Map-Reduce-Merge: Simplified Relational Data Processing on Large Clusters

A.Dasdan et al.

Presented by: Nedyalko Borisov
(slides adopted from Nate Roberts)

Instructor:
Jun Yang

Duke University
Computer Science
April 15, 2008
Outline

- Map-Reduce
- Features of Map-Reduce
- Map-Reduce-Merge
- Relational Algebra Operators using Map-Reduce-Merge
- Discussion
Map-Reduce Features

- Low cost unreliable commodity hardware
- Scalable RAIN Cluster
- Fault tolerant and easy to administer
- Simplified, restricted and powerful model
- Highly parallel and abstracted
- Shared disk storage and not-shared computing
Map-Reduce Features (cont.)

- High throughput
- High performance for large number of nodes
- Set-oriented keys and values
- Functional programming primitives
- Distributed partitioning/sorting framework
- Design for Search Engines, Generic data processing tasks
Map-Reduce-Merge
Map-Reduce-Merge

- Modifications to Map-Reduce
  - Reduce phase
  - Merge phase
  - Additional user-defined operations
    - Partition selector
    - Processor
    - Merger
    - Configurable iterators
# Map-Reduce-Merge Example

## LHS mapper computes emp bonuses

<table>
<thead>
<tr>
<th>emp-id</th>
<th>dept-id</th>
<th>bonus</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>B</td>
<td>$100</td>
</tr>
<tr>
<td>1</td>
<td>B</td>
<td>$50</td>
</tr>
<tr>
<td>2</td>
<td>A</td>
<td>$0</td>
</tr>
<tr>
<td>3</td>
<td>A</td>
<td>$150</td>
</tr>
<tr>
<td>3</td>
<td>A</td>
<td>$100</td>
</tr>
</tbody>
</table>

## RHS mapper retrieves bonus adjustment

<table>
<thead>
<tr>
<th>dept-id</th>
<th>bonus adjustment</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>1.1</td>
</tr>
<tr>
<td>A</td>
<td>0.9</td>
</tr>
</tbody>
</table>

## RHS reducer modified bonus adjustment and sorts on dept-id

<table>
<thead>
<tr>
<th>dept-id</th>
<th>bonus adjustment</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>1.15</td>
</tr>
<tr>
<td>A</td>
<td>0.95</td>
</tr>
</tbody>
</table>

## LHS reducer sorts on (dept-id, emp-id) pair and sums up emp bonuses

<table>
<thead>
<tr>
<th>emp-id</th>
<th>dept-id</th>
<th>bonus-sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>A</td>
<td>$0</td>
</tr>
<tr>
<td>3</td>
<td>A</td>
<td>$250</td>
</tr>
<tr>
<td>1</td>
<td>B</td>
<td>$150</td>
</tr>
</tbody>
</table>

## match keys on dept-id

<table>
<thead>
<tr>
<th>dept-id</th>
<th>bonus adjustment</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.95</td>
</tr>
<tr>
<td>B</td>
<td>1.15</td>
</tr>
</tbody>
</table>

## A sort-merge merger joins LHS and RHS reduced outputs, then computes final emp bonuses.

<table>
<thead>
<tr>
<th>emp-id</th>
<th>bonus</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>$0</td>
</tr>
<tr>
<td>3</td>
<td>$237.5</td>
</tr>
<tr>
<td>1</td>
<td>$172.5</td>
</tr>
</tbody>
</table>
Relational Algebra Operators

• Projection
  - emit subset of the data passed
  - mapper can do the work

• Aggregation
  - “group by” and “aggregate” operators
  - reducers can do the work
Relational Algebra Operators (cont.)

• Selection
  - only on one table attribute – mapper
  - on aggregate or group by attribute – reducer
  - attributes from both tables – mapper

• Union
  - reducers produce sorted output
  - merger just do the work
Relational Algebra Operators (cont.)

- Intersection
  - the two reducers produce sorted lists
  - the merger iterates on each input simultaneously and outputs the common tuples

- Difference
  - the two reducers produce sorted lists
  - the merger iterates on each input simultaneously and outputs the difference
Relational Algebra Operators (cont.)

• Cartesian product \((F \times S)\)
  - reducers should output the two set for the \(F\) and \(S\)
  - mergers get one part of \(F\) and the complete set from \(S\)
  - each merger produce \(F_p \times S\)

• Rename
  - trivial
Relational Algebra Operators (cont.)

• Sort-Merge Join
  - Map: partition records into key ranges according to the sorted attributes values, trying to achieve even distribution
  - Reduce: sort the data
  - Merge: join the sorted data for each key range
Relational Algebra Operators (cont.)

• Hash Join
  - Map: use one and the same hash function for both tables
  - Reduce: produce a hash table from the values mapped
  - Merge: use one bucket as a build set and the other as a probe
Relational Algebra Operators (cont.)

- Block Nested Loop Join
  - Map & Reduce the same as in the Hash Join
  - Merge: do the nested loop join for each of the buckets
Optimizations

- Optimal reduce-merge connections
  - complicated logic
- Combining phases
  - ReduceMap, MergeMap, ReduceMerge, ReduceMergeMap
- Library
- Changes of the workflow - future work
Discussion

• Is it worth the effort?
  – 3 table joins
  – what is saving us
• Union, intersection, difference
  – Map-Reduce can do them
• Optimized for specific case
• No real experiments