**Cheng, Loi**

9/12/2019

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.

|  |
| --- |
| *for example:*  Problem 1: 100% complete  Problem 2: 100% complete  Problem 3: 100% complete  Problem 4: 100% complete  Problem 5 Bonus: 100% complete |

**Problem 1: CPU Analysis** [points: 25]

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

|  |
| --- |
| #Loi Cheng  #9/12/2019  import time  import multiprocessing  import os  import psutil  import argparse  def fibonacciBillion():      '''      Sleeps for 1 sec      Prints its name/ID to the console      Performs CPU-intensive work, calculate a Fibonacci sequence for around one billion iterations      Does the above three things forever (you can use an endless while loop)      '''      while(True):          time.sleep(1)          processID = os.getpid()          process = psutil.Process(processID)          print(  "Process\_Name: " + process.name() + " Process\_ID: " + str(processID)  )          lastNum = 1          nextNum = 1          for i in range (10000000000):              newSum = lastNum + nextNum              lastNum = nextNum              nextNum = newSum      return  if \_\_name\_\_ == "\_\_main\_\_":      # get input arguments      parser = argparse.ArgumentParser(description='IntenseFibonacci')      parser.add\_argument('numThreads', type=int, help='number of threads')      args = parser.parse\_args()      # run specified number of numThreads      for i in range(args.numThreads):          p = multiprocessing.Process(target=fibonacciBillion)          p.start() |

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]

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 2 Threads (or “Processes”)      4 Threads      16 Threads       |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | 4 CPU Machine |  |  |  |  |  |  | |  | Utilization |  |  |  |  |  | | Number of Threads ("Processes") | CPU1 | CPU2 | CPU3 | CPU4 | Load Average | CPU% for each thread (average) | | 2 | 100% | 100% | 0% | 0% | 2 | 100% | | 4 | 100% | 100% | 100% | 100% | 4 | 100% | | 16 | 100% | 100% | 100% | 100% | 16 | 25% | |

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

|  |
| --- |
| All the threads are distributed to CPU’s to fully utilize all the CPU’s as much as possible. However, each thread (or “process”) can only be allocated to at most 1 CPU. So with 2 threads, each is assigned to its own CPU, leaving 2 CPU’s mostly not utilized. With 4 threads, all 4 CPU’s are fully utilized, each running 1 thread. With 16 threads, since the machine does not have 16 CPU’s, all the threads are squeezed into the 4 available CPU’s. The load average shows for 2 threads, the CPU’s are 2/4 = 50% utilized. Similarly, 4 threads creates 4/4 = 100% utilization. For 16 threads, 16/4 = 400% so the CPU’s are overutilized, and htop shows that each of the 16 threads can only use 25% of the CPU. |

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]

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 2 Threads (or “Processes”)      4 Threads      16 Threads       |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 8 CPU Machine |  |  |  |  |  |  |  |  |  |  | |  | CPU |  |  |  |  |  |  |  |  |  | | Number of Threads | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | Load Average | CPU% for each thread (average) | | 2 | 100% | 100% | 0% | 0% | 0% | 0% | 0% | 0% | 2 | 100% | | 4 | 0% | 100% | 100% | 100% | 0% | 0% | 0% | 100% | 4 | 100% | | 16 | 100% | 100% | 100% | 100% | 100% | 100% | 100% | 100% | 16 | 50% | |

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 for 2 thread and 4 threads are similar to the 4 CPU machine, with CPU% for each thread maxed at 100%. But when running 16 threads, the 4 CPU machine can only run with 25% CPU available for each thread, while the 8 CPU performs better with 50% CPU available. Still, both machines are overutilized when running the program with 16 threads. |

