# CS4513 Project 2 Report

**Mengwen Li (mli2)**

**Design:**

a). Script

For Network performance part:

To measure the amount of time required (the latency) to setup a connection to the server, authenticate and tear it down, the following script was used:

```
#!/bin/bash
# Author: Mengwen Li (mli2)

# First create file locally in order to measure sizes.
./removeFile.sh
./createFile.sh
# Measure the connection - tear down time.
ctTotalTime=0
# Do a connection with wrong password 100 times, then calculate the average.
# Since the connection is terminated as soon as the server validates the password,
# this time is the time required to setup a connection, authenticate and tear it down.
for i in {1..20}
do
    # measure the first time.
    t1=$(date +%s%3N)
    # run client and redirect the output message to a file.
    ./client 52.88.191.178 a b ls > expOut
    # measure the second time.
    t2=$(date +%s%3N)
    # calculate the time used.
    timeUse=$((t2-t1))
    # add to total time.
    ctTotalTime=$((ctTotalTime+timeUse))
    # print out the time used.
    echo $timeUse " milliseconds"
done
# print the total time used and the average time.
echo "total connection - tear down time is " $ctTotalTime " milliseconds"
echo "average connection - tear down time is " $((ctTotalTime/20)) " milliseconds"

```

Fig. 1. Script used to measure connection - tear down time.

The first two commands creates the files used at the server side locally in order to get the file sizes latter. In the for loop, the script first record the starting time, then, run the client with the ip address of AmazonEC2 Ubuntu instance, a legal user name, a illegal password, a command “ls”. The output was redirected to “expOut” file. By using a illegal password, only connection, authentication and connection tear down were performed, the server does not have to do an “execvp()”. Then, the second time is recorded, time used to perform connection - tear down was calculated and added to the total time, then, the time used is printed out for record. Twenty runs were performed and the average time was calculated and printed at the end.

To measure the throughput, the following script was used:

```
31 # Calculate the throughput.
32 tpTotalTime=0
33 # print the size of the file.
34 filesize=$(du -sb tpTest | cut -f1)
35 echo "Size of tpTest is $filesize bytes"
36 # Use cat command to get content for a 10MB file 20 times.
37 for i in {1..10}
38 do
39     # measure the first time.
40     t1=$(date +%s%3N)
41     # measure the second time.
42     ./client 52.88.191.178 a a cat tpTest > expOut
43     # measure the second time.
44     t2=$(date +%s%3N)
45     # calculate the time used.
46     timeUse=$((t2-t1))
47     # add to total time.
48     tpTotalTime=$((tpTotalTime+timeUse))
49     echo $timeUse " milliseconds"
50     rm expOut
51 done
52 # print the total time used, the average time and the throughput.
53 echo "total transfer time is " $tpTotalTime " milliseconds"
54 echo "average transfer time is " $((tpTotalTime/10)) " milliseconds"
55 echo "througput calculated is " $((filesize*8*10/(tpTotalTime/1000)) " bits/second"
```

Fig. 2. Script used to measure throughput.

Firtst, the size of the test file was calculated and printed out. Then, in the for loop, the start time is measured, then, run client with the ip address for Amazon EC2 ubuntu instance, correct user name and password, and “cat” command with the “testFile” as parameters. The output is redirected ot an output file. Then, the end time was measured, time used was calculated and the total time is updated. The time used was then printed out. Finally, the total time used for transfer, the average transfer time and the calculated throughput in “bits/second” is printed out.

To get the plot with x-axis being transfer size and y-axis being time, the following script was used:

```
57 # Get data to plot transfer size - time plot.
58 # Use cat command to get content for 10 files with size from 1MB to 10MB.
59 # The time required to get the data is measured.
60 for i in {1..10}
61 do
62     tmpTotalTime=0
63     # print the size of the file.
64     filename=testFile$i
65     filesize=$(du -sb $filename | cut -f1)
66     echo "Size of $filename is $filesize bytes"
67     for j in {1..10}
68     do
69         # measure the first time.
70         t1=$(date +%s%3N)
71         # run client and redirect the output message to a file.
72         ./client 52.88.191.178 a a cat testFile$i > expOut
73         # measure the second time.
74         t2=$(date +%s%3N)
75         # calculate the time used.
76         timeUse=$((t2-t1))
77         tmpTotalTime=$((tmpTotalTime+timeUse))
78         echo $timeUse " milliseconds"
79         rm expOut
80     done
81     echo "total transfer time is " $tmpTotalTime " milliseconds"
82     echo "average transfer time is " $((tmpTotalTime/10)) " milliseconds"
83 done
84 ./removeFile.sh
```

Fig. 3. Script used to collect data for plot.

