CSC8101 Cassandra Database Coursework

Introduction

Web sites often try to provide content recommendations based on user activity. This may be aggregate, as in 'most watched movies', 'bestsellers', or it may be personalized to varying degrees, as with 'more items like these' or 'customers like you also watched'.

The architecture of such applications includes a means to capture relevant events, such as which pages/items the users are viewing; a model to segment or group the events and/or users; a model to calculate predictions and a dynamic content delivery engine to personalise the site for the user based on the output of the model.

In this coursework you'll implement some of the pieces of such an application: a storage layer (Cassandra) to hold data on user actions both individually and in aggregate, and a program to make simple recommendations based on these. In a later coursework, you will use Apache Spark's streaming functionality to process an activity event stream and populate the Cassandra tables from those results instead.

Cassandra

Cassandra is a distributed NoSQL database commonly used for big data applications due to it fast write performance and high availability (fault tolerance).

In this practical you will learn how to connect to Cassandra, create keyspaces, tables, insert information and write select queries. Most of this will be done by directly interacting with the Cassandra database. The final task will involve using a language API (you may use Python, Java or Scala) to query the Cassandra database.

Links

- CSC8101 module site
- CSC8101 Cassandra Lecture Slides - Available on Blackboard
- Cassandra Documentations
- Python Cassandra Driver
- Java Cassandra Driver
- Scala Cassandra Driver

Setup

Connecting to Cassandra

Running on you VM

Cassandra is already installed on you personal VM. If you have not yet set up your VM follow the instructions here.

To interact with Cassandra, ssh into your VM and run the following command to start the CQL shell:

```
$ cqlsh
```

If Cassandra is running correctly you should see something similar to the prompt below:

```
Connected to Test Cluster at 127.0.0.1:9042.
[cqlsh 5.0.1 | Cassandra 3.9 | CQL spec 3.4.2 | Native protocol v4]
Use HELP for help.
cqlsh>
```

Remote connection from your local machine

If you prefer you can access the Cassandra database on your VM from your own local development machine.

1. Download Cassandra to your local machine.
2. Extract Cassandra into a folder on your machine.
3. Find the public IP address of your VM using the AWS console.
4. On the command line change directory into the extracted Cassandra folder and from their run the following command:

```
$ ./bin/cqlsh <YOUR-VM-PUBLIC-IP> 9042
```

Note: If you receive a protocol error then run the command above with the specific CQL version running on the VM (3.4.0) specified:

```
$ ./bin/cqlsh --cqlversion="3.4.0" <YOUR-VM-PUBLIC-IP> 9042
```

You should now be able to execute CQL commands, on the Cassandra instance running on your personal VM, from your local machine.

Set up your keyspace

CQL stores data in tables, whose schema defines the layout of said data in the table, and those tables are grouped in keyspaces. We will cover tables shortly, however first you need to create a keyspace to operate within.

A keyspace defines a number of options that applies to all the tables it contains, most prominently of which is the replication strategy used by the keyspace. It is generally encouraged to use one keyspace per application.

To set up a basic keyspace to operate in run the code below in the `cqlsh` prompt:

```
cqlsh> CREATE KEYSPACE csc8101 WITH replication = {'class': 'SimpleStrategy',
                                                   'replication_factor' : 1};
```

This command create a keyspace with the name "csc8101" and the options defined in the key, value map after the WITH clause. The exact operations and implications of this command are beyond the scope of this course. However, if you would like to know more, details are available in the Cassandra documentation.

If the create command executed correctly you should now be able to switch to the new keyspace using the following command:

```
cqlsh> USE csc8101;
```

The keyspace you are currently using should then be shown in the command prompt:

```
cqlsh:csc8101>
```

You are now ready to create tables, insert information and query the database.

Tasks

The log analysis system you are working receives JSON formatted messages from multiple servers in the following form:

```
{"client_id" : "4149719179064208",
 "timestamp" : "2017-01-25T14:03:32",
 "url" : {"topic" : "Action",
          "page" : "The Matrix"}
}
```

For the purposes of the following tasks, you can assume that an upstream system has extracted the relevant information from these messages (we are not expecting you to parse the raw JSON message). Therefore for the insert statements required in the task below you can simply provide your own dummy data.

The upstream system will collect messages into batches based on their arrival time. The timestamp values for the messages delivered to the database will be the same for all messages in the same batch and correspond to the start time of the collection window for that batch.

You should pay attention to the field names and data types in the original messages when forming your table schema, so that your database will be compatible with the upstream system. A later coursework will involve creating this upstream system so be sure to take care designing your schema as it will save you work later on.

Task 3: Recommend URLs for a User

Yes, we're are approaching these tasks in reverse order. Since Cassandra schema design is based on a 'query first' approach, you need to know what queries you are going to run before you can do the table design and testing. It is recommended that you read through all the tasks **before** you attempt to write the solutions.

You are required to design a system to recommend Pages for a given Client ID and Topic. The proposed process is that you should take the top N Pages for the given Topic from the most recent time period (batch). From this list you should exclude any pages that the Client ID has visited in the last 3 time periods. This will require you to use both the tables created in tasks 1 and 2.

To display the results of your query you are required to provide a simple program that can be used from the command line. It should accept a Client ID string, a Topic string and the value of N (the number of top URLs to use in the recommendation) as arguments. It should then return a recommended list of Pages.

For the purposes of this task your program can be very simple, you should just print the recommended URLs to the console. You can use either Python, Java or Scala to create your program. Links to the relevant Cassandra APIs for each are listed in the links section at the top of this page.

Task 2: URL distinct user counts

You are required to design a table schema to store the number of unique clients visiting each Topic and Page per time period.

For each event given to your database, you should assume that the upstream system will provide the Topic and Page as strings. The timestamp will be converted by the upstream system into a unix

timestamp (the number of milliseconds since 01 Jan 1970) so will need to be stored as a large integer. The user visit count will be supplied as an integer.

Once you have created your table, write a series of CQL statements to insert dummy data into the table.

To satisfy task 3, write a CQL query to list the top N Pages ranked by visitor count for a given Topic and time period. You can assume that the value of N will never be greater than 50.

Task 1: Client visits per URL

You are required to design a table schema to store the number of visits each Client ID makes to a given Topic and Page per batch time period.

For each event message supplied to your database, you should assume that the upstream system will provide a Client ID string, a batch time period unix timestamp as a large integer, a Topic string and a Page string. The visit count for each page will be supplied as an integer.

Once you have created your table, write a series of CQL statements to insert dummy data into it.

To satisfy task 3, write a CQL query to list for a given Client ID, Topic and batch time period, all the Pages they have visited.

Deliverables

Submission: Program code to provide the recommendations. Table definition statements in CQL. Test data that shows your program functions correctly in a variety of circumstances.

Viva: You may be expected to explain e.g. your table design decisions, how the database handles the queries at scale, how you chose your test data.