**BDS Assignment 2**

**Group 75**

**Nareshkumar P (2020FC04122)**

**Kommajyosula VNS Kanth (2020fc04120)**

**Dola Tejesh (2020fc04459)**

**Tools Used:**

**Oracle VM Virtualbox Manager** **->** used to create multiple nodes to communicate each other in the cluster. (For 8 nodes, created 8 new instances (m1,m2…m8) with 1 core, 1gb ram, 8gb harddisk)

**OS ->** Ubuntu 18.04

**HDFS ->** To load data in hdfs file system.

**Hive ->** To create relation tables for the given data file.

**Mysql db ->** Hive metastore

**Spark ->** To query the hive tables.

**Assignment Hands-on steps overview:**

**Step 1:** Using Oracle VM Virtualbox Manager, created namenodes and multiple datanodes with same configurations.

**Step2:** Installed Hadoop in all the nodes and update the datanode, namenode details in the configuration files. (core-site.xml, mapred-site.xml, hdfs-site.xml, yarn-site.xml)

**Step 3:** Updated the replication factor in hdfs-site xml file and load the data in to hdfs and verify the data loaded as per the replication factor value.

**Step 4:** To avoid the redundancy, created three new Hive tables to populate the normalized data of the given file (Coursera-1.csv). Tables are created using hive DDLs.

Hive tables: University\_Dim, Difficulty\_level\_Dim, Coursera\_Dim

**Step 5:** Verified the metadata of the table stored in metastore. (Mysql is our metastore of our Hive database)

**Step 6:** Loaded the normalized data into all the three new Hive tables using hive DMLs. Removed duplicates using ‘distinct’ keyword when inserting records from file to coursera\_dim hive table.

**Step 7:** Verified no duplicate records exists in the coursera\_dim hive table and dimension ids are properly populated.

**Step 8:** Used the ORC file format for the Hive tables. It heavily compresses the file size and also the performance of the column level manipulation/calculation is faster since the data stored in the columnar level.

**Step 9:** Verified the size of the hive data file with actual data file and found hive data file is five time lesser than the actual data size.

**Step 10:** Used Spark Sql to query the hive tables. Queried a table to retrieve data using an id column. Tried retrieving data from both hive CLI and spark sql.

1. **Storage:**
2. **Mutually Exclusive feature data (column value) which is not common across records (rows): private node:**

Update the replication factor as 1 in hdfs-site.xml to load the data in only node and for the security reasons, it won’t share to any other nodes.

**Hadoop Configurations:**

Having only one machine. (m1)

M1 -> it going to act as Namenode, Datanode, Resource Manager, Node Manager, Secondary Namenode

**Core-site.xml:**

<configuration>

<property>

<name>fs.defaultFS</name>

<value>hdfs://m1:9000</value>

</property>

</configuration>

**Mapred-site.xml:**

<configuration>

<property>

<name>mapred.framework.name</name>

<value>yarn</value>

</property>

</configuration>

**Hdfs-site.xml:**

<configuration>

<property>

<name>dfs.replication</name>

**<value>1</value>**

</property>

<property>

<name>dfs.namenode.name.dir</name>

<value>/credo/name</value>

<final>true</final>

</property>

<property>

<name>dfs.datanode.data.dir</name>

<value>/credo/data</value>

<final>true</final>

</property>

</configuration>

**Yarn-site.xml:**

<property>

<name>yarn.nodemanager.aux-services.mapreduce\_shuffle.class</name>

<value>org.apache.hadoop.mapred.ShuffleHandler</value>

</property>

<property>

<name>yarn.nodemanager.aux-services</name>

<value>mapreduce\_shuffle</value>

</property>

<property>

<name>yarn.nodemanager.vmem-check-enabled</name>

<value>false</value>

</property>

</configuration>

**Load file into single node cluster:**

Loading the given Coursera-1.csv file from local to HDFS. Coursera-1.csv

**hadoop fs -put Coursera-1.csv /user/hive/data**

The data is stored in only one datanode since the replication factor is 1.

