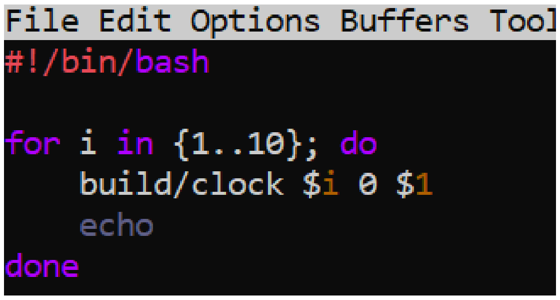
EVAL 1 Part 1 Solution A

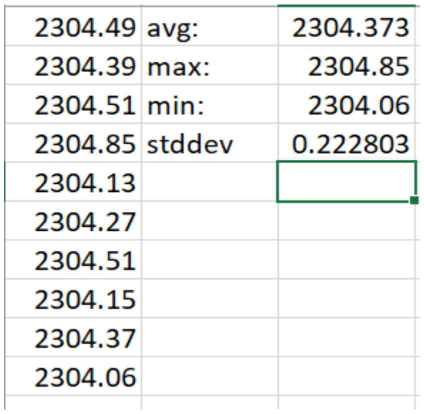
Using the following script (or something similar), we can obtain all 10 ./clock samples from either waiting method (by calling “./script.sh s/b”).



File Edit Options Buffers Tool

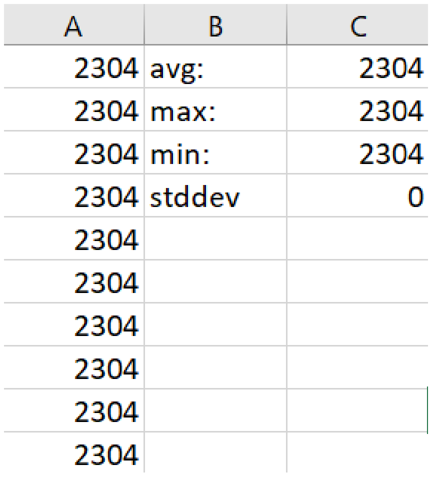
```
#!/bin/bash

for i in {1..10}; do
    build/clock $i 0 $1
    echo
done
```

We can then input the 10 clock frequencies into a spreadsheet, or a python script, and get our avg, max, min, and standard deviation.



2304.49	avg:	2304.373
2304.39	max:	2304.85
2304.51	min:	2304.06
2304.85	stddev	0.222803
2304.13		
2304.27		
2304.51		
2304.15		
2304.37		
2304.06		

SLEEP



A	B	C
2304	avg:	2304
2304	max:	2304
2304	min:	2304
2304	stddev	0
2304		
2304		
2304		
2304		
2304		
2304		

BUSYWAIT

By looking at the differences in our average and standard deviation, we can see than Busywait is by far the more precise and consistent, yielding the same exact speed everytime. Sleep is fairly precise but not nearly as accurate as busywait. My processor base speed is set to 2.30GHz, so in comparison they're both accurate. This is likely due to busywait constantly checking to exit the loop, whereas sleep does nothing for x amount of time, and entering and exiting nanosleep might take some time(overhead cost).

EVAL 1 Part 1 Solution B

a) Elapsed Sleep

Time (s)	**Clock Speed**
1	1000.45
2	1002.20
3	1003.27
4	1000.53
5	1001.07
6	1000.53
7	1000.71
8	1000.28
9	1000.41
10	1000.10

Calculating the average:
(1000.45 + 1002.20 + 1003.27 + 1000.53 + 1001.07 + 1000.53 + 1000.71 + 1000.28 + 1000.41 + 1000.10) / 10 = 1000.955

Standard Deviation: = (1000.45 - 1000.955)2 + ... + (1000.10 - 1000.955)2
σ = √0.90824500000001
= 0.95301888753582

Maximum: 1003.27
Minimum: 1000.10
Average: 1000.955
Standard Deviation = 0.95301888753582

b) BUSYWAIT

Time (s)	**Clock Speed**
1	1000.01
2	1000.01
3	1000.00
4	999.99

5	1000.00
6	1000.00
7	1000.00
8	1000.00
9	1000.00
10	1000.01

Standard Deviation =

$$\frac{(1000.01 - 1000.002)^2 + \ldots + (1000.01 - 1000.002)^2}{10}$$

$$\sigma = \sqrt{3.5999999999935E-5}$$
$$= 0.005999999999945$$

Maximum: 1000.01
Minimum: 999.99
Average: 1000.002
Standard Deviation =0.005999999999945

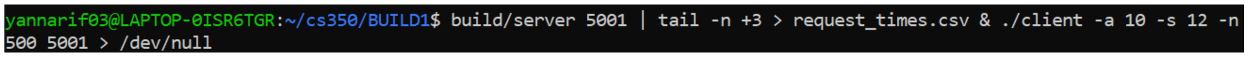
(c) If your conclusion is correctly derived from the data, then your solution is correct since there is no right or wrong answer for this question.

