Ravella, Kiran
09/17/2019
Harvard University Extension - Principles of Big Data Processing e88
Homework 2: Vertical and Horizontal Scaling , Shared State Management

This document is a template for your solutions submission. You are free to add additional information in this submission if you would like. Extra screenshots and extra documentation are appreciated. Screenshots must always be viewable. If a screenshot is too blurry or chopped off in a key area you will not receive full credit for it.

Make sure to also submit all your source code (.java files , .py files or whatever language you are using) - in a separate archive, named <LastName>_<FirstName>_HW2.zip

---

Please identify which problems were completed. If any were incomplete, please identify where you encountered problems.

Problem 1: 100% complete
Problem 2: 100% complete
Problem 3: 100% complete
Problem 4: 100% complete
Problem 5 Bonus: 0% Complete

Problem 1: CPU Analysis [points: 25]
Paste your source code into the following area [10 points]

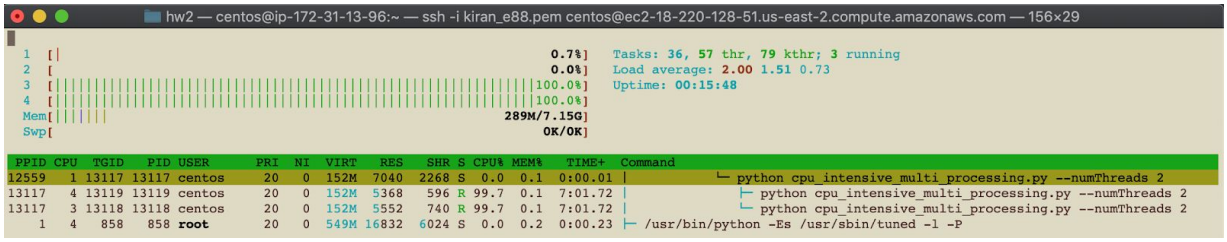
```
# This program creates a CPU-Intensive workload.
import argparse
import os
import time
import multiprocessing

#Using argparse library to parse the input arguments.
prog = "hw2"
desc = "Starts a specified number of threads that do CPU-Intesive work"
parser = argparse.ArgumentParser(prog=prog, description=desc)
parser.add_argument('--numThreads', '-n', default=4, type=int)

parsed_args = parser.parse_args()
numThreads = parsed_args.numThreads

print("Number of Threads: ", numThreads)

#Function that calculates the Fibonacci series
def Fibonacci(n):
    if n==1:
        return 0
    elif n==2:
        return 1
    else:
        return Fibonacci(n-1)+Fibonacci(n-2)

#Function that loops infinitely and puts load on the CPU.
```

```
def create_cpu_load(thread_num):
    while(True):
        time.sleep(1)
        print("In Thread :", thread_num, ", Process ID: ", os.getpid())
        Fibonacci(100)
        print("In Thread:", thread_num," ,Completed Fibonacci")

#The following code creates the number of threads specified in the input argument
threads= []
for thread_count_id in range(numThreads):
    t = multiprocessing.Process(target = create_cpu_load, args=(thread_count_id,))
    threads.append(t)
    t.start()

for current_thread in threads:
    current_thread.join()
```

Provide your table or graphs demonstrating the results of running this code with 2, 4, and 16 threads on a 4 CPU machine: [5 points]

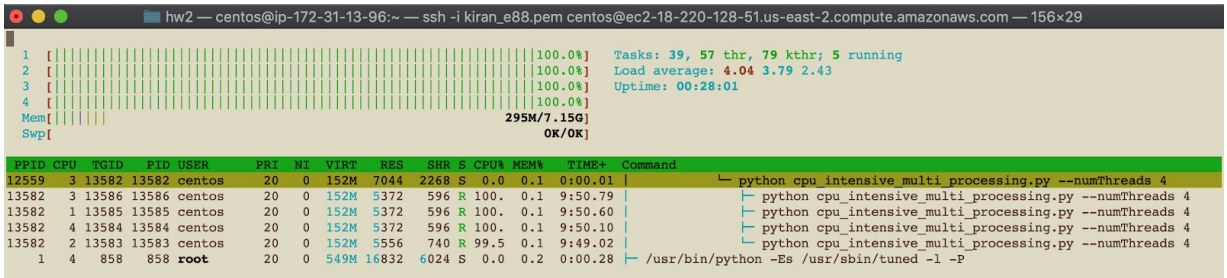
Running Code with 2 Threads on a c4.xlarge (4 CPU) instance:

```
[[centos@ip-172-31-13-96 ~]$ python cpu_intensive_multi_processing.py --numThreads 2
('Number of Threads: ', 2)
('In Thread :', 0, ', Process ID: ', 13118)
('In Thread :', 1, ', Process ID: ', 13119)
```



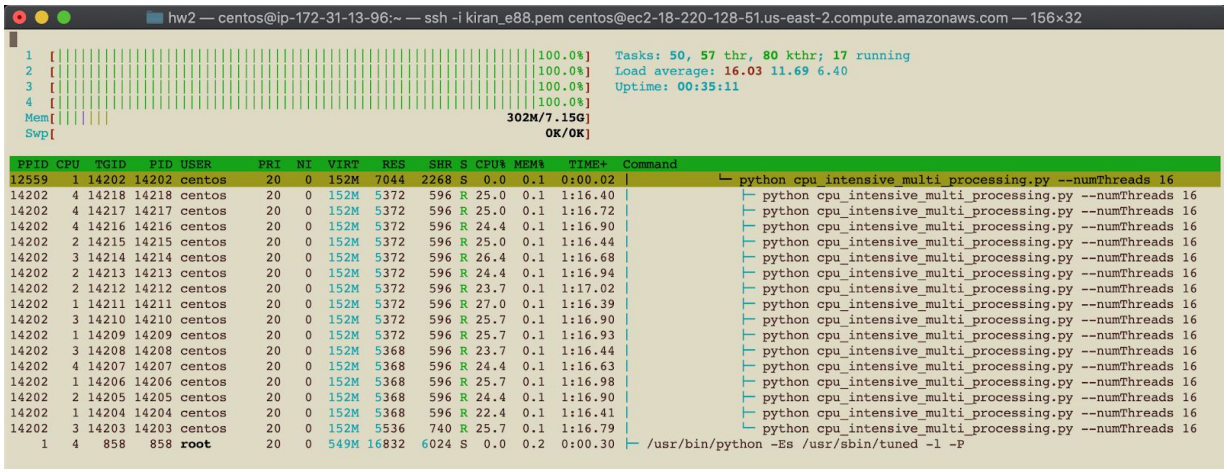
Running Code with 4 Threads on a c4.xlarge (4 CPU) instance:

```
[[centos@ip-172-31-13-96 ~]$ python cpu_intensive_multi_processing.py --numThreads 4
('Number of Threads: ', 4)
('In Thread :', 0, ', Process ID: ', 13583)
('In Thread :', 1, ', Process ID: ', 13584)
('In Thread :', 2, ', Process ID: ', 13585)
('In Thread :', 3, ', Process ID: ', 13586)
```