By using the outer for loop, 10 files with sizes from 1MB to 10MB was received from the server. The script first print out the size of the file, then, in the inner for loop, each file was transfered 10 times. The start time was first measured, then, run client with the ip address for Amazon EC2 ubuntu instance, correct user name and password, and “cat” command with the testing file as parameter and redirect the output to “expOut”. After that, the end time is measured, time used and total time was calculated. Finallly, the time used was printed out and “expOut” file was removed.
When the inner for loop is finished, the total transfer time is printed out and the average transfer time is calculated and printed.

To get the plot with x-axis being transfer size and y-axis being time for local host, the following script was used:

```
 86 # Get data to plot transfer size - time plot.
 87 # Use cat command to get content for 10 files with size from 1MB to 10MB.
 88 # The time required to get the data is measured.
 89 # This is for connecting to local server.
 90 for i in {1..10}
 91 do
 92     tmpTotalTime=0
 93     # print the size of the file.
 94     filename=testFile$i
 95     filesize=$(du -sb $filename | cut -f1)
 96     echo "Size of $filename is $filesize bytes"
 97     for j in {1..10}
 98     do
 99         # measure the first time.
100         t1=$(date +%s%3N)
101         # run client and redirect the output message to a file.
102         ./client localhost a a cat testFile$i > expOut
103         # measure the second time.
104         t2=$(date +%s%3N)
105         # calculate the time used.
106         timeUse=$((t2-t1))
107         tmpTotalTime=$((tmpTotalTime+timeUse))
108         echo $timeUse " milliseconds"
109         rm expOut
110     done
111     echo "total transfer time is " $tmpTotalTime " milliseconds"
112     echo "average transfer time is " $((tmpTotalTime/10)) " milliseconds"
113 done
```

Fig. 4. Script used to collect data on local host for plot.

The only changed line is on line 102, the ip address was changed to localhost.

For CPU and I/O performance part:

To measure the CPU performance on both my machine and the Amazon EC2 ubuntu instance, the following command was used:

```
sysbench --test=cpu --cpu-max-prime=20000 run
```

To measure the file I/O performance on both my machine and the Amazon EC2 ubuntu instance, the following commands was used:
First, the following command was used to prepare a 32G test file (2G test file for cloud server):

```
sysbench --test=fileio --file-total-size=32G prepare
```

Then, the following command was used to do the testing (2G for cloud server):

```
sysbench --test=fileio --file-total-size=16G --file-test-mode=rndrw --init-rng=on --max-time=30 --max-requests=0 run
```

Finally, the following command was used to clean up the testing file (2G for cloud server):

```
sysbench --test=fileio --file-total-size=16G cleanup
```

For Model part:

Excel is used for ploting the data points and claculating mean and standard deviation.

b). How many runs do you performed

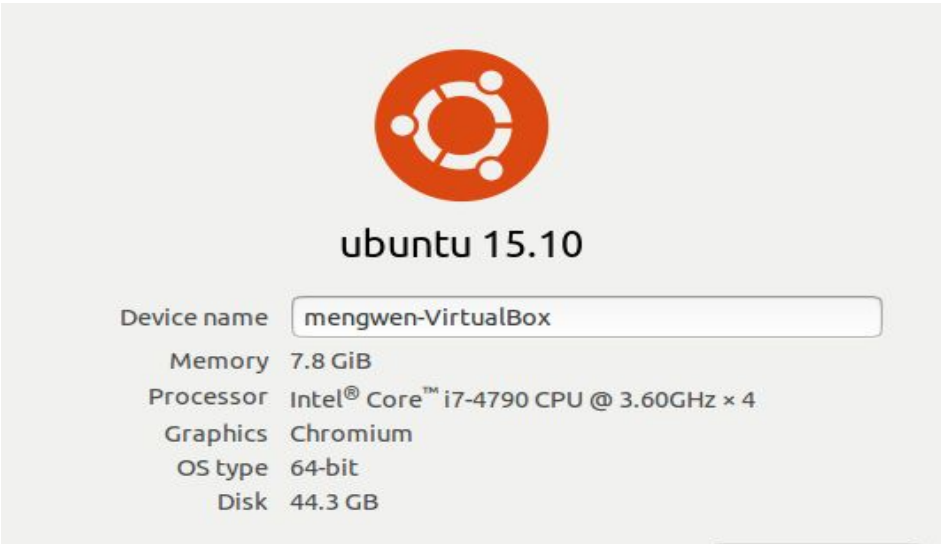
For measuring connection - tear down time, 20 times of running was performed and the average was taken to get the final result.
For measuring throughput, 1 run on 10 different size files were performed. The throughput was calculated by calculating the slope of the resulting graph.
For measuring CPU and I/O performance, 2 runs on each command was performed. One on my machine, the other on Amazon EC2 ubuntu instance.

