Skewed Join Optimization

Optimizing Skewed Joins

The Problem

A join of 2 large data tables is done by a set of MapReduce jobs which first sorts the tables based on the join key and then joins them. The Mapper gives all rows with a particular key to the same Reducer.

e.g., Suppose we have table A with a key column, "id" which has values 1, 2, 3 and 4, and table B with a similar column, which has values 1, 2 and 3. We want to do a join corresponding to the following query

- select A.id from A join B on A.id = B.id

A set of Mappers read the tables and gives them to Reducers based on the keys. e.g., rows with key 1 go to Reducer R1, rows with key 2 go to Reducer R2 and so on. These Reducers do a cross product of the values from A and B, and write the output. The Reducer R4 gets rows from A, but will not produce any results.

Now let's assume that A was highly skewed in favor of id = 1. Reducers R2 and R3 will complete quickly but R1 will continue for a long time, thus becoming the bottleneck. If the user has information about the skew, the bottleneck can be avoided manually as follows:

Do two separate queries

- select A.id from A join B on A.id = B.id where A.id <> 1;
- select A.id from A join B on A.id = B.id where A.id = 1 and B.id = 1;

The first query will not have any skew, so all the Reducers will finish at roughly the same time. If we assume that B has only few rows with B.id = 1, then it will fit into memory. So the join can be done efficiently by storing the B values in an in-memory hash table. This way, the join can be done by the Mapper itself and the data do not have to go to a Reducer. The partial results of the two queries can then be merged to get the final results.

- Advantages
  - If a small number of skewed keys make up for a significant percentage of the data, they will not become bottlenecks.
- Disadvantages
  - The tables A and B have to be read and processed twice.
  - Because of the partial results, the results also have to be read and written twice.
  - The user needs to be aware of the skew in the data and manually do the above process.

We can improve this further by trying to reduce the processing of skewed keys. First read B and store the rows with key 1 in an in-memory hash table. Now run a set of mappers to read A and do the following:

- If it has key 1, then use the hashed version of B to compute the result.
- For all other keys, send it to a reducer which does the join. This reducer will get rows of B also from a mapper.

This way, we end up reading only B twice. The skewed keys in A are only read and processed by the Mapper, and not sent to the reducer. The rest of the keys in A go through only a single Map/Reduce.

The assumption is that B has few rows with keys which are skewed in A. So these rows can be loaded into the memory.

Hive Enhancements

Original plan: ~~The skew data will be obtained from list bucketing (see the List Bucketing design document). There will be no additions to the Hive grammar.~~

Implementation: Starting in Hive 0.10.0, tables can be created as skewed or altered to be skewed (in which case partitions created after the ALTER statement will be skewed). In addition, skewed tables can use the list bucketing feature by specifying the STORED AS DIRECTORIES option. See the DDL documentation for details: Create Table, Skewed Tables, and Alter Table Skewed or Stored as Directories.