Running Code with 16 Threads on a c4.xlarge (4 CPU) instance:

```
[[centos@ip-172-31-13-96 ~]$ python cpu_intensive_multi_processing.py --numThreads 16
('Number of Threads: ', 16)
('In Thread :', 2, ', Process ID: ', 14205)
('In Thread :', 0, ', Process ID: ', 14203)
('In Thread :', 1, ', Process ID: ', 14204)
('In Thread :', 4, ', Process ID: ', 14207)
('In Thread :', 6, ', Process ID: ', 14209)
('In Thread :', 3, ', Process ID: ', 14206)
('In Thread :', 5, ', Process ID: ', 14208)
('In Thread :', 8, ', Process ID: ', 14211)
('In Thread :', 9, ', Process ID: ', 14212)
('In Thread :', 10, ', Process ID: ', 14213)
('In Thread :', 11, ', Process ID: ', 14214)
('In Thread :', 12, ', Process ID: ', 14215)
('In Thread :', 13, ', Process ID: ', 14216)
('In Thread :', 7, ', Process ID: ', 14210)
('In Thread :', 14, ', Process ID: ', 14217)
('In Thread :', 15, ', Process ID: ', 14218)
```



Number of Threads	**Load Average over last minute**
2	**2**
4	**4.04**
16	**16.03**

What can you summarize about the results? [3 points]

When the number of threads are less than the number of CPUs, then the program is executed in the same number of CPUs as the number of threads and those CPUs are fully utilized. The

remaining CPUs are not utilized. In our case, we ran the program with 2 threads and 2 CPUs were 100% Utilized whereas the rest were close to 0% utilization.

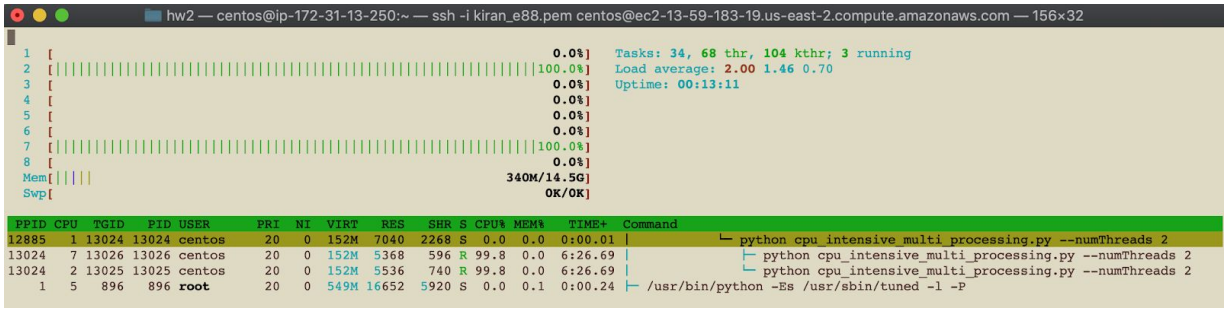
When the number of threads is equal to the number of CPUs, all the CPUs are fully utilized. In our case, we ran the program with 4 threads and all 4 CPUs were 100% utilized.

When the number of threads is more than the number of CPUs, then all the CPUs are fully utilized and the load average indicates that the CPUs are overloaded and that there are processes waiting for CPU time. In our case, we ran with 16 threads, so all 4 CPUs were 100% utilized and there were a lot of processes waiting for CPU time, which is more than 1200%.

Provide your table or graphs demonstrating the results of running this code with 2, 4, and 16 threads on an 8 CPU machine: [5 points]

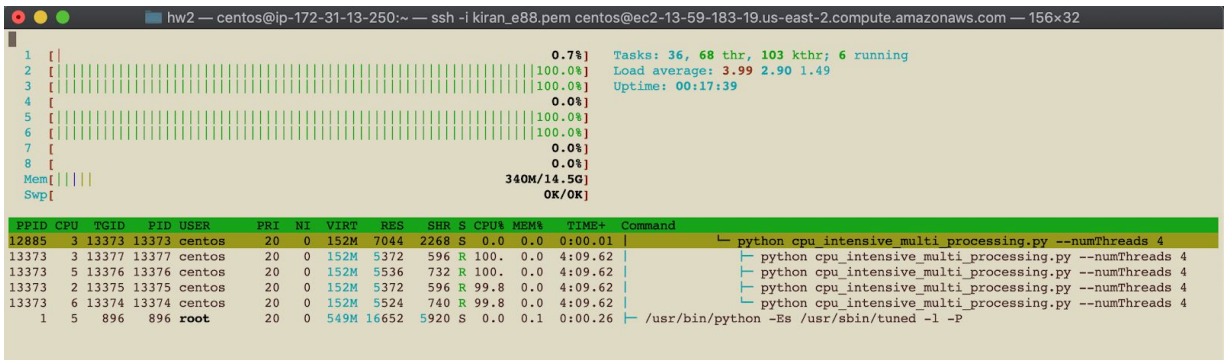
Running Code with 2 Threads on a c4.2xlarge (8 CPU) instance:

```
[centos@ip-172-31-13-250 ~]$ python cpu_intensive_multi_processing.py --numThreads 2
('Number of Threads: ', 2)
('In Thread :', 0, ', Process ID: ', 13025)
('In Thread :', 1, ', Process ID: ', 13026)
```

