Name: Huyen Nguyen

Report for Distributed Shell

Design

a) What programs/scripts you ran and what they did (use pseudo-code)

I used `measure_time` (compiled from `measure_time.c`) and `test_script.sh`.

Pseudocode for `test_script.sh`

```
# Measure the amount of time required (the latency, in milliseconds) to setup a connection to the server, authenticate, and tear it down
for i = 0 to 10
      call measure_time and execute an empty command

# Measure the maximum throughput (in bits per second) from the server to the client
for i = 0 to 10
      create file$i.txt of size (4 * i) bytes
      scp the file to the server
      call measure_time and execute “cat file$i.txt” command

# Cleaning up
Shut down the server by making client send “exit” command to the server
```

Pseudocode for `measure_time.c`

```
start = get current time
child_pid = Fork child process
If (child_pid == 0)
      Call rm using execvp()
      Print error // If execvp returns, it must have failed
Else
      Wait for child process
      Sync
      end = get current time
      print(start-end)
      Return child status
```

b) How many runs you performed

I executed the empty command 10 times to measure the amount of time required (the latency, in milliseconds) to setup a connection to the server, authenticate, and tear it down. Also, in order to measure the maximum throughput (in bits per second) from the server to the client, I created 10 files of different sizes and use `measure_time` to measure the duration to transfer each of them.

c) How you recorded your data

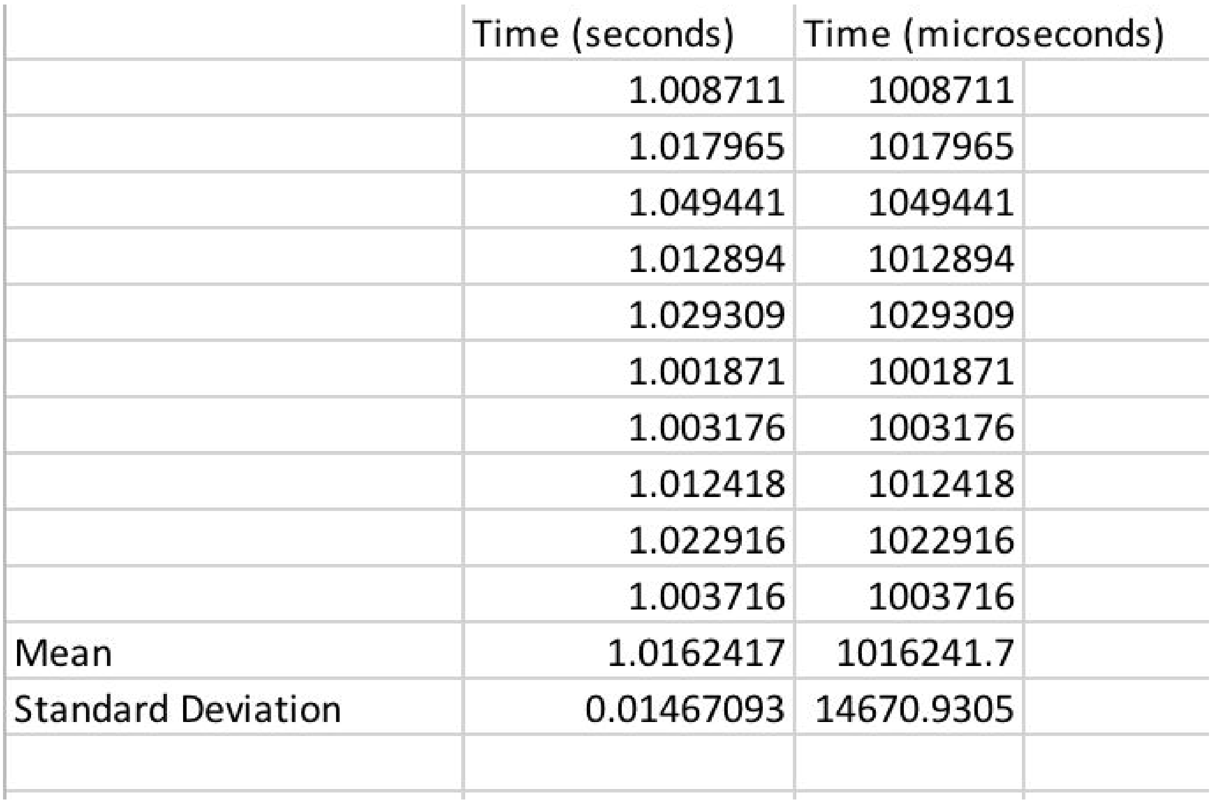
I used measure_time.c to record time. Specifically, for every file, I recorded the time right before running rm (1). Then, I created a child process to run rm and made the parent process wait for it. Then, when the child process is finished, I used sync(), and then measure the current time again (2). Finally, the time difference between (2)-(1) is the time taken to move the file to the dumpster.

d) What the system conditions were like

The experiment was conducted on MacBook Pro (Retina, 13-inch, Early 2015) with macOS High Sierra. I made sure that the scripts run on the Linux VM for this course. The reason why I decided to run the tests on my actual OS because I experienced problems with gettimeofday() in VMs in OS class.

