facebook
Join Strategies in Hive

Liyin Tang, Namit Jain
Software Engineer
Facebook
<table>
<thead>
<tr>
<th></th>
<th>Agenda</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Common Join</td>
</tr>
<tr>
<td>2</td>
<td>Map Join</td>
</tr>
<tr>
<td>3</td>
<td>Auto MapJoin</td>
</tr>
<tr>
<td>4</td>
<td>Bucket Map Join</td>
</tr>
<tr>
<td>5</td>
<td>Bucket Sort Merge Map Join</td>
</tr>
<tr>
<td>6</td>
<td>Skew Join</td>
</tr>
</tbody>
</table>
Common Join

Task A

Table X

Mapper
Mapper
Mapper

Common Join Task

Mapper
Mapper
Mapper

Shuffle

Reducer

Table Y
MapJoin

1. Spawn mapper based on the big table
2. All files of all small tables are replicated onto each mapper
Optimized Map Join

Task A

MapReduce Local Task

MapJoin Task

Mapper

Mapper

Mapper

Task C

Small Table Data

HashTable Files

Distributed Cache

Compressed & Archived

Big Table Data
Converting Common Join into Map Join

Previous Execution Flow

Optimized Execution Flow
Execution Time

SELECT * FROM SRC1 x JOIN SRC2 y ON x.key = y.key,

- **Task A**
  - **Conditional Task**
    - **MapJoinLocalTask**
      - **MapJoinTask**
      - **Task C**
    - **CommonJoinTask**
      - **Both tables are too big for map join**
Backup Task

- Task A
  - Conditional Task
    - Memory Bound
      - MapJoin LocalTask
        - MapJoinTask
      - Run as a Backup Task
        - CommonJoinTask
    - Task C
Performance Bottleneck

Distributed Cache is the potential performance bottleneck

- Large hashtable file will slow down the propagation of Distributed Cache
- Mappers are waiting for the hashtables file from Distributed Cache

Compress and archive all the hashtable file into a tar file.
Bucket Map Join

Why:

Total table/partition size is big, not good for mapjoin.

How:

set hive.optimize.bucketmapjoin = true;

1. Work together with map join
2. All join tables are bucketized, and each small table’s bucket number can be divided by big table’s bucket number.
3. Bucket columns == Join columns
Bucket Map Join

```sql
SELECT /*+ MAPJOIN(a,c)*/ a.*, b.*, c.*
a join b on a.key = b.key
join c on a.key = c.key;
```

Table a, b, c all bucketized by ‘key’
- a has 2 buckets, b has 2, and c has 1

1. Spawn mapper based on the big table
2. Only matching buckets of all small tables are replicated onto each mapper

Normally in production, there will be thousands of buckets!
Sort Merge Bucket Map Join

Why:

No limit on file/partition/table size.

How:

set hive.optimize.bucketmapjoin = true;
set hive.optimize.bucketmapjoin.sortedmerge = true;
set hive.input.format=org.apache.hadoop.hive.ql.io.BucketizedHiveInputFormat;

1. Work together with bucket map join

2. Bucket columns == Join columns == sort columns
Sort Merge Bucket Map Join

Table A

1, val_1
3, val_3
4, val_4
5, val_5

Table B

4, val_4
20, val_20
23, val_23

Table C

20, val_20
25, val_25

Small tables are read on demand
NOT hold entire small tables in memory
Can perform outer join

Facebook
Skew Join

Join bottlenecked on the reducer who gets the skewed key

set hive.optimize.skewjoin = true;
set hive.skewjoin.key = skew_key_threshold
**Skew Join**

set hive.optimize.skewjoin = true;
set hive.skewjoin.key = \texttt{Skew}\_\texttt{Key}\_\texttt{Threshold};

![Diagram of Skew Join process](image)

- **Job 1**
  - **Reducer 1**
    - 
    - **a-K 1**
    - **b-K 1**
    - **Write to HDFS**
  - **HDFS File**
    - **a-K1**
    - **b-K1**

- **Job 2**
  - **Map join**
    - **a-k1**
    - **Map join**
    - **b-k1**

**Final results**

---

**Diagram Explanation**

1. **Table A** and **Table B** are joined to form **A join B**.
2. The join results are processed by **Reducer 1** and **Reducer 2**.
3. **Reducer 1** processes **a-K 1** and **b-K 1** and writes the results to HDFS.
4. **Reducer 2** processes **a-K 2** and **b-K 2**.
5. **Join** results are mapped and merged by **Job 2**.
6. The final results are the **a-K 3** and **b-K 3**.