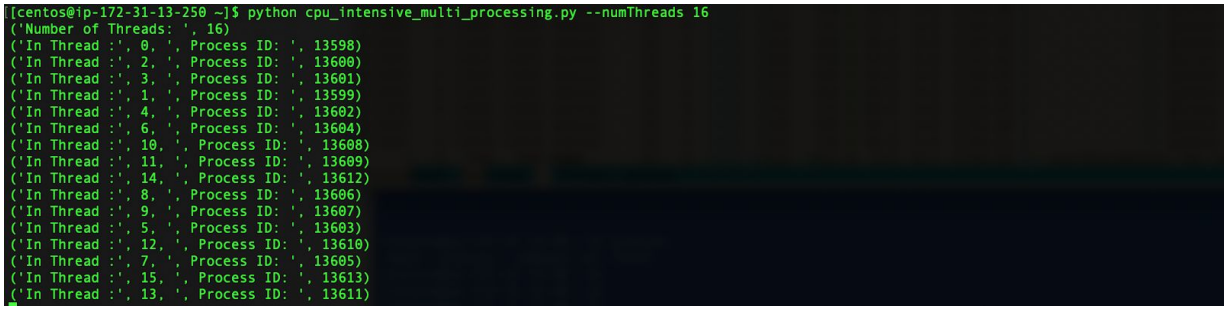


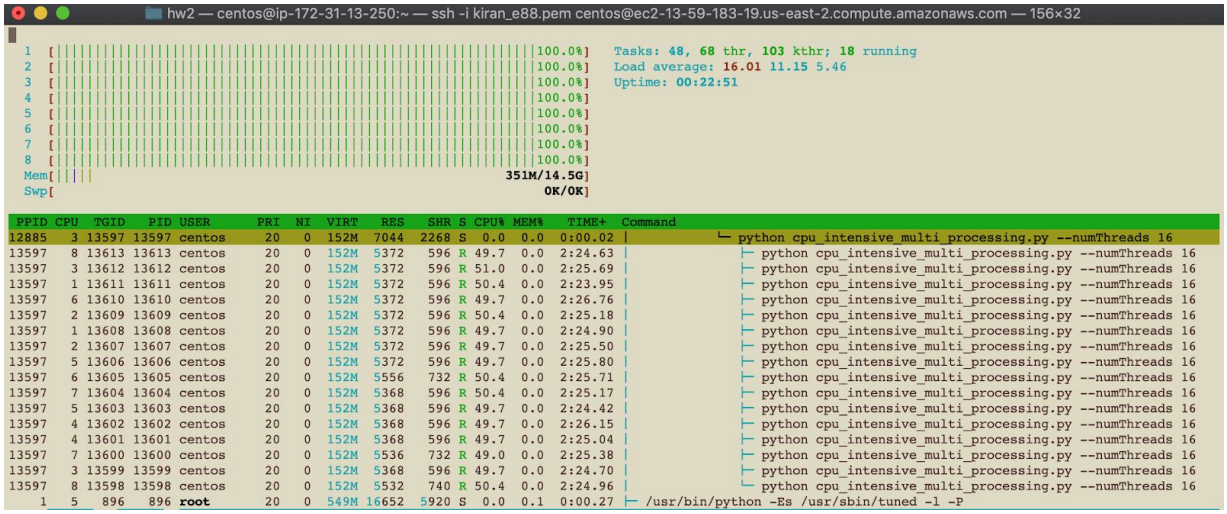
Running Code with 4 Threads on a c4.2xlarge (8 CPU) instance:

```
[centos@ip-172-31-13-250 ~]$ python cpu_intensive_multi_processing.py --numThreads 4
('Number of Threads: ', 4)
('In Thread :', 0, ', Process ID: ', 13374)
('In Thread :', 2, ', Process ID: ', 13376)
('In Thread :', 1, ', Process ID: ', 13375)
('In Thread :', 3, ', Process ID: ', 13377)
```



Running Code with 16 Threads on a c4.2xlarge (8 CPU) instance:





Number of Threads	Load Average over last minute
2	2
4	3.99
16	16.01

What can you summarize about the results? How does a 4 CPU machine compare to an 8 CPU machine in this exercise? [2 points]

Just like in case of 4 CPU machine, when the number of threads is less than or equal to the number of CPUs, then all threads get a full CPU and are able to utilize those CPUs fully. So in our case, when the number of threads were 2 and 4, then 2 and 4 CPUs were fully utilized whereas the rest were not utilized at all. Also, when we ran the program with 16 threads, all 8 CPUs were fully utilized and there was processes waiting for CPU time due to overloading.

4CPU vs 8 CPU

I think the main difference would be when we ran the program with 16 threads, in case of 4 CPU machine, only 4 threads would run at any time and the rest of 12 threads would be waiting for CPU time. Whereas in case of a 8 CPU machine, 8 threads would be running and only 8 threads would be waiting for CPU time. So, it would speed up the processing.

When we ran the program with 2 and 4 threads, both machines had equal or more CPUs than the number of threads and there were no other programs running on those machines, so all threads got full CPUs and I don't think the processing would have been any faster on the 8 CPU compared to 4 CPU machine.

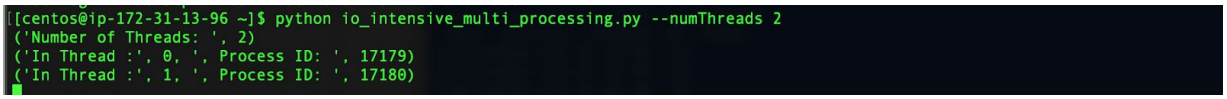
Problem 2: I/O Analysis [points: 25]

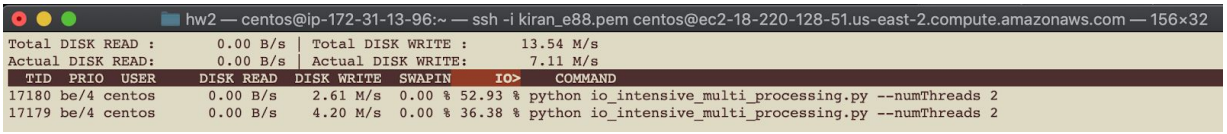
Paste your source code into the following area. Make sure you clarify what you did to programmatically create an I/O intensive process. [10 points]

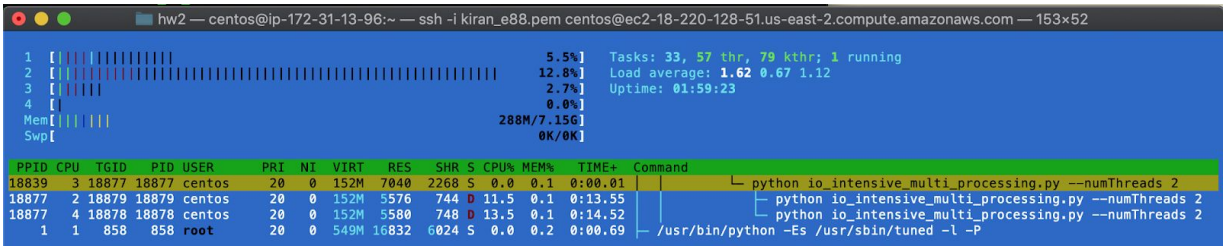
```
# This program creates a IO-Intensive workload.
# It calculates the Fibonacci series and writes a lot of text to files on disk

import argparse
import os
import time
import multiprocessing


#Using argparse library to parse the input arguments.
prog = "hw2"
desc = "Starts a specified number of threads that do CPU-Intesive work"
parser = argparse.ArgumentParser(prog=prog, description=desc)
parser.add_argument('--numThreads', '-n', default=4, type=int)

parsed_args = parser.parse_args()
numThreads = parsed_args.numThreads

print("Number of Threads: ", numThreads)

#Function that calculates the Fibonacci series
def Fibonacci_to_file(n):
    file_to_process = open("output.txt","w")
    file_to_process.write("Inside fibonacci function")
```

