CS5590 BigData Programming - Lab Assignment 1

1. Finding facebook mutual friends using map reduce
2. Comparison of Hbase and Cassandra based on a usecase

Team Id : 14
Member 1 : Ruthvic Punyamurtula
Class Id : 16
Member 2 : Shankar Pentyala
Class Id : 15

Source Code :

https://github.com/Ruthvicp/CS5590_BigDataProgramming/tree/mas

Video/Demo : https://youtu.be/ppwkEDYgeio

Introduction

This lab assignment deals with understanding the concepts of hadoop - map reduce and also implementing a map reduce algorithm to find the mutual friends concept. The second part deals with comparison of Hbase and Cassandra based on a use case and user own data set.

Objective

1. Implement MapReduce algorithm for finding Facebook common friends problem and run the MapReduce job on Apache Hadoop. Show your implementation through map-reduce diagram

Approach

First, take the input as discussed in the use case as “A -> B C D, B -> A C D E, C -> A B D E, D -> A B C E, E -> B C D”
Then in map phase find the mutual friends of two people. Group them based on the mutual pair key and finally reduce them to get the mutual friends list.

Workflow

Create a mapper class as shown in the code snippet below. Each line of the input file is split based on “tab”. Then its length is computed = 2, where the first part is source or base user and the rest of the split is considered as list of friends of the user. Then the keys are prepared as (A,B) or (B,A) based on the integer values of A & B in the input.

```
    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
        String[] line = value.toString().split( regex: "\t");
        if (line.length == 2) {
            String friend1 = line[0];
            List<String> values = Arrays.asList(line[1].split( regex: ","));
            for (String friend2 : values) {
                int f1 = Integer.parseInt(friend1);
                int f2 = Integer.parseInt(friend2);
                if (f1 < f2)
                    word.set(friend1 + "," + friend2);
                else
                    word.set(friend2 + "," + friend1);
                context.write(word, new Text(line[1]));
            }
        }
    }

}
```

Create a reducer class where the data is grouped based on the key values (A,B) or (B,C) and their list of friends as produced. Then finally reduced to find the mutual friends of (A,B).

```
public static class Reduce extends Reducer<Text, Text, Text, Text> {
    private Text result = new Text();

    public void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {
        HashMap<String, Integer> map = new HashMap<String, Integer>();
        StringBuilder sb = new StringBuilder();
        for (Text friends : values) {
            List<String> temp = Arrays.asList(friends.toString().split( regex: ","));
            for (String friend : temp) {
                if (map.containsKey(friend))
                    sb.append(friend + ','); // append to string if friend already present
                else
                    map.put(friend, 1);

            }
        }
        if (sb.lastIndexOf( str: ",") > -1) {
            sb.deleteCharAt(sb.lastIndexOf( str: ","));
        }

        result.set(new Text(sb.toString()));
        context.write(key, result);
    }
}
```

A main method which acts as a driver to set mapper and reducer class which takes the input and produces the output.

```
    Job job = new Job(conf,  jobName: "MutualFriend");
    job.setJarByClass(MutualFriend.class);
    job.setMapperClass(Map.class);
    job.setReducerClass(Reduce.class);

    job.setOutputKeyClass(Text.class);

    job.setOutputValueClass(Text.class);
    // set the HDFS path of the input data
    FileInputFormat.addInputPath(job, new Path( pathString: "Input12"));
    // set the HDFS path for the output
    FileOutputFormat.setOutputPath(job, new Path( pathString: "Output123"));
    // Wait till job completion
    System.exit(job.waitForCompletion( verbose: true) ? 0 : 1);
}
```