1. **Feature data common in two records: 2-way shared node:**

Change the replication factor as 2 in hdfs-site.xml to load and share the data between only 2 nodes. In case of any issue on one node, other node’s data will be available.

**Hadoop Configurations:**

Having two machines -> m1, m2

M1 -> it going to act as Namenode, Datanode, Resource Manager, Node Manager

M2 -> It going to act as Datanode, Node Manager, Secondary Namenode

**Core-site.xml:**

<configuration>

<property>

<name>fs.defaultFS</name>

<value>hdfs://m1:9000</value>

</property>

</configuration>

**Mapred-site.xml:**

<configuration>

<property>

<name>mapred.framework.name</name>

<value>yarn</value>

</property>

</configuration>

**Hdfs-site.xml:**

<configuration>

<property>

<name>dfs.replication</name>

<value>2</value>

</property>

<property>

<name>dfs.namenode.name.dir</name>

<value>/credo/name</value>

<final>true</final>

</property>

<property>

<name>dfs.datanode.data.dir</name>

<value>/credo/data</value>

<final>true</final>

</property>

<property>

<name>dfs.namenode.http-address</name>

<value>m1:50070</value>

</property>

<property>

<name>dfs.namenode.secondary.http-address</name>

<value>m2:50090</value>

</property>

<property>

<name>dfs.namenode.checkpoint.period</name>

<value>600</value>

</property>

</configuration>

**Yarn-site.xml:**

<configuration>

<property>

<name>yarn.resourcemanager.address</name>

<value>m1:9001</value>

</property>

<property>

<name>yarn.resourcemanager.resource-tracker.address</name>

<value>m1:8031</value>

</property>

<property>

<name>yarn.nodemanager.aux-services.mapreduce\_shuffle.class</name>

<value>org.apache.hadoop.mapred.ShuffleHandler</value>

</property>

<property>

<name>yarn.nodemanager.aux-services</name>

<value>mapreduce\_shuffle</value>

</property>

<property>

<name>yarn.nodemanager.pmem-check-enabled</name>

<value>false</value>

</property>

<property>

<name>yarn.nodemanager.vmem-check-enabled</name>

<value>false</value>

</property>

<property>

<name>yarn.log-aggregation-enable</name>

<value>true</value>

</property>

<property>

<name>yarn.nodemanager.log-aggregation.roll-monitoring-interval-seconds</name>

<value>3600</value>

</property>

</configuration>

**Slaves:**

m1

m2

**Machine 2 (m2) Configuration:**

All the configurations are same as machine 1 (m1) except few configurations in hdfs-site.xml

**Hdfs-site.xml:**

<configuration>

<property>

<name>dfs.replication</name>

<value>2</value>

</property>

<property>

<name>dfs.datanode.data.dir</name>

<value>/credo/data2</value>

<final>true</final>

</property>

<property>

<name>dfs.namenode.http-address</name>

<value>m1:50070</value>

</property>

<property>

<name>dfs.namenode.secondary.http-address</name>

<value>m2:50090</value>

</property>

<property>

<name>dfs.namenode.checkpoint.period</name>

<value>600</value>

</property>

</configuration>

**Load file 2-way shared node:**

Loading the given Coursera-1.csv file from local to HDFS. Coursera-1.csv

**hadoop fs -put Coursera-1.csv /user/hive/data**

The data stored as two copies and each copy loaded into two different datanodes since the replication factor is 2.

1. **Feature data common in four records: 4-way shared node:**

Change the replication factor as 4 in hdfs-site.xml to load and share the data between four different datanodes. High availability is guaranteed incase of any node failures.

