Bucketing in Hive

Bucketing

Bucketing is a method to evenly distribute the data across many files. Create multiple buckets and then place each record into one of the buckets based on some logic mostly some hashing algorithm.

Bucketing feature of Hive can be used to distribute/organize the table/partition data into multiple files such that similar records are present in the same file. While creating a Hive table, a user needs to give the columns to be used for bucketing and the number of buckets to store the data into. Which records go to which bucket are decided by the Hash value of columns used for bucketing.

[Hash(column(s))] MOD [Number of buckets]

Hash value for different columns types is calculated differently. For int columns, the hash value is equal to the value of int. For String columns, the hash value is calculated using some computation on each character present in the String.

Data for each bucket is stored in a separate HDFS file under the table directory on HDFS. Inside each bucket, we can define the arrangement of data by providing the SORT BY column while creating the table.

For partitioned table, we will have n buckets created per partition.

Example

**Creating a Hive table using bucketing**

For creating a bucketed table, we need to use CLUSTERED BY clause to define the columns for bucketing and provide the number of buckets. Following query creates a table Employee bucketed using the ID column into 5 buckets.

CREATE TABLE Employee(

ID BIGINT,

NAME STRING,

AGE INT,

SALARY BIGINT,

DEPARTMENT STRING

)

COMMENT 'This is Employee table stored as textfile clustered by id into 5 buckets'

CLUSTERED BY(ID) INTO 5 BUCKETS

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

STORED AS TEXTFILE;

**Inserting data into a bucketed table**

We have following data in Employee\_old table.

CREATE TABLE Employee\_old(

employee\_oldID BIGINT,

employee\_oldNAME STRING,

employee\_oldAGE INT,

employee\_oldSALARY BIGINT,

employee\_oldDEPARTMENT STRING

)

COMMENT 'This is Employee table stored as textfile clustered by id into 5 buckets'

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

STORED AS TEXTFILE;

0: jdbc:hive2://localhost:10000> select \* from employee\_old;

+------------------+--------------------+-------------------+----------------------+--------------------------+--+

| employee\_old.id  | employee\_old.name  | employee\_old.age  | employee\_old.salary  | employee\_old.department  |

+------------------+--------------------+-------------------+----------------------+--------------------------+--+

| 1                | Sudip              | 34                | 62000                | HR                       |

| 2                | Suresh             | 45                | 76000                | FINANCE                  |

| 3                | Aarti              | 25                | 37000                | BIGDATA                  |

| 4                | Neha               | 27                | 39000                | FINANCE                  |

| 5                | Rajesh             | 29                | 59000                | BIGDATA                  |

| 6                | Suman              | 37                | 63000                | HR                       |

| 7                | Paresh             | 42                | 71000                | BIGDATA                  |

| 8                | Rami               | 33                | 56000                | HR                       |

| 9                | Arpit              | 41                | 46000                | HR                       |

| 10               | Sanjeev            | 51                | 99000                | FINANCE                  |

| 11               | Sanjay             | 32                | 67000                | FINANCE                  |

+------------------+--------------------+-------------------+----------------------+--------------------------+--+

We will select data from the table Employee\_old and insert it into our bucketed table Employee.

We need to set the property ‘hive.enforce.bucketing‘ to true while inserting data into a bucketed table. This will enforce bucketing, while inserting data into the table.

#Set the property

0: jdbc:hive2://localhost:10000> set hive.enforce.bucketing=true;

#Insert data into Bucketed table employee

0: jdbc:hive2://localhost:10000> INSERT OVERWRITE TABLE Employee SELECT \* from Employee\_old;

**Verify the Data in Buckets**

Once we execute the INSERT query, we can verify that 5 files are created Under the Employee table directory on HDFS.

**Name        Type**

000000\_0    file

000001\_0    file

000002\_0    file

000003\_0    file

000004\_0    file

Each file represents a bucket. Let us see the contents of these files.

Content of 000000\_0

All records with Hash(ID) mod 5 == 0 goes into this file.

5,Rajesh,29,59000,BIGDATA

10,Sanjeev,51,99000,FINANCE

Content of 000001\_0

All records with Hash(ID) mod 5 == 1 goes into this file.

1,Sudip,34,62000,HR

6,Suman,37,63000,HR

11,Sanjay,32,67000,FINANCE

Content of 000002\_0

All records with Hash(ID) mod 5 == 2 goes into this file.

2,Suresh,45,76000,FINANCE

7,Paresh,42,71000,BIGDATA

Content of 000003\_0

All records with Hash(ID) mod 5 == 3 goes into this file.

3,Aarti,25,37000,BIGDATA

8,Rami,33,56000,HR

Content of 000004\_0

All records with Hash(ID) mod 5 == 4 goes into this file.

4,Neha,27,39000,FINANCE

9,Arpit,41,46000,HR

Advantages

1. Fast Map side Joins – If two tables are bucketed by the same column(s) into same number of buckets and the join is performed on the bucketed column(s), then hive can do efficient map side join by reading the same bucket from both the tables and performing a join, as all the data for similar records will be present in the corresponding bucket from both the tables. If the records are sorted inside each bucket, then hive can join the data using merge, which is a linear time operation. Bucketing will help only when the join key and bucketing key are the same.
2. Efficient Group by – If the group by is performed on the bucketed column(s), then aggregations can be performed in the combiner. This will reduce network traffic by sending less data to reducers.
3. Sampling – Using Bucketing we can run queries on a sample of data from the table. This is beneficial while testing, so that we need not run our queries on whole data.