For example, say all your machines had a similar clock speed range (eg. 2.7000-2.7003 Ghz) then your intuition of all machines being identical is correct. Similarly if your clock speeds varied (eg. 2.4 Ghz to 2.8 Ghz) then *clearly* not all machines are identical.

Eval 1 Part 2 Solution A (bash scripts)

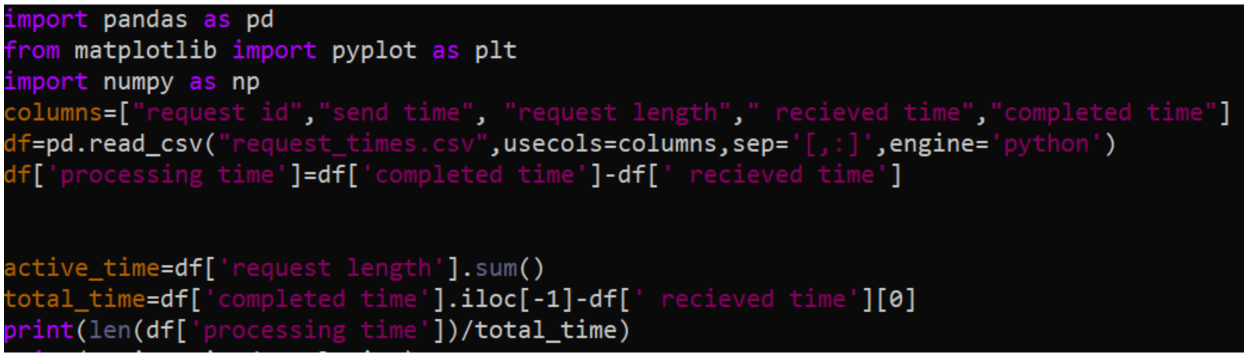
The following solution uses bash scripts.

The following command takes the output of server, truncates the first 3 lines (the port initialization output), and redirects the rest into a csv file. Additionally, it discards the client output.

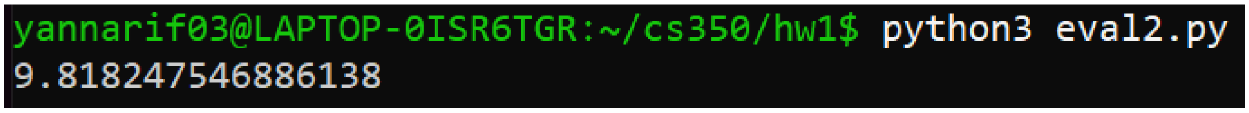


```
yannarif03@LAPTOP-0ISR6TGR:~/cs350/BUILD1$ build/server 5001 | tail -n +3 > request_times.csv & ./client -a 10 -s 12 -n
500 5001 > /dev/null
```

By modifying the command that was given in the instructions, i was able to redirect the server output into a csv file, and work on it in a python script. By using the pandas package, we can open our csv files as data frames, and preform operations on entire columns.



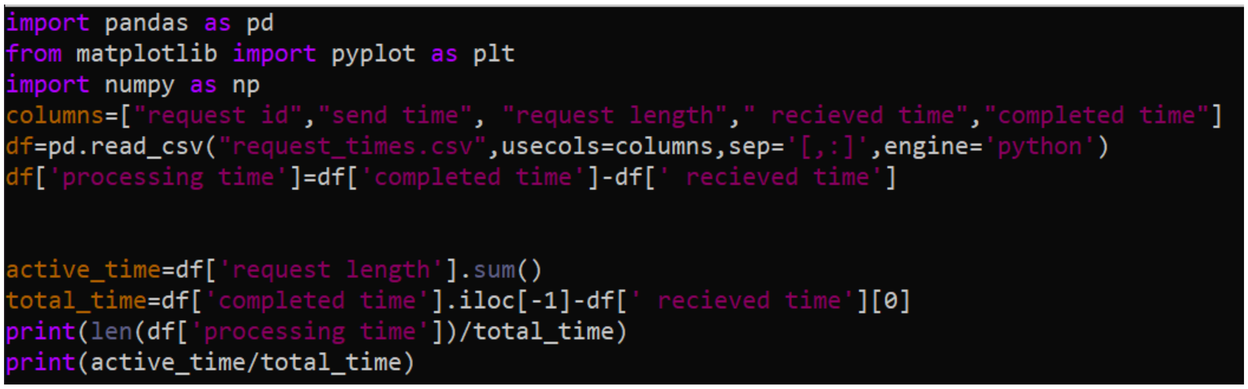
```
import pandas as pd
from matplotlib import pyplot as plt
import numpy as np
columns=["request id","send time", "request length"," recieved time","completed time"]
df=pd.read_csv("request_times.csv",usecols=columns,sep='[,:]',engine='python')
df['processing time']=df['completed time']-df[' recieved time']


active_time=df['request length'].sum()
total_time=df['completed time'].iloc[-1]-df[' recieved time'][0]
print(len(df['processing time'])/total_time)
```



