The first **`Time per request`** metric doesn't take the concurrency value into account:

```
mean1 = timetaken * 1000 / done
```

The second version of **`Time per request`** accounts for the number of concurrent connections the user has configured **ab** to make, using the **`-c`** option (**concurrency**):

```
mean2 = concurrency * timetaken * 1000 / done
```

This metric(Time per request) gives a rough indicator of web server performance under specific levels of load. Tracking these values over time can help you assess your optimization efforts, and unusually high or low values can point to CPU saturation on the web server, code changes, or other events that you’ll want to investigate in more detail.

For the aggregated connection times total(the time elapsed from the moment ab attempts to make the connection to the time the connection closes), ab aggregates the data into min,mean,sd(standard deviation),median, and max of each stage, all in milliseconds.

The values( mean and statistics) are not stable with different runs of the threaded and forked servers with the big and small files. Values fluctuate but the general trend is that the threaded server performs slightly better than the forked server.

#### 3.1.2 Discussion of Results for the first set of experiments (small,big vs threaded, forked)

The values( mean and statistics) are not stable with different runs of the threaded and forked servers with the big and small files. Values fluctuate but the general trend is that the threaded server performs slightly better than the forked server.

### 3.2 Experimenting Servers with dcollect.sh script

At this point, we test the servers using a script file dcollect.sh which basically iterates over the concurrency levels(1-30) and the number of repetitions(1-32), calculates and plots average and standard deviation value for requests/s against concurrency values.

### 3.2.1 Results for the experimental setup with dcollect.sh and dcollect.p(concurrency level)

Below are the statistics figures for the forked and threaded servers respectively.