**Hadoop Configurations:**

Having two machines -> m1, m2, m3, m4

M1 -> it going to act as Namenode, Datanode, Resource Manager, Node Manager

M2 -> It going to act as Datanode, Node Manager, Secondary Namenode

M3 -> It going to act as Datanode, Node Manager

M4 -> It going to act as Datanode, Node Manager

**Core-site.xml:**

<configuration>

<property>

<name>fs.defaultFS</name>

<value>hdfs://m1:9000</value>

</property>

</configuration>

**Mapred-site.xml:**

<configuration>

<property>

<name>mapred.framework.name</name>

<value>yarn</value>

</property>

</configuration>

**Hdfs-site.xml:**

<configuration>

<property>

**<name>dfs.replication</name>**

**<value>4</value>**

</property>

<property>

<name>dfs.namenode.name.dir</name>

<value>/credo/name</value>

<final>true</final>

</property>

<property>

<name>dfs.datanode.data.dir</name>

<value>/credo/data</value>

<final>true</final>

</property>

<property>

<name>dfs.namenode.http-address</name>

<value>m1:50070</value>

</property>

<property>

<name>dfs.namenode.secondary.http-address</name>

<value>m2:50090</value>

</property>

<property>

<name>dfs.namenode.checkpoint.period</name>

<value>600</value>

</property>

</configuration>

**Yarn-site.xml:**

<configuration>

<property>

<name>yarn.resourcemanager.address</name>

<value>m1:9001</value>

</property>

<property>

<name>yarn.resourcemanager.resource-tracker.address</name>

<value>m1:8031</value>

</property>

<property>

<name>yarn.nodemanager.aux-services.mapreduce\_shuffle.class</name>

<value>org.apache.hadoop.mapred.ShuffleHandler</value>

</property>

<property>

<name>yarn.nodemanager.aux-services</name>

<value>mapreduce\_shuffle</value>

</property>

<property>

<name>yarn.nodemanager.pmem-check-enabled</name>

<value>false</value>

</property>

<property>

<name>yarn.nodemanager.vmem-check-enabled</name>

<value>false</value>

</property>

<property>

<name>yarn.log-aggregation-enable</name>

<value>true</value>

</property>

<property>

<name>yarn.nodemanager.log-aggregation.roll-monitoring-interval-seconds</name>

<value>3600</value>

</property>

</configuration>

**Slaves:**

m1

m2

m3

m4

**Machine 2,3,4 (m2,m3,m4) Configuration:**

All the configurations are same as machine 1 (m1) except few configurations in hdfs-site.xml

**Hdfs-site.xml:**

<configuration>

<property>

<name>dfs.replication</name>

**<value>4</value>**

</property>

<property>

<name>dfs.datanode.data.dir</name>

<value>/credo/data3</value>

<final>true</final>

</property>

<property>

<name>dfs.namenode.http-address</name>

<value>m1:50070</value>

</property>

<property>

<name>dfs.namenode.secondary.http-address</name>

<value>m2:50090</value>

</property>

<property>

<name>dfs.namenode.checkpoint.period</name>

<value>600</value>

</property>

</configuration>

**Load file 4-way shared node:**

Loading the given Coursera-1.csv file from local to HDFS. Coursera-1.csv

**hadoop fs -put Coursera-1.csv /user/hive/data**

The data stored as four copies and each copy loaded into four different datanodes since the replication factor is 4.

1. **Feature data common in 8 records: 8-way shared node:**

Change the replication factor as 8 in hdfs-site.xml to load and share the data between eight different datanodes. It gives high availability during multiple node failures but occupies more space in the cluster.