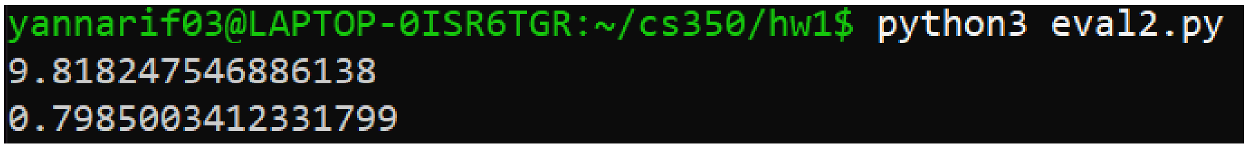
```
yannarif03@LAPTOP-0ISR6TGR:~/cs350/hw1$ python3 eval2.py
9.818247546886138
```

A little less than 10 requests per second. This makes sense because with a client parameter of -a 10, the client sends 10 requests per second.

After that. I added code to my python file to calculate the percentage of active time during the process, yielding me a percentage of about…



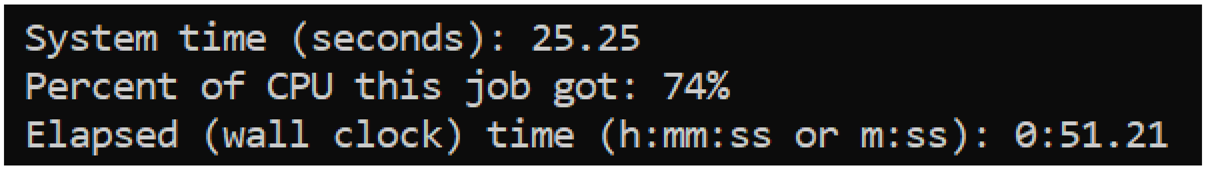
```
import pandas as pd
from matplotlib import pyplot as plt
import numpy as np
columns=["request id","send time", "request length"," recieved time","completed time"]
df=pd.read_csv("request_times.csv",usecols=columns,sep='[,:]',engine='python')
df['processing time']=df['completed time']-df[' recieved time']


active_time=df['request length'].sum()
total_time=df['completed time'].iloc[-1]-df[' recieved time'][0]
print(len(df['processing time'])/total_time)
print(active_time/total_time)
```



```
yannarif03@LAPTOP-0ISR6TGR:~/cs350/hw1$ python3 eval2.py
9.818247546886138
0.7985003412331799
```

80%

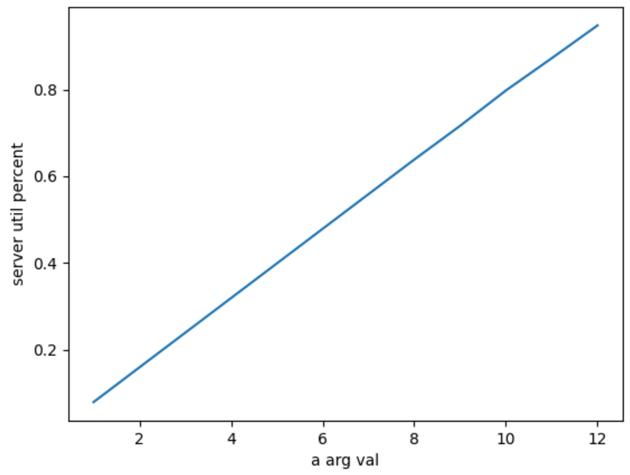
Then by using the time function in bash, i get:



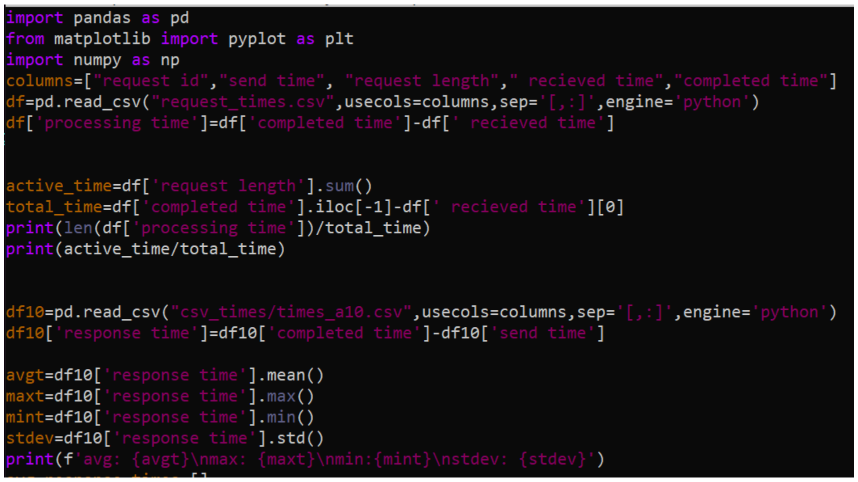
```
System time (seconds): 25.25
Percent of CPU this job got: 74%
Elapsed (wall clock) time (h:mm:ss or m:ss): 0:51.21
```

74%. It's not a perfect match but it's fairly close together.

