CS 378 – Big Data Programming

Lecture 15
Join Patterns
Review

• Assignment 6 – User Sessions

• We’ll look at implementation details of:
  – Parsing logs
  – Avro schema
  – Populating Avro object with data
  – Mapper
  – Combiner
    • Should we use one? Can we use one?
  – Reducer
Issues

• Grabbing data from a builder before `build()` is called.
• `getConf()` not defined
  – Your main class must `extend` Configured
• Map output other than `Text`, `AvroValue<Session>`
  – How is session data passed to reducers?
• Getting enum values given a string
Join Patterns

• It is almost always the case that our “big data” is coming from multiple sources
  – Web logs (of different types)
  – Databases (RDBMS, column-oriented DB, NoSQL, …)
  – Key/value store (redis, …)
  – …

• And we need to combine/integrate this data for
  – Reporting, analysis, BI, ETL, …
Join Patterns

• We’re familiar with “join” in RDBMS (using SQL)

• A join combines data from two or more data sets
  – Based on a field or set of fields – the foreign key
  – In RDBMS, the foreign key field matches values in the column of another table
  – Effectively a cross-reference between two or more tables
Join Example
Tables 5-1, 5-2 from MapReduce Design Patterns

**Table 5-1. Table A**

<table>
<thead>
<tr>
<th>User ID</th>
<th>Reputation</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3738</td>
<td>New York, NY</td>
</tr>
<tr>
<td>4</td>
<td>12946</td>
<td>New York, NY</td>
</tr>
<tr>
<td>5</td>
<td>17556</td>
<td>San Diego, CA</td>
</tr>
<tr>
<td>9</td>
<td>3443</td>
<td>Oakland, CA</td>
</tr>
</tbody>
</table>

**Table 5-2. Table B**

<table>
<thead>
<tr>
<th>User ID</th>
<th>Post ID</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>35314</td>
<td>Not sure why this is getting downvoted.</td>
</tr>
<tr>
<td>3</td>
<td>48002</td>
<td>Hehe, of course, it’s all true!</td>
</tr>
<tr>
<td>5</td>
<td>44921</td>
<td>Please see my post below.</td>
</tr>
<tr>
<td>5</td>
<td>44920</td>
<td>Thank you very much for your reply.</td>
</tr>
<tr>
<td>8</td>
<td>48675</td>
<td>HTML is not a subset of XML!</td>
</tr>
</tbody>
</table>
Join Example – Inner Join

Table 5-3 from MapReduce Design Patterns

<table>
<thead>
<tr>
<th>A.User ID</th>
<th>A.Reputation</th>
<th>A.Location</th>
<th>B.User ID</th>
<th>B.Post ID</th>
<th>B.Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3738</td>
<td>New York, NY</td>
<td>3</td>
<td>35314</td>
<td>Not sure why this is getting downvoted.</td>
</tr>
<tr>
<td>3</td>
<td>3738</td>
<td>New York, NY</td>
<td>3</td>
<td>48002</td>
<td>Hehe, of course, it’s all true!</td>
</tr>
<tr>
<td>5</td>
<td>17556</td>
<td>San Diego, CA</td>
<td>5</td>
<td>44921</td>
<td>Please see my post below.</td>
</tr>
<tr>
<td>5</td>
<td>17556</td>
<td>San Diego, CA</td>
<td>5</td>
<td>44920</td>
<td>Thank you very much for your reply.</td>
</tr>
</tbody>
</table>
Join Example – Left Outer Join
Table 5-4 from MapReduce Design Patterns

<table>
<thead>
<tr>
<th>A.User ID</th>
<th>A.Reputation</th>
<th>A.Location</th>
<th>B.User ID</th>
<th>B.Post ID</th>
<th>B.Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3738</td>
<td>New York, NY</td>
<td>3</td>
<td>35314</td>
<td>Not sure why this is getting downvoted.</td>
</tr>
<tr>
<td>3</td>
<td>3738</td>
<td>New York, NY</td>
<td>3</td>
<td>48002</td>
<td>Hehe, of course, it’s all true!</td>
</tr>
<tr>
<td>4</td>
<td>12946</td>
<td>New York, NY</td>
<td>null</td>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td>5</td>
<td>17556</td>
<td>San Diego, CA</td>
<td>5</td>
<td>44921</td>
<td>Please see my post below.</td>
</tr>
<tr>
<td>5</td>
<td>17556</td>
<td>San Diego, CA</td>
<td>5</td>
<td>44920</td>
<td>Thank you very much for your reply.</td>
</tr>
<tr>
<td>9</td>
<td>3443</td>
<td>Oakland, CA</td>
<td>null</td>
<td>null</td>
<td>null</td>
</tr>
</tbody>
</table>
Reduce Side Join

• The simplest implementation of mapReduce join is the *reduce side join*

• Reduce side join can accomplish any of the joins we discussed
  – Inner join, full outer join, left outer join, right outer join
  – Anti-join, full Cartesian product

• Requires that all data be sent over the network to reducers
• Can join as many data sets as you need (needs common key)
Reduce Side Join - Steps

• Mapper reads its data source, extracts *foreign key*
  – Foreign key output as the key, input record as value

• A partitioner can be used
  – If you understand the key distribution
  – Want to equally distribute keys across reducers

• Reducer performs the join operation
  – Inner join, outer join, .....
Reduce Side Join - Data Flow

Figure 5-1 from MapReduce Design Patterns
Reduce Side Join

• Join multiple types of logs into user sessions
  – Impression logs
  – Lead logs

• Key is userID and apikey
  – Just like we used to create sessions with impressions

• Join output is sessions with
  – Array of impressions
  – Array of leads
Assignment 7

• Extend the Avro object for user session
  – Include an array of leads
  – Leads ordered by timestamp
  – Format of lead log is the same as impression log
  – Lead log entry has different fields
  – Some fields are the same, but have different name (userid)

• Join impression logs and lead logs into one session
  – Use userId:apikey as the key

• Use MultipleInputs to handle different log formats
MultipleInputs

- The `MultipleInputs` class allows different mappers to handle different input data

```java
MultipleInputs.addInputPath(job,
    new Path(args[0]), TextInputFormat.class,
    ImpressionMapper.class);

MultipleInputs.addInputPath(job,
    new Path(args[1]), TextInputFormat.class,
    LeadMapper.class);
```