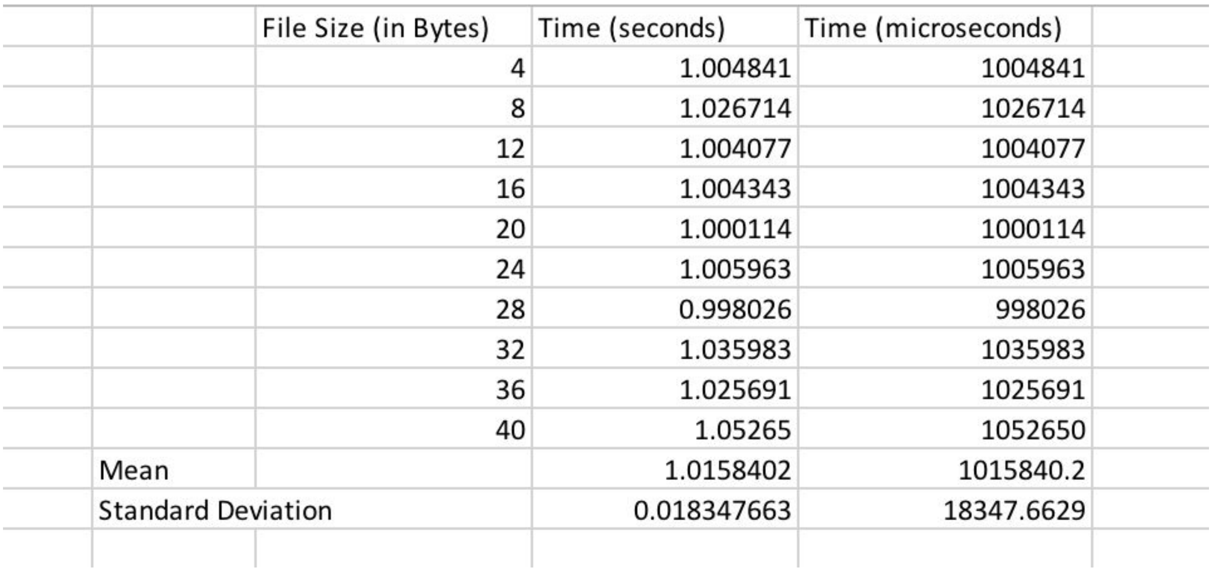
Results

Network Performance



	Time (seconds)	Time (microseconds)
	1.008711	1008711
	1.017965	1017965
	1.049441	1049441
	1.012894	1012894
	1.029309	1029309
	1.001871	1001871
	1.003176	1003176
	1.012418	1012418
	1.022916	1022916
	1.003716	1003716
Mean	1.0162417	1016241.7
Standard Deviation	0.01467093	14670.9305

Figure 1: The amount of time required (the latency, in seconds) to setup a connection to the Amazon EC2 server, authenticate, and tear it down (10 iterations)



	File Size (in Bytes)	Time (seconds)	Time (microseconds)
	4	1.004841	1004841
	8	1.026714	1026714
	12	1.004077	1004077
	16	1.004343	1004343
	20	1.000114	1000114
	24	1.005963	1005963
	28	0.998026	998026
	32	1.035983	1035983
	36	1.025691	1025691
	40	1.05265	1052650
Mean		1.0158402	1015840.2
Standard Deviation		0.018347663	18347.6629

Figure 2: The amount of time required to transfer a file of various sizes from the Amazon EC2 server to the client