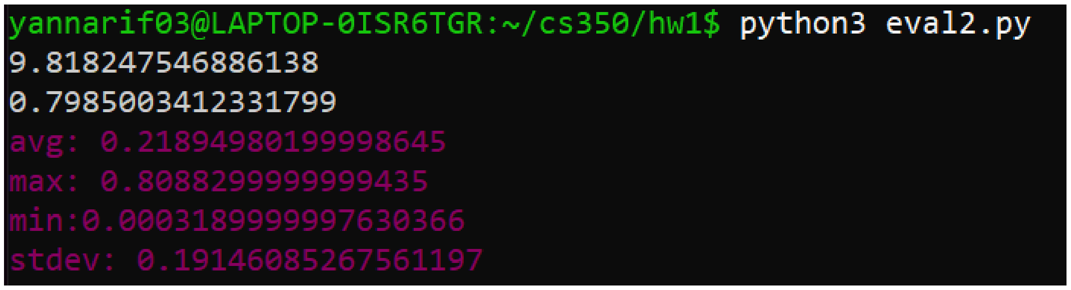
We then update the bash script again to run all 12 versions of the server and store the server output in one set of csv files, and the bash output in another set of .txt files. Using bash commands i was able to quickly gather the cpu% from the txt files and plot them using pyplot:



Then to get the stats on the a=10 server: new python code



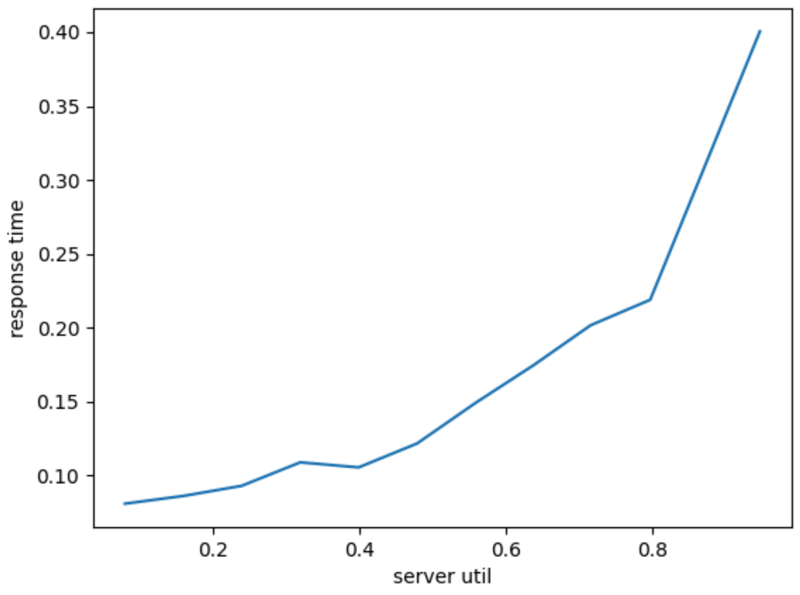
```
import pandas as pd
from matplotlib import pyplot as plt
import numpy as np
columns=["request id","send time", "request length"," recieved time","completed time"]
df=pd.read_csv("request_times.csv",usecols=columns,sep='[,:]',engine='python')
df['processing time']=df['completed time']-df[' recieved time']


active_time=df['request length'].sum()
total_time=df['completed time'].iloc[-1]-df[' recieved time'][0]
print(len(df['processing time'])/total_time)
print(active_time/total_time)


df10=pd.read_csv("csv_times/times_a10.csv",usecols=columns,sep='[,:]',engine='python')
df10['response time']=df10['completed time']-df10['send time']

avgt=df10['response time'].mean()
maxt=df10['response time'].max()
mint=df10['response time'].min()
stdev=df10['response time'].std()
print(f'avg: {avgt}\nmax: {maxt}\nmin:{mint}\nstdev: {stdev}')
```



