Benchmarking

CS553 Assignment #1

Maulik Patel A20334253

The assignment is about benchmarking different parts of computer. I have benchmarked three parts of the computer as follows

1. Network Performance.
2. Disk Performance.
3. CPU Performance (Operations executed per second).

All the programs are designed in following structure -:

- The program when executed will ask to enter the number of threads and based upon the input it will run the program with given threads.
- The various parameters like latency, throughput, IOPS, FLOPS are calculated and displayed on console window at the end of execution.
- All the programs are in Java and I have used the inbuilt Java timer for calculating the time taken for execution.
- In network when the program is executed it will ask for the number of threads to run and will give the throughput and latency of varying size block data. In disk performance it will ask for entering the number of threads and will give the latency and throughput of sequential write, sequential read, random write, random read. In CPU there are three programs giving IOPS, FLOPS and giving 600 samples for running the program for 10 minutes.
- All the benchmarking was done by running programs on Amazon EC2 micro instance and for networking the benchmarking was done by connecting two different micro instance of Amazon EC2.

For performing the benchmarking of the different part of computer the program is designed as follows-:

1. Networking –:
    - The network part is divided into two parts TCP and UDP. Each UDP and TCP have server and client program which is written in java.
    - The program is designed such that it supports the multithreading and based on the user input how many threads he want to create the server opens the server socket on the different port and continuously listen for the incoming client request on that port .

- The client side program will ask for IP address and port number of the server we want to connect and will create a socket connection in TCP and in UDP it will create a datagram Packet to be send to the server.
- For sending the data the string of specified size is created at client side and send wrapping as object in TCP and datagram packet in UDP
- Both TCP and UDP client program will send the data of size 1 Byte, 1 KiloByte and 64 KiloByte to the server in TCP as in form of object and in UDP inform of packet and the server send back the same data to the client giving us the round trip time calculated at the client side.
- Throughput in Mbps and latency in milliseconds are calculated and displayed in console.
- For two thread the client will start two different ports and will send data into two threads. So we will get two different values for throughput and latency for each thread. In that case we have to take the biggest value of throughput and smallest value of latency.

2. Disk Performance -:
    - The program is designed to perform the four function to measure the disk performance sequential write, sequential read, random write and random write.
    - The program is written in java with multithreading support and the threads are started in the constructor based on the user input.
    - Program is written in Java and it consists of four methods and they are called in the run function which is executed each time we start the new thread.
    - Throughput is calculated in MegaByte per second and latency is calculated in milliseconds.
    - For two threads it will give the throughput and latency of two different threads and we have to take the average of the value.
3. CPU Performance -:
    - The program is designed to do the various operations like multiplication and addition supporting multithreading. The program displays the values of instruction executed per second in Giga-FLOPS and Giga-IOPS. Two different programs are written to calculate the Integer Operations Per Second(IOPS) and Floating Point Operations per second(FLOPS). The inbuilt timer in java is used for calculating the time. The duration for calculating the operation is calculated in the main thread.
    - The user has to pass the number of threads to run and based on that it will create the array of the threads for executing the operations.
    - A separate java program is designed for getting the number of operations performed from 4 threads and then adding it up for and getting sample for every second taking 600 samples and plotting the graph for it.

Improvement and Extension to the program-:

The further improvements and modification that we can do in our program can be

Network Performance -:

- We can increase the concurrency level up to 4 and 8 threads to compare the throughput and latency.
- We can increase the buffer size from 64KB to 1MB and 100MB and measure various performance parameters for better comparison.

Disk Performance -:

- We can increase the concurrency from 2 threads to 4 and 8 threads.
- In my program I am writing with the buffer size of 1 Byte, 1 KB and 1 MB we can increase the buffer size to 10MB and 100MB and compare the performance.
- In the disk there is possibility of having the bad sectors resulting in increased latency so we have to take in account for that too.

CPU Performance -:

- The CPU program is doing the simple arithmetic operations like addition and multiplication without much dependencies. We can increase the dependencies and make the operations more complex and calculating the GIOPS and GFLOPS values.
- The benchmarking is performed only on the ec2 micro instance so we can perform it on various other instance and compare the results to calculate the variations based on the hardware.

Performance Evaluation

. All the programs are executed for three times and the average values is taken as the results. First the environment setup of the system on which the benchmarking is performed is described. The detailed description of each part is described and finally the average data values are described in the table and then plotted in form of graph for better visualization.

System Environment -:

All the experiments are performed on Amazon EC2 t2 micro instance in Ubuntu AMI. It has 1 core with 1GB RAM and Intel Xeon processor with 2.5 GHz speed.

Overview of the Experiments -:

- Network Benchmarking is done for both TCP and UDP for varying concurrency of up to two threads. The experiments are performed using two AWS t2 micro instance and the latency and throughput are calculated for threads(1,2) for block size of 1 Byte, 1 KB and 64 KB.
- Disk Benchmarking is for four different parameters of sequential write, sequential read, random write and random read with varying concurrency of 1 and 2 threads. The average throughput and latency value is calculated for each parameter.
- CPU Benchmarking is done to calculate the giga-IOPS and giga-FLOPS for varying concurrency of 1,2 and 4 threads. The separate program is executed to get the 600 samples for every second.

Results and Analysis -:

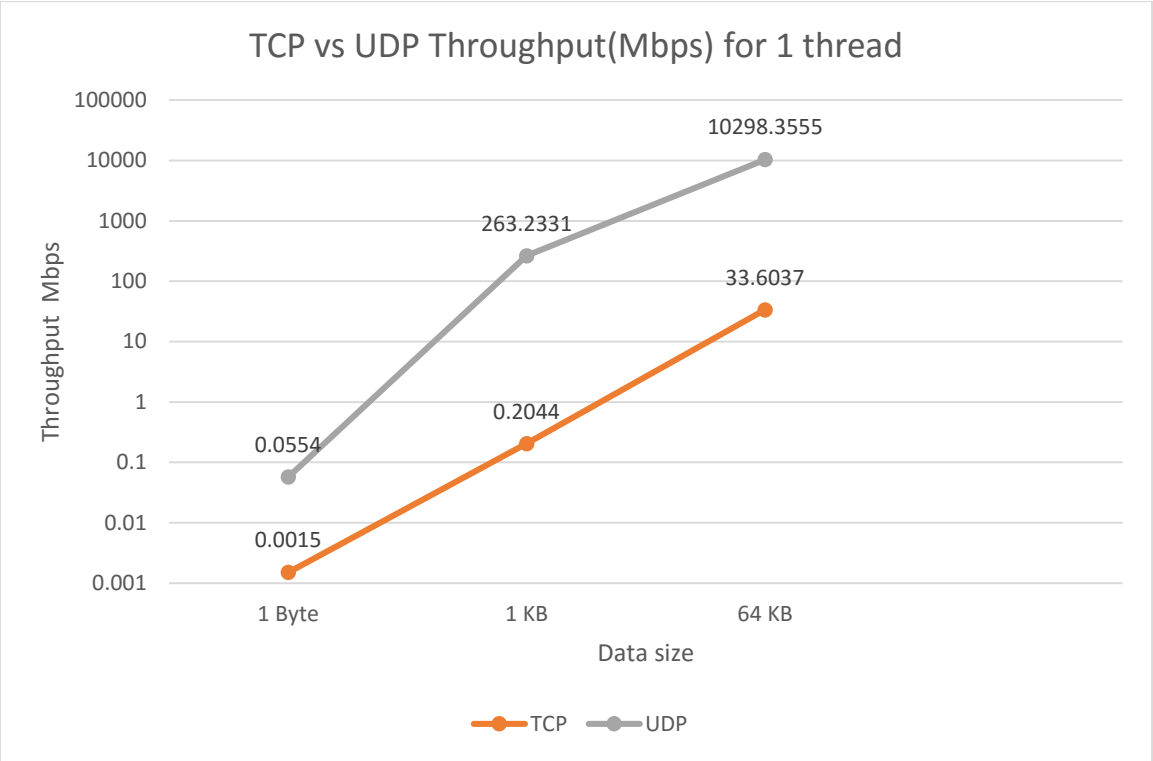
Network -:

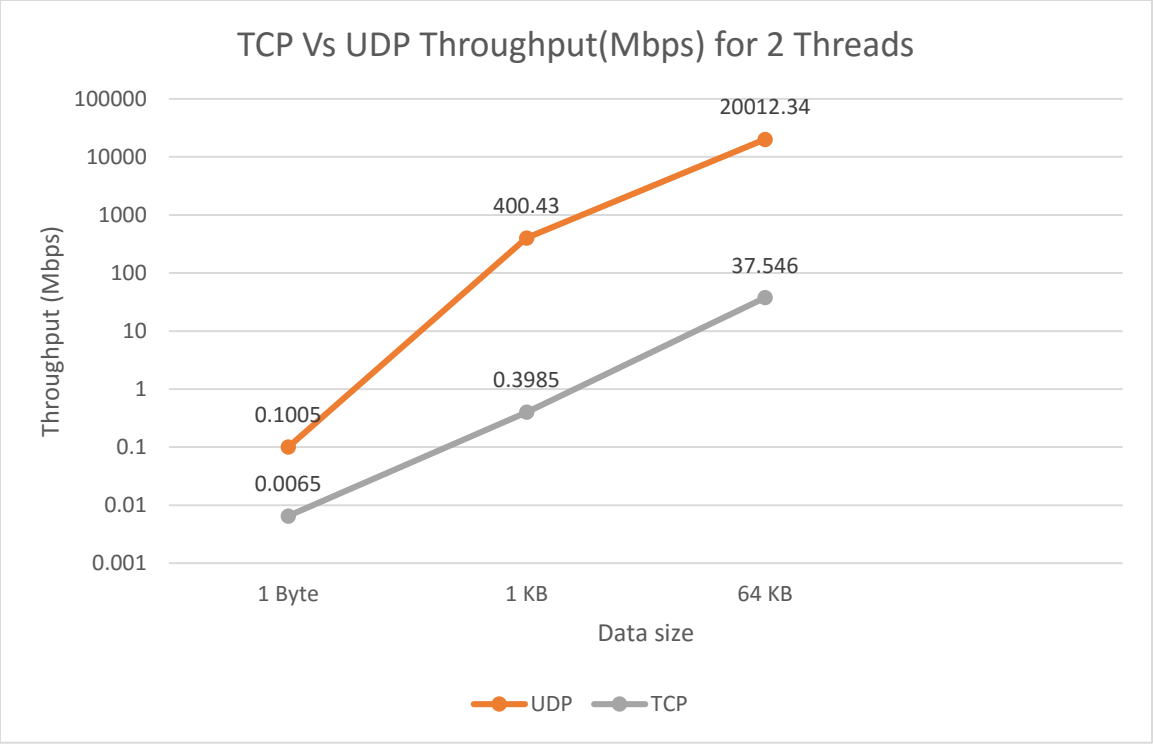
- The experiment is performed three times and then taken the average to get the consistent value.
- Two EC2 t2micro instance are used for both TCP and UDP. One instance the server is started on different ports and it continuously listen to the incoming request from the client. On the other instance client is started and is connected to the server.
- The throughput and latency values are calculated for various parameters for TCP and UDP which clearly states that the throughput of UDP is a way more when compared to TCP.
- The following table shows the different values for TCP and UDP for 1 and 2 threads.

Throughput Result -:

Size of blocks	Average Throughput TCP 1-Thread (Mbps)	Average Throughput UDP 1-Thread (Mbps)	Average Throughput TCP 2-Threads (Mbps)	Average Throughput UDP 2-Threads (Mbps)
1 B	0.0015	0.0554	0.0065	0.1005
1 KB	0.2044	263.2331	0.3985	400.43
64 KB	33.6037	10298.3555	37.546	20012.34

Size of blocks	Standard - Deviation Throughput TCP 1-Thread (Mbps)	Standard - Deviation Throughput UDP 1-Thread (Mbps)	Standard - Deviation Throughput TCP 2-Threads (Mbps)	Standard -Deviation Throughput UDP 2-Threads (Mbps)
1 B	0.0024	0.0562	0.0031	0.0103
1 KB	0.0193	30.356	0.1029	48.43
64 KB	3.965	207.56	5.673	107.59





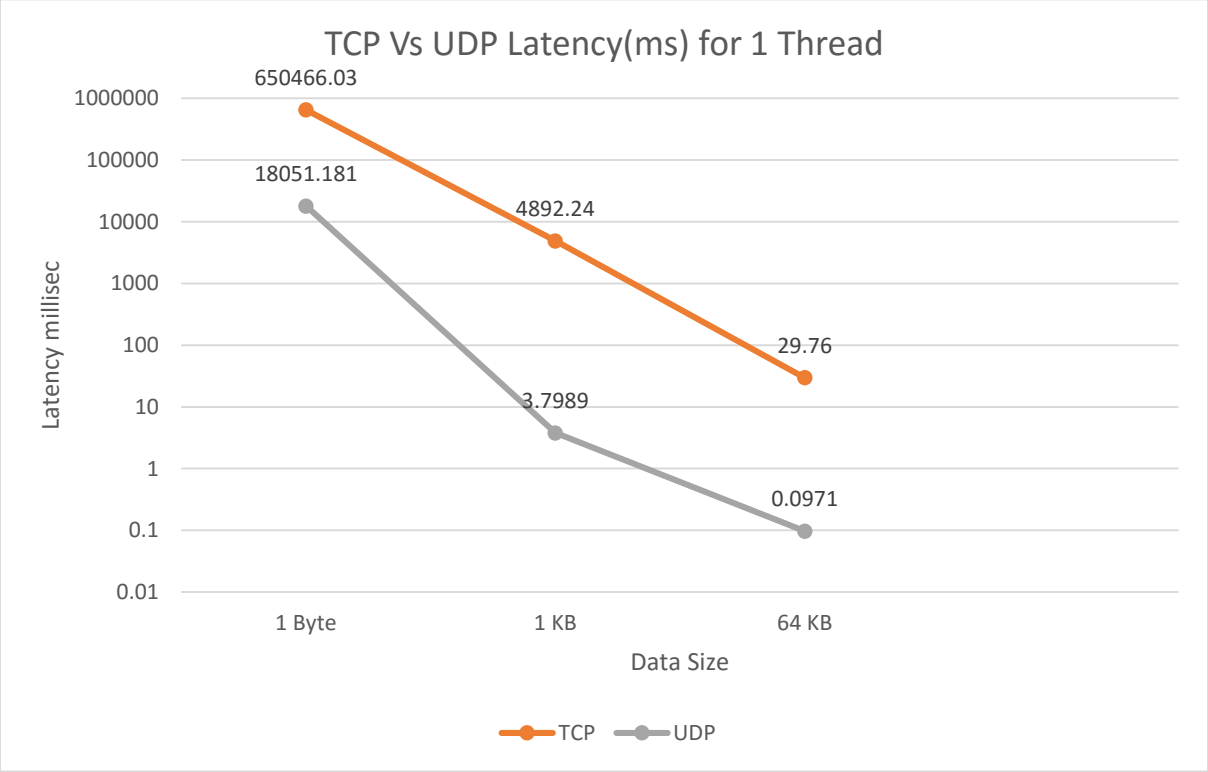
Analysis -:

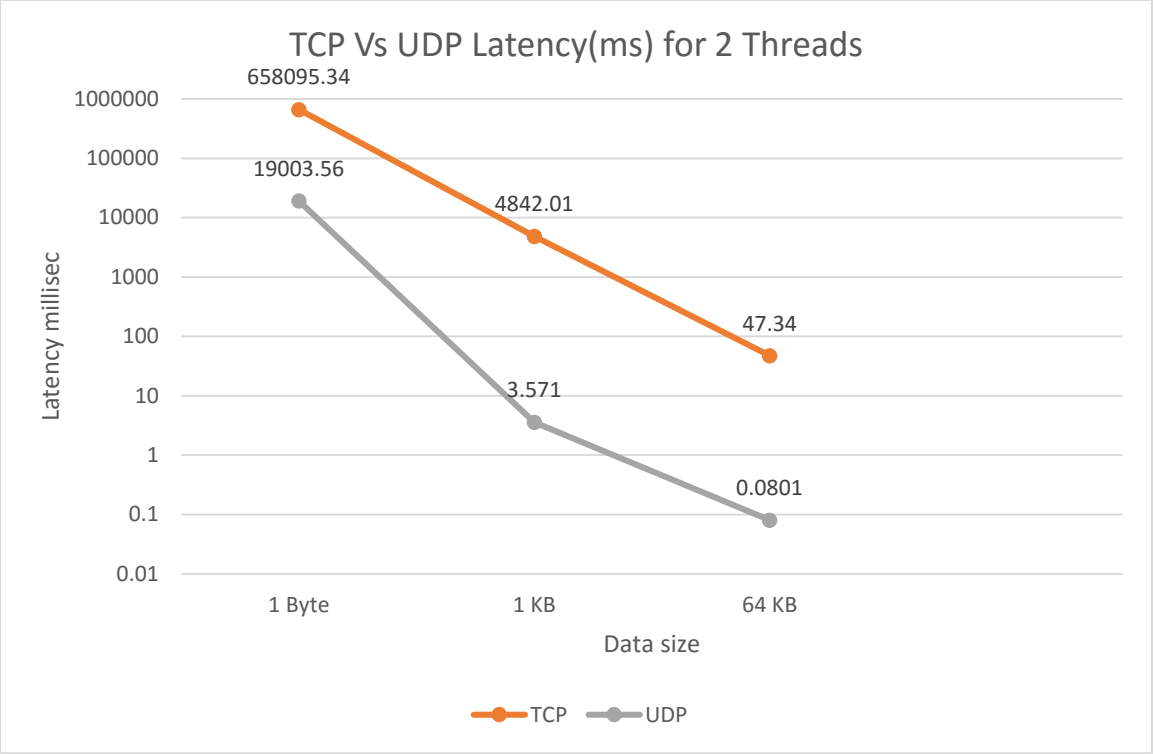
- As seen in the graph with increasing the packet size the throughput is increased with a larger values.
- Increasing the thread increases the throughput almost to double the value of throughput obtained from single thread.
- Comparing the UDP and TCP we can easily see that the throughput of UDP is having a much greater value as compared to TCP. TCP is connection oriented protocol compared to UDP which is connectionless protocol, so there will be a lot of overhead in TCP compared to UDP decreasing the throughput.

Latency Results -:

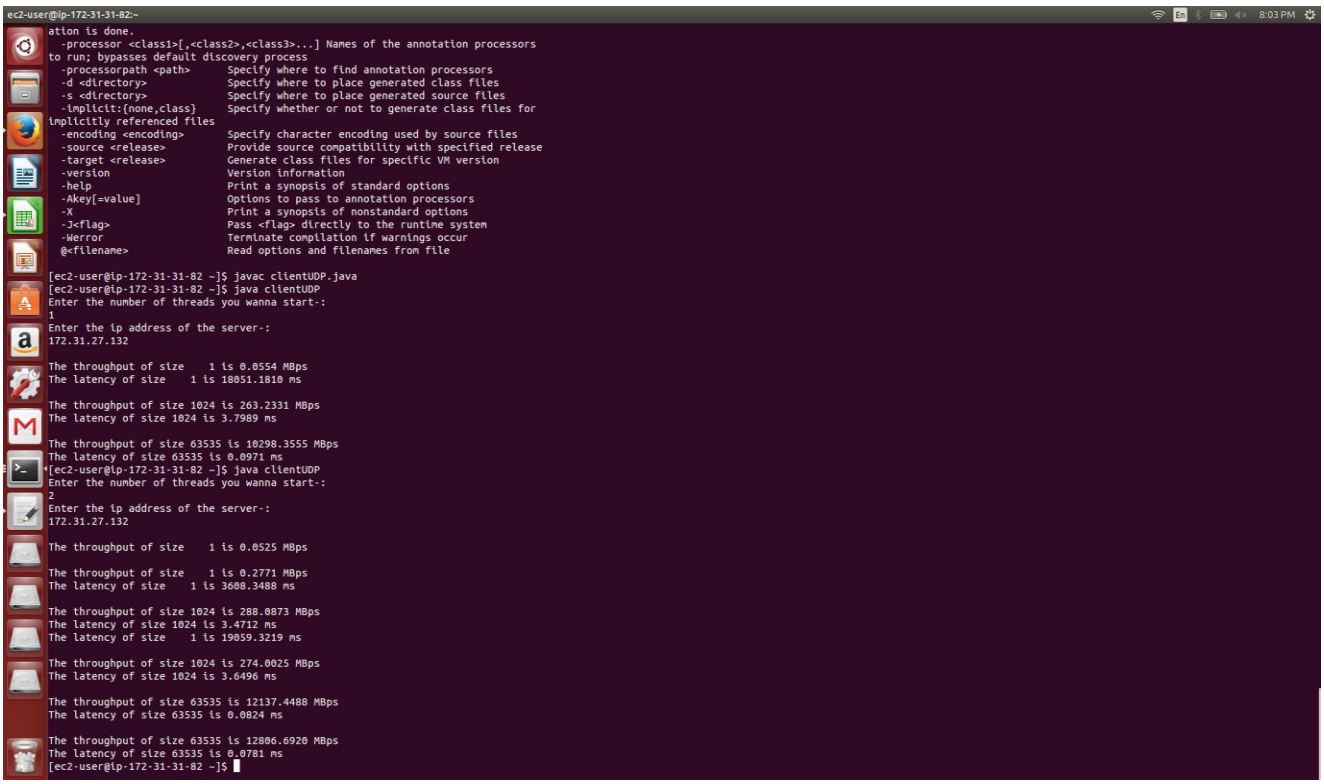
Size of blocks	**Latency TCP 1-Thread (millisecond)**	**Latency UDP 1-Thread (millisecond)**	**Latency TCP 2-Threads (millisecond)**	**Latency UDP 2-Threads (millisecond)**
1 B	650466.03	18051.181	658095.34	19003.56
1 KB	4892.24	3.7989	4842.01	3.571
64 KB	29.76	0.0971	47.34	0.0801

Size of blocks	Standard-Deviation Latency TCP 1-Thread (millisecond)	Standard-Deviation Latency UDP 1-Thread (millisecond)	Standard-Deviation Latency TCP 2-Threads (millisecond)	Standard-Deviation Latency UDP 2-Threads (millisecond)
1 B	90653.67	349.71	98653.56	2561.76
1 KB	308.45	1.563	342.78	1.31
64 KB	2.47	0.1045	3.71	0.134




TCP Vs UDP Latency(ms) for 2 Threads
658095.34
19003.56
4842.01
47.34
3.571
0.0801
1000000
100000
10000
1000
100
10
1
0.1
0.01
Latency millisec
1 Byte
1 KB
64 KB
Data size
TCP
UDP

UDP output -:


ec2-user@ip-172-31-31-82:~
8:03 PM
ation is done.
-processor <class1>[,<class2>,<class3>...] Names of the annotation processors
to run; bypasses default discovery process
-processorpath <path> Specify where to find annotation processors
-d <directory> Specify where to place generated class files
-s <directory> Specify where to place generated source files
-implicit:{none,class} Specify whether or not to generate class files for
implicitly referenced files
-encoding <encoding> Specify character encoding used by source files
-source <release> Provide source compatibility with specified release
-target <release> Generate class files for specific VM version
-version Version information
-help Print a synopsis of standard options
-Akey[=value] Options to pass to annotation processors
-X Print a synopsis of nonstandard options
-J<flag> Pass <flag> directly to the runtime system
-Werror Terminate compilation if warnings occur
@<filename> Read options and filenames from file
[ec2-user@ip-172-31-31-82 ~]$ javac clientUDP.java
[ec2-user@ip-172-31-31-82 ~]$ java clientUDP
Enter the number of threads you wanna start-:
1
Enter the ip address of the server-:
172.31.27.132
The throughput of size 1 is 0.0554 MBps
The latency of size 1 is 18051.1810 ms
The throughput of size 1024 is 263.2331 MBps
The latency of size 1024 is 3.7989 ms
The throughput of size 63535 is 10298.3555 MBps
The latency of size 63535 is 0.0971 ms
[ec2-user@ip-172-31-31-82 ~]$ java clientUDP
Enter the number of threads you wanna start-:
2
Enter the ip address of the server-:
172.31.27.132
The throughput of size 1 is 0.0525 MBps
The throughput of size 1 is 0.2771 MBps
The latency of size 1 is 3608.3488 ms
The throughput of size 1024 is 288.0873 MBps
The latency of size 1024 is 3.4712 ms
The latency of size 1 is 19059.3219 ms
The throughput of size 1024 is 274.0025 MBps
The latency of size 1024 is 3.6496 ms
The throughput of size 63535 is 12137.4488 MBps
The latency of size 63535 is 0.0824 ms
The throughput of size 63535 is 12806.6920 MBps
The latency of size 63535 is 0.0781 ms
[ec2-user@ip-172-31-31-82 ~]$

