Skew Join Optimization in Hive

Ambrish Bhargava Follow
Oct 4, 2018 · 8 min read

Skewed Data

Data can be “skewed”, meaning it tends to have a **long tail** on one side or the other.
Example of long tail towards positive side:

Example

Assume, we have the following tables in system:

TABLE 1: Fact

- Contains 100M records.
- Has field CODE_ID as 1 of the field.
- The datatype is INTEGER and has values ranging 1–10K.
- 99M records have value as 250 for CODE_ID field.
- 1M records have rest of the values with equal probability.

TABLE 2: Dimension

- Contains 10k records
- Table big enough so that MAP side join is not possible.
- Has UNIQUE values for field CODE_ID.

If we look closely, we can say that 99% of records in FACT table has only 1 value (CODE_ID = 250). And hence we can say that FACT table has skewed data in CODE_ID field.

Hive Issues With Skewed Data

While executing following query (based on sample table):

```
select
 *
from
 FACT f
 left join DIMENSION d
   on f.CODE_ID = d.CODE_ID
```

We may notice that it progresses to 99% reduce stage quite fast and then gets stuck. Those 99% of reducers are corresponding to the values which have consolidated 1M records in FACT table. Following is the graphical example of the same:

There will be 1 (or few) reducer(s), which will get all the rows corresponding to skewed value (all rows with CODE_ID = 250). This single reducer will be in running state for very long time (say 1–2 hrs) whereas all other will be completed in 1–2 minutes.

Something like this will be the status for long time

```
2017-12-16 21:30:14,527 Map 1: 6283/6283 Map 4: 9/9 Reducer 2:
577(+1)/578
....
```

```
2017-12-16 21:34:47,360 Map 1: 6283/6283 Map 4: 9/9 Reducer 2:
577(+1)/578
2017-12-16 21:34:50,397 Map 1: 6283/6283 Map 4: 9/9 Reducer 2:
577(+1)/578
....
2017-12-16 21:35:51,145 Map 1: 6283/6283 Map 4: 9/9 Reducer 2:
577(+1)/578
2017-12-16 21:35:51,554 Map 1: 6283/6283 Map 4: 9/9 Reducer 2:
578/578
```

This single reducer issue can also be called SKEWED JOIN issue.

Existing Solutions

There are few ways to optimize the Skew join issue in HIVE. Following are some:

Separate Queries

You can split the query into queries and run them separately avoid the skew join.

Example:
Considering our sample tables, we need to write 2 queries to avoid skew join:

Query 1

Execute the query excluding the SKEWED values. Something like:

```
select
  *
from
  FACT f
  left join DIMENSION d
    on f.CODE_ID = d.CODE_ID
where
  f.CODE_ID <> 250
```

Query 2

Execute the query with only SKEWED values. Something like:

```
select
  *
from
  FACT f
```

```
    left join DIMENSION d
      on f.CODE_ID = d.CODE_ID
  where
    f.CODE_ID = 250
    and d.CODE_ID = 250
```

> *Advantages*

- Simple change in the query will avoid the skew join.
- Helpful when query is simple.

> *Disadvantages*

- You need to re-write same query twice.
- It will be harder to write 2 separate queries if original query is complex.
- Whenever you want to modify the query, it needs to be done at 2 separate places. If something is missing then it will be hard to debug.

Using Hive Configuration

You can enable Skew join optimization using hive configuration. Applicable settings are:

```
set hive.optimize.skewjoin=true;
set hive.skewjoin.key=500000;
set hive.skewjoin.mapjoin.map.tasks=10000;
set hive.skewjoin.mapjoin.min.split=33554432;
```

Configuration settings description

hive.optimize.skewjoin

Whether to enable skew join optimization. The algorithm is as follows: At runtime, detect the keys with a large skew. Instead of processing those keys, store them temporarily in an HDFS directory. In a follow-up map-reduce job, process those skewed keys. The same key need not be skewed for all the tables, and so, the follow-up map-reduce job (for the skewed keys) would be much faster, since it would be a map-join.

hive.skewjoin.key

Determine if we get a skew key in join. If we see more than the specified number of rows with the same key in join operator, we think the key as a skew join key.

hive.skewjoin.mapjoin.map.tasks

Determine the number of map task used in the follow up map join job for a skew join. It should be used together with hive.skewjoin.mapjoin.min.split to perform a fine grained control.

hive.skewjoin.mapjoin.min.split

Determine the number of map task at most used in the follow up map join job for a skew join by specifying the minimum split size. It should be used together with hive.skewjoin.mapjoin.map.tasks to perform a fine grained control.