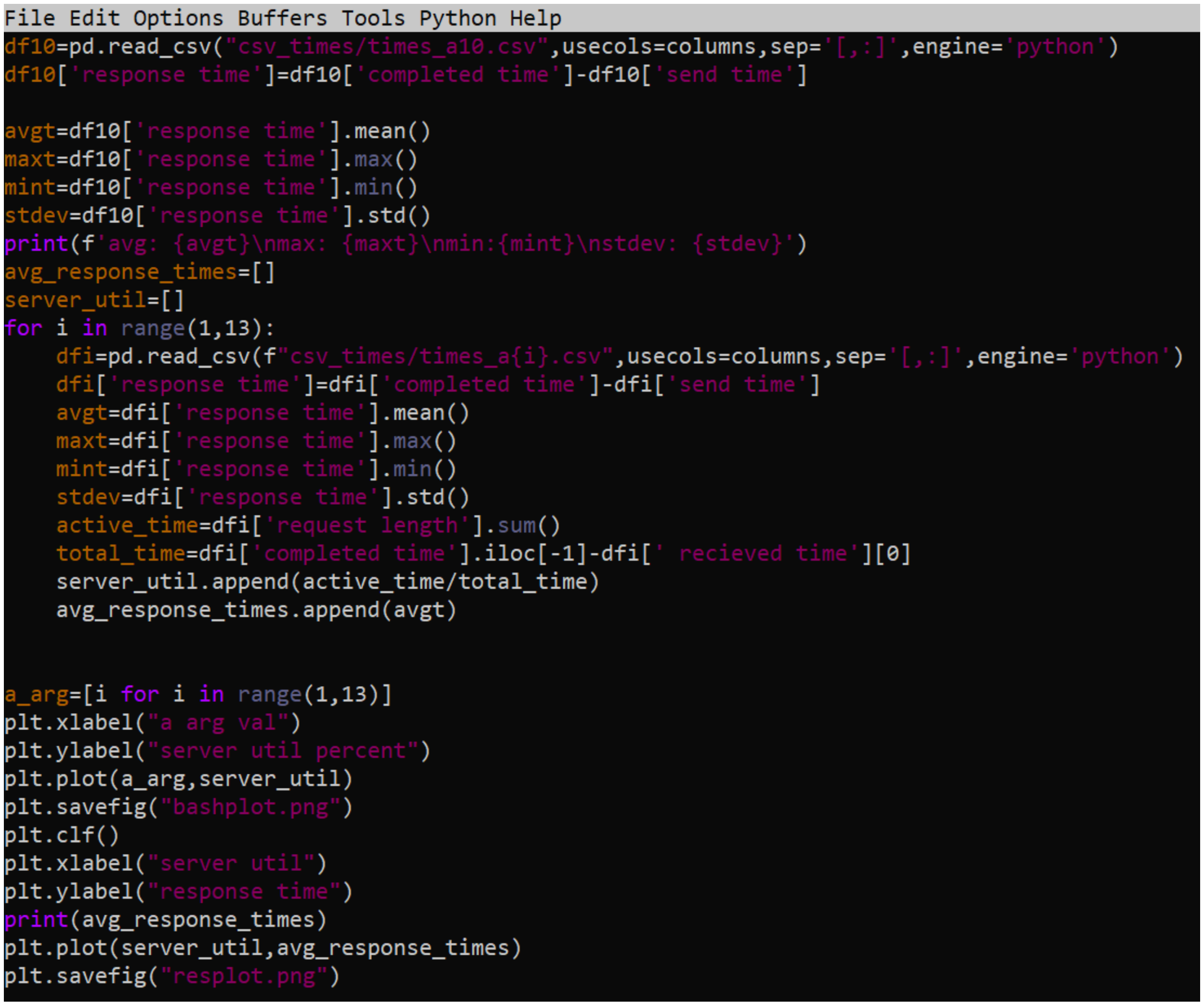
```
yannarif03@LAPTOP-0ISR6TGR:~/cs350/hw1$ python3 eval2.py
9.818247546886138
0.7985003412331799
avg: 0.21894980199998645
max: 0.8088299999999435
min:0.00031899999997630366
stdev: 0.19146085267561197
```

We can then use that same data in python to calculate and plot the response times with respect to the -a parameter:

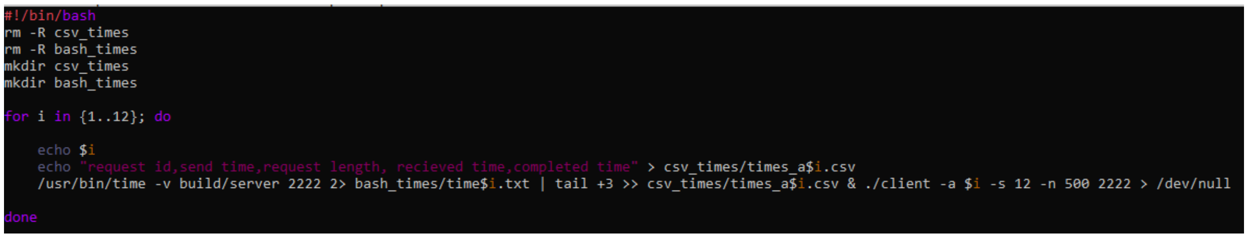
RESPONSE TIME GRAPH:



We discover that there is an exponential relationship between an increased utilization of the server and the average response time of each request.



```python
df10=pd.read_csv("csv_times/times_a10.csv",usecols=columns,sep='[,:]',engine='python')
df10['response time']=df10['completed time']-df10['send time']

avgt=df10['response time'].mean()
maxt=df10['response time'].max()
mint=df10['response time'].min()
stdev=df10['response time'].std()
print(f'avg: {avgt}\nmax: {maxt}\nmin:{mint}\nstdev: {stdev}')
avg_response_times=[]
server_util=[]
for i in range(1,13):
    dfi=pd.read_csv(f"csv_times/times_a{i}.csv",usecols=columns,sep='[,:]',engine='python')
    dfi['response time']=dfi['completed time']-dfi['send time']
    avgt=dfi['response time'].mean()
    maxt=dfi['response time'].max()
    mint=dfi['response time'].min()
    stdev=dfi['response time'].std()
    active_time=dfi['request length'].sum()
    total_time=dfi['completed time'].iloc[-1]-dfi[' recieved time'][0]
    server_util.append(active_time/total_time)
    avg_response_times.append(avgt)


a_arg=[i for i in range(1,13)]
plt.xlabel("a arg val")
plt.ylabel("server util percent")
plt.plot(a_arg,server_util)
plt.savefig("bashplot.png")
plt.clf()
plt.xlabel("server util")
plt.ylabel("response time")
print(avg_response_times)
plt.plot(server_util,avg_response_times)
plt.savefig("resplot.png")
```



```bash
#!/bin/bash
rm -R csv_times
rm -R bash_times
mkdir csv_times
mkdir bash_times

for i in {1..12}; do

    echo $i
    echo "request id,send time,request length, recieved time,completed time" > csv_times/times_a$i.csv
    /usr/bin/time -v build/server 2222 2> bash_times/time$i.txt | tail +3 >> csv_times/times_a$i.csv & ./client -a $i -s 12 -n 500 2222 > /dev/null

done
```

Eval 1 Part 2 Solution B (python scripts)

The following solution uses mainly python scripts

(a) THROUGHPUT

Receipt/sent timestamp of the first request:5862.315197003
Completion timestamp of the last request: 5913.654858569

Therefore, to get the time elapsed, we take the difference.
Therefore, we get 5913.654858569 - 5862.315197003 = **51.339661566 s**

Therefore, the throughput is 500/51.339 since that is the requests per second since there are 500 requests.
Therefore, 500/51.339 =**9.73918463546 req/sec**

(b) UTILIZATION

The total time is the same as the time elapsed, we take the difference. Therefore, we get 5913.654858569 - 5862.315197003 = **51.339661566 seconds**

40.66409 seconds is the **busy time** as calculated on my python script. I wrote a script that sums all the Request lengths by running server and putting the content into an output.txt file.

Now, to get the utilization, which is busy time/total time, we do **40.66409/51.339661566 = 0.79205995442 = 79.205995442%**

Total Utilization: 79.2% -> 79%

CODE:

```
import pandas as pd

# Define the column names
column_names = [
        'RequestID',
        'ClientRequestTimestamp',
        'ClientRequestLength',
        'ReceiptTimestamp',
        'CompletionTimestamp'
]

# Initialize empty lists to store data
data = {col: [] for col in column_names}
```

```
# Read the text file line by line
with open('output.txt', 'r') as file:
        for line in file:
        # Split the line into t
        t = line.strip().split(',')
        if len(t) != 4:
        continue

        # Extract data and append to respective lists
        request_id_and_timestamp, length, timestamp2, completion_timestamp = t
        request_id, client_request_timestamp = request_id_and_timestamp.split(':')

        data['RequestID'].append(request_id.strip())
        data['ClientRequestTimestamp'].append(float(client_request_timestamp.strip()))
        data['ClientRequestLength'].append(float(length.strip()))
        data['ReceiptTimestamp'].append(float(timestamp2.strip()))
        data['CompletionTimestamp'].append(float(completion_timestamp.strip()))  #

# Create a Pandas DataFrame
df = pd.DataFrame(data)

# calculating the total busy time which is consistent for all files
total_busy_time = 40.664097

# Calculate the total time (last CompletionTimestamp - first ClientRequestTimestamp)
total_time = df['CompletionTimestamp'].iloc[-1] - df['ClientRequestTimestamp'].iloc[0]

# Calculate utilization
utilization = total_busy_time / total_time

# Calculating the response time for each request
# First, calculate the queueing time
queueingTime = df['ReceiptTimestamp'] - df['ClientRequestTimestamp']
# Then, calculate the service time to get the total response time
serviceTime = df['CompletionTimestamp'] - df['ReceiptTimestamp']
df['ResponseTime'] = queueingTime + serviceTime

# Calculate statistics for response times
average_response_time = df['ResponseTime'].mean()
std_dev_response_time = df['ResponseTime'].std()
max_response_time = df['ResponseTime'].max()
min_response_time = df['ResponseTime'].min()

# Display the calculated values
print("Total Busy Time:", total_busy_time)
print("Total Time:", total_time)
print("Utilization:", utilization)
print("Average Response Time:", average_response_time)
print("Standard Deviation of Response Time:", std_dev_response_time)
print("Maximum Response Time:", max_response_time)
print("Minimum Response Time:", min_response_time)
```