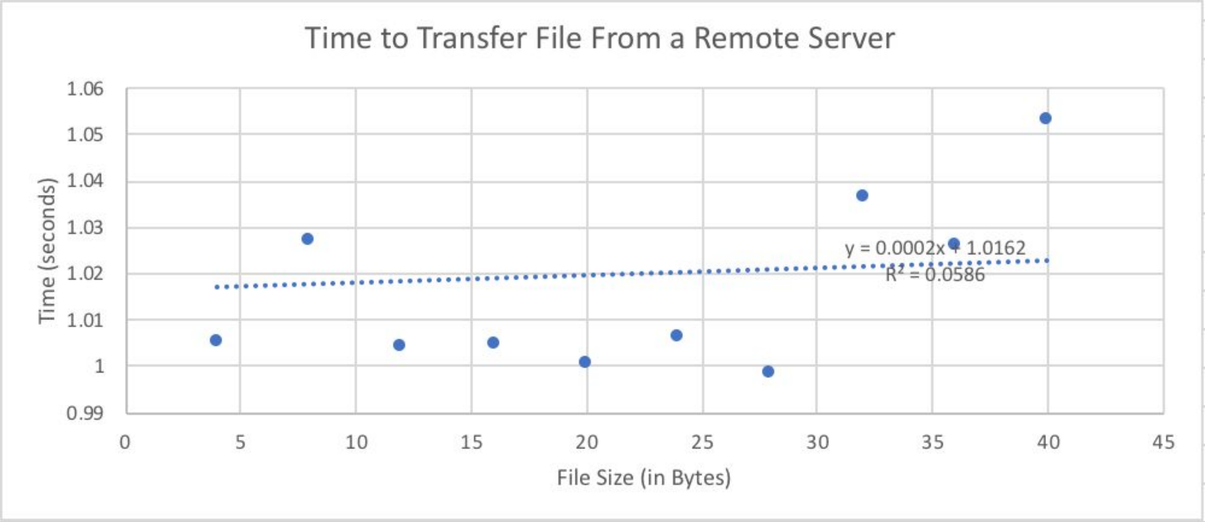


Figure 3: The linear equation with time (in seconds) and file size (in bytes) using data from figure 2 and the mean value from figure 1 as the y-intercept

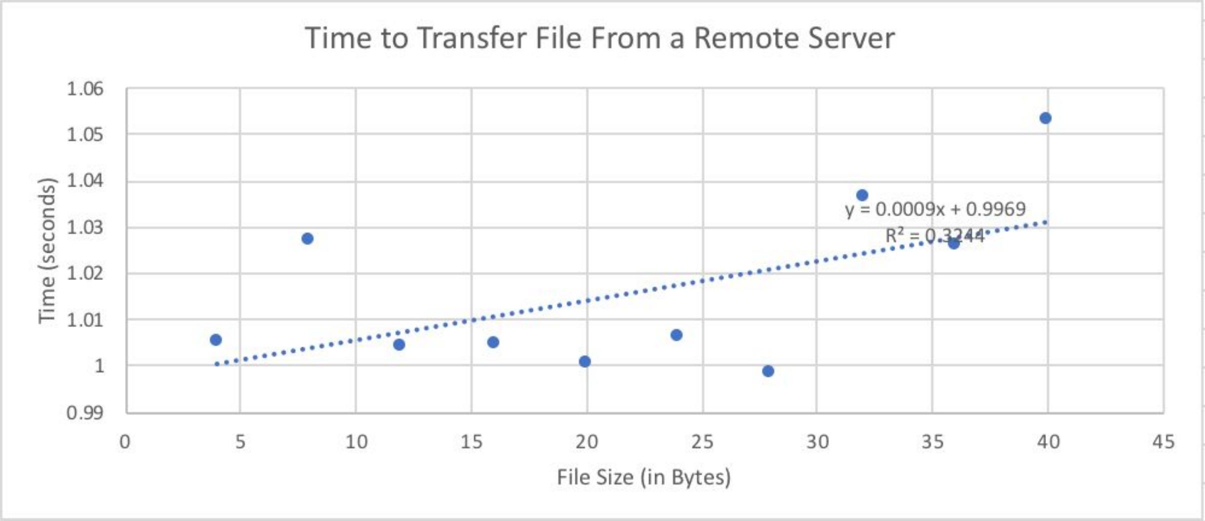


Figure 4: The linear equation with time (in seconds) and file size (in bytes) using data from figure 2 only