Data set and Parameter

The input file for this can be found at

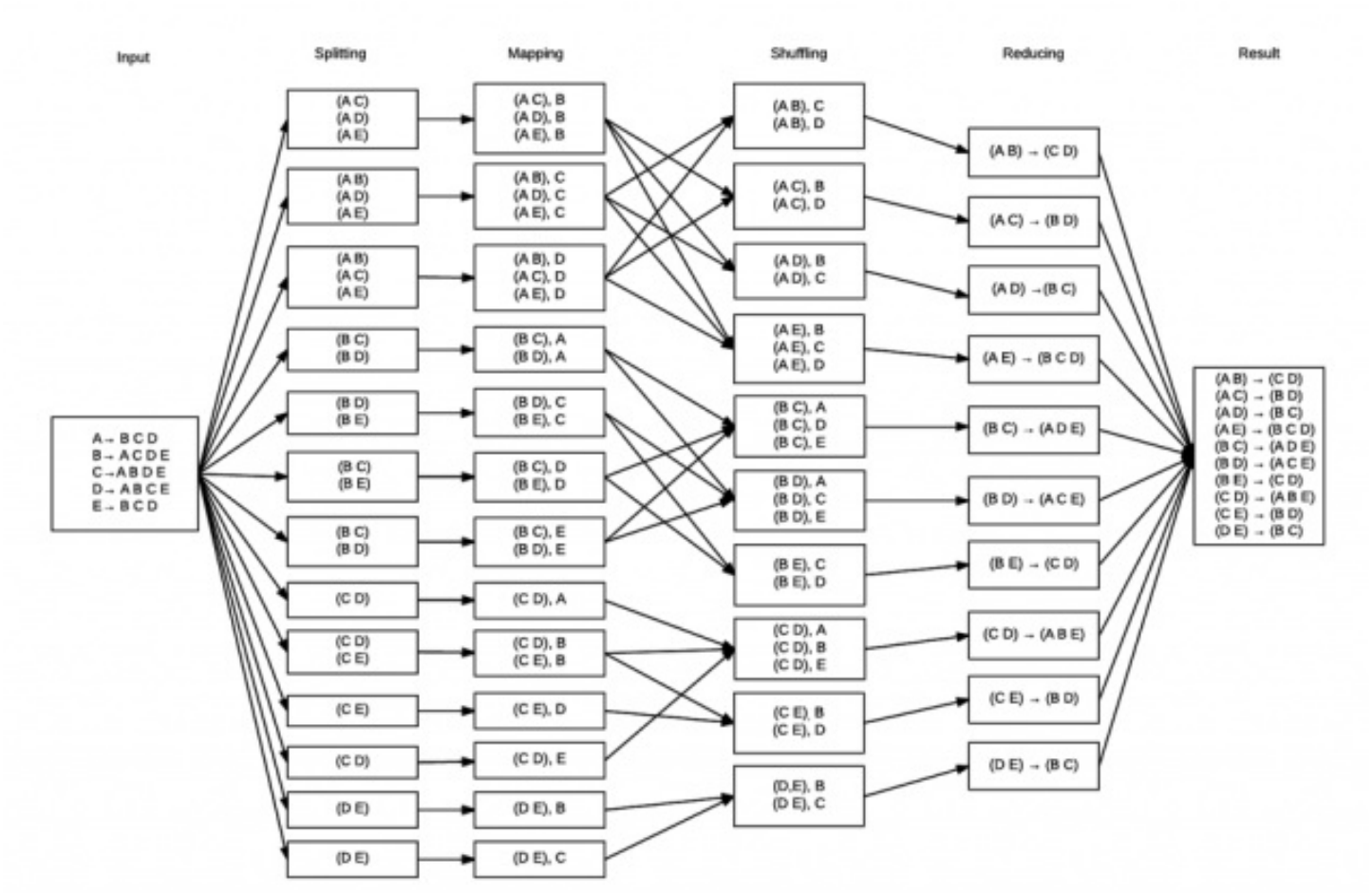
https://github.com/Ruthvicp/CS5590_BigDataProgramming/raw/master/Lab/

Evaluation

Hadoop map reduce is very efficient in finding the common/mutual friends of two users when their list of friends are grouped together and filtered using reduce operation

conclusion

Representing the mutual friends problem using the map reduce diagram



---

2. No SQL comparison - Cassandra Vs

Hbase

Introduction

Consider the use case of Netflix. We create a Netflix users database model in order to find the users based on region, last activity of trial users, find the paid users and their favorite genre to provide recommendations to the user.

Objectives

a)Consider netflix use case and use a simple data set. Describe the use case considered based on your assumptions, report the data set, its fields, datatype etc.

b)Use HBase to implement a Solution for the use case. Report at least 3 queries, their input and output. The query’s relevance towards solving the use case is important.

c)Use Cassandra to implement a Solution for the use case. Report at least 3 queries, their input and output. The query’s relevance towards solving the use case is important.

d)Compare Cassandra and HBase for your use case. Present a table with comparison of your use case being implemented in both NO SQL Systems.