(c) The result of the statistics were:

```
Command being timed: "./server 2222"
	User timroot@85153189150e:/workspace/build# e (seconds): 40.71
	System time (seconds): 0.02
	Percent of CPU this job got: 79%
	Elapsed (wall clock) time (h:mm:ss or m:ss): 0:51.33
	Average shared text size (kbytes): 0
	Average unshared data size (kbytes): 0
	Average stack size (kbytes): 0
	Average total size (kbytes): 0
	Maximum resident set size (kbytes): 8248
	Average resident set size (kbytes): 0
	Major (requiring I/O) page faults: 1
	Minor (reclaiming a frame) page faults: 1115
	Voluntary context switches: 134
	Involuntary context switches: 113
	Swaps: 0
	File system inputs: 0
	File system outputs: 0
	Socket messages sent: 0
	Socket messages received: 0
	Signals delivered: 0
	Page size (bytes): 4096
	Exit status: 0
```

And, from that, we can extract the percentage of CPU this job got: 79% as highlighted in bold, which is accurate to what I calculated in (b), therefore it matches.

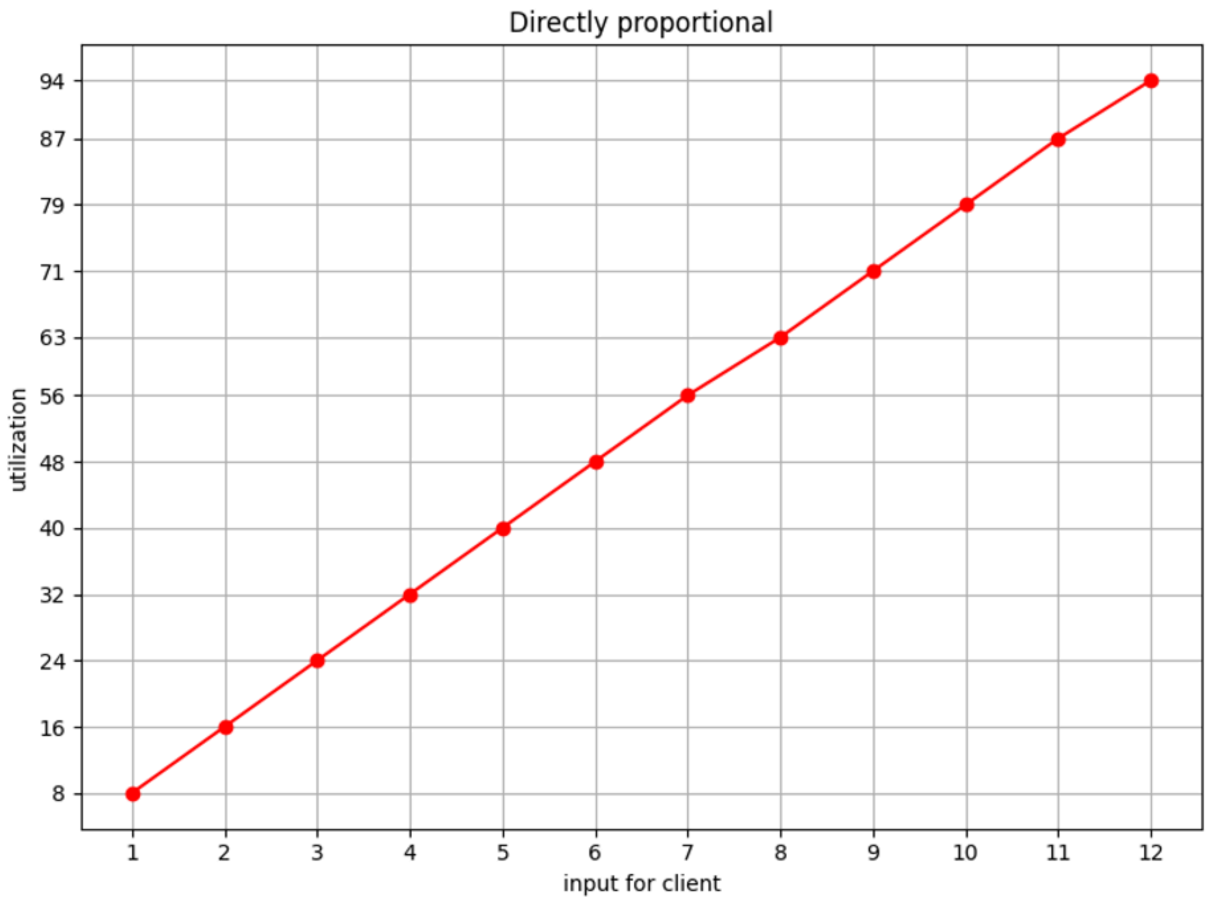
(d)