Analysis -:

- Analyzing the graph we can see that the latency remains almost same even if we increase the threads.
- Comparing TCP and UDP we can easily see that latency of TCP is having a bigger value compared to UDP because of the overhead in TCP protocol for establishing the secure connection for data transfer.
- Also as increase the data size the latency decreases at a greater value. The bigger the data size the less is the latency.

Extra Credit -:

Performing benchmarking through ipref in AWS micro instance

Server Side

install iperf package in the system
Run the server
iperf3 -s

```
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
Accepted connection from ::1, port 56391
[  5] local ::1 port 5201 connected to ::1 port 56392
[ ID] Interval           Transfer     Bandwidth
[  5]   0.00-1.00   sec  4.89 GBytes  42.0 Gbits/sec
[  5]   1.00-2.00   sec  5.07 GBytes  43.5 Gbits/sec
[  5]   2.00-3.00   sec  5.04 GBytes  43.3 Gbits/sec
[  5]   3.00-4.00   sec  4.84 GBytes  41.6 Gbits/sec
[  5]   4.00-5.00   sec  4.89 GBytes  42.0 Gbits/sec
[  5]   4.00-5.00   sec  4.89 GBytes  42.0 Gbits/sec
- - - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth
[  5]   0.00-5.00   sec  0.00 Bytes  0.00 bits/sec                  sender
[  5]   0.00-5.00   sec  29.0 GBytes  49.8 Gbits/sec                  receiver
iperf3: the client has terminated
-----------------------------------------------------------
Server listening on 5201
-----------------------------------------------------------
Accepted connection from 172.31.72.5, port 56061
[  5] local 172.31.72.5 port 5201 connected to 172.31.72.5 port 56062
[ ID] Interval           Transfer     Bandwidth
[  5]   0.00-1.00   sec  4.80 GBytes  41.3 Gbits/sec
[  5]   1.00-2.00   sec  4.95 GBytes  42.5 Gbits/sec
[  5]   2.00-3.00   sec  4.98 GBytes  42.7 Gbits/sec
[  5]   3.00-4.00   sec  4.99 GBytes  42.8 Gbits/sec
[  5]   4.00-5.00   sec  4.84 GBytes  41.5 Gbits/sec
[  5]   5.00-6.00   sec  4.84 GBytes  41.6 Gbits/sec
[  5]   6.00-7.00   sec  4.87 GBytes  41.8 Gbits/sec
[  5]   7.00-8.00   sec  4.94 GBytes  42.4 Gbits/sec
[  5]   8.00-9.00   sec  4.97 GBytes  42.6 Gbits/sec
```

```
[  5]   9.00-10.00  sec  4.93 GBytes  42.4 Gbits/sec
[  5]  10.00-10.04  sec   134 MBytes  29.9 Gbits/sec
- - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth       Retr
[  5]   0.00-10.04  sec  49.2 GBytes  42.1 Gbits/sec    0             sender
[  5]   0.00-10.04  sec  49.2 GBytes  42.1 Gbits/sec                  receive
```

Client Side -:

```
iperf3 -c 172.31.72.5
Connecting to host 172.31.72.5, port 5201
[  4] local 172.31.72.5 port 56062 connected to 172.31.72.5 port 5201
[ ID] Interval           Transfer     Bandwidth       Retr  Cwnd
[  4]   0.00-1.00   sec  4.94 GBytes  42.5 Gbits/sec    0   3.18 MBytes
[  4]   1.00-2.00   sec  4.95 GBytes  42.5 Gbits/sec    0   3.18 MBytes
[  4]   2.00-3.00   sec  4.98 GBytes  42.7 Gbits/sec    0   3.18 MBytes
[  4]   3.00-4.00   sec  4.99 GBytes  42.9 Gbits/sec    0   3.18 MBytes
[  4]   4.00-5.00   sec  4.83 GBytes  41.5 Gbits/sec    0   3.18 MBytes
[  4]   5.00-6.00   sec  4.84 GBytes  41.6 Gbits/sec    0   3.18 MBytes
[  4]   6.00-7.00   sec  4.87 GBytes  41.8 Gbits/sec    0   3.18 MBytes
[  4]   7.00-8.00   sec  4.94 GBytes  42.4 Gbits/sec    0   3.18 MBytes
[  4]   8.00-9.00   sec  4.96 GBytes  42.6 Gbits/sec    0   3.18 MBytes
[  4]   9.00-10.00  sec  4.92 GBytes  42.3 Gbits/sec    0   3.18 MBytes
- - - - - - - - - - - - - - - - - - - - - - - -
[ ID] Interval           Transfer     Bandwidth       Retr
[  4]   0.00-10.00  sec  49.2 GBytes  42.3 Gbits/sec    0             sender
[  4]   0.00-10.00  sec  49.2 GBytes  42.3 Gbits/sec                  receiver
```

The maximum throughput we are getting is 42.2 Gbits/s

The efficiency when compared to the ipref is given as

Efficiency = (Calculated maximum throughput*100/maximum troughput in Ipref)

= (19.54*100)/42.2

Efficiency = 46.30

Disk Performance -:

- The experiment was performed three times and its average was taken to make the result more consistent and the graphs are plotted on that results.
- All the experiments were performed on Amazon t2 micro instance.

- The results will give throughput and latency as the output for all the four functions sequential write, sequential read, random write and random read for 1 and 2 threads.
- The data is plotted in the table and for better visualization it is plotted in graphs.

For Sequential write:

Block size	**Average(Throughput-Sequential Write)**	**Standard Deviation(Throughput-Sequential Write)**	**Average(Latency-Sequential write)**	**Standard Deviation((Latency-Sequential Write)**
1 Byte(1 thread)	42.46	21.3	23.55	10.13
1 KB (1 thread)	349.46	103.5	2.86	1.42
1 MB(1 Thread)	229.72	109.7	4.35	3.42
1 Byte(2 Thread)	36.67	14.6	27.63	18.71
1 KB (2 Thread)	439.29	102.5	2.38	1.12
1 MB(2 Thread)	367.54	131.7	3.15	2.17

For Sequential Read:

Number of Thread	**Average(Throughput-Sequential Read)**	**Standard Deviation(Throughput-Sequential Read)**	**Average(Latency-Sequential Read)**	**Standard Deviation((Latency-Sequential Read)**
1 Byte(1 thread)	3.18	2.17	314.79	117.3
1 KB (1 thread)	739.4	302.1	1.35	1.07
1 MB(1 Thread)	1357.52	438.98	0.74	0.53
1 Byte(2 Thread)	3.2	2.13	309.51	102.5
1 KB (2 Thread)	1396.22	432.54	0.89	0.74
1 MB(2 Thread)	2010.14	783.23	0.76	0.94

