**Cheng, Loi**

11/17/2019

Harvard Extension - Big Data Principles e88

Homework 10: Data Modeling and Programming with Cassandra

Please identify which problems were completed:

|  |
| --- |
| Problem 1: 100%  Problem 2: 100%  Problem 3: 100%  Problem 4: 100%  Bonus Problem 5:  Bonus Problem 6: |

**Problem 1 [25 points] data modeling with Cassandra**

Using CQLSH and the DESCRIBE command, show your keyspace and table info [8 points]

|  |
| --- |
|  |

Explain the design of your table’s Primary Key [5 points]

|  |
| --- |
| The url, country, and date is chosen as the primary key. The url and country is chosen because the queries are based on url and country, and so if we design the system to store entries with the same url and country pair in the same set of nodes, it should enable faster query time.  However, with only url and country pair as key, all the entries with the same key would always be stored in one partition, which may cause hot spotting if a url is frequently visited by a single country every day. So, adding date to the primary key can reduce hotspotting, as entries for the same url and country but with different dates could be stored in different partition. Also, with date as primary key, we can run faster range queries by url, country and date, since Cassandra knows which node stores these entries without visiting every single node to check.  Time is included as a clustering key so we can efficiently perform a time range query. Id is also in the clustering key so that entries with the same url, country, hour, time but different uuid would not overwrite each other. |

Show a screenshot / screen copy of Query 1 being run with its results [6 points]

|  |
| --- |
|  |

Show a screenshot / screen copy of Query 2 being run with its results [6 points]

|  |
| --- |
|  |

**Problem 2 [25 Points] range queries with Cassandra**

Using CQLSH and the DESCRIBE command, show your keyspace and table info [7 points]

|  |
| --- |
|  |

Explain the design of your table’s Primary Key [5 points]

|  |
| --- |
| The url, country, and hour is chosen as the primary key. The url and date is chosen because the queries are based on url and country, and so if we design the system to store entries with the same url and country pair in the same set of nodes, it should enable faster query time.  However, with only url and country pair as key, all the entries with the same key would always be stored in one partition. Because we are getting too many events for each url and country combination, we also add hour as the primary key so that entries for the same url and country but with different hours could be stored in different partition.  Time is included as a clustering key so we can efficiently perform a time range query. Id is also in the clustering key so that entries with the same url, country, hour, time but different uuid would not overwrite each other. |

Show a screenshot / screen copy of Query 1 being run with its results [5 points]

|  |
| --- |
|  |

Show a screenshot / screen copy of Query 2 being run with its results [5 points]

|  |
| --- |
|  |

What is the main difference between your tables for problem 1 and problem 2? [3 points]

|  |
| --- |
| For problem 1, the partition key is (url, country, date), so we can efficiently query the database for specific url + country + date combinations without visiting every node. Since the query can return results for an entire day, we can reduce the result to specific time ranges within the 24hr period.  For problem 2, the partition key is (url, country, hour), so we can only efficiently query the data base for a specific url + country + hour combination without visiting every node. Since the query can only return results for one hour, we can only reduce the result to a timeframe less than 60 min within this one-hour time frame. If we are looking for results in a time range that span multiple hours, we may need to download the data off Cassandra for the url + country + hours of interest, and import it into another tool like pandas dataframe to do further aggregation.  Given the same type of data to store, each partition in hw10\_p1 would be larger than those in hw10\_p2 |

**Problem 3 [30 Points] programming with Cassandra drivers**

Copy and paste yourcode here [10 points]