> *Advantages*

- No need to change the query.
- Simple change in the setting will improve the performance.

> *Disadvantages*

- Above solution is not consistent and we do not see improvement in performance every time.

Optimizing Skew Join — Solution

Based on above, we can say that the whole issue is related to amount of data going to the single reducer (See Image 2). If we can distribute the data corresponding to SKEWED values to multiple reducer then we can overcome the problem.

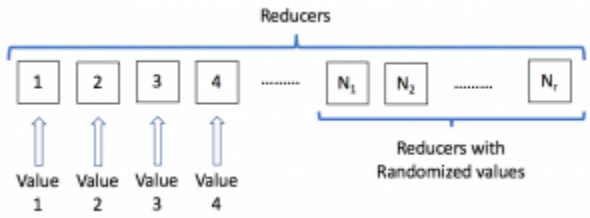
Considering the same tables (FACT & DIMENSION), if we distribute CODE_ID = 250 to (r) separate reducers with equal probability then load to single reducer will also be shared by r different reducer.

Data distribution with equal probability

The question arises: How can we distribute the data with equal probability.

To achieve this, we can follow the steps as explained below:

- Write a function to generate random numbers between 1 to (r) with equal probability.
- Create a new field (say **CODE_ID_RANDOM**) in the **FACT** table. The values can be defined as:
- If value is not skewed then simply copy the value as is.
- If value is identified as skewed (CODE_ID = 250) then **CODE_ID_RANDOM= concat_ws("~~~", cast(CODE_ID as String), random_r())**
- e.g. Field **CODE_ID_RANDOM** will have values from **250~~~1** to **250~~~r** for CODE_ID = 250.
- Total number of rows in the FACT table will remain unchanged but will have 1 extra field.
- In DIMENSION table, create new field as following:
- If value is not skewed then simply copy the value as is.
- If value is identified as skewed (CODE_ID = 250) then create (r) duplicate rows as following



To achieve this, we can use a simple random number generator which will generate 1 to (r) values with equal distribution. We need to follow steps provided in Housekeeping section (see below).

Housekeeping

Pre-steps

Metastore table

First we need to create a metastore where we will keep all the skewed values corresponding to the DIMENSION table. Something like:

```
CREATE TABLE skew_join_metastore (
  table_name string,
  field_name string,
  skewed_values string
)
ROW FORMAT DELIMITED
  FIELDS TERMINATED BY '\t'
STORED AS TEXTFILE
location 'SOME_LOCATION_IN_AWS_OR_HDFS';
```

The table needs to be in TEXT format as we will be reading the content directly.

Skew Value extraction

We need to extract skewed values from the FACT table so that it can be used later to optimize the join. Simple HQL to extract skewed values:

```
INSERT OVERWRITE TABLE skew_join_metastore
select
  "DIMENSION" as table_name, -- Dimension table on which optimization
is been done
  "CODE_ID" as field_name, -- Field for which table is optimized
  concat_ws(',', collect_set(CODE_ID)) as skewed_values
from (
  SELECT
    CODE_ID
  FROM
    FACT
  GROUP BY
    CODE_ID
  HAVING
    count(*) > 1000000 -- Any value above this will be considered as
skewed value
) code_id_skewed
```

UDF to handle Skew values

We need to create a UDF which will consume the SKEWED values (extracted in previous step) and apply them during the JOIN. Signature of this UDF can be:

```
udf_skewjoin_optimizer(
  DIM_TABLE_NAME,
  DIM_FIELD_NAME,
  ACTUAL_FIELD_NAME,
```

```
  IS_FACT_TABLE[optional]
)
```

Parameter description

- DIM_TABLE_NAME: Dimension table on which optimization is been done.
- DIM_FIELD_NAME: Field in the Dimension table on which optimization is been done.
- ACTUAL_FIELD_NAME: Field name in FACT or DIMENSION table to be used during join.
- IS_FACT_TABLE: A boolean to identified if the source table is FACT or DIMENSION. Default is **True**, saying the table is FACT.