Approach

Cassandra :

Create a table in Cassandra to store the data set as shown below.

```
cqlsh:kstest> create table netflix_users (user_id int, user_type text, user_name text, joined_date date, last_activity d
ate, last_watched text, country text, genre text, Primary key (user_id));
cqlsh:kstest> describe tables;

department  employee  employees  netflix_users
```

Hbase :

In Hbase as well, create a similar table to process the Netflix users data.

```
hduser@ruthvic-VirtualBox:/usr/lib/hbase/hbase-0.94.8/bin$ ./hbase shellHBase Shell; enter 'help<RETURN>' for list of supported commands.
Type "exit<RETURN>" to leave the HBase Shell
Version 0.94.8, r1485407, Wed May 22 20:53:13 UTC 2013

hbase(main):001:0> create 'netflix_users','user_details','user_activity'
0 row(s) in 3.4560 seconds

hbase(main):002:0> put 'netflix_users',1,'user_details:user_name','ruth'
0 row(s) in 0.4270 seconds

hbase(main):003:0> put 'netflix_users',1,'user_details:user_type','trial'
0 row(s) in 0.0460 seconds

hbase(main):004:0> put 'netflix_users',1,'user_details:country','india'
0 row(s) in 0.0120 seconds

hbase(main):005:0> put 'netflix_users',1,'user_details:genre','sci-fi'
0 row(s) in 0.0300 seconds

hbase(main):006:0> get 'netflix_users',1
COLUMN                CELL
 user_details:country timestamp=1529284949220, value=india
 user_details:genre   timestamp=1529284968441, value=sci-fi
 user_details:user_na timestamp=1529284863167, value=ruth
 me
 user_details:user_ty timestamp=1529284878198, value=trial
 pe
4 row(s) in 0.1720 seconds
```

Workflow

**Cassandra Queries : **

- Insert data

```
cqlsh:kstest> Insert into netflix_users (user_id, user_type, user_name, joined_date, last_activity, last_w
atched, country, genre) values (5,'inactive','thanos','2015-04-14','2015-05-02','jurassic park','usa','com
edy');
cqlsh:kstest> select * from netflix_users;

 user_id | country | genre    | joined_date | last_activity | last_watched         | user_name | user_type

---------+---------+----------+-------------+---------------+----------------------+-----------+----------
-
       5 |     usa |   comedy |  2015-04-14 |    2015-05-02 |        jurassic park |    thanos |  inactive

       1 |   india |   sci-fi |  2017-07-20 |    2018-06-16 | marvel infinity wars |      ruth |     trial

       2 |   india | thriller |  2018-05-15 |    2018-06-17 |              rampage |     punya |     trial

       4 |  europe |   horror |  2017-01-06 |    2018-05-03 |            conjuring |     chris |      paid

       3 |     usa | animated |  2016-02-27 |    2018-06-13 |            baby boss |       sam |      paid
```

- Find inactive users

```
cqlsh:kstest> select * from netflix_users where user_type = 'inactive';
InvalidRequest: Error from server: code=2200 [Invalid query] message="Cannot execute this query as it migh
t involve data filtering and thus may have unpredictable performance. If you want to execute this query de
spite the performance unpredictability, use ALLOW FILTERING"
cqlsh:kstest> select * from netflix_users where user_type = 'inactive' allow filtering;

 user_id | country | genre  | joined_date | last_activity | last_watched  | user_name | user_type
---------+---------+--------+-------------+---------------+---------------+-----------+-----------
       5 |     usa | comedy |  2015-04-14 |    2015-05-02 | jurassic park |    thanos |  inactive
```

- Find paid users

```
cqlsh:kstest> select last_activity,genre  from netflix_users where user_type='paid' allow filtering;

 last_activity | genre
---------------+----------
    2018-05-03 |   horror
    2018-06-13 | animated
```

- Find trial end date of a new user

```
cqlsh:kstest> select trial_end_date,genre,joined_date from netflix_users where user_id=2;

 trial_end_date | genre    | joined_date
----------------+----------+-------------
     2018-07-15 | thriller |  2018-05-15
```