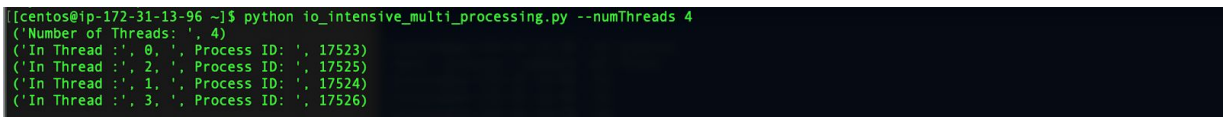
```
    file_to_process.close()
    if n==1:
        return 0
    elif n==2:
        return 1
    else:
        return Fibonacci_to_file(n-1)+Fibonacci_to_file(n-2)

#Function that loops infinitely and puts load on the CPU.
def create_cpu_load(thread_num):
    while(True):
        time.sleep(1)
        print("In Thread :", thread_num, ", Process ID: ", os.getpid())
        Fibonacci_to_file(100)
        print("In Thread:", thread_num," ,Completed Fibonacci")

#The following code creates the number of threads specified in the input argument
threads= []
for thread_count_id in range(numThreads):
    t = multiprocessing.Process(target = create_cpu_load, args=(thread_count_id,))
    threads.append(t)
    t.start()

for current_thread in threads:
    current_thread.join()
```

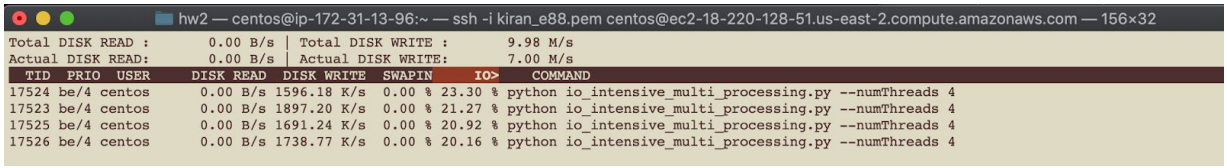
Provide your table or graphs demonstrating the results of running this code with 2, 4 and 16 threads on a 4 CPU machine: [5 points]