**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]

|  |
| --- |
| #Loi Cheng  #9/13/2019  import time  import multiprocessing  import os  import psutil  import argparse  import random  def bigIOtask():      '''      Continuously create, write 100000 random numbers to, close and delete a file      '''      processID = os.getpid()      process = psutil.Process(processID)      print(  "Process\_Name: " + process.name() + " Process\_ID: " + str(processID)  )      while(True):          fileName = "big\_IO\_file\_" + process.name() + "\_" + str(processID) + ".txt"          # open a (new) file to write          outF = open(fileName, "w")          for i in range (100000):              # write line to output file              outF.write( str(random.randint(1,100000)) )              outF.write("\n")          outF.close()          os.remove(fileName)      return  if \_\_name\_\_ == "\_\_main\_\_":      # get input arguments      parser = argparse.ArgumentParser(description='bigIOtask')      parser.add\_argument('numThreads', type=int, help='number of threads')      args = parser.parse\_args()      # run specified number of numThreads      for i in range(args.numThreads):          p = multiprocessing.Process(target=bigIOtask)          p.start() |

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]

|  |
| --- |
| 2 threads        4 threads        16 threads |

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

|  |
| --- |
| The CPU utilization from htop shows nearly identical behavior as seen in Problem 1. IO usage from iotop shows low utilization at 2 threads, where DISK WRITE goes up to about 43k/s for a brief moment then drops to zero for about a second. The DISK WRITE is frequently at 0 for both threads. At 4 threads, DISK WRITE is often active on 2 or 3 threads, and is seldomly at 0 for all threads, which occurs about once every 5-10 seconds. At 16 threads, IO usage is constantly active for about at least half the threads, and does not seem to ever show 0 for all the threads. |

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]

|  |
| --- |
| 2 threads        4 threads        16 threads |

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 CPU utilization from htop shows nearly identical behavior as seen in Problem 1. At 2 threads, the iotop IO is nearly identical between 4 CPU and 8 CPU. At 4 threads, the 8 CPU seems to perform slightly better, with DISK WRITE more frequently at 0 than when using the 4 CPU. At 16 threads, again the iotop IO is nearly identical between 4 CPU and 8 CPU, the IO usage is constantly active for about at least half the threads, and does not seem to ever show 0 for all the threads. |

**Problem 3: unique counts** [points: 25]

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

|  |
| --- |
| #Loi Cheng  #9/13/2019  import multiprocessing  import pandas as pd  def processData(fileName, hourUrlList, hourUrlUserList, hourUrlUUIDList):      """      reduces a set of data to useful lists that will be combined with similar lists generated from other threads      """      df = pd.read\_csv(fileName,header=None)      df["DateTime"] = pd.to\_datetime(df[1])      df["DateHour"] = df["DateTime"].dt.date.astype("str") + ":" + df["DateTime"].dt.strftime("%H").astype("str")      df['DateHour\_URL'] = df['DateHour'].astype('str') + ":" + df[2].astype('str')      df = df.rename(columns={0:"UUID", 2: "url", 3: "user"})      # Query1      # reduce to list with hour and unique url      dfQueryOne = df[["DateHour","url"]].drop\_duplicates()      hourUrlList += dfQueryOne.values.tolist()        # Query2      # reduce data to list with hour unique URL, and unique visitor list      dfQueryTwo = df[['DateHour\_URL','user']].drop\_duplicates()      hourUrlUserList += dfQueryTwo.values.tolist()      # Query3      # Reduce data to table with count of unique (by UUID) events/clicks per URL per hour per day      # <date:hour:url>,  Total Event\_count      dfQueryThree = df[['DateHour\_URL','UUID']].groupby('DateHour\_URL').count().reset\_index()      hourUrlUUIDList += dfQueryThree.values.tolist()      return  if \_\_name\_\_ == "\_\_main\_\_":      with multiprocessing.Manager() as manager:          # creating lists in server process memory          hourUrlList = manager.list()          hourUrlUserList = manager.list()          hourUrlUUIDList = manager.list()            # creating new processes          p1 = multiprocessing.Process(target=processData, args=("input\_files/file-input1.csv", hourUrlList, hourUrlUserList, hourUrlUUIDList))          p2 = multiprocessing.Process(target=processData, args=("input\_files/file-input2.csv", hourUrlList, hourUrlUserList, hourUrlUUIDList))          p3 = multiprocessing.Process(target=processData, args=("input\_files/file-input3.csv", hourUrlList, hourUrlUserList, hourUrlUUIDList))          p4 = multiprocessing.Process(target=processData, args=("input\_files/file-input4.csv", hourUrlList, hourUrlUserList, hourUrlUUIDList))            # running process          p1.start()          p2.start()          p3.start()          p4.start()          # wait to finish          p1.join()          p2.join()          p3.join()          p4.join()            # process joined data for Query1          df = pd.DataFrame.from\_records(hourUrlList)          df = df.drop\_duplicates().groupby(0).count().reset\_index()          df = df.rename(columns={0: "<date\_hour>", 1: "<url\_count>"})          df.to\_csv("date\_hour--url\_count.csv", index=None)          # output joined data for Query2          df = pd.DataFrame.from\_records(hourUrlUserList)          df = df.drop\_duplicates().groupby(0).count().reset\_index()          df = df.rename(columns={0: "<date:hour:url>", 1: "unique\_user\_count"})          df.to\_csv("date\_hour\_url--unique\_user\_count.csv", index=None)          # output joined data for Query3          df = pd.DataFrame.from\_records(hourUrlUUIDList)          df = df.groupby(0).sum().reset\_index()          df = df.rename(columns={0: "<date:hour:url>", 1: "Total Event\_count"})          df.to\_csv("date\_hour\_url--Total\_Event\_count.csv", index=None) |