HBase Queries :

- Find trial users

```
hbase(main):024:0> scan 'netflix_users', {FILTER => "SingleColumnValueFilter('user_details','user_type',=,'binary:trial')"}
ROW                                           COLUMN+CELL
 1                                            column=user_activity:joined_date, timestamp=1529285225368, value=2017-07-20
 1                                            column=user_activity:last_activity, timestamp=1529285282283, value=2018-06-16
 1                                            column=user_activity:last_watched, timestamp=1529285419798, value=marvel infinity wars
 1                                            column=user_details:country, timestamp=1529284949220, value=india
 1                                            column=user_details:genre, timestamp=1529284968441, value=sci-fi
 1                                            column=user_details:user_name, timestamp=1529284863167, value=ruth
 1                                            column=user_details:user_type, timestamp=1529284878198, value=trial
 2                                            column=user_activity:joined_date, timestamp=1529285796755, value=2018-06-17
 2                                            column=user_activity:last_activity, timestamp=1529285852987, value=2018-05-15
 2                                            column=user_activity:last_watched, timestamp=1529285958765, value=rampage
 2                                            column=user_activity:trial_end_date, timestamp=1529286007198, value=2018-07-15
 2                                            column=user_details:country, timestamp=1529285656798, value=india
 2                                            column=user_details:genre, timestamp=1529285697334, value=thriller
 2                                            column=user_details:user_name, timestamp=1529285578228, value=punya
 2                                            column=user_details:user_type, timestamp=1529285604073, value=trial
2 row(s) in 0.4780 seconds
```

- Find users who watched a particular movie on netflix

```
hbase(main):025:0> scan 'netflix_users', {FILTER => "SingleColumnValueFilter('user_activity','last_watched',=,'binary:rampage')"}
ROW                                           COLUMN+CELL
 2                                            column=user_activity:joined_date, timestamp=1529285796755, value=2018-06-17
 2                                            column=user_activity:last_activity, timestamp=1529285852987, value=2018-05-15
 2                                            column=user_activity:last_watched, timestamp=1529285958765, value=rampage
 2                                            column=user_activity:trial_end_date, timestamp=1529286007198, value=2018-07-15
 2                                            column=user_details:country, timestamp=1529285656798, value=india
 2                                            column=user_details:genre, timestamp=1529285697334, value=thriller
 2                                            column=user_details:user_name, timestamp=1529285578228, value=punya
 2                                            column=user_details:user_type, timestamp=1529285604073, value=trial
1 row(s) in 0.1560 seconds
```

- Find the region and other personal details of user

```
hbase(main):022:0> get 'netflix_users',1,'user_details:country'
COLUMN                                        CELL
 user_details:country                         timestamp=1529284949220, value=india
1 row(s) in 0.0650 seconds
```

Datasets & parameters

The input data set for both cassandra and hbase can be found here (Netflix data)

Evaluation

HBase vs Cassandra Comparison Table

Points	HBase	Cassandra
CAP Theorem	Consistency & Availability	Availability and Partition Tolerance
Coprocessor	Yes	No
Rebalancing	HBase provides Automatic rebalancing within a cluster.	Cassandra also provides rebalancing but not for overall cluster
Architecture Model	It is based on Master-Slave Architecture Model	Cassandra is based on Active-Active Node Modal
Base of Database	It is based on Google BigTable	Cassandra is based on Amazon DynamoDB
SPoF (Single Point of Failure)	If Master Node is not available entire cluster will not be accessible	All nodes having the same role within cluster so no SPoF
DR (Disaster Recovery)	DR is possible if Two Master Nodes are configured.	Yes, as all nodes having the same role
HDFS Compatibility	Yes, As HBase stores all meta-data in HDFS	No
Consistency	Strong	Not Strong as HBase

Conclusion

Cassandra Key characteristics involve High Availability, Minimal administration and No SPoF (Single Point of Failure) other side HBase is good for faster reading and writing the data with linear scalability.

References

- https://www.educba.com/hbase-vs-cassandra/
- https://www.linkedin.com/pulse/real-comparison-nosql-databases-hbase-cassandra-mongodb-sahu/
- http://stevekrenzel.com/finding-friends-with-mapreduce