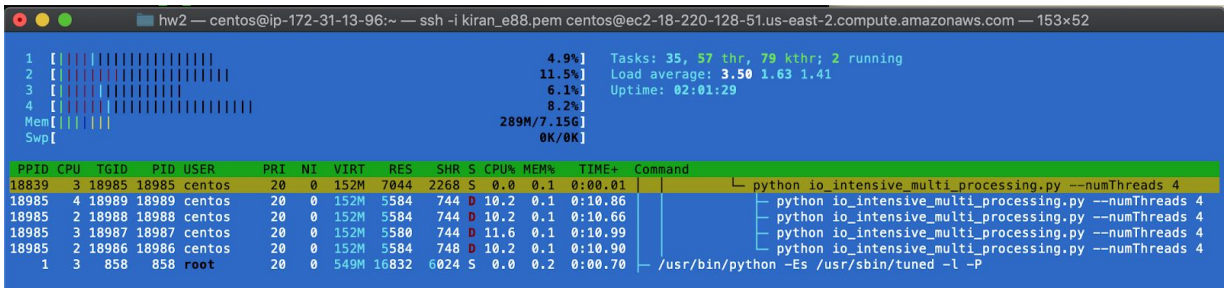


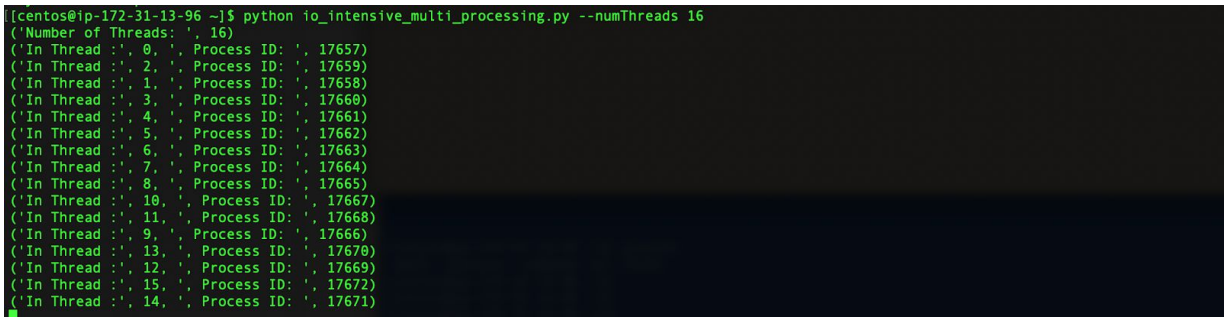


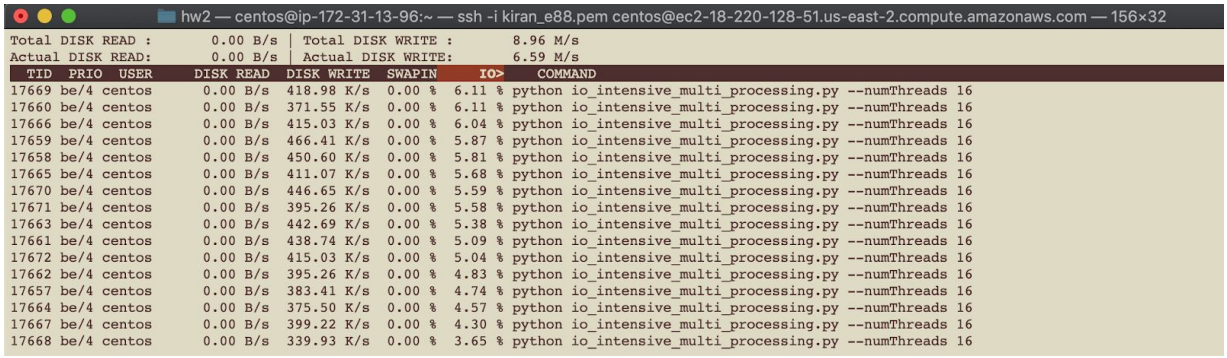


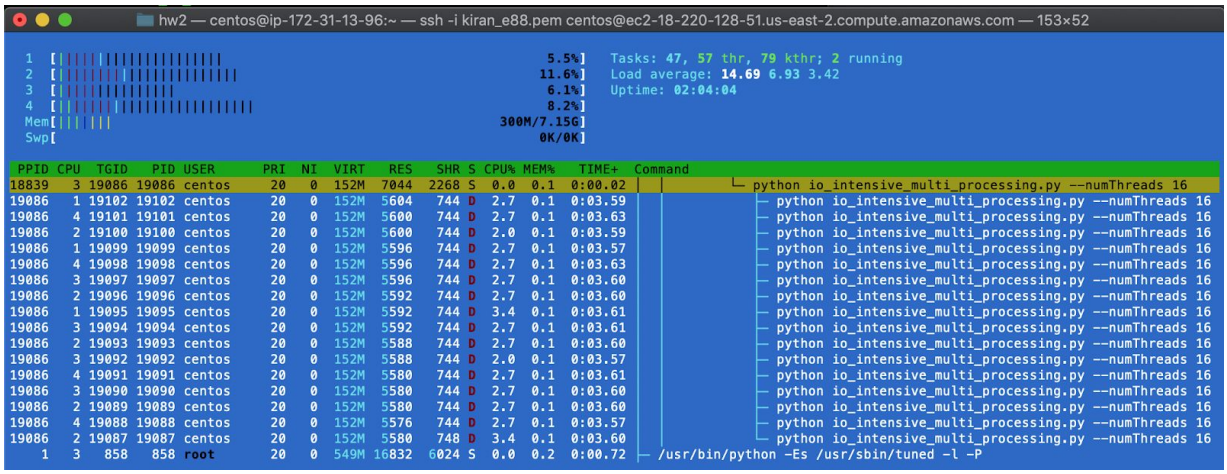



hw2 — centos@ip-172-31-13-96:~ — ssh -i kiran_e88.pem centos@ec2-18-220-128-51.us-east-2.compute.amazonaws.com — 156×32
Total DISK READ : 0.00 B/s | Total DISK WRITE : 9.98 M/s
Actual DISK READ: 0.00 B/s | Actual DISK WRITE: 7.00 M/s
TID PRIO USER DISK READ DISK WRITE SWAPIN IO> COMMAND
17524 be/4 centos 0.00 B/s 1596.18 K/s 0.00 % 23.30 % python io_intensive_multi_processing.py --numThreads 4
17523 be/4 centos 0.00 B/s 1897.20 K/s 0.00 % 21.27 % python io_intensive_multi_processing.py --numThreads 4
17525 be/4 centos 0.00 B/s 1691.24 K/s 0.00 % 20.92 % python io_intensive_multi_processing.py --numThreads 4
17526 be/4 centos 0.00 B/s 1738.77 K/s 0.00 % 20.16 % python io_intensive_multi_processing.py --numThreads 4


hw2 — centos@ip-172-31-13-96:~ — ssh -i kiran_e88.pem centos@ec2-18-220-128-51.us-east-2.compute.amazonaws.com — 153×52
1 [4.9%]
2 [11.5%]
3 [6.1%]
4 [8.2%]
Mem[289M/7.15G]
Swp[0K/0K]
Tasks: 35, 57 thr, 79 kthr; 2 running
Load average: 3.50 1.63 1.41
Uptime: 02:01:29
PPID CPU TGID PID USER PRI NI VIRT RES SHR S CPU% MEM% TIME+ Command
18839 3 18985 18985 centos 20 0 152M 7044 2268 S 0.0 0.1 0:00.01 └─ python io_intensive_multi_processing.py --numThreads 4
18985 4 18989 18989 centos 20 0 152M 5584 744 D 10.2 0.1 0:10.86 ├─ python io_intensive_multi_processing.py --numThreads 4
18985 2 18988 18988 centos 20 0 152M 5584 744 D 10.2 0.1 0:10.66 ├─ python io_intensive_multi_processing.py --numThreads 4
18985 3 18987 18987 centos 20 0 152M 5580 744 D 11.6 0.1 0:10.99 ├─ python io_intensive_multi_processing.py --numThreads 4
18985 2 18986 18986 centos 20 0 152M 5584 748 D 10.2 0.1 0:10.90 └─ python io_intensive_multi_processing.py --numThreads 4
1 3 858 858 root 20 0 549M 16832 6024 S 0.0 0.2 0:00.70 ├─ /usr/bin/python -Es /usr/sbin/tuned -l -P


[centos@ip-172-31-13-96 ~]$ python io_intensive_multi_processing.py --numThreads 16
('Number of Threads: ', 16)
('In Thread :', 0, ', Process ID: ', 17657)
('In Thread :', 2, ', Process ID: ', 17659)
('In Thread :', 1, ', Process ID: ', 17658)
('In Thread :', 3, ', Process ID: ', 17660)
('In Thread :', 4, ', Process ID: ', 17661)
('In Thread :', 5, ', Process ID: ', 17662)
('In Thread :', 6, ', Process ID: ', 17663)
('In Thread :', 7, ', Process ID: ', 17664)
('In Thread :', 8, ', Process ID: ', 17665)
('In Thread :', 10, ', Process ID: ', 17667)
('In Thread :', 11, ', Process ID: ', 17668)
('In Thread :', 9, ', Process ID: ', 17666)
('In Thread :', 13, ', Process ID: ', 17670)
('In Thread :', 12, ', Process ID: ', 17669)
('In Thread :', 15, ', Process ID: ', 17672)
('In Thread :', 14, ', Process ID: ', 17671)