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

|  |
| --- |
| For the program, manager.list() was used as the data structure for shared state management. Since the data being delivered is in the form of key:value of unknown length, any with Array and List being the only two known options, it was easier to store the data as a list than an array. |

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]

|  |
| --- |
| #Loi Cheng  #9/13/2019  import multiprocessing  import pandas as pd  def processData(fileName, hourCountryUrlList, t1, t2):      '''      reduces a set of data to useful lists that will be combined with similar lists generated from other threads      '''      #read file, add the DateTime      df = pd.read\_csv(fileName,header=None)      df['DateTime'] = pd.to\_datetime(df[1])        #trim file by t1 t2      df = df.set\_index(df['DateTime'])      df = df.loc[t1:t2]      #create the DateHour\_Country column      df = df.rename(columns={2: 'url',4:'country'})      df['DateHour'] = df['DateTime'].dt.date.astype('str') + ':' + df['DateTime'].dt.strftime('%H').astype('str')      df['DateHour\_Country'] = df['DateHour'].astype('str') + ',' + df['country'].astype('str')      # Query4      # reduce to list with hour,country and unique url      dfQueryFour = df[['DateHour\_Country','url']].drop\_duplicates()      hourCountryUrlList += dfQueryFour.values.tolist()        return  if \_\_name\_\_ == '\_\_main\_\_':      #specify time range      t1 = '2019-09-13 17:00:00'      t2 = '2019-09-14 09:00:00'      with multiprocessing.Manager() as manager:          # creating lists in server process memory          hourCountryUrlList = manager.list()            # creating new processes          p1 = multiprocessing.Process(target=processData, args=('input\_files/file-input1.csv', hourCountryUrlList, t1, t2) )          p2 = multiprocessing.Process(target=processData, args=('input\_files/file-input2.csv', hourCountryUrlList, t1, t2) )          p3 = multiprocessing.Process(target=processData, args=('input\_files/file-input3.csv', hourCountryUrlList, t1, t2) )          p4 = multiprocessing.Process(target=processData, args=('input\_files/file-input4.csv', hourCountryUrlList, t1, t2) )            # running process          p1.start()          p2.start()          p3.start()          p4.start()          # wait to finish          p1.join()          p2.join()          p3.join()          p4.join()            # process joined data for Query4          df = pd.DataFrame.from\_records(hourCountryUrlList)          df = df.drop\_duplicates().groupby(0).count().reset\_index()          df = df.rename(columns={0: '<date\_hour\_country>', 1: '<url\_count>'})          df.to\_csv('date\_hour\_country--url\_count.csv', index=None) |

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

|  |
| --- |
| The problem 3 and 4 implementation are very similar. The main difference is the filtering of the data by date and time in problem 4 reduced the overall amount of data considered. |