c). How do you recorded your data

The data used was printed out either by the script or by the command to the terminal so it can be recorded.

d). What the system conditions were like

The system I used to run the program was shown in the following graph:



**Results:**

For network performance part:

The running result I got for measuring the time required to setup a connection to the server, authenticate and tear it down was shown in the following table:

| Run number | Time used (ms) |
|---|---|
| 1 | 776 |
| 2 | 784 |
| 3 | 786 |
| 4 | 775 |
| 5 | 775 |
| 6 | 768 |
| 7 | 770 |
| 8 | 781 |
| 9 | 762 |
| 10 | 777 |
| 11 | 790 |
| 12 | 773 |
| 13 | 777 |
| 14 | 784 |
| 15 | 776 |
| 16 | 774 |
| 17 | 775 |
| 18 | 771 |
| 19 | 775 |
| 20 | 781 |
| Mean | 776.5 |
| Standard diviation | 6.36 |

The running result I got for measuring throughput was shown in the following table:

| Run number | Time measured (ms) |
|---|---|
| 1 | 5767 |
| 2 | 6758 |
| 3 | 6747 |
| 4 | 6541 |
| 5 | 6031 |
| 6 | 4225 |
| 7 | 3592 |
| 8 | 6234 |
| 9 | 3983 |
| 10 | 4303 |
| File size (MB) | 10 |
| Throughput (bits/s) | 15094339 |
| Mean | 5418.1 |
| Standard diviation | 1185.17 |

The data I got for the plot with x-axis being transfer size and y-axis being time was shown in the following table:

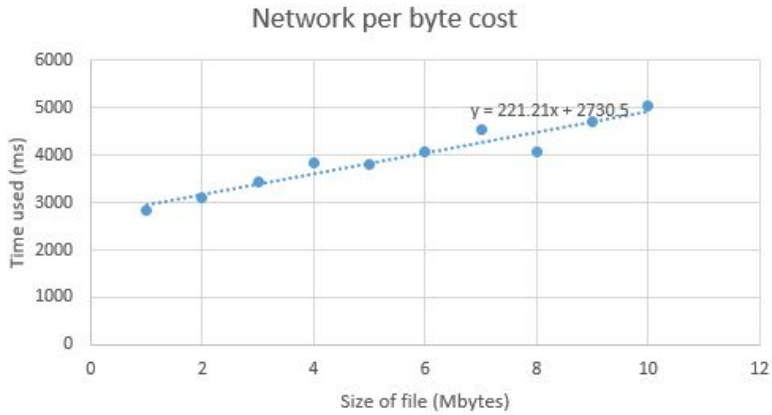
| File size (MB) | Run number | Measured time (ms) |
|---|---|---|
| 1 | 1 | 2901 |
| | 2 | 2829 |
| | 3 | 2822 |
| | 4 | 2784 |
| | 5 | 2826 |
| | 6 | 2832 |
| | 7 | 2816 |
| | 8 | 2864 |
| | 9 | 2825 |
| | 10 | 2856 |
| Mean | | 2835.5 |
| Standard diviation | | 29.99 |
| 2 | 1 | 3680 |

