Introduction to Parallel Execution

Query or Spark Program Q

Answers

Data Partition

Data resides on one or more machines

Spark Driver

Translates Q into an execution plan and runs the plan
Example: At a Company

Query 1: Is there an employee named “Nemo”?  
Query 2: What is “Nemo’s” salary?  
Query 3: How many departments are there in the company?  
Query 4: What is the name of “Nemo’s” department?  
Query 5: How many employees are there in the “Accounts” department?

<table>
<thead>
<tr>
<th>Employee</th>
<th></th>
<th>Department</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Name</td>
<td>DeptID</td>
</tr>
<tr>
<td>10</td>
<td>Nemo</td>
<td>12</td>
</tr>
<tr>
<td>20</td>
<td>Dory</td>
<td>156</td>
</tr>
<tr>
<td>40</td>
<td>Gill</td>
<td>89</td>
</tr>
<tr>
<td>52</td>
<td>Ray</td>
<td>34</td>
</tr>
</tbody>
</table>

...
Counting the number of records that will be read or transferred over the network in a parallel execution

- We have a dataset R with two attributes A and B
- There are 10000 records in R, with 2500 unique values of A and 5000 unique values of B
- See Figure 1 on next slide
### Records in R

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>

R has a total of \( T(R) = 10000 \) records.

### Algebraic representation of records in R

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x )</td>
<td>( 2x - 1 )</td>
</tr>
<tr>
<td>( x )</td>
<td>( 2x - 1 )</td>
</tr>
<tr>
<td>( x )</td>
<td>( 2x )</td>
</tr>
<tr>
<td>( x )</td>
<td>( 2x )</td>
</tr>
</tbody>
</table>

For \( x \) in 1, 2, 3, ..., 2499, 2500

---

**Figure 1:** Figure showing the contents of records in R.
Counting exercise (contd.)

- Dataset R has 10 partitions
- R is stored on 10 machines, with one partition per machine
- We want to find the result of:

```
Select A, MAX(B)
From R
Where B >= 1000 and B < 2000
Group By A
```

- See Figure on next slide
M1:
1 \leq A \leq 250

M2:
251 \leq A \leq 500

M5:
1001 \leq A \leq 1250

M6:
1251 \leq A \leq 1500

M7:
1501 \leq A \leq 1750

M10:
2251 \leq A \leq 2500

Send records with
1 \leq A \leq 1250

Send records with
1251 \leq A \leq 2500
Partitioning and Load Balancing

• Looking at data as Key-Value pairs
• The Map Vs. Reduce model of parallel execution
• Partitioning:
  – Range Partitioning
  – Hash partitioning
  – List partitioning
• The Shuffle step in parallel execution
• Load balancing and skew problems
Same example (contd.)

• Suppose records in R are partitioned randomly across M1-M10. What, if anything, changes?
Other Common Tasks that Need Parallel Execution

- Word counting
- Inverted indexes
Other Common Tasks that Need Parallel Execution

• TF-IDF
Other Common Tasks that Need Parallel Execution

• Sorting
• PageRank (will consider this when we study Graph Processing)
• …
Join Example: Store that Sells Cars

Owners of Honda Accords who are <= 23 years old

Join (Cars.OwnerID = Owners.ID)

Filter (Make = Honda and Model = Accord)

Filter (Age <= 23)

<table>
<thead>
<tr>
<th>Make</th>
<th>Model</th>
<th>OwnerID</th>
<th>ID</th>
<th>Name</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>Honda</td>
<td>Accord</td>
<td>12</td>
<td>12</td>
<td>Nemo</td>
<td>22</td>
</tr>
<tr>
<td>Honda</td>
<td>Accord</td>
<td>156</td>
<td>156</td>
<td>Dory</td>
<td>21</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>Nemo</td>
<td>22</td>
</tr>
<tr>
<td>34</td>
<td>Ray</td>
<td>42</td>
</tr>
<tr>
<td>89</td>
<td>Gill</td>
<td>36</td>
</tr>
<tr>
<td>156</td>
<td>Dory</td>
<td>21</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Make</th>
<th>Model</th>
<th>OwnerID</th>
<th>ID</th>
<th>Name</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>Honda</td>
<td>Accord</td>
<td>12</td>
<td>12</td>
<td>Nemo</td>
<td>22</td>
</tr>
<tr>
<td>Honda</td>
<td>Accord</td>
<td>156</td>
<td>156</td>
<td>Dory</td>
<td>21</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>Nemo</td>
<td>22</td>
</tr>
<tr>
<td>34</td>
<td>Ray</td>
<td>42</td>
</tr>
<tr>
<td>89</td>
<td>Gill</td>
<td>36</td>
</tr>
<tr>
<td>156</td>
<td>Dory</td>
<td>21</td>
</tr>
</tbody>
</table>
Parallel Execution of Joins in MapReduce Style

- Two typical types of join
  - Map-side join
  - Reduce-side join
Map-side Join

Map tasks:

Table L

Table R

Duplicate

Split 1

Split 2

Split 3

Split 4
**REDUCE-SIDE JOIN**

**Input**
- Pairs: (key, targeted record)
- L: ratings.dat
  - 1193::5::978300760
  - 661::3::978302109
  - 661::3::978301968
  - 661::4::978300275
  - 1193::5::97882429
- R: movies.dat
  - 661::James and the Glant...
  - 914::My Fair Lady..
  - 1193::One Flew Over the...
  - 2355::Bug's Life, A...
  - 3408::Erin Brockovich...

**Map**
- Output key is changed to a composite of the join key and the table tag.

**Shuffle**
- Group by join key
  - {661::James...} X
    - (1::661::3::978300760), (661::3::978302109), (661::3::978301968), (661::4::978300275), (1193::5::97882429)

**Reduce**
- Buffers records into two sets according to the table tag + Cross-product
  - Drawback: all records may have to be buffered
    - Out of memory
    - The key cardinality is small
    - The data is highly skewed

**Output**

<table>
<thead>
<tr>
<th>Phase /Function</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map Function</td>
<td>Output key is changed to a composite of the join key and the table tag.</td>
</tr>
<tr>
<td>Partitioning function</td>
<td>Hashcode is computed from just the join key part of the composite key</td>
</tr>
<tr>
<td>Grouping function</td>
<td>Records are grouped on just the join key</td>
</tr>
</tbody>
</table>