---

**Code Example**

```java
// Example code for Skew Join optimization

// Set optimization configuration
hive.optimize.skewjoin = true;
hive.skewjoin.key = "Skew\_Key\_Threshold";

// Skew Join operation
A join B
```
Clusters

2000 live nodes cluster

**Commodity machines**

- CPU: 2 Intel@Xeon X5650
- Memory: 48G
- Disk: 2 TP*12 Disks
- CentOS
## Performance Evaluation I

<table>
<thead>
<tr>
<th>Small Table</th>
<th>Big Table</th>
<th>Join Condition</th>
<th>Average Map Join Execution time</th>
<th>Average New Optimized Map Join Execution time</th>
<th>Performance Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>75 K rows; 383K file size</td>
<td>130 M rows; 3.5G file size;</td>
<td>1 join key, 2 join value</td>
<td>1032 sec</td>
<td>79 sec</td>
<td>+ 1206%</td>
</tr>
<tr>
<td>500 K rows; 2.6M file size</td>
<td>130 M rows; 3.5G file size</td>
<td>1 join key, 2 join value</td>
<td>3991 sec</td>
<td>144 sec</td>
<td>+2671 %</td>
</tr>
<tr>
<td>75 K rows; 383K file size</td>
<td>16.7 B rows; 459 G file size</td>
<td>1 join key, 2 join value</td>
<td>4801 sec</td>
<td>325 sec</td>
<td>+1377 %</td>
</tr>
</tbody>
</table>
# Performance Evaluation II

<table>
<thead>
<tr>
<th>Small Table</th>
<th>Big Table</th>
<th>Join Condition</th>
<th>Average Join Execution Time Without Compression</th>
<th>Average Join Execution Time With Compression</th>
<th>Performance Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>75 K rows; 383K file size</td>
<td>130 M rows; 3.5G file size</td>
<td>1 join key, 2 join value</td>
<td>106 sec</td>
<td>73 sec</td>
<td>+ 45%</td>
</tr>
<tr>
<td>500 K rows; 2.6M file size</td>
<td>130 M rows; 3.5G file size</td>
<td>1 join key, 2 join value</td>
<td>129 sec</td>
<td>106 sec</td>
<td>+21 %</td>
</tr>
<tr>
<td>75 K rows; 383K file size</td>
<td>16.7 B rows; 459 G file size</td>
<td>1 join key, 2 join value</td>
<td>441 sec</td>
<td>326 sec</td>
<td>+ 35 %</td>
</tr>
<tr>
<td>500 K rows; 2.6M file size</td>
<td>16.7 B rows; 459 G file size</td>
<td>1 join key, 2 join value</td>
<td>326 sec</td>
<td>251 sec</td>
<td>+30 %</td>
</tr>
<tr>
<td>1M rows; 10M file size</td>
<td>16.7 B rows; 459 G file size</td>
<td>1 join key, 3 join value</td>
<td>495 sec</td>
<td>266 sec</td>
<td>+86 %</td>
</tr>
<tr>
<td>1M rows; 10M file size</td>
<td>16.7 B rows; 459 G file size</td>
<td>2 join key, 2 join value</td>
<td>425 sec</td>
<td>255 sec</td>
<td>+67 %</td>
</tr>
</tbody>
</table>
## Performance Evaluation III

<table>
<thead>
<tr>
<th>Small Table</th>
<th>Big Table</th>
<th>Join Condition</th>
<th>Previous Common Join</th>
<th>Optimized Common Join</th>
<th>Performance Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>75 K rows; 383K file size</td>
<td>130 M rows; 3.5G file size</td>
<td>1 join key, 2 join value</td>
<td>169 sec</td>
<td>79 sec</td>
<td>+ 114%</td>
</tr>
<tr>
<td>500 K rows; 2.6M file size</td>
<td>130 M rows; 3.5G file size</td>
<td>1 join key, 2 join value</td>
<td>246 sec</td>
<td>144 sec</td>
<td>+71%</td>
</tr>
<tr>
<td>75 K rows; 383K file size</td>
<td>16.7 B rows; 459 G file size</td>
<td>1 join key, 2 join value</td>
<td>511 sec</td>
<td>325 sec</td>
<td>+57%</td>
</tr>
<tr>
<td>500 K rows; 2.6M file size</td>
<td>16.7 B rows; 459 G file size</td>
<td>1 join key, 2 join value</td>
<td>502 sec</td>
<td>305 sec</td>
<td>+64%</td>
</tr>
<tr>
<td>1M rows; 10M file size</td>
<td>16.7 B rows; 459 G file size</td>
<td>1 join key, 3 join value</td>
<td>653 sec</td>
<td>248 sec</td>
<td>+163%</td>
</tr>
<tr>
<td>1M rows; 10M file size</td>
<td>16.7 B rows; 459 G file size</td>
<td>2 join key, 2 join value</td>
<td>1117 sec</td>
<td>536 sec</td>
<td>+108%</td>
</tr>
</tbody>
</table>
Summary & Future Work

Mapjoin supported since Hive 0.5

New map join launched @Facebook since Jan, 2011

Set hashtable file replica number based on the number of Mappers

Tune the limit of small table data size by sampling

Memory efficient hashtable