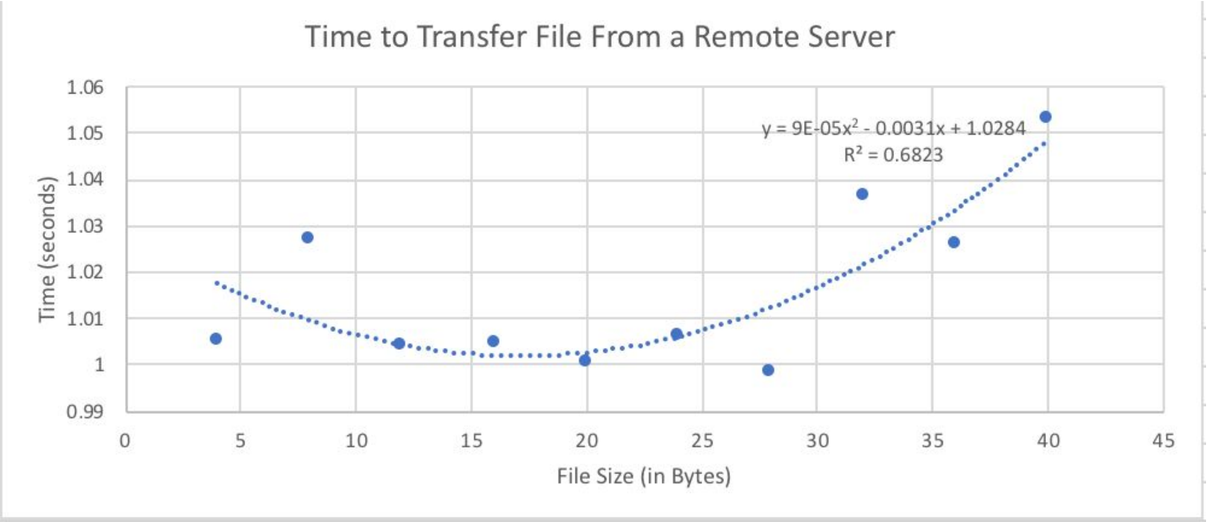


Figure 4: The quadratic equation with time (in seconds) and file size (in bytes) using data from figure 2 only

CPU and I/O Performance

Local Machine

CPU Performance

```
> sysbench --test=cpu --cpu-max-prime=20000 run
WARNING: the --test option is deprecated. You can pass a script name or path on the
command line without any options.
sysbench 1.1.0-27a5b99 (using bundled LuaJIT 2.1.0-beta3)

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


Prime numbers limit: 20000

Initializing worker threads...

Threads started!

CPU speed:
    events per second:   551.23

Throughput:
    events/s (eps):                551.2305
    time elapsed:                  10.0013s
    total number of events:              5513

Latency (ms):
         min:                            1.41
         avg:                            1.81
         max:                           30.54
         95th percentile:                2.76
         sum:                          9992.30

Threads fairness:
    events (avg/stddev):           5513.0000/0.00
    execution time (avg/stddev):   9.9923/0.00
```

I/O Performance

```
> sysbench --test=fileio --file-total-size=1G prepare
> sysbench --test=fileio --file-total-size=1G --file-test-mode=rndrw --max-time=30
--max-requests=0 run
Throughput:
         read:  IOPS=4461.73 69.71 MiB/s (73.10 MB/s) (1)
         write: IOPS=2974.49 46.48 MiB/s (48.73 MB/s) (2)
         fsync: IOPS=9514.30

Latency (ms):
         min:                            0.00
         avg:                            0.06
         max:                           47.21
         95th percentile:                0.19
         sum:                        29526.98 (3)

> sysbench --test=fileio --file-total-size=1G cleanup
```

Amazon Linux Instance

CPU Performance

```
> sysbench --test=cpu --cpu-max-prime=20000 run
WARNING: the --test option is deprecated. You can pass a script name or path on the
command line without any options.
sysbench 1.1.0-e5c8052 (using bundled LuaJIT 2.1.0-beta3)

Running the test with following options:
Number of threads: 1
Initializing random number generator from current time
```

```
Prime numbers limit: 20000

Initializing worker threads...

Threads started!

CPU speed:
    events per second:   340.73

Throughput:
    events/s (eps):              340.7268
    time elapsed:                10.0021s
    total number of events:         3408

Latency (ms):
         min:                      2.66
         avg:                      2.93
         max:                       3.23
         95th percentile:                3.07
         sum:                    9997.33

Threads fairness:
    events (avg/stddev):          3408.0000/0.00
    execution time (avg/stddev):   9.9973/0.00
```

I/O Performance

```
> sysbench --test=fileio --file-total-size=6G prepare
> sysbench --test=fileio --file-total-size=6G --file-test-mode=rndrw --max-time=30
--max-requests=0 run

Throughput:
         read:  IOPS=1328.00 20.75 MiB/s (21.76 MB/s) (1)
         write: IOPS=885.33 13.83 MiB/s (14.51 MB/s) (2)
         fsync: IOPS=2831.49

Latency (ms):
         min:                    0.00
         avg:                    0.20
         max:                    16.70
         95th percentile:              0.73
         sum:                 29759.11 (3)

> sysbench --test=fileio --file-total-size=6G cleanup
```

Analysis

Network Performance

The linear equation on figure 3 has an R-squared value of 0.0586, whereas the one on figure 4 has an R-squared value of 0.3244. This is indicative that the relation between the amount of time to transfer a file and the file size might not be linear. In fact, the quadratic equation on figure 5 has an R-squared value of 0.6823, which is greater than the linear equations' R-squared values on both figures 3 and 4. In fact, in order to predict the amount of time to transfer a file, other factors could have impacts, such as the OS, the processor speed, etc.

