**Choi, Mason**

September 19, 2022

Harvard University Extension - Principles of Big Data Processing e88

Homework 3: Shared state with Redis; HDFS

* **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>\_HW3.zip**
* **Make sure to add full result files into that archive as well**

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

|  |
| --- |
| Problem 1: 100% complete  Problem 1B: Bonus: 100% complete  Problem 2: 100% complete  Problem 3: Bonus: 0% complete – not attempted  Problem 4: 100% complete  Problem 5: 100% complete |

**Problem 1: unique counts** [points: 40]

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

|  |
| --- |
| *# Copyright (c) 2022 CSCIE88 Marina Popova*  *# Code adapted by Mason Choi in September of 2022 for CSCIE88*  *# {"2022-09-04:07": {"url1": [clicks, {user-01, user-02}], "url2": [{...}], ...}, ...}*  *# redis hash: python dictionary{string: list[string, set]}*  """  python problem1.py -f ../logs/file-input1.csv  python problem1.py -f ../logs/file-input2.csv  python problem1.py -f ../logs/file-input3.csv  python problem1.py -f ../logs/file-input4.csv  ^C  """  import argparse  from collections import namedtuple  import redis  import json  event\_fields = ['uuid', 'timestamp', 'url', 'userid', 'country', 'ua\_browser', 'ua\_os', 'response\_status', 'TTFB']  Event = namedtuple('Event', event\_fields)  **def** parse\_arguments():      prog = "counter\_process\_redis"      desc = "application that reads a file, parses all lines, counts the lines and " \             "stores/increments the counter maintained in Redis"      parser = argparse.ArgumentParser(prog=prog, description=desc)  *# name of a simple String field in Redis - that will be use as a shared counter*      parser.add\_argument('--file\_name', '-f', required=False, default="../logs/file-input1.csv",                          help="a csv log file to process")      parser.add\_argument('--redis\_url', '-ru', required=False, default="redis://localhost:6379",                          help="Redis end point url; Eg: redis://localhost:6379")      parsed\_args = parser.parse\_args()      return parsed\_args  **def** do\_work(redis\_url, file\_name):      redis\_client = redis.Redis.from\_url(redis\_url)      with open(file\_name) as file\_handle:          events = map(parse\_line, file\_handle)          for event in events:              date\_hour = event[1][:event[1].find(":")]              date\_hour = date\_hour.replace("T", ":")  *# parse datetime into date:hour form*              url = event[2]              user = event[3]  *# aggregate relevant data into dict of dicts*              if redis\_client.exists(date\_hour):                  if bytes(url, encoding="utf-8") in redis\_client.hgetall(date\_hour):                      old\_url\_value = redis\_client.hget(date\_hour, url)                      old\_url\_value = json.loads(old\_url\_value)  *# load stringified data*                      old\_url\_value[0] += 1                      old\_url\_value[1].append(user)                      old\_url\_value[1] = list(set(old\_url\_value[1]))                      old\_url\_value = json.dumps(old\_url\_value)  *# restringify*                      new\_url\_value = old\_url\_value                      redis\_client.hset(date\_hour, url, new\_url\_value)  *# update hm value*                  else:                      new\_url\_value = json.dumps([1, list(set([user]))])                      redis\_client.hset(date\_hour, url, new\_url\_value)              else:                  new\_datehour\_value = json.dumps([1, list(set([user]))])                  redis\_client.hset(date\_hour, url, new\_datehour\_value)      redis\_client.close()  **def** parse\_line(line):      return Event(\*line.split(','))  **def** main():      parsed\_args = parse\_arguments()      file\_name = parsed\_args.file\_name      redis\_url = parsed\_args.redis\_url      do\_work(redis\_url, file\_name)  if \_\_name\_\_ == '\_\_main\_\_':      main() |

Explain your choice of the Redis data structures for shared state management for each Query [10 points]

|  |
| --- |
| I used the Redis hash which maps keys to hash maps, or rather a key to a key to a value. I initially wanted to create a data structure looking like {date\_hour:{url:[count, {userID, userID}]}}  But since Redis does not support nested data structures, I used JSON to convert the last value (a list) into a string. Thus my Redis data structure was a key to a hash, which mapped a key to a string. |

Show a screenshot of your Redis server running, and 4 instances of your application [5 points]

|  |
| --- |
| Graphical user interface, website  Description automatically generated with medium confidence |