|  |
| --- |
| from datetime import datetime, timedelta  from cassandra.cluster import Cluster  from cassandra import util  import pandas as pd  import numpy as np  import uuid  import random  # Initialize the connection and session with Cassandra on localhost  cluster = Cluster(['127.0.0.1'])  session = cluster.connect('hw10')  def generate\_data(num\_rows):  #generate uuid  uuid\_list = []  for \_ in range(num\_rows):  uuid\_list.append(uuid.uuid4())  df = pd.DataFrame(uuid\_list)  #generate incremental time  time\_list = [pd.Timestamp('2019-11-17T01Z')]  for i in range(1,len(df)):  time\_list.append( time\_list[-1] + pd.Timedelta(seconds=random.randint(1,50)) )  df[1] = time\_list  #generate url  new\_url\_list = ['http://example.com/?url=091','http://example.com/?url=095','http://example.com/?url=099']  df[2] = np.random.choice(new\_url\_list, size=len(df))  #generate user  df[3] = np.random.choice(['user-028','user-039','user-052','user-065','user-099',], size=len(df))  #generate country  df[4] = np.random.choice(['ER','SJ','MA','GD','ZW'], size=len(df))  #generate browser  new\_browser\_list = ['Chrome','Firefox','Edge','IE']  df[5] = np.random.choice(new\_browser\_list, size=len(df))  #generate OS  df[6] = np.random.choice(['Mac','Linux','iPhone','Windows'], size=len(df))  #generate Response  df[7] = np.random.choice([501,307,510,208], size=len(df))  #generate TTFB  df[8] = np.random.uniform(size=len(df))  df[8] = df[8].round(3)  #extract hour  df[9] = df[1].astype(str).apply(lambda x: x[:13]+':00:00Z')  return df  def insert\_cassandra(df):  """  Table Reference  CREATE TABLE IF NOT EXISTS hw10.hw10\_p2 (  id UUID,  time timestamp,  url text,  userId text,  country text,  ua\_browser text,  ua\_os text,  response\_status int,  TTFB float,  hour timestamp,  PRIMARY KEY ((url, country, hour), time, id)  );  """  #clean the table for homework purposes  session.execute(  """  TRUNCATE hw10.hw10\_p2  """  )  #run insertions  for i in range(len(df)):  #format each entry for insertion  #proper format example  #event\_string = "8e66dea6-2a91-4ba6-9e48-c534c705574e, '2019-09-14 03:56:26Z', 'http://example.com/?url=078', 'user-079', 'SN', 'Firefox', 'Mac', 201, 0.2186, '2019-09-14 03:00:00Z'"  event\_string = str(list(df.iloc[i]))[1:-1]  event\_string = event\_string.replace("'","",2)  event\_string = event\_string.replace("UUID(","",)  event\_string = event\_string.replace("Timestamp(","",)  event\_string = event\_string.replace(", tz='UTC'","",)  event\_string = event\_string.replace("+0000","Z",)  event\_string = event\_string.replace(")","",)  print("event", event\_string )    session.execute(  """  INSERT INTO hw10.hw10\_p2  (id, time, url, userId, country, ua\_browser, ua\_os, response\_status, TTFB, hour)  VALUES ( %s )  """  %event\_string  )    print ("inserted")  def read\_cassandra():    row = session.execute(  """  SELECT url, country, count(\*), AVG(TTFB) FROM hw10.hw10\_p2  WHERE hour = '2019-11-17 01:00:00Z' and  country = 'GD' and  url = 'http://example.com/?url=095' and  time >= '2019-11-17 01:00:00Z' and time <= '2019-11-17 01:40:00Z'  """  )  for items in row:  print(items)  row = session.execute(  """  SELECT url, country, count(\*), AVG(TTFB) FROM hw10.hw10\_p2  WHERE hour = '2019-11-17 03:00:00Z' and  country = 'ZW' and  url = 'http://example.com/?url=099' and  time >= '2019-11-17 03:10:00Z' and time <= '2019-11-17 03:50:00Z'  """  )  for items in row:  print(items)  def main():  df = generate\_data(500)  insert\_cassandra(df)  read\_cassandra()  print("Job Completed")  if \_\_name\_\_ == "\_\_main\_\_":  main() |

Show execution of your application and a sample of the data inserted into your table [10] points

|  |
| --- |
|  |

Show execution and results of your queries [10] points

|  |
| --- |
|  |

**Problem 4 [20 Points] data consistency with Cassandra**

Enter your code - highlight changes you made to handle consistency [10 points]