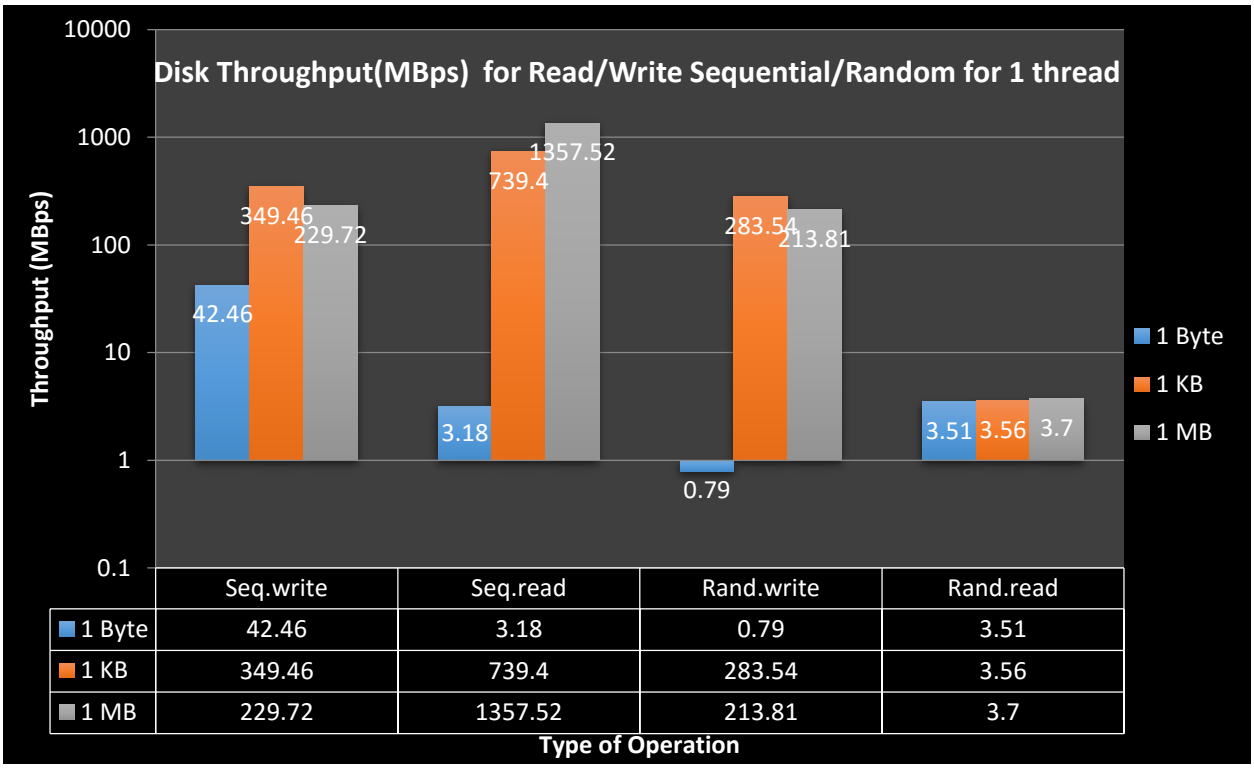
For Random write :

Number of Thread	Average(Throughput-Random write)	Standard Deviation(Throughput-Random write)	Average(Latency-Random write)	Standard Deviation((Latency-Random write)
1 Byte(1 thread)	0.79	0.93	1265.48	234.6
1 KB (1 thread)	283.54	108.42	3.53	1.76
1 MB(1 Thread)	213.81	78.56	4.68	2.35
1 Byte(2 Thread)	0.81	0.34	1238.25	308.78
1 KB (2 Thread)	273.92	103.56	3.71	1.39
1 MB (2 Thread)	256.76	106.56	5.18	2.87

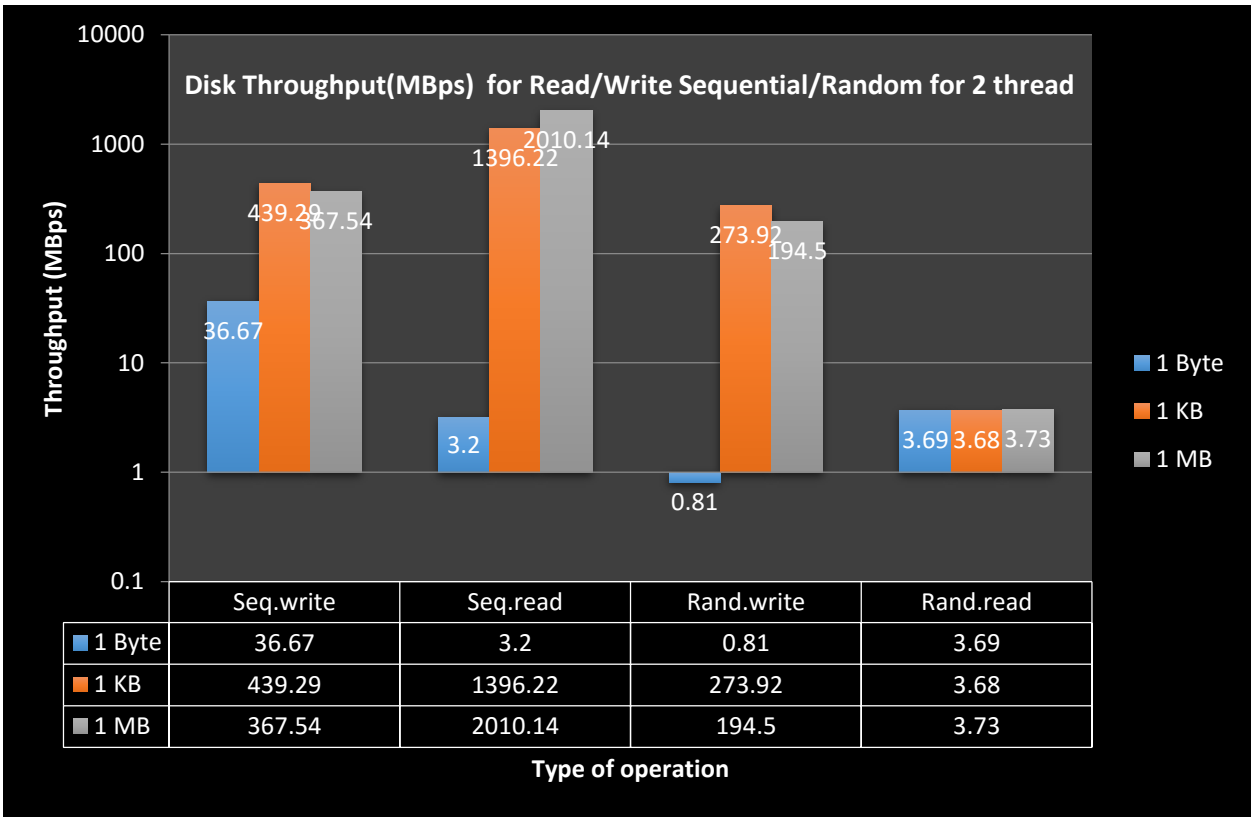
For Random Read -:

Number of Thread	Average(Throughput-Random Read)	Standard Deviation(Throughput-Random Read)	Average(Latency-Random Read)	Standard Deviation((Latency-Random Read)
1 Byte(1 thread)	3.51	0.87	284.66	76.45
1 KB (1 thread)	3.56	0.93	280.9	67.87
1 MB(1 Thread)	3.7	1.03	270.37	79.67
1 Byte(2 Thread)	3.69	0.56	274.38	98.56
1 KB (2 Thread)	3.68	0.46	277.22	87.56
1 MB(2 Thread)	3.73	0.78	268.39	94.53

Throughput Results -:



	Seq.write	Seq.read	Rand.write	Rand.read
1 Byte	42.46	3.18	0.79	3.51
1 KB	349.46	739.4	283.54	3.56
1 MB	229.72	1357.52	213.81	3.7