Show results of your queries in the Redis CLI, for the specified keys [15 points]

|  |
| --- |
| Query 1:  <date\_hour>, <url\_count>  2022-09-03:16 185  2022-09-04:02 185  2022-09-04:19 184  2022-09-05:01 193  2022-09-06:08 186  Query 2  <date:hour:url>, unique\_user\_count  2022-09-03:16:[http://example.com/?url=06](http://example.com/?url=064)5, 3  2022-09-04:02:http://example.com/?url=110, 3  2022-09-04:19:http://example.com/?url=100, 6  2022-09-05:01:http://example.com/?url=018, 4  2022-09-06:08:http://example.com/?url=013, 2  Query 3  <date:hour:url>, event\_count  2022-09-03:16:[http://example.com/?url=06](http://example.com/?url=064)5, 3  2022-09-04:02:http://example.com/?url=110, 3  2022-09-04:19:http://example.com/?url=100, 6  2022-09-05:01:http://example.com/?url=018, 4  2022-09-06:08:http://example.com/?url=013, 2 |

**Problem 1B: Bonus [**points: 10**]**

* in this problem - we will use a programmatic access to Redis to get results of queries 1-3 - instead of running the queries manually from a Redis CLI client
* write an application that implements Queries 1, 2 and 3 - by reading data from the Redis server
* run this application after all data processing is finished by the applications from Problem 1

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

|  |
| --- |
| import redis  import json  redis\_client = redis.Redis.from\_url("redis://localhost:6379")  date\_times = ["2022-09-03:16", "2022-09-04:02", "2022-09-04:19", "2022-09-05:01", "2022-09-06:08"]  urls = ["http://example.com/?url=065", "http://example.com/?url=110", "http://example.com/?url=100", "http://example.com/?url=018", "http://example.com/?url=013"]  *# query 1*  for date\_time in date\_times:      print(**f**"{date\_time} {len(redis\_client.hgetall(date\_time))}")  print()  *# query 2*  for key in zip(date\_times, urls):      user\_count = len(json.loads(redis\_client.hget(key[0], key[1]))[1])      print(**f**"{key[0]}:{key[1]}, {user\_count}")  print()  *# query 3*  for key in zip(date\_times, urls):      event\_count = json.loads(redis\_client.hget(key[0], key[1]))[0]      print(**f**"{key[0]}:{key[1]}, {event\_count}")  redis\_client.close() |

Show results of your query 1, 2 and 3. Use the same data points as in problem 1 [5 points]

|  |
| --- |
| Text  Description automatically generated |

**Problem 2: time range queries** [points: 30]

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

|  |
| --- |
| *# Copyright (c) 2022 CSCIE88 Marina Popova*  *# Code adapted by Mason Choi in September of 2022 for CSCIE88*  *# {"2022-09-04:07": {"country": {url1, url2}, "country": {...}, ...}, ...}*  *# # redis hash: python dictionary{string: set}*  """  python problem2.py -tr 2022-09-04T02 2022-09-05T22 -f ../logs/file-input1.csv  python problem2.py -tr 2022-09-04T02 2022-09-05T22 -f ../logs/file-input2.csv  python problem2.py -tr 2022-09-04T02 2022-09-05T22 -f ../logs/file-input3.csv  python problem2.py -tr 2022-09-04T02 2022-09-05T22 -f ../logs/file-input4.csv  ^C  """  import argparse  from collections import namedtuple  import redis  import json  from dateutil.parser import parse  event\_fields = ['uuid', 'timestamp', 'url', 'userid', 'country', 'ua\_browser', 'ua\_os', 'response\_status', 'TTFB']  Event = namedtuple('Event', event\_fields)  **def** parse\_arguments():      prog = "counter\_process\_redis"      desc = "application that reads a file, parses all lines, counts the lines and " \             "stores/increments the counter maintained in Redis"      parser = argparse.ArgumentParser(prog=prog, description=desc)  *# name of a simple String field in Redis - that will be use as a shared counter*      parser.add\_argument('--file\_name', '-f', required=False, default="../logs/file-input1.csv",                          help="a csv log file to process")      parser.add\_argument('--time-range', '-tr', required=True, type=str, nargs=2,                      help="Time range for query, in form YEAR-MONTH-DAYTHOUR. Inclusive.")      parser.add\_argument('--redis\_url', '-ru', required=False, default="redis://localhost:6379",                          help="Redis end point url; Eg: redis://localhost:6379")      parsed\_args = parser.parse\_args()      return parsed\_args  **def** do\_work(redis\_url, file\_name, time\_range):      redis\_client = redis.Redis.from\_url(redis\_url)      with open(file\_name) as file\_handle:          events = map(parse\_line, file\_handle)          for event in events:              date\_hour = event[1][:event[1].find(":")]  *# check if time is within specified range*              if parse(date\_hour) >= time\_range[0] and parse(date\_hour) <= time\_range[1]:                  date\_hour = date\_hour.replace("T", ":")  *# parse datetime into date:hour form*                  country = event[4]                  url = event[2]  *# aggregate relevant data into dict of dicts*                  if redis\_client.exists(date\_hour):                      if bytes(country, encoding="utf-8") in redis\_client.hgetall(date\_hour):                          old\_country\_value = redis\_client.hget(date\_hour, country)                          old\_country\_value = json.loads(old\_country\_value)  *# load stringified data*                          old\_country\_value.append(url)                          old\_country\_value = json.dumps(list(set(old\_country\_value)))  *# restringify & dedup*                          new\_country\_value = old\_country\_value                          redis\_client.hset(date\_hour, country, new\_country\_value)  *# update hm value*                      else:                          new\_country\_value = json.dumps([url])                          redis\_client.hset(date\_hour, country, new\_country\_value)                  else:                      new\_datehour\_value = json.dumps([url])                      redis\_client.hset(date\_hour, country, new\_datehour\_value)      redis\_client.close()  **def** parse\_line(line):      return Event(\*line.split(','))  **def** main():      parsed\_args = parse\_arguments()      file\_name = parsed\_args.file\_name      redis\_url = parsed\_args.redis\_url      time\_range = [parse(parsed\_args.time\_range[0]), parse(parsed\_args.time\_range[1])]      do\_work(redis\_url, file\_name, time\_range)  if \_\_name\_\_ == '\_\_main\_\_':      main() |