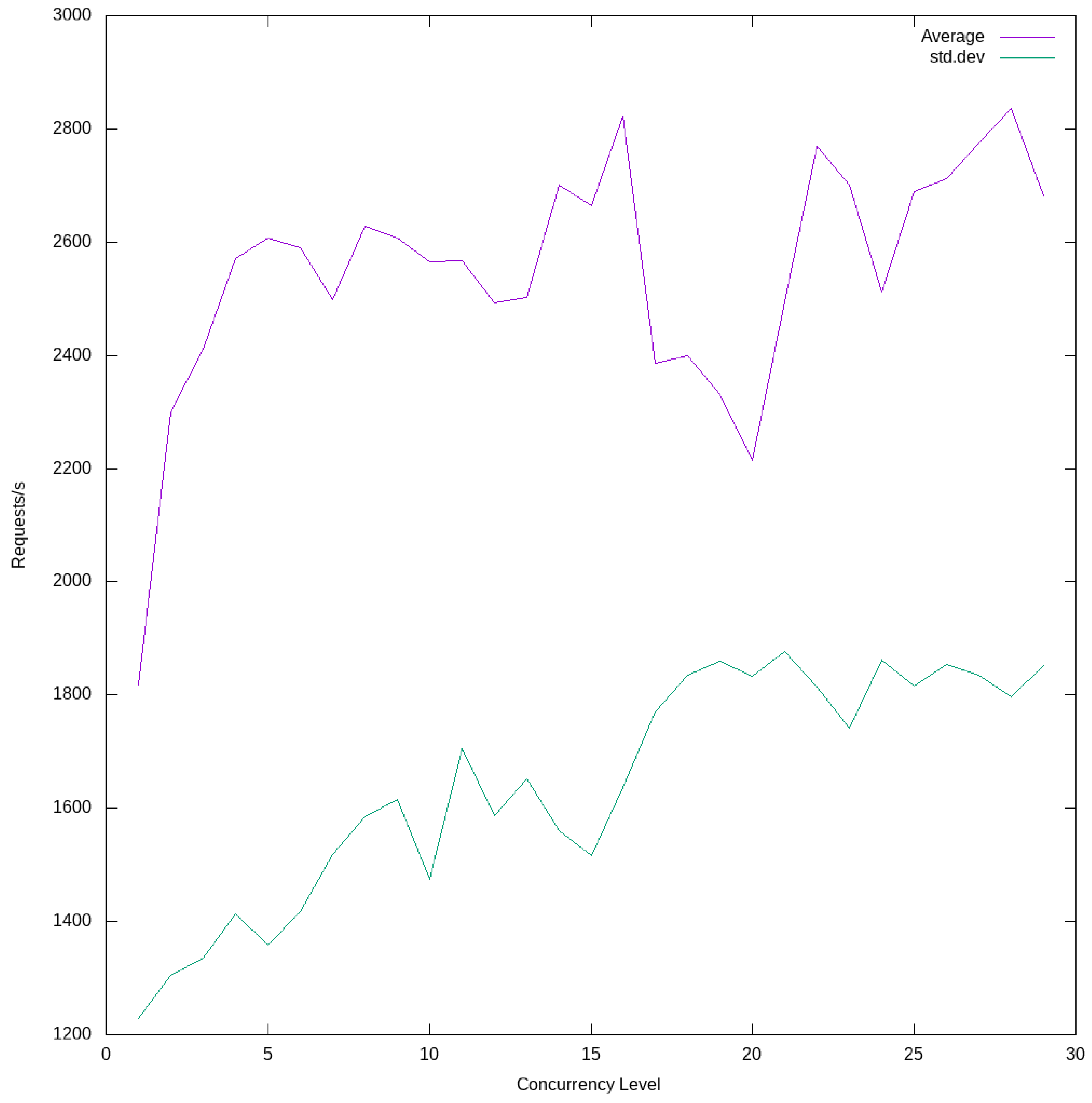


*Figure 1. Request(Average and standard deviation) vs Concurrency Level for the forked web server.*

It's log file is statistics1.log(see submission folder)