Internally, in UDF we have use the random number generator function and will distribute the Skewed values among multiple reducers.

Code snippet of UDF

```
// Reading the skewed values form the metastore location
private Map<Key, Set<String>> initMap() {
  Map<Key, Set<String>> skewValuesMap = Maps.newHashMap();
  try (BufferedReader lineReader = new BufferedReader(getReader())) {
    String line = null;
    while ((line = lineReader.readLine()) != null) {
      if (StringUtils.isBlank(line)) {
        continue;
      }
      String[] pair = split(line, '\t');
      if (pair.length < 3) {
        continue;
      }

      Key key = new Key(pair[0], pair[1]);
      Set<String> values = getSkewedValues(pair[2]);
      skewValuesMap.put(key, values);
    }
  } catch (IOException e) {}
  return skewValuesMap;
}

// Generating random join keys
Key key = new Key(tableName, fieldName);
Set<Text> retVal = Sets.newHashSet();
```

```
Text inputValText = new Text(fieldValue);

// If table is FACT, then simply generate the random keys.
if (randomizeInput) {
  if (retVal.contains(inputValText)) {
    int randomVal = RANDOM_GENERATOR.nextInt(SEED_VALUE);
    inputValText = new Text(inputValText + SEED_DELIMITER +
randomVal);
  }
  return inputValText;
}

List<Text> repInputValues = Lists.newArrayList();

// Generating range from DIMENSION table
if (retVal.contains(inputValText)) {
  for (int num = 0; num < SEED_VALUE; num++) { // SEED_VALUE is the
distribution number
    repInputValues.add(new Text(fieldValue + SEED_DELIMITER + num));
  }
} else {
  repInputValues.add(inputValText);
}

// Returning final list of join keys
return repInputValues;
```

Optimized View

A simple wrapper view to add an extra field to hold optimized join key. This view will hide the complexity involved related to DIMENSION table.

```
drop view if exists DIMENSION_OPTIMIZED;

create view DIMENSION_OPTIMIZED as
select
  `(code_id_arr)?+.+`
from (
  select
    *,
    udf_skewjoin_optimizer('dimension', 'code_id', code_id, false)
code_id_arr
  from
    DIMENSION
) optimized
lateral view explode(code_id_arr) t as code_id_optimized
```

Examples

Without Optimization

Following is the original query which involves SKEW join operation

```
select
  f.CODE_ID,
  count(*) as code_count
from
  FACT f left join
  DIMENSION d
    on f.CODE_ID = d.CODE_ID
group by
  f.CODE_ID
```

With Optimization

Following query is the optimized version of the query which has SKEW join issue (above query)

```
select
  f.CODE_ID,
  count(*) as code_count
from
  FACT f left join
  DIMENSION_OPTIMIZED d
    on udf_skewjoin_optimizer('dimension', 'code_id', f.CODE_ID) =
d.CODE_ID_OPTIMIZED
group by
  f.CODE_ID
```

Advantages

- No need to worry about stuck queries because of Skew joins.
- With minimal changes in the query, we can get atleast 50% performance improvement.

Disadvantages

- Need to do housekeeping to make this working.
- Initial setup is required to identify SKEW values.

- Whenever there is requirement for new DIMENSION table, we need to make changes in the code. But it is one time thing.

Future Enhancements

We can add a dimension table and its column to the skew join optimizer if it satisfies the below conditions:

- If all the values of the column that needed to be added to UDF in the dimension table are unique.
- If the data in the FACT table on which the join has to happen is skewed.

Reference

- https://en.wikipedia.org/wiki/Skewness
- https://cwiki.apache.org/confluence/display/Hive/Skewed+Join+Optimization
- https://cwiki.apache.org/confluence/display/Hive/Configuration+Properties
- https://weidongzhou.wordpress.com/2017/06/08/join-type-in-hive-skewed-join/

Programming Skewed Data Hive Software Engineering Big Data