| | | |
|---|---|---|
| | 2 | 3324 |
| | 3 | 3152 |
| | 4 | 3038 |
| | 5 | 3016 |
| | 6 | 2990 |
| | 7 | 2993 |
| | 8 | 2972 |
| | 9 | 2940 |
| | 10 | 2961 |
| Mean | | 3106.6 |
| Standard diviation | | 220.14 |
| 3 | 1 | 3697 |
| | 2 | 3316 |
| | 3 | 3252 |
| | 4 | 3205 |
| | 5 | 3671 |
| | 6 | 4474 |
| | 7 | 3333 |
| | 8 | 3182 |
| | 9 | 3134 |
| | 10 | 3145 |
| Mean | | 3440.9 |
| Standard diviation | | 394.1 |
| 4 | 1 | 4978 |
| | 2 | 3566 |
| | 3 | 3544 |
| | 4 | 3589 |
| | 5 | 3208 |
| | 6 | 3794 |
| | 7 | 3369 |
| | 8 | 3426 |
| | 9 | 4946 |
| | 10 | 3975 |
| Mean | | 3839.5 |
| Standard diviation | | 596.4 |
| 5 | 1 | 3701 |
| | 2 | 3370 |

| | | |
|---|---|---|
| | 3 | 5227 |
| | 4 | 3586 |
| | 5 | 3327 |
| | 6 | 4963 |
| | 7 | 3586 |
| | 8 | 3719 |
| | 9 | 3373 |
| | 10 | 3293 |
| Mean | | 3814.5 |
| Standard diviation | | 658.9 |
| 6 | 1 | 3616 |
| | 2 | 3330 |
| | 3 | 5408 |
| | 4 | 3730 |
| | 5 | 5241 |
| | 6 | 3643 |
| | 7 | 4757 |
| | 8 | 3586 |
| | 9 | 4058 |
| | 10 | 3522 |
| Mean | | 4089.1 |
| Standard diviation | | 722.2 |
| 7 | 1 | 5850 |
| | 2 | 4331 |
| | 3 | 3586 |
| | 4 | 3383 |
| | 5 | 5706 |
| | 6 | 5649 |
| | 7 | 3683 |
| | 8 | 3483 |
| | 9 | 5821 |
| | 10 | 3869 |
| Mean | | 4536.1 |
| Standard diviation | | 1026.68 |
| 8 | 1 | 3678 |
| | 2 | 3398 |
| | 3 | 3409 |

| | | |
|---|---|---|
| | 4 | 6088 |
| | 5 | 3772 |
| | 6 | 5152 |
| | 7 | 3790 |
| | 8 | 3502 |
| | 9 | 3334 |
| | 10 | 4494 |
| Mean | | 4061.7 |
| Standard diviation | | 865.78 |
| 9 | 1 | 3832 |
| | 2 | 3487 |
| | 3 | 3891 |
| | 4 | 6130 |
| | 5 | 6427 |
| | 6 | 6386 |
| | 7 | 3874 |
| | 8 | 4659 |
| | 9 | 3683 |
| | 10 | 4777 |
| Mean | | 4714.6 |
| Standard diviation | | 1116.75 |
| 10 | 1 | 4502 |
| | 2 | 3586 |
| | 3 | 4852 |
| | 4 | 5449 |
| | 5 | 5801 |
| | 6 | 3865 |
| | 7 | 5206 |
| | 8 | 5996 |
| | 9 | 4934 |
| | 10 | 6136 |
| Mean | | 5032.7 |
| Standard diviation | | 819.72 |

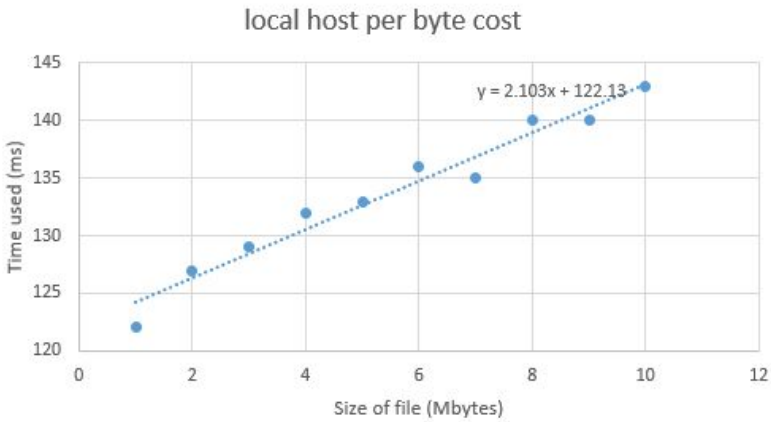
The network per byte cost I got from the above table was shown in the following:



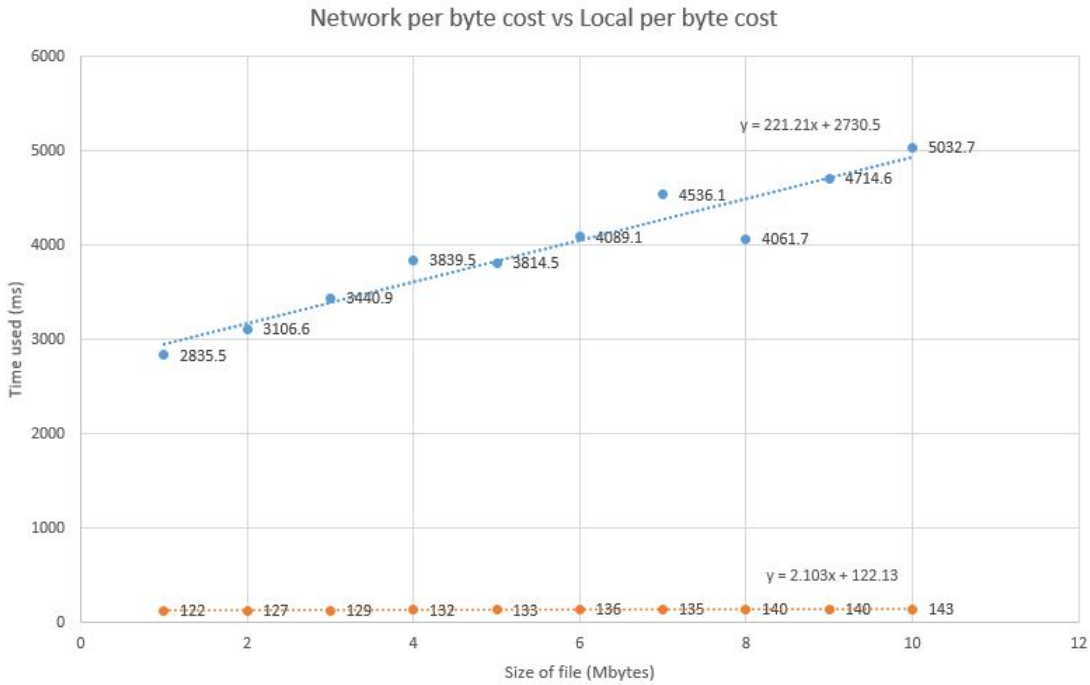

The data I got for the plot with x-axis being transfer size and y-axis being time for local host was shown in the following table:

| File size (MB) | Average Time measured (ms) |
|---|---|
| 1 | 122 |
| 2 | 127 |
| 3 | 129 |
| 4 | 132 |
| 5 | 133 |
| 6 | 136 |
| 7 | 135 |
| 8 | 140 |
| 9 | 140 |
| 10 | 143 |

The plot for the table above was shown below:



The plot with both lines together in one graph was shown below:
(blue line for cloud, orange line for local)