**Hadoop Configurations:**

Having two machines -> m1, m2, m3, m4, m5, m6, m7, m8

M1 -> it going to act as Namenode, Datanode, Resource Manager, Node Manager

M2 -> It going to act as Datanode, Node Manager, Secondary Namenode

M3 -> It going to act as Datanode, Node Manager

M4 -> It going to act as Datanode, Node Manager

M5 -> It going to act as Datanode, Node Manager

M6 -> It going to act as Datanode, Node Manager

M7 -> It going to act as Datanode, Node Manager

M8 -> It going to act as Datanode, Node Manager

**Core-site.xml:**

<configuration>

<property>

<name>fs.defaultFS</name>

<value>hdfs://m1:9000</value>

</property>

</configuration>

**Mapred-site.xml:**

<configuration>

<property>

<name>mapred.framework.name</name>

<value>yarn</value>

</property>

</configuration>

**Hdfs-site.xml:**

<configuration>

<property>

**<name>dfs.replication</name>**

**<value>8</value>**

</property>

<property>

<name>dfs.namenode.name.dir</name>

<value>/credo/name</value>

<final>true</final>

</property>

<property>

<name>dfs.datanode.data.dir</name>

<value>/credo/data</value>

<final>true</final>

</property>

<property>

<name>dfs.namenode.http-address</name>

<value>m1:50070</value>

</property>

<property>

<name>dfs.namenode.secondary.http-address</name>

<value>m2:50090</value>

</property>

<property>

<name>dfs.namenode.checkpoint.period</name>

<value>600</value>

</property>

</configuration>

**Yarn-site.xml:**

<configuration>

<property>

<name>yarn.resourcemanager.address</name>

<value>m1:9001</value>

</property>

<property>

<name>yarn.resourcemanager.resource-tracker.address</name>

<value>m1:8031</value>

</property>

<property>

<name>yarn.nodemanager.aux-services.mapreduce\_shuffle.class</name>

<value>org.apache.hadoop.mapred.ShuffleHandler</value>

</property>

<property>

<name>yarn.nodemanager.aux-services</name>

<value>mapreduce\_shuffle</value>

</property>

<property>

<name>yarn.nodemanager.pmem-check-enabled</name>

<value>false</value>

</property>

<property>

<name>yarn.nodemanager.vmem-check-enabled</name>

<value>false</value>

</property>

<property>

<name>yarn.log-aggregation-enable</name>

<value>true</value>

</property>

<property>

<name>yarn.nodemanager.log-aggregation.roll-monitoring-interval-seconds</name>

<value>3600</value>

</property>

</configuration>

**Slaves:**

m1

m2

m3

m4

m5

m6

m7

m8

**Machine 2,3,4,5,6,7,8 (m2,m3,m4,m5,m6,m7,m8) Configuration:**

All the configurations are same as machine 1 (m1) except few configurations in hdfs-site.xml

**Hdfs-site.xml:**

<configuration>

<property>

<name>dfs.replication</name>

**<value>8</value>**

</property>

<property>

<name>dfs.datanode.data.dir</name>

<value>/credo/data5</value>

<final>true</final>

</property>

<property>

<name>dfs.namenode.http-address</name>

<value>m1:50070</value>

</property>

<property>

<name>dfs.namenode.secondary.http-address</name>

<value>m2:50090</value>

</property>

<property>

<name>dfs.namenode.checkpoint.period</name>

<value>600</value>

</property>

</configuration>

**Load file 8-way shared node:**

Loading the given Coursera-1.csv file from local to HDFS. Coursera-1.csv

**hadoop fs -put Coursera-1.csv /user/hive/data**

The data stored as eight copies and each copy loaded into eight different datanodes since the replication factor is 8.

1. **Metadata:**

**DB used: Hive**

**Metadata Storage: Mysql db**

Table structure of given coursera csv file.

**Coursera\_data:**

|  |  |
| --- | --- |
| **Column\_Name** | **Datatype** |
| Course\_name | String |
| University | String |
| Difficulty\_level | String |
| Course\_Rating | String |
| Course\_URL | String |
| Course\_Description | String |
| Skills | String |

Created a hive table to load the input csv file in Hadoop cluster.