|  |
| --- |
| Changes to handle consistency is highlighted  from datetime import datetime, timedelta  from cassandra.cluster import Cluster  from cassandra import util  import pandas as pd  import numpy as np  import uuid  import random  import time  from cassandra import ConsistencyLevel  from cassandra.query import SimpleStatement  # Initialize the connection and session with Cassandra on localhost  cluster = Cluster(['127.0.0.1'])  session = cluster.connect('hw10')  def handle\_success\_insert(data,event,insertions):  print(event)  insertions[0] += 1  print('async insertion ok')  def handle\_error\_insert(error,event):  print(error)  print(event)  print('async insertion error')  def generate\_data(num\_rows):  #generate uuid  uuid\_list = []  for \_ in range(num\_rows):  uuid\_list.append(uuid.uuid4())  df = pd.DataFrame(uuid\_list)  #generate incremental time  time\_list = [pd.Timestamp('2019-11-17T01Z')]  for i in range(1,len(df)):  time\_list.append( time\_list[-1] + pd.Timedelta(seconds=random.randint(1,50)) )  df[1] = time\_list  #generate url  new\_url\_list = ['http://example.com/?url=091','http://example.com/?url=095','http://example.com/?url=099']  df[2] = np.random.choice(new\_url\_list, size=len(df))  #generate user  df[3] = np.random.choice(['user-028','user-039','user-052','user-065','user-099',], size=len(df))  #generate country  df[4] = np.random.choice(['ER','SJ','MA','GD','ZW'], size=len(df))  #generate browser  new\_browser\_list = ['Chrome','Firefox','Edge','IE']  df[5] = np.random.choice(new\_browser\_list, size=len(df))  #generate OS  df[6] = np.random.choice(['Mac','Linux','iPhone','Windows'], size=len(df))  #generate Response  df[7] = np.random.choice([501,307,510,208], size=len(df))  #generate TTFB  df[8] = np.random.uniform(size=len(df))  df[8] = df[8].round(3)  #extract hour  df[9] = df[1].astype(str).apply(lambda x: x[:13]+':00:00Z')  return df  def insert\_cassandra(df,timeout):  """  Table Reference  CREATE TABLE IF NOT EXISTS hw10.hw10\_p4 (  id UUID,  time timestamp,  url text,  userId text,  country text,  ua\_browser text,  ua\_os text,  response\_status int,  TTFB float,  hour timestamp,  PRIMARY KEY ((url, country, hour), time, id)  );  """  #clean the table for homework purposes only  session.execute(  """  TRUNCATE hw10.hw10\_p4  """  )  #insertions counter, used for pass by reference later  insertions = [0]  futures = []  #run insertions  for i in range(len(df)):  #format each entry for insertion  #proper format example  #event\_string = "8e66dea6-2a91-4ba6-9e48-c534c705574e, '2019-09-14 03:56:26Z', 'http://example.com/?url=078', 'user-079', 'SN', 'Firefox', 'Mac', 201, 0.2186, '2019-09-14 03:00:00Z'"  event\_string = str(list(df.iloc[i]))[1:-1]  event\_string = event\_string.replace("'","",2)  event\_string = event\_string.replace("UUID(","",)  event\_string = event\_string.replace("Timestamp(","",)  event\_string = event\_string.replace(", tz='UTC'","",)  event\_string = event\_string.replace("+0000","Z",)  event\_string = event\_string.replace(")","",)    query = SimpleStatement(  """  INSERT INTO hw10.hw10\_p4  (id, time, url, userId, country, ua\_browser, ua\_os, response\_status, TTFB, hour)  VALUES ( %s )  """  %event\_string,  consistency\_level=ConsistencyLevel.QUORUM  )  #async append  futures.append (  session.execute\_async(query)  )  #setup callbacks  futures[-1].add\_callbacks(callback = handle\_success\_insert, errback = handle\_error\_insert, callback\_args=(event\_string,insertions,),errback\_args=(event\_string,))  wait\_timeout = 0  while insertions[0] < len(df):  print ('insertions completed', insertions[0])  time.sleep(1)  wait\_timeout += 1  if wait\_timeout > timeout:  print('timeout, some insertions were not made')  break    def read\_cassandra():  query = SimpleStatement(  """  SELECT url, country, count(\*), AVG(TTFB) FROM hw10.hw10\_p4  WHERE hour = '2019-11-17 01:00:00Z' and  country = 'ER' and  url = 'http://example.com/?url=095' and  time >= '2019-11-17 01:00:00Z' and time <= '2019-11-17 01:40:00Z'  """,  consistency\_level=ConsistencyLevel.QUORUM  )  row = session.execute(query)  for items in row:  print(items)  query = SimpleStatement(  """  SELECT url, country, count(\*), AVG(TTFB) FROM hw10.hw10\_p4  WHERE hour = '2019-11-17 03:00:00Z' and  country = 'SJ' and  url = 'http://example.com/?url=099' and  time >= '2019-11-17 03:10:00Z' and time <= '2019-11-17 03:50:00Z'  """,  consistency\_level=ConsistencyLevel.QUORUM  )  row = session.execute(query)  for items in row:  print(items)  def main():  df = generate\_data(500)  #async insert, but will either wait until all inserts are done, or timeout  timeout = 60  insert\_cassandra(df,timeout)  #read  read\_cassandra()  print("Job Completed")  if \_\_name\_\_ == "\_\_main\_\_":  main() |

Show execution and results of your queries [10] points

|  |
| --- |
|  |

**Problem 5 [Bonus +20] Real-Time views with Spark and Cassandra**

Show your keyspace and table info; explain design of your table's primary keys [4 points]

|  |
| --- |
|  |

Enter your job's code [4 points]

|  |
| --- |
|  |

Demo your setup - input data, execution of your job, with a few batch outputs and a sample of the data inserted into your table [7] points

|  |
| --- |
|  |

Show execution and results of your queries [5] points

|  |
| --- |
|  |

**Problem 6 [Bonus +15] Cassandra cluster**

Explain how you created your Cassandra cluster [4 points]

|  |
| --- |
|  |

Show commands you used to create the keyspace and your table; show test data you inserted into the table [3 points]

|  |
| --- |
|  |

Demo using the 'nodetool' to inspect the status of your cluster [4 points]

|  |
| --- |
|  |

Demo using the 'nodetool' to inspect location of your data rows [4 points]

|  |
| --- |
|  |