hw2 — centos@ip-172-31-13-96:~ — ssh -i kiran_e88.pem centos@ec2-18-220-128-51.us-east-2.compute.amazonaws.com — 156×32
Total DISK READ : 0.00 B/s | Total DISK WRITE : 8.96 M/s
Actual DISK READ: 0.00 B/s | Actual DISK WRITE: 6.59 M/s
TID PRIO USER DISK READ DISK WRITE SWAPIN IO> COMMAND
17669 be/4 centos 0.00 B/s 418.98 K/s 0.00 % 6.11 % python io_intensive_multi_processing.py --numThreads 16
17660 be/4 centos 0.00 B/s 371.55 K/s 0.00 % 6.11 % python io_intensive_multi_processing.py --numThreads 16
17666 be/4 centos 0.00 B/s 415.03 K/s 0.00 % 6.04 % python io_intensive_multi_processing.py --numThreads 16
17659 be/4 centos 0.00 B/s 466.41 K/s 0.00 % 5.87 % python io_intensive_multi_processing.py --numThreads 16
17658 be/4 centos 0.00 B/s 450.60 K/s 0.00 % 5.81 % python io_intensive_multi_processing.py --numThreads 16
17665 be/4 centos 0.00 B/s 411.07 K/s 0.00 % 5.68 % python io_intensive_multi_processing.py --numThreads 16
17670 be/4 centos 0.00 B/s 446.65 K/s 0.00 % 5.59 % python io_intensive_multi_processing.py --numThreads 16
17671 be/4 centos 0.00 B/s 395.26 K/s 0.00 % 5.58 % python io_intensive_multi_processing.py --numThreads 16
17663 be/4 centos 0.00 B/s 442.69 K/s 0.00 % 5.38 % python io_intensive_multi_processing.py --numThreads 16
17661 be/4 centos 0.00 B/s 438.74 K/s 0.00 % 5.09 % python io_intensive_multi_processing.py --numThreads 16
17672 be/4 centos 0.00 B/s 415.03 K/s 0.00 % 5.04 % python io_intensive_multi_processing.py --numThreads 16
17662 be/4 centos 0.00 B/s 395.26 K/s 0.00 % 4.83 % python io_intensive_multi_processing.py --numThreads 16
17657 be/4 centos 0.00 B/s 383.41 K/s 0.00 % 4.74 % python io_intensive_multi_processing.py --numThreads 16
17664 be/4 centos 0.00 B/s 375.50 K/s 0.00 % 4.57 % python io_intensive_multi_processing.py --numThreads 16
17667 be/4 centos 0.00 B/s 399.22 K/s 0.00 % 4.30 % python io_intensive_multi_processing.py --numThreads 16
17668 be/4 centos 0.00 B/s 339.93 K/s 0.00 % 3.65 % python io_intensive_multi_processing.py --numThreads 16

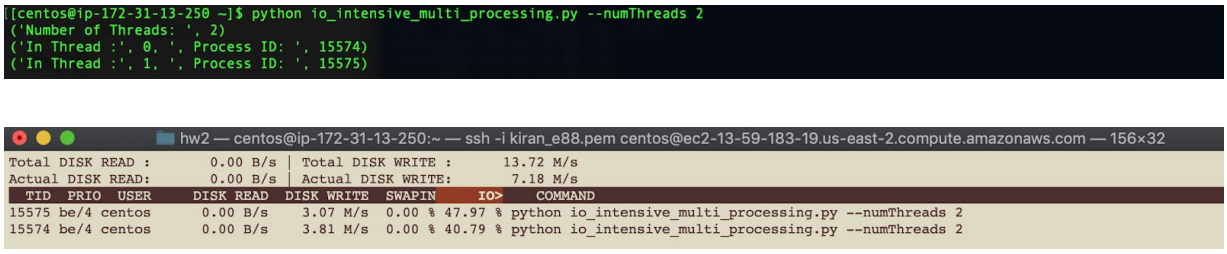


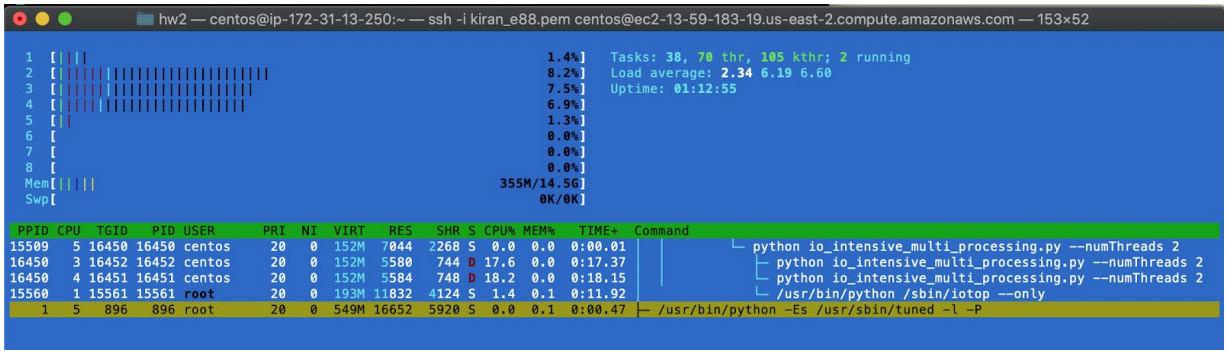
Number of Threads	Actual DISK Write	Load over last min
2	7.11 M/s	1.62
4	7.00 M/s	3.50
16	6.59 M/s	14.69

What can you summarize about the results? [3 points]

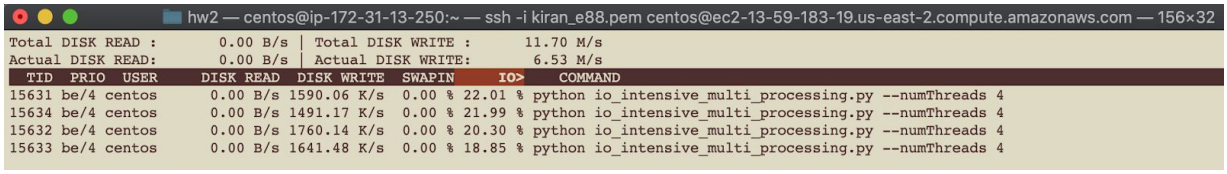
It appears that the actual disk writes is decreasing as the number of threads increases. Also CPUs are not fully utilized even when number of threads is greater than or equal to number of CPUs.

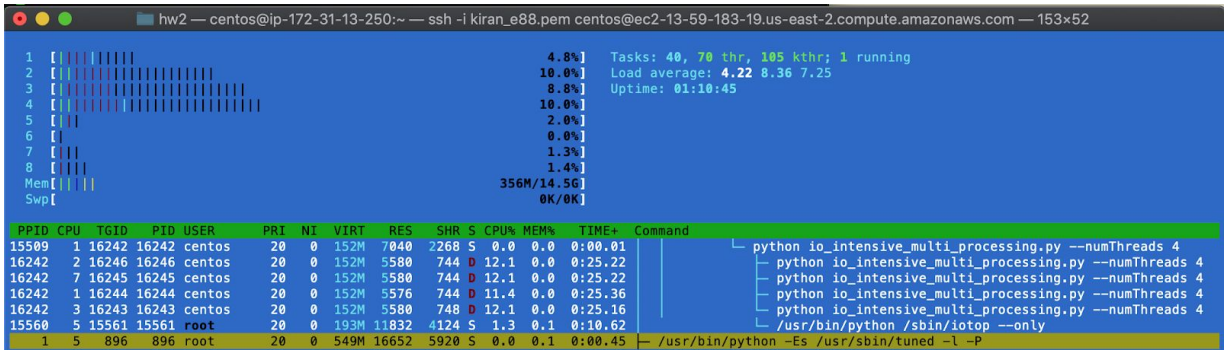
Provide your table or graphs demonstrating the results of running this code with 2, 4 and 16 threads on an 8 CPU machine: [5 points]