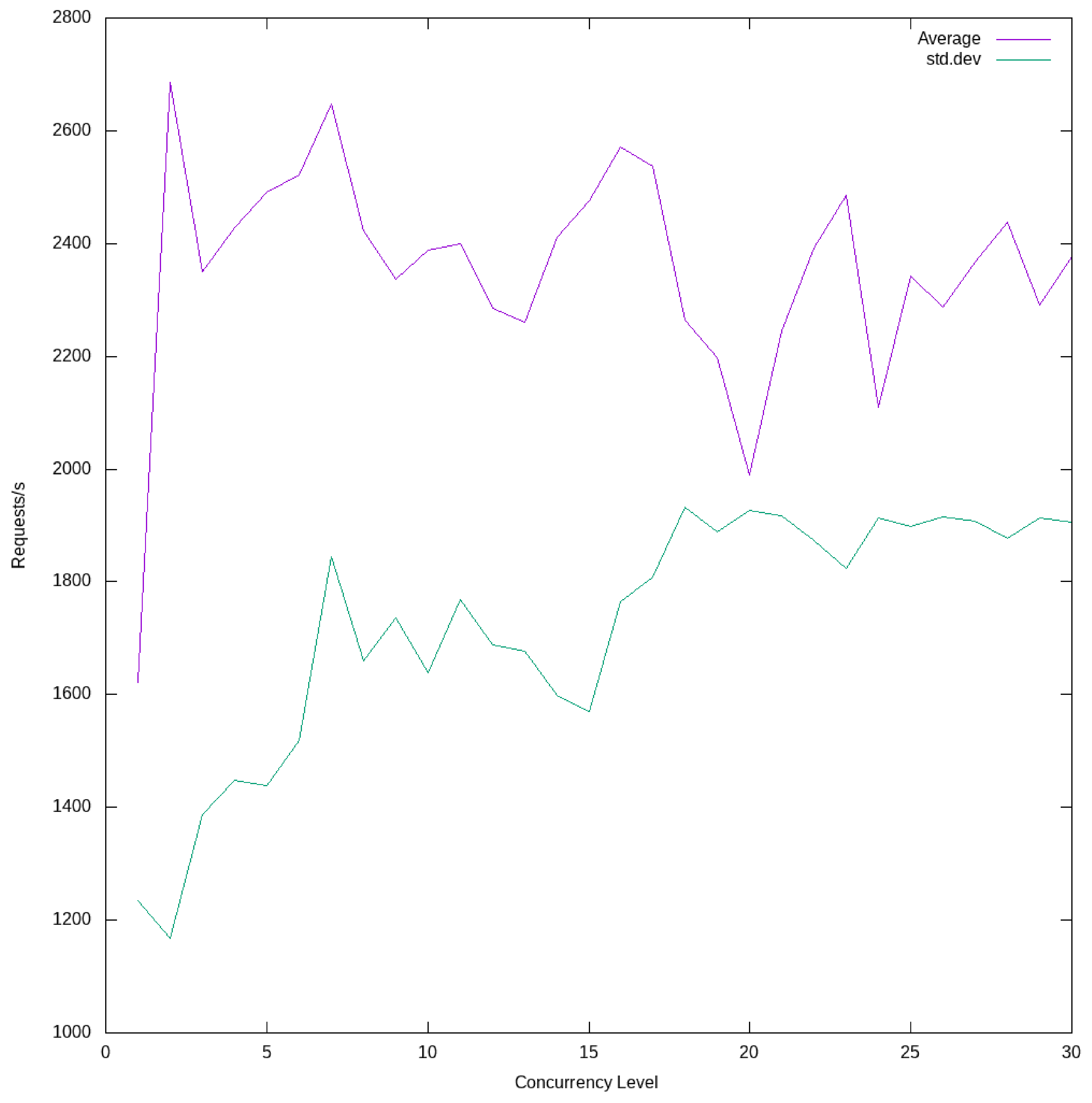


*Figure 2:* Request(Average and standard deviation) vs Concurrency level for the threaded web server.

It's log file is statistics2.log(see submission folder)

**3.2.1 Discussion of plots for the two servers**

The figures of request(Average and standard deviation) vs Concurrency level for both threaded and forked web servers show similar trends. At small levels of concurrency(concurrency less than 5), average request per second increases with Concurrency but starts to fluctuate at higher concurrency.
Checking further, we can see that the forking server slightly performs better than the threaded server at higher concurrency. This can be attributed to CPU saturation as concurrency increases. Threads are more efficient on multi processor(CPU) systems while processes(forking) is faster than threading on single cpu as there are no locking over-heads or context switching.

The system used for testing here is a dual core processor system

```
CPU:       Dual core Intel Core i3-6006U (-MT-MCP-) cache: 3072 KB
           clock speeds: max: 2000 MHz 1: 2000 MHz 2: 2000 MHz 3: 2000 MHz
           4: 2000 MHz
```

So at low concurrency, threading is performing better than forking(the system is dual core and at low concurrency, threads are exploiting the dual core processor, taking advantage of it and performing better than forking ). However,at higher concurrency,there's CPU saturation(system becomes more like a single CPU system) causing deadlocks and race conditions rendering threads less effective as compared to processes(forking).
Another practical observation is that running the script(dcollect.sh) against threaded server, the system always gets overloaded  taking lots a very long time to complete all iterations and sometimes it becomes frozen  requiring a complete system reboot for the system to become responsive but running the script(dcollect.sh) against fork server, everything goes smoothly and all iterations are completed on a reasonable amount of time.

## 4. CONCLUSION

Threads are most effective on multi-processor or multi-core systems when the system has enough available resources but are less effective at system saturation (deadlock). But when a system is limited in resources(CPU), forking is a better approach.

## REFERENCES

[1] https://www.datadoghq.com/blog/apachebench/
[2] http://www.geekride.com/fork-forking-vs-threading-thread-linux-kernel/