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  2018-09-13:19,IQ, 1  2018-09-13:19,IR, 4  2018-09-13:19,IS, 9  2018-09-13:19,IT, 2  2018-09-13:19,JE, 4 |

**Problem 5: Bonus: Top N queries** [15 points]

Paste your source code into the following area [5 points]

|  |
| --- |
| #Loi Cheng  #9/13/2019  import multiprocessing  import pandas as pd  def processData(fileName, dateUrlTTFBList):      '''      reduces a set of data to useful lists that will be combined with similar lists generated from other threads      '''      #read file, add the DateTime      df = pd.read\_csv(fileName,header=None)      df['DateTime'] = pd.to\_datetime(df[1])        #create the DateURL column      df = df.rename(columns={2:"url",8:"TTFB"})      df['Date'] = df['DateTime'].dt.date      df['DateURL'] = df['Date'].astype('str') + "::" + df['url'].astype('str')      # Query5      # reduce to list DateURL and Sum TTFB and Count TTFB      dfDateUrlTTFB = df[['DateURL','TTFB']]      dfDateUrlTTFBSum = dfDateUrlTTFB.groupby('DateURL').sum().reset\_index()      dfDateUrlTTFBSum = dfDateUrlTTFBSum.rename(columns={'TTFB':'TTFBSum'})      dfDateUrlTTFBCount = dfDateUrlTTFB.groupby('DateURL').count().reset\_index()      dfDateUrlTTFBCount = dfDateUrlTTFBCount.rename(columns={'TTFB':'TTFBCount'})      dfQueryFive = dfDateUrlTTFBSum.set\_index('DateURL').join(dfDateUrlTTFBCount.set\_index('DateURL')).reset\_index()      dateUrlTTFBList += dfQueryFive.values.tolist()      return  if \_\_name\_\_ == '\_\_main\_\_':      with multiprocessing.Manager() as manager:          # creating lists in server process memory          dateUrlTTFBList = manager.list()            # creating new processes          p1 = multiprocessing.Process(target=processData, args=('input\_files/file-input1.csv', dateUrlTTFBList)  )          p2 = multiprocessing.Process(target=processData, args=('input\_files/file-input2.csv', dateUrlTTFBList)  )          p3 = multiprocessing.Process(target=processData, args=('input\_files/file-input3.csv', dateUrlTTFBList)  )          p4 = multiprocessing.Process(target=processData, args=('input\_files/file-input4.csv', dateUrlTTFBList)  )            # running process          p1.start()          p2.start()          p3.start()          p4.start()          # wait to finish          p1.join()          p2.join()          p3.join()          p4.join()            # process joined data for Query4          df = pd.DataFrame.from\_records(dateUrlTTFBList)          df = df.groupby(0).sum().reset\_index()          df3 = df[0].str.split("::", n = 1, expand = True)          df3['averageTTFB']=df[1]/df[2]          df3 = df3.rename(columns={0:"date",1:"url"})          outDF = df3.sort\_values(['date','averageTTFB']).groupby('date').head(5)          outDF.to\_csv('date--fiveUrl--LowestAvgTTFB.csv', index=None) |

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

|  |
| --- |
| Unlike problem 3 and problem 4, this implementation reduces the data by the most as the time aggregation of the data is done by each day instead of each hour |

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 | http://example.com/?url=039 | 0.402814286 | | 9/13/19 | http://example.com/?url=110 | 0.429469355 | | 9/13/19 | http://example.com/?url=005 | 0.430635849 | | 9/13/19 | http://example.com/?url=006 | 0.431381159 | | 9/13/19 | http://example.com/?url=028 | 0.431906452 |  |  |  |  | | --- | --- | --- | | 9/14/19 | http://example.com/?url=123 | 0.405598077 | | 9/14/19 | http://example.com/?url=070 | 0.418589333 | | 9/14/19 | http://example.com/?url=124 | 0.423315873 | | 9/14/19 | http://example.com/?url=107 | 0.426570588 | | 9/14/19 | http://example.com/?url=101 | 0.429436923 | |