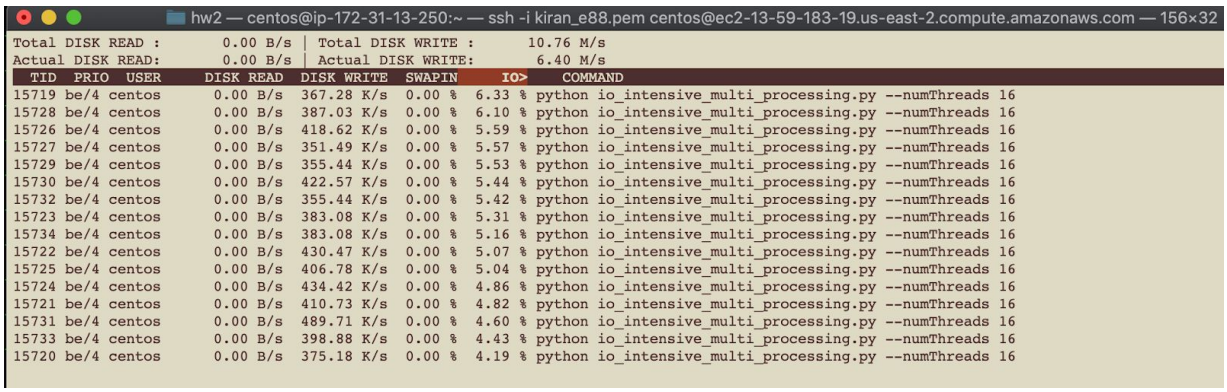


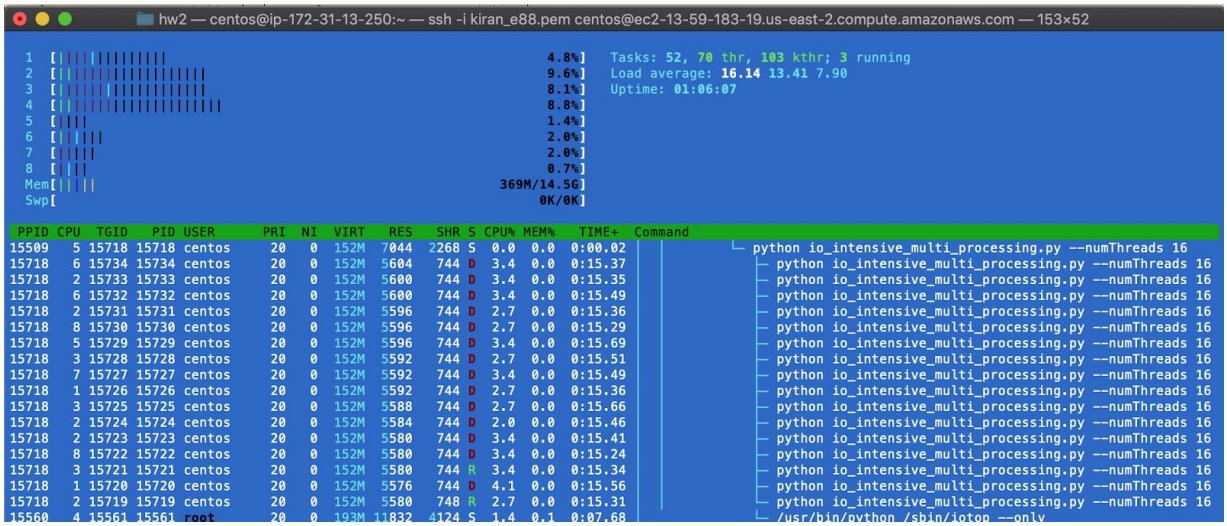

```
[centos@ip-172-31-13-250 ~]$ python io_intensive_multi_processing.py --numThreads 4
('Number of Threads: ', 4)
('In Thread :', 0, ', Process ID: ', 15631)
('In Thread :', 2, ', Process ID: ', 15633)
('In Thread :', 1, ', Process ID: ', 15632)
('In Thread :', 3, ', Process ID: ', 15634)
```






```
[centos@ip-172-31-13-250 ~]$ python io_intensive_multi_processing.py --numThreads 16
('Number of Threads: ', 16)
('In Thread :', 0, ', Process ID: ', 15719)
('In Thread :', 2, ', Process ID: ', 15721)
('In Thread :', 1, ', Process ID: ', 15720)
('In Thread :', 4, ', Process ID: ', 15723)
('In Thread :', 3, ', Process ID: ', 15722)
('In Thread :', 6, ', Process ID: ', 15725)
('In Thread :', 5, ', Process ID: ', 15724)
('In Thread :', 8, ', Process ID: ', 15727)
('In Thread :', 7, ', Process ID: ', 15726)
('In Thread :', 10, ', Process ID: ', 15729)
('In Thread :', 9, ', Process ID: ', 15728)
('In Thread :', 12, ', Process ID: ', 15731)
('In Thread :', 11, ', Process ID: ', 15730)
('In Thread :', 14, ', Process ID: ', 15733)
('In Thread :', 13, ', Process ID: ', 15732)
('In Thread :', 15, ', Process ID: ', 15734)
```





Number of Threads	Actual DISK write	Load over last minute
2	7.18 M/s	2.34
4	6.53 M/s	4.22
16	6.40 M/s	16.14

What can you summarize about the results? How does a 4 CPU machine compare to an 8 CPU machine in this exercise? [2 points]

The results from 4 CPU machine are very similar to 8 CPU machines and it appears we have reached the IO limit in both cases. Also, for both machines as the number of threads increased the actual disk write has decreased.

Problem 3: unique counts [points: 25]