CPU and I/O Performance

Solve equation

Local_CPU + Local_File_I/O = n * Network + Remote_CPU + Remote_File_I/O

- Local_CPU = 10.0013 (number is highlighted in blue on page 4)
- Local_File_I/O = Reading time + Writing time = (1) x (3)/1000 + (2) x (3)/1000 ≈ 3597(MB) (the numbers are highlighted in red on page 5). Also, I used (3)/1000 since I want the unit to be in seconds so it can be multiplied by (1), whose unit is (MB/s).
- Network = 1.0162417 (number is the mean on figure 1)
- Remote_CPU = 10.0021 (number is highlighted in blue on page 6)
- Remote_File_I/O = Reading time + Writing time = (1) x (3)/1000 + (2) x (3)/1000 ≈ 1079(MB) (the numbers are highlighted in red on page 6)

So n ≈2478 (MB) or 2.478x10^9 (bytes).
However, due to variations between computer configurations (e.g. CPU, disk, etc.), the equation might not be the most accurate to model the relations among the variables. However, the fact that this yields a positive for n makes sense. In fact, for a decent computer, local transfer should be faster than remote transfer. However, the number seems quite big. The fact that I used different file sizes for local (1 GB) and server (6 GB) might have caused this.

Other Comments

- The `--init-rng=on` option in the command `$ sysbench --test=fileio --file-total-size=16G --file-test-mode=rndrw --init-rng=on --max-time=30 --max-requests=0 run` is not present in the sysbench Github documentation. Therefore, I removed it to perform the experiments.
- The command `$ sysbench --test=fileio --file-total-size=16G prepare` gave this error `FATAL: Failed to write file! errno = 28 (No space left`

`on device`). Therefore, I had to use 6GB for the Amazon EC2 instance, and only 1GB for my computer based on the amount of free storage I have (using `$ df -h`).

Sample Session

```
$ ./server
SOCKET CREATED SUCCESSFULLY! PORT NUMBER IS 1024
./server activating
        PORT:1024

directory:/Users/huyennguyen/Documents/Documents/distributed-systems-wpi/distributed-shell
Socket ready to go! Accepting connections....


************Listening for requests...************

$ ./client -s 127.0.0.1 -c "ls"
Client output:
Done. Creating socket...
Created. Trying connection to server...
Sending username...
Username sent successfully!
876817783
Client: Encrypted password 87FXVT5PVcoXQ
Server said credentials were ok
Command is ls
Command sent successfully!
Makefile
Project2_DistributedShell.pdf
Project2_DistributedShell_slides.pdf
README.md
abc.txt
client
client.c
client.h
client.o
distributed-systems-wpi.pem
server
server.c
server.h
server.o
shared.h
test
test_file.0
test_file.1
test_file.2
test_file.3
test_file.4
test_file.5
test_file.6
test_file.7
test_file.8
test_file.9
ubuntu@54.149.136.240
ubuntu@ec2-54-149-136-240.us-west-2.compute.amazonaws.com
Finished printing command output from server.
```

Server output:

```
Received connection
Here is the username: huyen
876817783
Here is the encrypted password: 87FXVT5PVcoXQ
Here is the encrypted password on the server: 87FXVT5PVcoXQ
Correct credentials!
Here is the command: ls
child with pid 5869 exited with status of 0

************Listening for requests...************
```

```
$ ./client -s 127.0.0.1 -c "exit"
```

Client output:

```
Done. Creating socket...
Created. Trying connection to server...
Sending username...
Username sent successfully!
878078308
Client: Encrypted password 87FXVT5PVcoXQ
Server said credentials were ok
Command is exit
Command sent successfully!
Finished printing command output from server.
```

Server output:

```
Received connection
Here is the username: huyen
878078308
Here is the encrypted password: 87FXVT5PVcoXQ
Here is the encrypted password on the server: 87FXVT5PVcoXQ
Correct credentials!
Here is the command: exit
```

```
Server exiting...
[1]    5831 killed     ./server
```