Explain your choice of Redis data structures [5 points]

|  |
| --- |
| In problem 5 of the last homework assignment I worked on a similar question, using the data structure looking like {date\_time: {country: {url1, url2, …}…}…} or more generally dict{dict{set}}. I used a Redis hash and stringified the set using JSON, resulting in a Redis structure of “date\_time”:{“country”: “{url1, url2, …}”…}… |

Show results of your query for the specified time range [10 points]

|  |
| --- |
| * time range:   + t1 = 2022-09-04 2 AM UTC and   + t2 = 2022-09-05 10 PM UTC   <date,hour,country>, url\_count  2022-09-04:03, KI, 1  2022-09-04:10, DJ, 3  2022-09-05:01, AS, 1  2022-09-05:16, CH, 2  2022-09-05:20, VE, 2 |

**Problem 3: Bonus: Top N queries** [10 points]

Paste your source code into the following area [3 points]

|  |
| --- |
|  |

Explain the changes you made to the code and why ? [2 points]

|  |
| --- |
|  |

Show results of your query [5 points]

|  |
| --- |
| Date URL Average\_TTFB  2022-09-04  2022-09-04  2022-09-04  2022-09-04  2022-09-04  2022-09-05  2022-09-05  2022-09-05  2022-09-05  2022-09-05  2022-09-06  2022-09-06  2022-09-06  2022-09-06  2022-09-06 |

**Problem 4: [15 points] AWS EMR cluster and HDFS**

Include screenshot of the created EMR cluster - with all cluster information; clearly identify master and slave nodes - either from the EC2 instances views or from the cluster view [5 points]

|  |
| --- |
| A screenshot of a computer  Description automatically generated |

List commands you used to ssh to your master node and view HDFS version; include screenshots of the hdfs version output [5 points]

|  |
| --- |
| Text  Description automatically generated |

List commands you used to demo basic (create, get/view, delete file) HDFS operations - include a screenshot of the terminal where you were running the commands, with their output [5 points]

|  |
| --- |
| Text  Description automatically generated |

**Problem 5: [15 points]**

Explain the difference between idempotent and non-idempotent operations [5 points]

|  |
| --- |
| In the context of computing, an idempotent operation is some operation or action that is guaranteed to produce the same result no matter how many times it is invoked. A non-idempotent operation can produce different results when invoked more than one time, so to get the desired result it must only be invoked once as opposed to an idempotent one. |

Give an example of a non-idempotent operation [5 points]

|  |
| --- |
| Removing a specified element from a list (ie. using list.remove(x) in Python) is generally considered non-idempotent because if that element appears more than once then the list’s contents will change when calling the operation more than once. In addition, if the element only appears once or not at all, multiple operations will definitely raise an error. |

give an example of an idempotent operation [5 points]

|  |
| --- |
| Removing a specified element from a Python set using the set.discard() method is considered an idempotent operation because no matter how many times it is invoked, the set will remain the same (times >= 1). This is because sets only contain unique elements, and set.discard() does not raise any errors if an element does not exist in the set. |