Paste your source code into the following area [10 points]

```
import csv
from dateutil.parser import parse
import time
from multiprocessing import Process, Lock, Manager

files = ['file-input1.csv', 'file-input2.csv', 'file-input3.csv', 'file-input4.csv']

# Following function reads each line in the above files and forms a data structure of all data
def readFile(inputFile, lock, shared_dict):
	lock.acquire()
	with open(inputFile) as csvfile:
		readCSV = csv.reader(csvfile, delimiter=',')
		for row in readCSV:
			dt = parse(row[1])
			dt_str = str(dt.date())+":"+str(dt.time().hour)
			url = row[2]
			user = row[3]
			if dt_str in shared_dict:
				if url in shared_dict[dt_str]:
					if user in shared_dict[dt_str][url]:
						dictCopy = shared_dict[dt_str][url]
						dictCopy[user] = dictCopy[user]+1
						dictSecondCopy = shared_dict[dt_str]
						dictSecondCopy[url] = dictCopy
						shared_dict[dt_str] = dictSecondCopy
						shared_dict[dt_str][url] = dictCopy
					else:
						### First modify at the user level.
						dictCopy = shared_dict[dt_str][url]
						dictCopy[user] = 1
						### Next modify at the URL level
						dictSecondCopy = shared_dict[dt_str]
						dictSecondCopy[url] = dictCopy
						shared_dict[dt_str] = dictSecondCopy
						shared_dict[dt_str][url] = dictCopy

				else:
					dictCopy = shared_dict[dt_str]
					dictCopy[url] = {user: 1}
					shared_dict[dt_str] = dictCopy
			else:
				shared_dict[dt_str] = {url: {user: 1}}
	lock.release()

manager = Manager()
shared_dict = manager.dict()
lock = Lock()

threads = []
for inputFile in files:
	procs = Process(target=readFile, args=[inputFile, lock, shared_dict])
	threads.append(procs)
	procs.start()

for t in threads : t.join()


####Query 1 ######
print("############################ Query 1 OUTPUT ################################")
for key in sorted(shared_dict.keys()):
	print (key, len(shared_dict[key]))
```

```
####Query 2 ######

print("############################ Query 2 OUTPUT ################################")
for key in sorted(shared_dict.keys()):
        for url in shared_dict[key]:
                print (key, url , len(shared_dict[key][url]))

####Query 3 ######

print("############################ Query 3 OUTPUT ################################")

for key in sorted(shared_dict.keys()):
        for url in shared_dict[key]:
                counter = 0
                for user in shared_dict[key][url]:
                        counter = counter + shared_dict[key][url][user]
                print (key, url , counter)
```

Explain your choice of the data structures for shared state management [5 points]

I chose a nested dictionary with the following format:
{date : { URL : { user : clicks}}}

Using this data structure allows us to answer all the following queries using the same program/data structure in an efficient manner.

What are the results of your queries for the following specified keys? [10 points]
The expected output for the first value is provided for your reference.

Query 1:
<date_hour>, <url_count>

```
2019-09-12:13, 185
2019-09-12:14, 186
2019-09-12:15, 185
2019-09-12:16, 190
2019-09-12:17, 189
```

Query 2
<date:hour:url>, unique_user_count

```
2019-09-12:02:http://example.com/?url=003, 1
2019-09-12:02:http://example.com/?url=004, 3
2019-09-12:02:http://example.com/?url=005, 4
2019-09-12:02:http://example.com/?url=006, 10
```

Query 3
<date:hour:url>, event_count

```
2019-09-12:02:http://example.com/?url=003, 1
2019-09-12:02:http://example.com/?url=004, 3
2019-09-12:02:http://example.com/?url=005, 5
```

```
2019-09-12:02:http://example.com/?url=006, 10
```

Problem 4: time range queries [points: 25]

Paste your source code into the following area [15 points]

```
import csv
from dateutil.parser import parse
from datetime import datetime
import time
from multiprocessing import Process, Lock, Manager

files = ['file-input1.csv', 'file-input2.csv', 'file-input3.csv', 'file-input4.csv']

# Following function reads each line in the above files and forms a data structure of all the
data
def readFile(inputFile, lock, shared_dict):
        lock.acquire()
        with open(inputFile) as csvfile:
                readCSV = csv.reader(csvfile, delimiter=',')
                for row in readCSV:
                        dt = parse(row[1])
                        dt_str = str(dt.date())+":"+str(dt.time().hour)
                        url = row[2]
                        country= row[4]

                        start_date = datetime(2019, 9, 13, 17)
                        end_date = datetime(2019, 9, 14, 9)
                        current_line_date = datetime(dt.year, dt.month, dt.day, dt.time().hour)
                        if (start_date <= current_line_date <= end_date):
                                if dt_str in shared_dict:
                                        if country in shared_dict[dt_str]:
                                                if url in shared_dict[dt_str][country]:
                                                        dictCopy = shared_dict[dt_str][country]
                                                        dictCopy[url] = dictCopy[url]+1
                                                        dictSecondCopy = shared_dict[dt_str]
                                                        dictSecondCopy[country] = dictCopy
                                                        shared_dict[dt_str] = dictSecondCopy
                                                        shared_dict[dt_str][country] = dictCopy
                                                else:
                                                        ### First modify at the user level.
                                                        dictCopy = shared_dict[dt_str][country]
                                                        dictCopy[url] = 1
                                                        ### Next modify at the URL level
                                                        dictSecondCopy = shared_dict[dt_str]
                                                        dictSecondCopy[country] = dictCopy
                                                        shared_dict[dt_str] = dictSecondCopy
                                                        shared_dict[dt_str][country] = dictCopy

                                        else:
                                                dictCopy = shared_dict[dt_str]
                                                dictCopy[country] = {url: 1}
                                                shared_dict[dt_str] = dictCopy
                                else:
                                        shared_dict[dt_str] = {country: {url: 1}}
        lock.release()

manager = Manager()
shared_dict = manager.dict()
lock = Lock()
```

```
threads = []
for inputFile in files:
        procs = Process(target=readFile, args=[inputFile, lock, shared_dict])
        threads.append(procs)
        procs.start()

for t in threads : t.join()


####Query 1 ######
print("############################ Query 1 OUTPUT ################################")
for key in sorted(shared_dict.keys()):
        for country in shared_dict[key]:
                print (key, country, len(shared_dict[key][country]))
```

What are the main differences with the Problem 3 implementation? [5 points]

I modified the data structure as follows:
{date: {country: {url: count}}}

Also, I have to filter out the data to make sure we only get the time range we want. This is done before adding the data to dictionary.

What are the results of your query for the specified keys ? [5 points]
The expected output for the first value is provided for your reference.

<date,hour,country>, url_count
2019-09-13:19,IQ, 1
2019-09-13:19,IR, 4
2019-09-13:19,IS, 9
2019-09-13:19,IT, 2
2019-09-13:19,JE, 4

Problem 5: Bonus: Top N queries [15 points]
Paste your source code into the following area [5 points]

What are the main differences with the Problem 3 and 4 implementation? [5 points]

What are the results of your query? [5 points] The expected 5 values for 9/12 are provided, please fill in the values for avg TTFB and the URLs for 9/13 and 9/14.

Date	URL	Average_TTFB
9/12/19	http://example.com/?url=114	0.393101408

9/12/19 http://example.com/?url=101 0.402545
9/12/19 http://example.com/?url=133 0.413317187
9/12/19 http://example.com/?url=033 0.418867857
9/12/19 http://example.com/?url=157 0.419289394

9/13/19
9/13/19
9/13/19
9/13/19
9/13/19

9/14/19
9/14/19
9/14/19
9/14/19
9/14/19