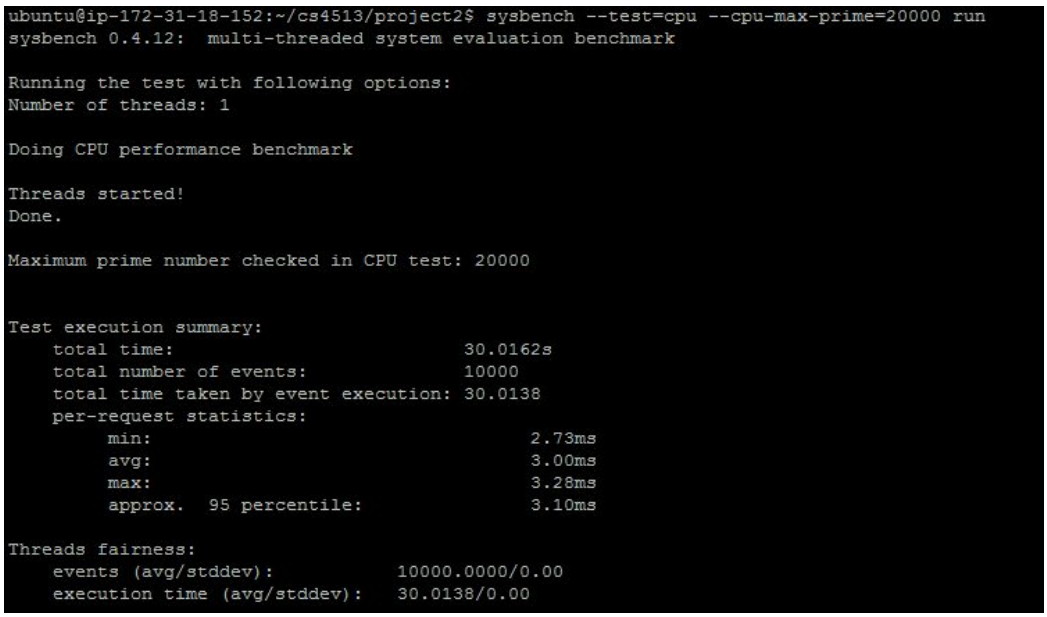



For CPU and I/O performance part:

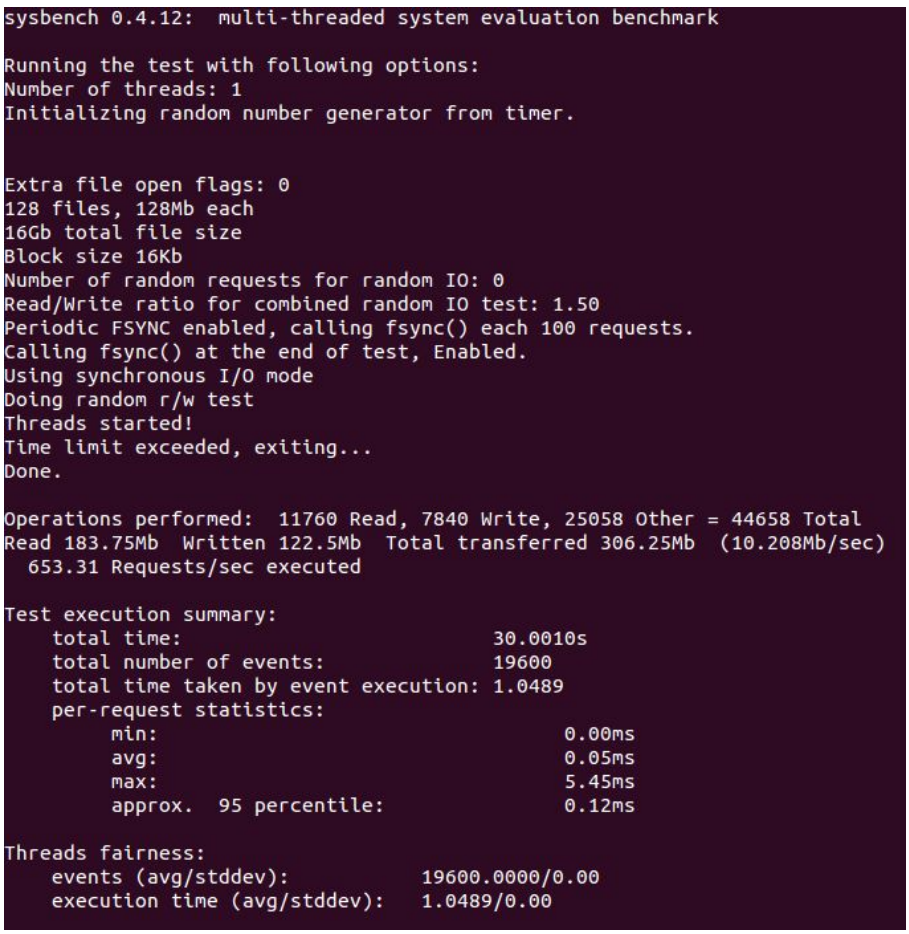
The result I get for benchmark the CPU performance for my local machine was shown in the following figure:

```
mengwen@mengwen-VirtualBox:~/Desktop/cs4513/Project2$ sysbench --test=cpu --cpu-max-prime=20000 run
sysbench 0.4.12:  multi-threaded system evaluation benchmark

Running the test with following options:
Number of threads: 1

Doing CPU performance benchmark

Threads started!
Done.

Maximum prime number checked in CPU test: 20000


Test execution summary:
    total time:                          22.6628s
    total number of events:              10000
    total time taken by event execution: 22.6605
    per-request statistics:
         min:                                  2.07ms
         avg:                                  2.27ms
         max:                                 24.96ms
         approx.  95 percentile:               2.52ms

Threads fairness:
    events (avg/stddev):           10000.0000/0.00
    execution time (avg/stddev):   22.6605/0.00
```