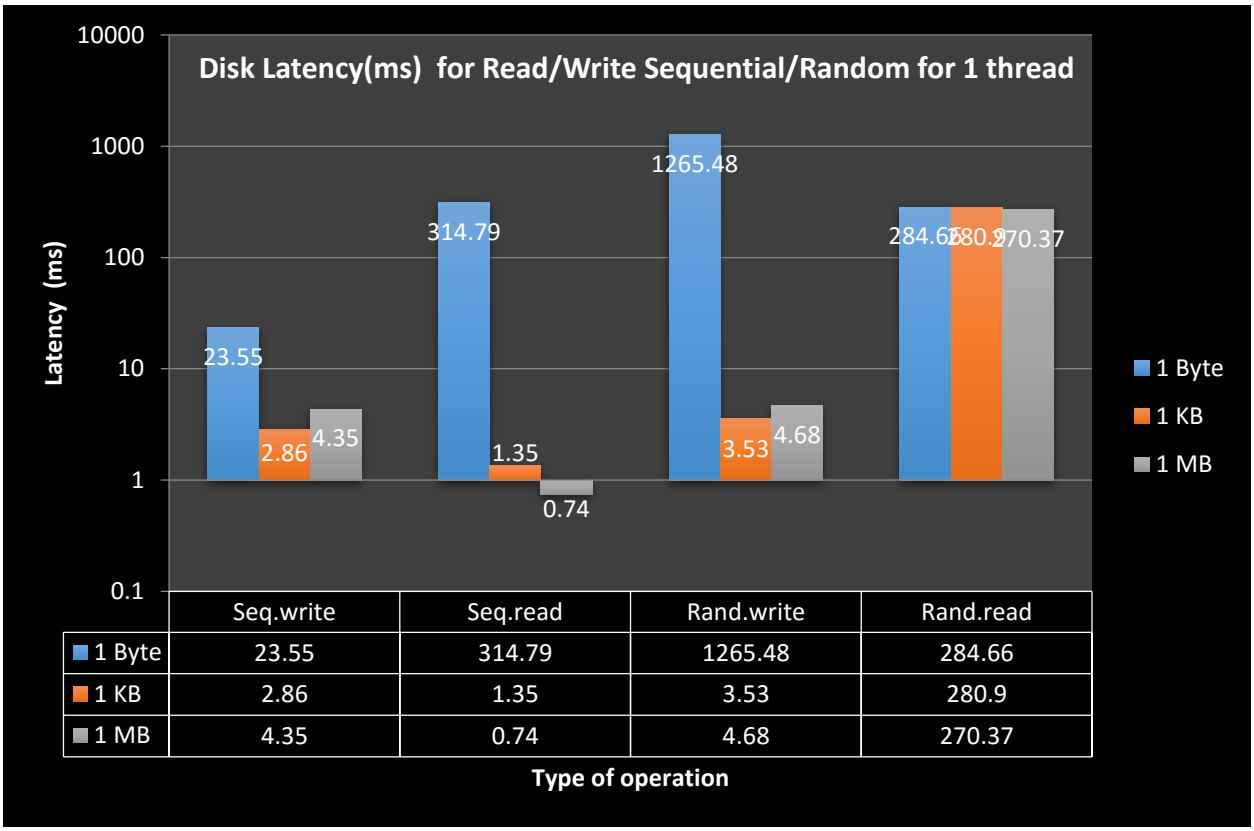


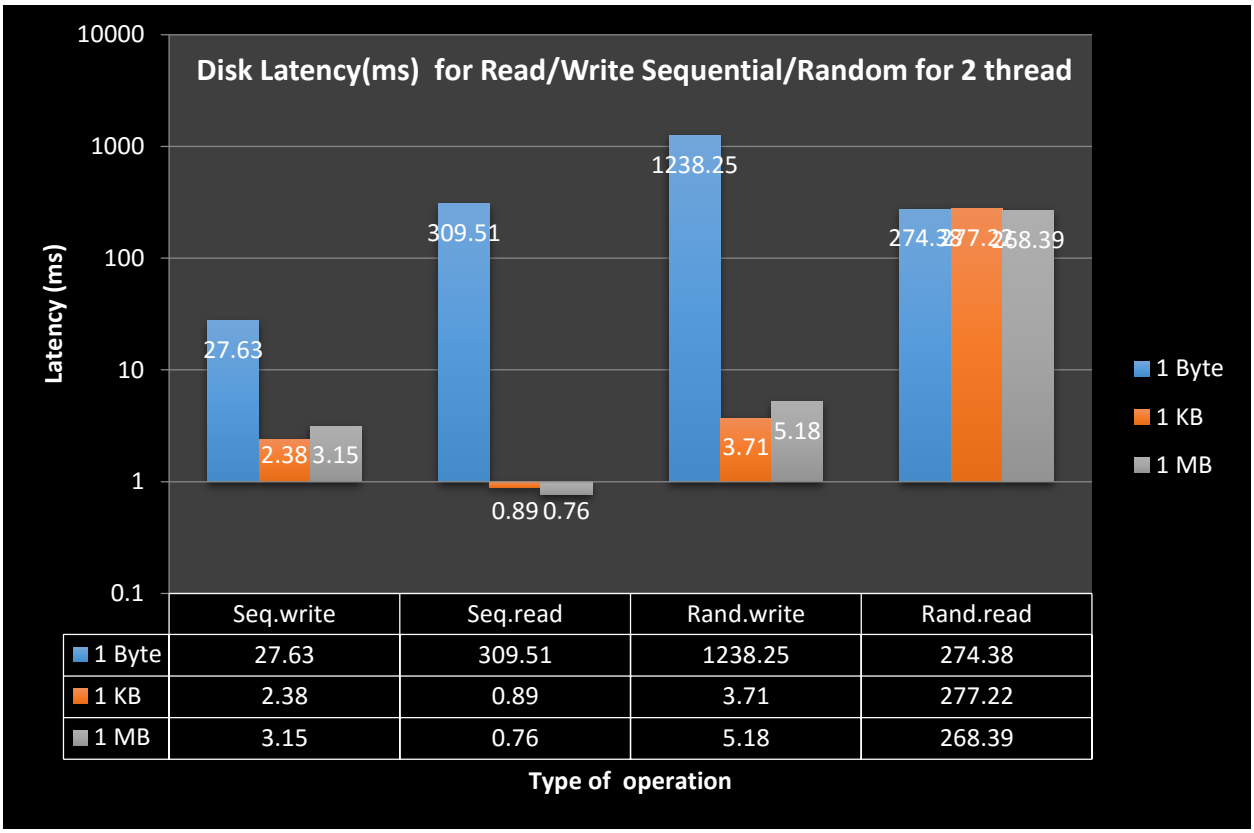
	Seq.write	Seq.read	Rand.write	Rand.read
1 Byte	36.67	3.2	0.81	3.69
1 KB	439.29	1396.22	273.92	3.68
1 MB	367.54	2010.14	194.5	3.73

Throughput Analysis -:

- Comparing the graph and the table in the graph the throughput results for the sequential write is much greater then the random write as in sequential write the data is written sequentially whereas in random first we have to place the pointer and then write the data decreasing the throughput.
- Similarly, for reading the sequential read is having much greater value compared to the random read because for random read the data is stored in form of byte array and so we have to first jump to the position from where the data is to be read and then read the data creating a lot of overhead decreasing the throughput at a greater extent.
- For random read the throughput value is very less compared to the sequential read as it has to set the pointer every time to new position and then read creating an overhead resulting in the performance.
- For two threads the value of the sequential read and write increases whereas for random read and write it is almost the same.
- It can be seen from the graph increasing the buffer size does not increase the throughput. It will increase the throughput to a certain size and then the value will go down.

Latency Results -:





Latency Analysis -:

- Comparing the graph we can see that the values of the latency decreases as we increase the buffer size. The latency for the sequential write is lower than the random write and similarly for sequential read the value is lower than the random read.
- Increasing the thread does not decrease the latency to a larger extent but a minor change is observed.

Extra Credit -:

IOZONE

```
sudo ./iozone -a -s 1024
Iozone: Performance Test of File I/O
        Version $Revision: 3.283 $
Compiled for 32 bit mode.
Build: linux

Contributors:William Norcott, Don Capps, Isom Crawford, Kirby Collins
             Al Slater, Scott Rhine, Mike Wisner, Ken Goss
             Steve Landherr, Brad Smith, Mark Kelly, Dr. Alain CYR,
             Randy Dunlap, Mark Montague, Dan Million,
             Jean-Marc Zucconi, Jeff Blomberg, Benny Halevy,
             Erik Habbinga, Kris Strecker, Walter Wong.

Run began: Fri Feb 12 22:07:36 2016
```

Auto Mode
File size set to 1024 KB
Command line used: ./iozone -a -s 1024
Output is in Kbytes/sec
Time Resolution = 0.000001 seconds.
Processor cache size set to 1024 Kbytes.
Processor cache line size set to 32 bytes.
File stride size set to 17 * record size.

