MapReduce:
Algorithm Design for
Relational Operations

Juliana Freire & Cláudio Silva

Some slides borrowed from Jimmy Lin, Jeff Ullman, Jerome Simeon, and Jure Leskovec
Projection

\[ \pi_{\square \bigcirc} \]
Projection in MapReduce

- Easy
  - Map over tuples, emit new tuples with appropriate attributes
  - No reducers, unless for regrouping or re-sorting tuples
  - Alternative: do projection in reducer, after some other processing

- Limited by HDFS streaming speeds
  - Speed of encoding/decoding tuples becomes important
  - Semi-structured (XML) data? No problem!
Selection
Selection in MapReduce

• Easy
  • Map over tuples, emit new tuples with appropriate attributes
  • No reducers, unless for regrouping or re-sorting tuples
  • Alternative: do projection in reducer, after some other processing

• Limited by HDFS streaming speeds
  • Speed of encoding/decoding tuples becomes important
  • Semi-structured (XML) data? No problem!
Aggregation in MapReduce

- Example – Log analysis: What is the average time spent per URL?
- In SQL:
  - SELECT url, AVG(time) FROM visits GROUP BY url
- In MapReduce:
  - Map over tuples, emit time, keyed by url
  - Framework automatically groups values by keys
  - Compute average in reducer
  - How can you make this more efficient?
Relational Joins

$R_1$  $S_1$
$R_2$  $S_2$
$R_3$  $S_3$
$R_4$  $S_4$
Running Example

• Two tables:
  – S: User demographics (user_id, gender, age, income, etc.)
  – R: User page visits (user_id, URL, time spent, etc.)

• Analyses we might want to perform:
  – Statistics on demographic characteristics
  – Statistics on page visits
  – Statistics on page visits by URL
  – Statistics on page visits by demographic characteristic

• Need to join S and R!
Join Algorithms in MapReduce

- Reduce-side join
- Map-side join
- In-memory join
Reduce-Side Join

• Basic idea: *group by join key*
  – Map over both sets of tuples
  – Emit tuple as value with *join key as the intermediate key*
  – Execution framework brings together tuples sharing the same key
  – Perform actual join in reducer
  – Similar to a “sort-merge join” in database terminology

• Two variants
  – 1-to-1 joins
  – 1-to-many and many-to-many joins
Reduce-Side Join: 1-to-1

Map

```
R1  
R4  
S2  
S3  
```

Reduce

```
keys                       values
R1                         S1
S3                         R4
```

Note: no guarantee if R is going to come first or S
Reduce-Side Join: 1-to-1

• At most one tuple from R and one tuple from S share the same join key

• Reducer will receive keys in the following format
  
  \[
  \begin{align*}
  k_{23} & \rightarrow [(r_{64}, R_{64}), (s_{84}, S_{84})] \\
  k_{37} & \rightarrow [(r_{68}, R_{68})] \\
  k_{59} & \rightarrow [(s_{97}, S_{97}), (r_{81}, R_{81})] \\
  k_{61} & \rightarrow [(s_{99}, S_{99})]
  \end{align*}
  \]

• If there are two values associated with a key, one must be from R and the other from S

• What should the reduced do for \( k_{37} \)?
Reduce-Side Join: 1-to-many

Map

R1  S2  S3  S9

keys  values

Reduce

keys  values

R1  S2  S3  ...

R is the one side, S is the many
Reduce-Side Join: 1-to-many

- Reducer will receive keys in the following format:
  - $k_{23} \rightarrow [(r_{64}, R_{64}), (s_{84}, S_{84}), (r_{65}, R_{65})]$  
  - $k_{37} \rightarrow [(r_{68}, R_{68})]$  
  - $k_{59} \rightarrow [(s_{97}, S_{97}), (s_{98}, S_{98}), (s_{99}, S_{99}), (s_{100}, S_{100}), (r_{81}, R_{81})]$  
  - $k_{61} \rightarrow [(s_{99}, S_{99})]$  

- There can be many values from $S$ for a given key, we do not know when the value for $R$ will be encountered.
Reduce-Side Join: 1-to-many

Map

R

\[ R_1 \]

S

\[ S_2 \]

\[ S_3 \]

\[ S_9 \]

Buffer all values in memory, pick out the tuple from R, and then cross it with every tuple from S to perform the join

Reduce

R

\[ R_1 \]

S

\[ S_2 \]

\[ S_3 \]

\[ \ldots \]

R is the one side, S is the many
Reduce-Side Join: 1-to-many

Map

Buffer all values in memory, pick out the tuple from R, and then cross it with every tuple from S to perform the join

Reduce

What’s the problem?