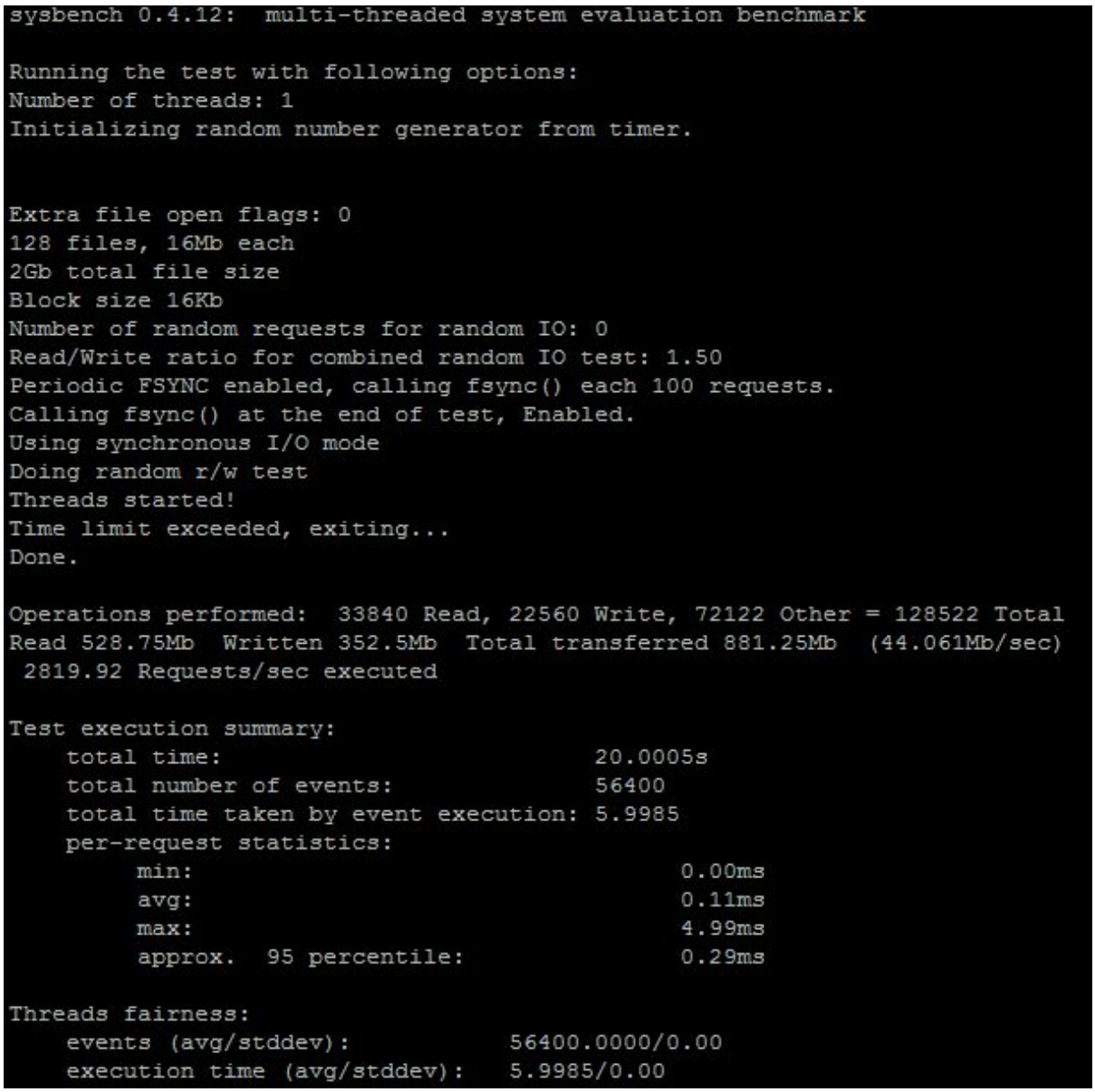
The result I get for benchmark the CPU performance for Amazon EC2 ubuntu instance was shown in the following figure:




The result I get for benchmark the file I/O performance for my local machine was shown in the following figure:



The result I get for benchmark the file I/O performance for Amazon EC2 ubuntu instance was shown in the following figure:

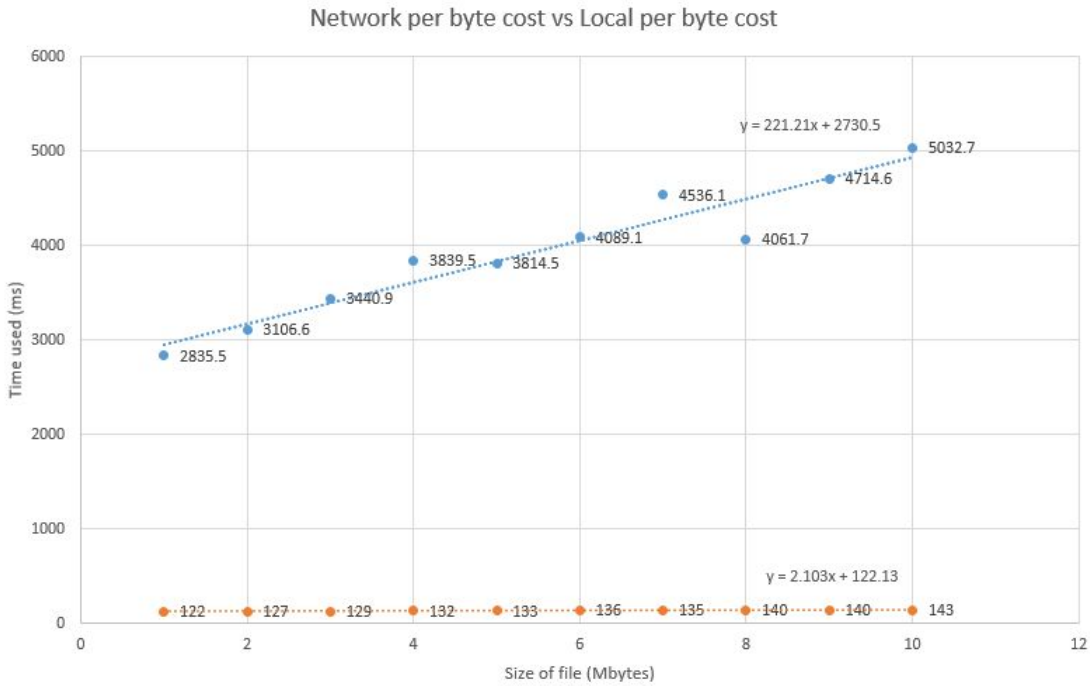


```
sysbench 0.4.12:  multi-threaded system evaluation benchmark

Running the test with following options:
Number of threads: 1
Initializing random number generator from timer.


Extra file open flags: 0
128 files, 16Mb each
2Gb total file size
Block size 16Kb
Number of random requests for random IO: 0
Read/Write ratio for combined random IO test: 1.50
Periodic FSYNC enabled, calling fsync() each 100 requests.
Calling fsync() at the end of test, Enabled.
Using synchronous I/O mode
Doing random r/w test
Threads started!
Time limit exceeded, exiting...
Done.

Operations performed:  33840 Read, 22560 Write, 72122 Other = 128522 Total
Read 528.75Mb  Written 352.5Mb  Total transferred 881.25Mb  (44.061Mb/sec)
 2819.92 Requests/sec executed

Test execution summary:
    total time:                          20.0005s
    total number of events:              56400
    total time taken by event execution: 5.9985
    per-request statistics:
         min:                                  0.00ms
         avg:                                  0.11ms
         max:                                  4.99ms
         approx.  95 percentile:               0.29ms

Threads fairness:
    events (avg/stddev):           56400.0000/0.00
    execution time (avg/stddev):   5.9985/0.00
```

**Analysis:**

Calculation for n:

22.6628 + n / 10.208 = n * 0.221 + 30.0162 + n / 44.06
n = -50.457




From the above figure, we can also tell that the intersection for these too lines is less than 0.

The result means that for any positive bytes being transfered, the local performance is always better than the cloud performane.
From the benchmark, we can tell that the local CPU is faster than the cloud CPU. This may be one explanation for this result. From the standard diviation calculated in the tables above, we can tell that the network environment is not very stable. Also, the connection - tear down time is relative big. This may also cause the cloud performance to be worse.