KB	reclen	write	rewrite	read	reread	random read	random write	bkwd read	record rewrite	stride read	fwrite	frewrite	fread	freread
1024	4	1790324	4473789	8539920	9135494	5785308	3938463	6023528	5046840	6244111	2359447	3230284	7757577	8605043
1024	8	2375759	4992577	12475806	12637282	9660377	5785312	9481480	6826831	9488127	2844443	501960911377778	11906965	
1024	16	2652675	6280904	12656903	1314224611252747	687247810666662	832125610907392	2730667	497087511906981	12641968				
1024	32	2462180	6736016	13479939	1382483412799994	664934911770118	921557612341997	2876405	581817912641968	13473683				
1024	64	2681108	7058019	12037255	1462858014222227	812698112799994	975871115287023	3210031	620606113473683	14628580				
1024	128	2813107	7161740	13492056	1381690713298699	825806312047065	975703314628604	3047619	624390312487797	13128201				
1024	256	2774446	6731990	11911786	1149525911906981	691891411010750	846100212349292	2790190	522449011252747	12190483				
1024	512	2738563	6695627	12195013	1234071612190483	781678911505619	799818511632857	2687664	624389811636366	12337358				
1024	1024	2540610	6605091	11908545	1265887012337340	660645011252747	758496311649218	3093655	664934911636366	12337340				

iozone test complete.

Theoretical Disk Performance is obtained by the following command :
“dd if=/dev/zero of=speedtest bs=1 count=3200 conv=fdatasync”
Throughput = 62.0 MBps
Latency = 16.12 ms

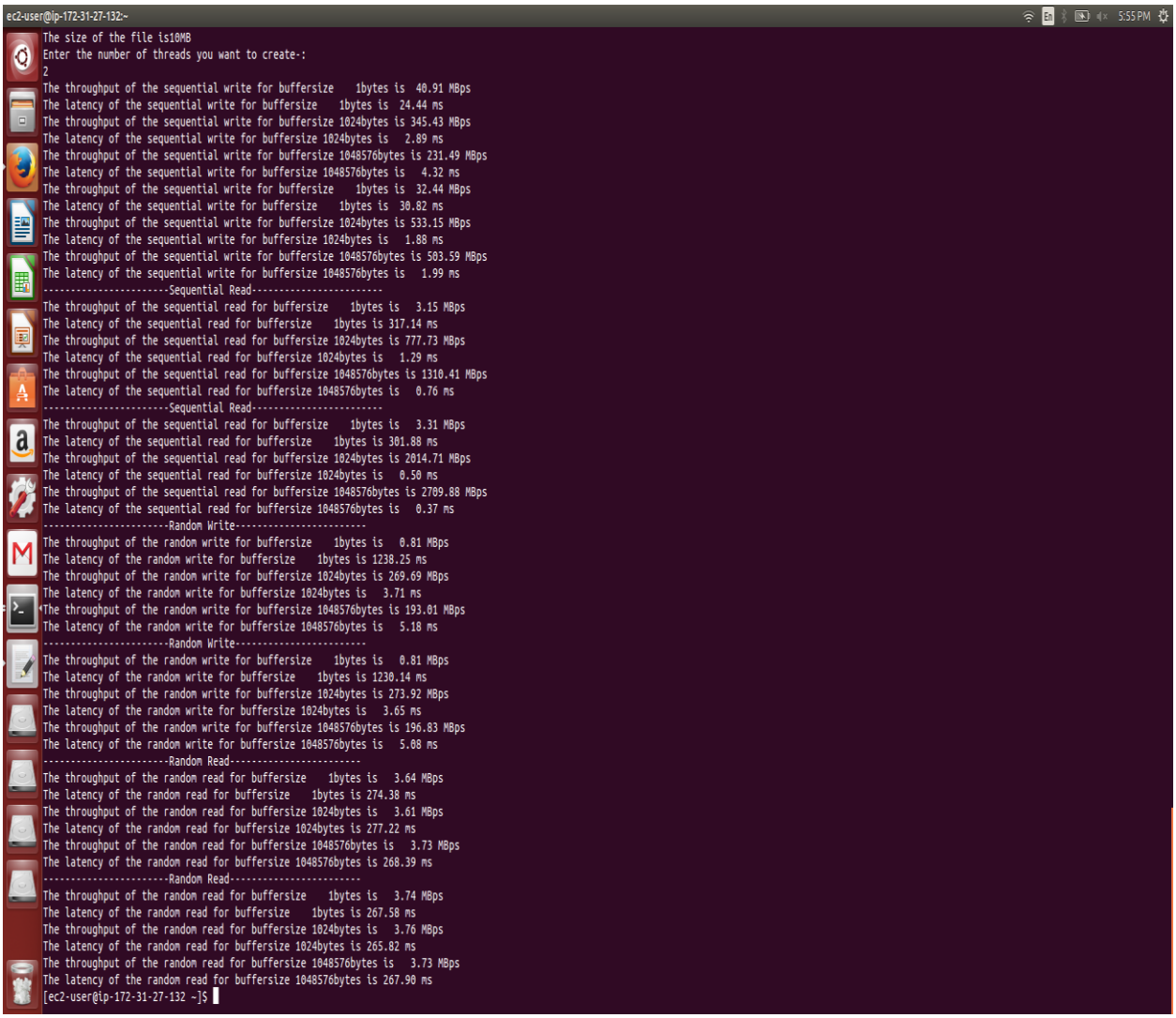
Now the efficiency for 1 byte data is calculated by equation

