Hive Join Strategies

hkropp General, Hive October 9, 2016 4 Minutes

Hive (http://hive.apache.org/) joins (https://en.wikipedia.org/wiki/Join_(SQL)) are executed by MapReduce jobs through different execution engines like for example Tez (http://tez.apache.org/), Spark (http://spark.apache.org/) or MapReduce (http://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html). Joins even of multiple tables can be achieved by one job only. Since it's first release many optimizations have been added to Hive giving users various options for query improvements of joins.

Understanding how joins are implemented with MapReduce helps to recognize the different optimization techniques in Hive (http://hive.apache.org/) today.

MapReduce Joins

Joins with MapReduce can be achieved in two ways, either during the map phase (map-side) or during the reduce phase (reduce-side). Assuming the following sample data both approaches are explained below.

join-sample-data

Map-Side Joins

The map-side join can only be achieved if it is possible to join the records by key during read of the input files, so before the map phase. Additionally for this to work the input files need to be sorted by the same join key. Further more both inputs need to have the same number of partitions.

Reaching these strict constraints is commonly hard to achieve. The most likely scenario for a map-side join is when both input tables were created by (different) MapReduce jobs having the same amount of reducers using the same (join) key.

The CompositeInputFormat (http://hadoop.apache.org/docs/current/api/org/apache/hadoop/mapred/join/CompositeInputFormat.html) is capable of performing joins over a set of data sources sorted and partitioned the same way. They are fairly efficient as a subsequent reduce phase with a expensive shuffle phase can be omitted.

Note: Hive map-side joins work differently.

Reduce-Side Joins

Reduce-side joins do not impose any such constraints of map-side joins on the format of the input sources. Because reading from multiple different sources can be achieved by using a MultiInputs (https://hadoop.apache.org/docs/current/api/org/apache/hadoop/mapreduce/lib/input/MultipleInputs.html) file format, one MapRedcue job is enough to join both input tables. In the map phase values are emitted based on the join key ensuring that each reducer gets the same – based on the join key – entries from both inputs.

While during the Sort-and-Shuffle phase (Hive Shuffle-Join) entries will be sorted by the join key at each reducer, one would still need to ensure that the entry of one side comes before the entries of the other side. This is achieved through a technique often referred to as *Secondary Indexes*(* (https://vangjee.wordpress.com/2012/03/20/secondary-sorting-aka-sorting-values-in-hadoops-mapreduce-programming-paradigm/)). In this method a composed key is being created where each entry is first sorted by the first and then by the second component of the key.

Hive Shuffle-Join

The Shuffle-Join is the default and was for long the only join implementation in Hive. It is completely based on the reduce-side join of MapReduce where during the reduce phase entries are joined during the shuffle phase, hence the name of the join strategy. Please refer to the detailed explanation above for further insides.

Looking at the way the reduce-side join works, it becomes clear why Hive only supports *equi-joins*, meaning that only the equal conditions (=) are allowed for the ON expression. Also conditions can only be combined with an AND operator.

Partitioning entries in the shuffle phase based on other operations then the equal operator would be fairly difficult to achieve and possibly also limiting the amount of reducers, hence sacrificing scalability and performance.

Hive Map-Side Join (Broadcast Join)

The shuffle phase is a fairly expensive operation in all available execution engines. Avoiding the shuffle phase would result in great performance advantages. Especially for some joins where one of the tables is reasonable small the burden of a shuffle phase becomes unacceptable.

Often joins are executed on a so called lookup table with limited entries. These fact tables are often being used for feature enrichment during the processing. Imagine for example a table of cities and a list of POIs (point of interests) in those cities only referenced by ID. Enriching the POI entries with the information of the cities could be achieved by joining the two tables. To optimize for such scenarios where on of the tables is fairly small (for example a lookup or fact table) Hive can used map-side joins, which work differently from plain MapReduce map-side joins.

In Hive the small table is distributed and send to all the nodes processing the join making it available during the map phase. Originally each Mapper would read the complete table before the execution of the map function from HDFS. This was changed when noticing that large amounts of mappers reading the same file could become a bottleneck.

Overcoming the initial drawback of the map-side join implementation the distributed cache (http://hadoop.apache.org/docs/current/api/org/apache/hadoop/filecache/DistributedCache.html) is being used today. The smaller table is turned into a hash table. This hash table is being serialized and shipped as distributed cache prior to the job execution. Because of this this the join strategy is also often referred to as *Broadcast Join*.

Note: Good when one of the tables is small enough to fit in RAM.

Hive Settings

To activate broadcast joins in Hive the configuration hive.auto.convert.join needs to be activated. For long users needed to specify in the SQL query which table was small enough to be used for broadcasting. This can now be set with the hive.auto.convert.join.nonconditionaltask.size value allowing Hive to determine which tables it can use for broadcasting.

```
hive.auto.convert.join
```

De-/Activates broadcast (map-side) joins in Hive.

```
hive.auto.convert.join.nonconditionaltask.size
```

Determines the size of a table should have to be converted to broadcast join. The default is 10MB which is fairly small and values from 256MB up to 512MB are acceptable values depending on your container size.

Hive Bucket Join

Buckets add an additional structure to the way the data is organized across *HDFS*. Entries with identical values in the columns used bucketing are stored in the same file partitions. Tables bucketed based on columns that include the join key can make advantage of Hive Bucket Joins.

The trick of Bucket Join in Hive is that the join of bucketed files having the same join key can efficiently be implemented as map-side joins. As previously explained do map-side joins impose strict constrains on the way the data needs to be organized. It needs to be sorted and partitioned identically. This is what can be achieved with bucketed Hive tables.

bucket-join-hive

Further Readings

- Hadoop: The Definitive Guide (https://www.amazon.de/gp/product/1491901632/ref=as_li_qf_sp_asin_il_tl?ie=UTF8&camp=1638&creative=6742&creativeASIN=1491901632&linkCode=as2&tag=henningkroppo-21) (Amazon)
- Programming Hive (https://www.amazon.de/gp/product/149193445X/ref=as_li_qf_sp_asin_il_tl?ie=UTF8&camp=1638&creative=6742&creativeASIN=149193445X&linkCode=as2&tag=henningkroppo-21) (Amazon)
- SQL Joins (https://en.wikipedia.org/wiki/Join_(SQL))
- Hive Joins (https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Joins)
- Join Optimization in Apache Hive (https://www.facebook.com/notes/facebook-engineering/join-optimization-in-apache-hive/470667928919/)
- Hive Join Optimizations (https://cwiki.apache.org/confluence/display/Hive/LanguageManual+JoinOptimization)

- Hive Outer Join Behavior (https://cwiki.apache.org/confluence/display/Hive/OuterJoinBehavior)

Tagged:
hive,
join,
sql-join

Published by hkropp

View all posts by hkropp

4 thoughts on “Hive Join Strategies”

Pingback: Broadcast Join with Spark – Hadoop by Passion, Big Data for Love!
Ram says:
April 27, 2017 at 7:05 pm
Good information on joins, If u have 1 million records in one table which i want to do join operation wit table having 50 million reocrds, which approach is better as part of optimization.

↪ Reply
Pingback: Hive – Big Data
Pingback: Material de estudo para o exame 70-775 Perform Data Engineering on Microsoft Azure HDInsight – Thomaz