**DDL:**

create table if not exists bds.coursera\_data

(

course\_name string,

university string,

difficulty\_level string,

course\_rating string,

course\_url string,

course\_description string,

skills string

)

row format delimited

fields terminated by ',' escaped by '\\'

location '/user/hive/bds/coursera\_data'

tblproperties ("skip.header.line.count"="1");

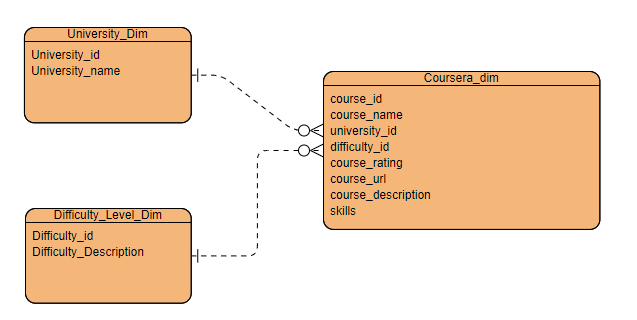
**Load command to populate data into Hadoop/HDFS:**

load data inpath '/user/hive/data/Coursera-1.csv' into table bds.coursera\_data;

**Metadata design change to decrease the storage size:**

1. **Normalizing the Data:**

Created two new dimension tables (Difficulty\_level\_dim, University\_dim) to avoid the redundancy in the coursera data. Choosing the two columns which has redundant data and creating two reference tables for the coursera data to refer their values using their ids.



**Difficulty\_level\_dim:**

|  |  |
| --- | --- |
| Column\_Name | Datatype |
| Difficulty\_id | int |
| Difficulty\_description | string |

**DDL:**

create table if not exists bds.difficulty\_level\_dim

(

difficulty\_id int,

difficulty\_description string

)

location '/user/hive/bds/difficulty\_level\_dim';

**DML:**

insert overwrite table bds.difficulty\_level\_dim

select ROW\_NUMBER() OVER () AS id, sub.difficulty\_level from (select distinct difficulty\_level from bds.coursera\_data where difficulty\_level is not null) sub;

**University\_dim:**

|  |  |
| --- | --- |
| **Column Name** | **Datatype** |
| University\_id | int |
| University\_name | String |

**DDL:**

create table if not exists bds.university\_dim

(

university\_id int,

university\_name string

)

location '/user/hive/bds/university\_dim';

**DML:**

insert overwrite table bds.university\_dim

select ROW\_NUMBER() OVER () AS id, sub.university from (select distinct university from bds.coursera\_data where university is not null) sub;

**Coursera\_dim:**

Created a new column course\_id which is a unique identifier of a course\_name. It helps to retrieve data very quicker.

|  |  |
| --- | --- |
| **Column Name** | **Datatype** |
| Course\_id | Int |
| Course\_name | String |
| University\_id | Int |
| Difficulty\_id | Int |
| Course\_rating | String |
| Course\_url | String |
| Course\_description | String |
| skills | string |

**DDL:**

create table if not exists bds.coursera\_dim

(

course\_id int,

course\_name string,

university\_id int,

difficulty\_id int,

course\_rating string,

course\_url string,

course\_description string,

skills string

)

row format delimited

fields terminated by ','

stored as orc

location '/user/hive/bds/coursera\_dim';

**DML:**

insert overwrite table bds.coursera\_dim

select ROW\_NUMBER() OVER () AS course\_id, course\_name, university\_id, difficulty\_id, course\_rating, course\_url, course\_description, skills from

(select distinct cd.course\_name, ud.university\_id, dld.difficulty\_id, cd.course\_rating, cd.course\_url, cd.course\_description, cd.skills from bds.coursera\_data cd

left outer join bds.difficulty\_level\_dim dld on cd.difficulty\_level=dld.difficulty\_description

left outer join bds.university\_dim ud on cd.university=ud.university\_name) sub;

1. **Removing duplicate records**

We could see around 100s of duplicate records present in the given dataset hence we have ignored the duplicate records while loading coursera data to coursera\_dim table and reduced the storage space.

**DML: (Removed duplicate records based on the below query)**

insert overwrite table bds.coursera\_dim

select ROW\_NUMBER() OVER () AS course\_id, course\_name, university\_id, difficulty\_id, course\_rating, course\_url, course\_description, skills from

(select distinct cd.course\_name, ud.university\_id, dld.difficulty\_id, cd.course\_rating, cd.course\_url, cd.course\_description, cd.skills from bds.coursera\_data cd

left outer join bds.difficulty\_level\_dim dld on cd.difficulty\_level=dld.difficulty\_description

left outer join bds.university\_dim ud on cd.university=ud.university\_name) sub;

1. **Saved as ORC file format**

ORC file format helps to compress the size of the file. It’s a row columnar file and it helps to load the record in column level storage. It has own storing logic to retrieve data very faster and also do aggregate calculations on columns very faster.