R is the one side, S is the many
Sorting: Use Values

– “Value-to-key conversion” design pattern: form composite intermediate key, \((k, v_1)\)

– Let execution framework do the sorting: Sort by join attribute, then sort all tuple ids from \(R\) before \(S\)

Before \(k \rightarrow (s97, S_{97}), (s98, S_{98}), (s99, S_{99}), (s100, S_{100}), (r81, R_{81})\)… Values arrive in arbitrary order…

After \((k, r81) \rightarrow (r81, R_{81})\) \((k, s97) \rightarrow (s97, S_{97})\) Values arrive in sorted order of tuple id \((k, s98) \rightarrow (s98, S_{98})\) Process by preserving state across \((k, s99) \rightarrow (s99, S_{99})\) multiple keys … Remember to partition correctly!
Reduce-Side Join: Value-to-Key Conversion

In reducer...

- New key encountered: hold in memory
- Cross with records from other set

https://www.inkling.com/read/hadoop-definitive-guide-tom-white-3rd/chapter-8/example-8-9
Reduce-Side Join Many-to-Many

In reducer...

<table>
<thead>
<tr>
<th>keys</th>
<th>values</th>
</tr>
</thead>
<tbody>
<tr>
<td>R₁</td>
<td></td>
</tr>
<tr>
<td>R₅</td>
<td></td>
</tr>
<tr>
<td>R₈</td>
<td></td>
</tr>
<tr>
<td>S₂</td>
<td></td>
</tr>
<tr>
<td>S₃</td>
<td></td>
</tr>
<tr>
<td>S₉</td>
<td></td>
</tr>
</tbody>
</table>

Hold in memory

Cross with records from other set

R is the smaller dataset
Reduce-Side Join Many-to-Many

In reducer...

<table>
<thead>
<tr>
<th>keys</th>
<th>values</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_1$</td>
<td></td>
</tr>
<tr>
<td>$R_5$</td>
<td></td>
</tr>
<tr>
<td>$R_8$</td>
<td></td>
</tr>
<tr>
<td>$S_2$</td>
<td></td>
</tr>
<tr>
<td>$S_3$</td>
<td></td>
</tr>
<tr>
<td>$S_9$</td>
<td></td>
</tr>
</tbody>
</table>

Hold in memory

Cross with records from other set

What’s the problem?

R is the smaller dataset
Reduce-Side Join

• What are the limitations?
  – Both datasets are transferred over the network!
Map-Side Join: Basic Idea

Assume two datasets are sorted by the join key:

A sequential scan through both datasets to join (called a “merge join” in database terminology)
Map-Side Join: Parallel Scans

• If datasets are **sorted by join key**, join can be accomplished by a scan over both datasets
• How can we accomplish this in parallel?
  – Partition and sort both datasets in the same manner
• In MapReduce:
  – Map over one dataset, read from other corresponding partition
  – No reducers necessary (unless to repartition or resort)
• **Consistently partitioned datasets**: realistic to expect?
  – Depends on the workflow
  – For ad hoc data analysis, reduce-side are more general, although less efficient
References

• Data Intensive Text Processing with MapReduce, Lin and Dyer (Chapter 3)
• Mining of Massive Data Sets, Rajaraman et al. (Chapter 2)
• Hadoop tutorial: https://developer.yahoo.com/hadoop/tutorial/module4.html
• Jeffrey Dean and Sanjay Ghemawat, MapReduce: Simplified Data Processing on Large Clusters http://labs.google.com/papers/mapreduce.html