Utilization for each:

a=1:0.0798892662441442 = 8%
a=2: 0.15973050592890847 = 16%
a= 3: 0.23954017797102417 = 24%
a=4: 0.3193053949526752 = 32%
a=5: 0.3990886326787358 = 40%
a=6: 0.47875465361269953 = 48%
a=7: 0.5584589903120173 = 56%

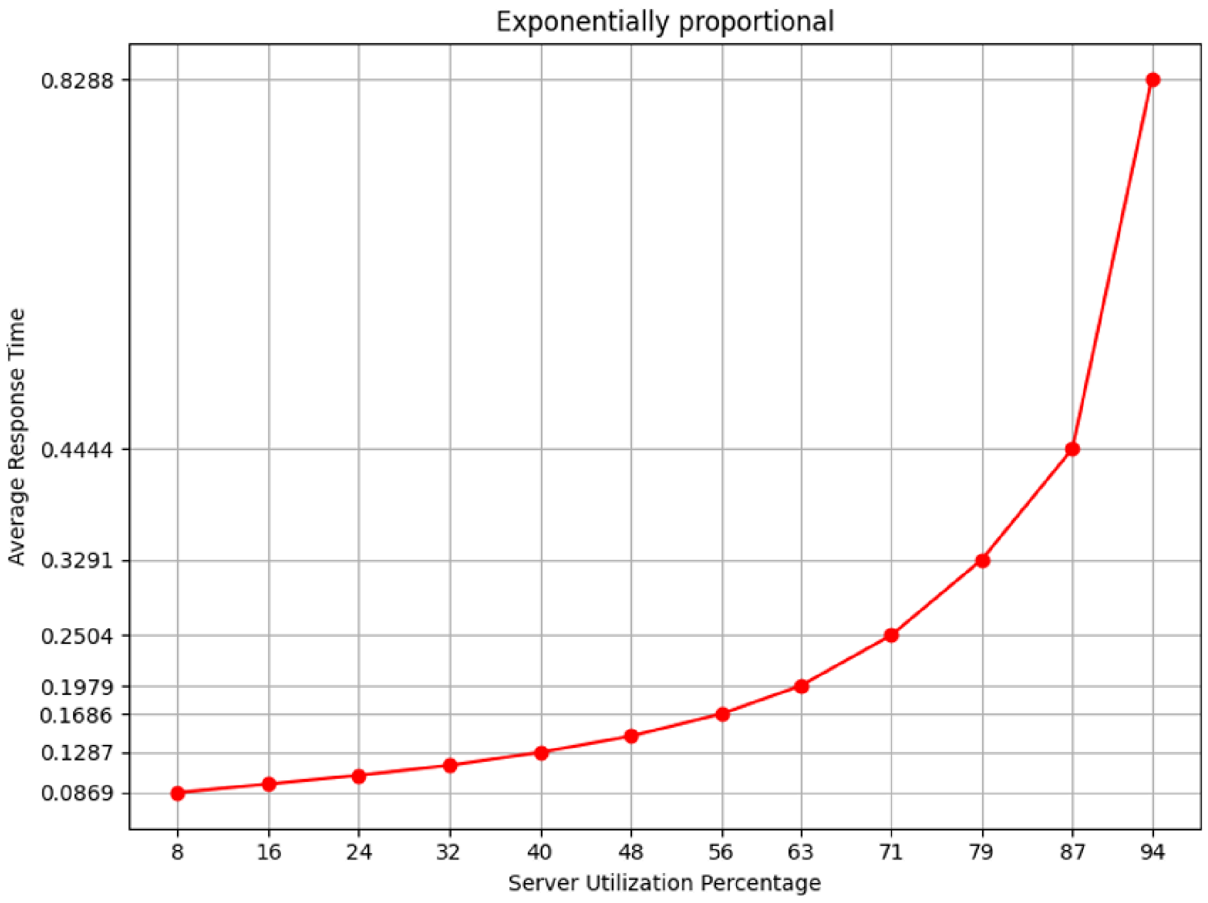
a=8: 0.6374793408368947 = 63%
a=9: 0.7154083665035419 = 72%
a=10: 0.7923298236879591 = 79%
a=11: 0.8686553294998192 = 87%
a=12: 0.9438751679324104 = 94%

From running the server and client 12 times, we can see from the graph below that there is a steady increase rate, making the utilization directly proportional to the arrival rate (input for client), since when arrival rate increases, utilization increases.



(e) Total Busy Time: 40.664097
Total Time: 51.32218399999874
Utilization: 0.7923298236879591
Average Response Time: 0.3291007599999648
Standard Deviation of Response Time: 0.3148110082315718
Maximum Response Time: 1.4485970000005182
Minimum Response Time: 0.0004439999993337551
All coded on python.

(f) The below graph plotted on python is the avg response time vs server utilization. As seen below, the relation between the avg response time and server utilization is exponentially proportional. With an increase in server utilization, the greater the avg response time and it increases on and on (as discussed in class as well).


Exponentially proportional
Average Response Time
Server Utilization Percentage