DDL: (Stored as ORC command helps to load the records in ORC format)

create table if not exists bds.coursera\_dim

(

course\_id int,

course\_name string,

university\_id int,

difficulty\_id int,

course\_rating string,

course\_url string,

course\_description string,

skills string

)

row format delimited

fields terminated by ','

stored as orc

location '/user/hive/bds/coursera\_dim';

Actual file size: 5mb

ORC file size: 1 mb

**We could see the enormous reduction in ORC file size. Its compressed 5 times lesser than the actual file size.**

1. **Retrieval**

Fetching the records in coursera\_dim table using newly created column course\_id (has unique id) helps to retrieve the records quicker.

**Fetching record using course\_id:**

hive> select \* from bds.coursera\_dim where course\_id=1500;

OK

1500 Ruby on Rails: An Introduction 118 2 4.5 https://www.coursera.org/learn/ruby-on-rails-intro "Did you ever want to build a web application? Perhaps you even started down that path in a language like Java or C#, when you realized that there was so much �climbing the mountain� that you had to do? Maybe you have heard about web services being all the rage, but thought they were too complicated to integrate into your web application. Or maybe you wondered how deploying web applications to the cloud works, but there was too much to set up just to get going. In this course, we will explore how to build web applications with the Ruby on Rails web application framework, which is geared towards rapid prototyping. Yes, that means building quickly! At the conclusion of this course, you will be able to build a meaningful web application and deploy it to the �cloud� using a Heroku PaaS (Platform as a Service). Best of all, it will almost feel effortless� Really! �But wait�, you will say, �there is no way that we can build a useful application if there is no database involved. You need the data for an application to be useful.� Great point! But what if� instead of getting the data from the database, we get it from the internet by tapping into one of the web services out there that readily provides data needed by our application? �Ok, but that�s probably very complicated�, you will say. Take this course and you will be pleasantly surprised at just how easy it is!" language github web unit testing ruby (programming language) rubygems ruby on rails Web Development rspec Computer Programming computer-science mobile-and-web-development

Time taken: 0.816 seconds, Fetched: 1 row(s)

**Fetching record based on course\_name:**

hive> select \* from bds.coursera\_data where course\_name='Ruby on Rails: An Introduction';

OK

Ruby on Rails: An Introduction Johns Hopkins University Intermediate 4.5 https://www.coursera.org/learn/ruby-on-rails-intro "Did you ever want to build a web application? Perhaps you even started down that path in a language like Java or C#, when you realized that there was so much �climbing the mountain� that you had to do? Maybe you have heard about web services being all the rage, but thought they were too complicated to integrate into your web application. Or maybe you wondered how deploying web applications to the cloud works, but there was too much to set up just to get going. In this course, we will explore how to build web applications with the Ruby on Rails web application framework, which is geared towards rapid prototyping. Yes, that means building quickly! At the conclusion of this course, you will be able to build a meaningful web application and deploy it to the �cloud� using a Heroku PaaS (Platform as a Service). Best of all, it will almost feel effortless� Really! �But wait�, you will say, �there is no way that we can build a useful application if there is no database involved. You need the data for an application to be useful.� Great point! But what if� instead of getting the data from the database, we get it from the internet by tapping into one of the web services out there that readily provides data needed by our application? �Ok, but that�s probably very complicated�, you will say. Take this course and you will be pleasantly surprised at just how easy it is!" language github web unit testing ruby (programming language) rubygems ruby on rails Web Development rspec Computer Programming computer-science mobile-and-web-development

Time taken: 0.969 seconds, Fetched: 1 row(s)

**When comparing both retrieving queries, the query which uses course\_id column performs better than the query which uses course\_name column.**