Efficiency = (Calculated Disk Throughput*100)/Throughput obtained by command

Efficiency = 42.46*100/62

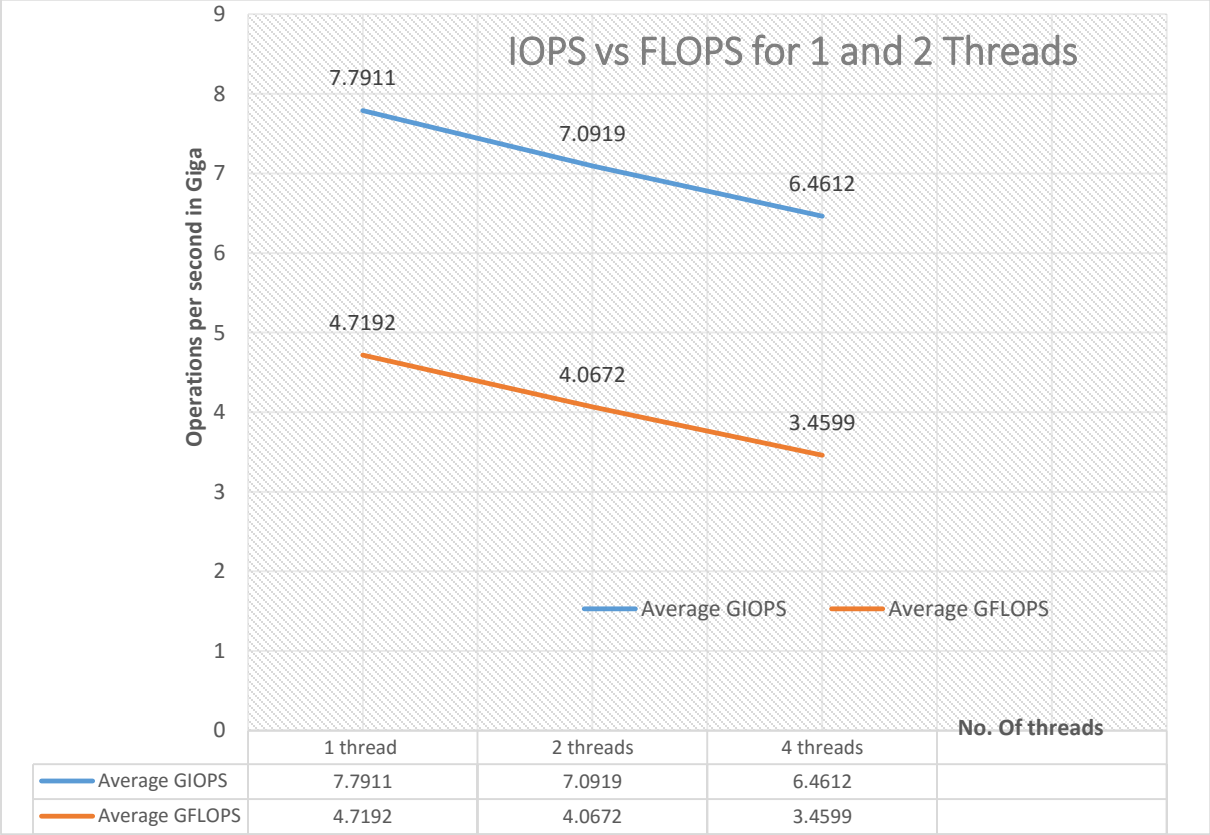
Efficiency = 74.93

Sample Output of Disk Program -:



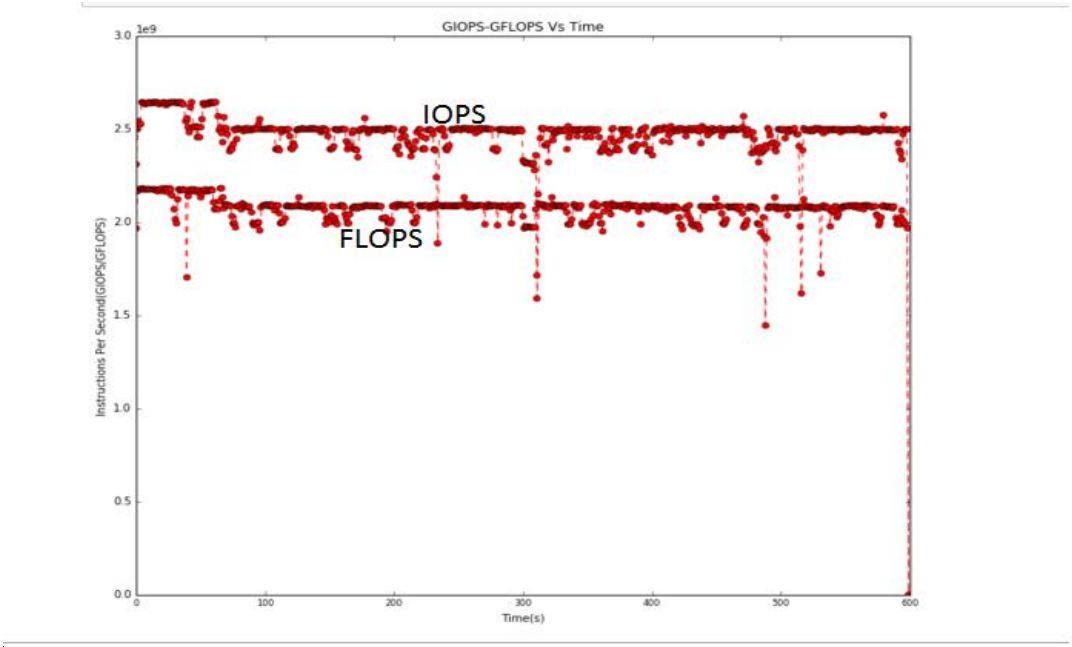
CPU Performance -:

- The experiment was performed three times and its average was taken to make the result more consistent and the graphs are plotted on that results.
- All the experiments were performed on Amazon t2 micro instance.
- The output is measured in gig-IOPS and giga-FLOPS. The performance java program will run for 10 minutes and will create a file in which all the 600 samples taken at every second is stored.



- Analyzing the graph we can see that with increase in the threads the FLOPS and IOPS are decreasing. The main reason behind this is because we are having the system of one core and also we have not taken the multiplication factor with number of threads.
- Compared to IOPS the FLOPS are having smaller value because floating point operations take more time compared to integer.
- The program is having a lot of dependencies which result in lesser number of operations executed.

The following graph is plotted by taking the value of operations performed every second.



- The graph shows the 600 samples value of the operations performed per second in GIOPS FLOPS.
- Comparing the result, we can see that the IOPS values are greater then FLOPS
- We can see that the value obtained is approximately equal to 2.5 IOPS and 2.3FLOPS.

The theoretical performance can be calculated as follows :

= (CPU speed in GHz) * (number of CPU cores) *(CPU instructions per node)

= 2.5*1*8*2

= 40 GFLOPS/sec.

Efficiency -:

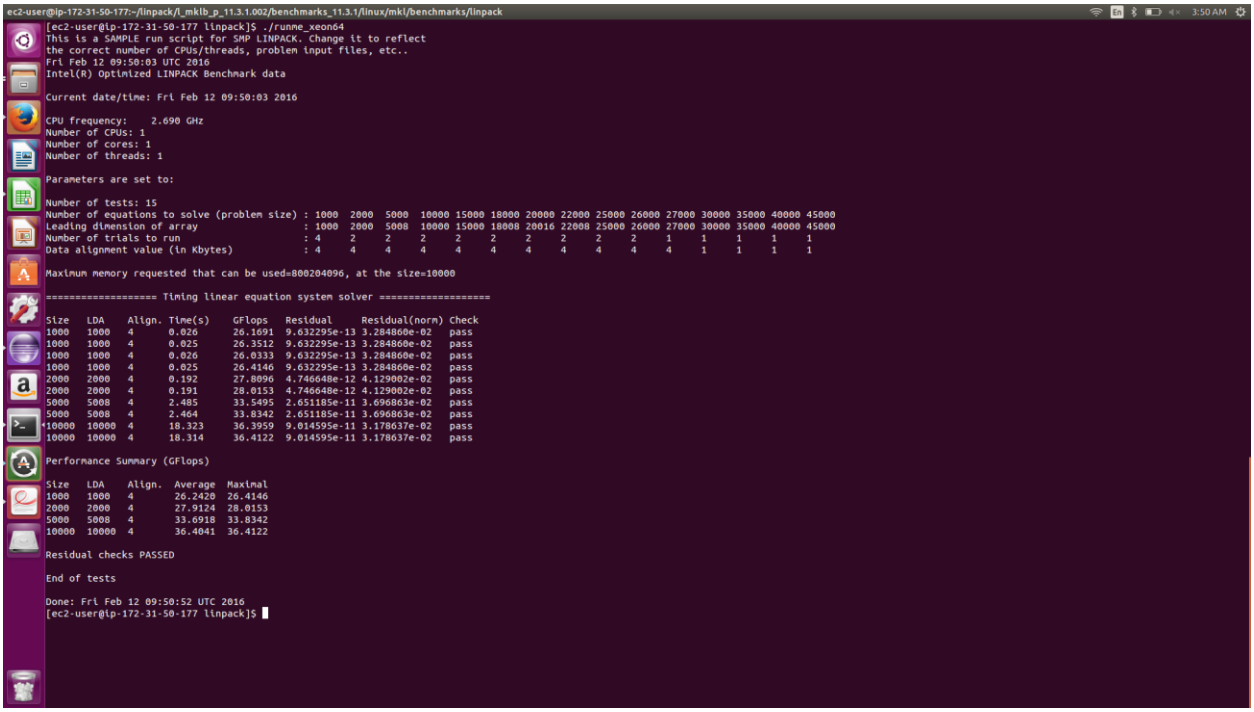
= (Obtained Performance*100)/Theoretical performance

= 11.77 %

The theoretical efficiency is the ideal efficiency when no process are running and calculating the simplest floating point operations without any dependencies. In my program I have made many dependencies which makes next to wait for previous operation and as the reason the compute cycles are wasted.

Extra Credit -:

The linpack screenshot is as follows -:



- The maximum value of GFLOPS obtained will be 36.4122 in amazon EC2 t2 micro instance while running the linpack.