CS 378 – Big Data Programming

Lecture 18

Join Patterns

Review

- Assignment 7 – User Sessions
  - Reduce side join (impressions and leads)

- 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

Join Patterns

- For Assignment 8, we'll add replicated join to our session generator (Assignment 7)
  - Map from ZIP code to DMA code
  - DMA = Demographic Marketing Area (Nielsen)
  - Should we do this in `map()` or `reduce()`?

- Write sessions to different files
  - Based on session characteristics

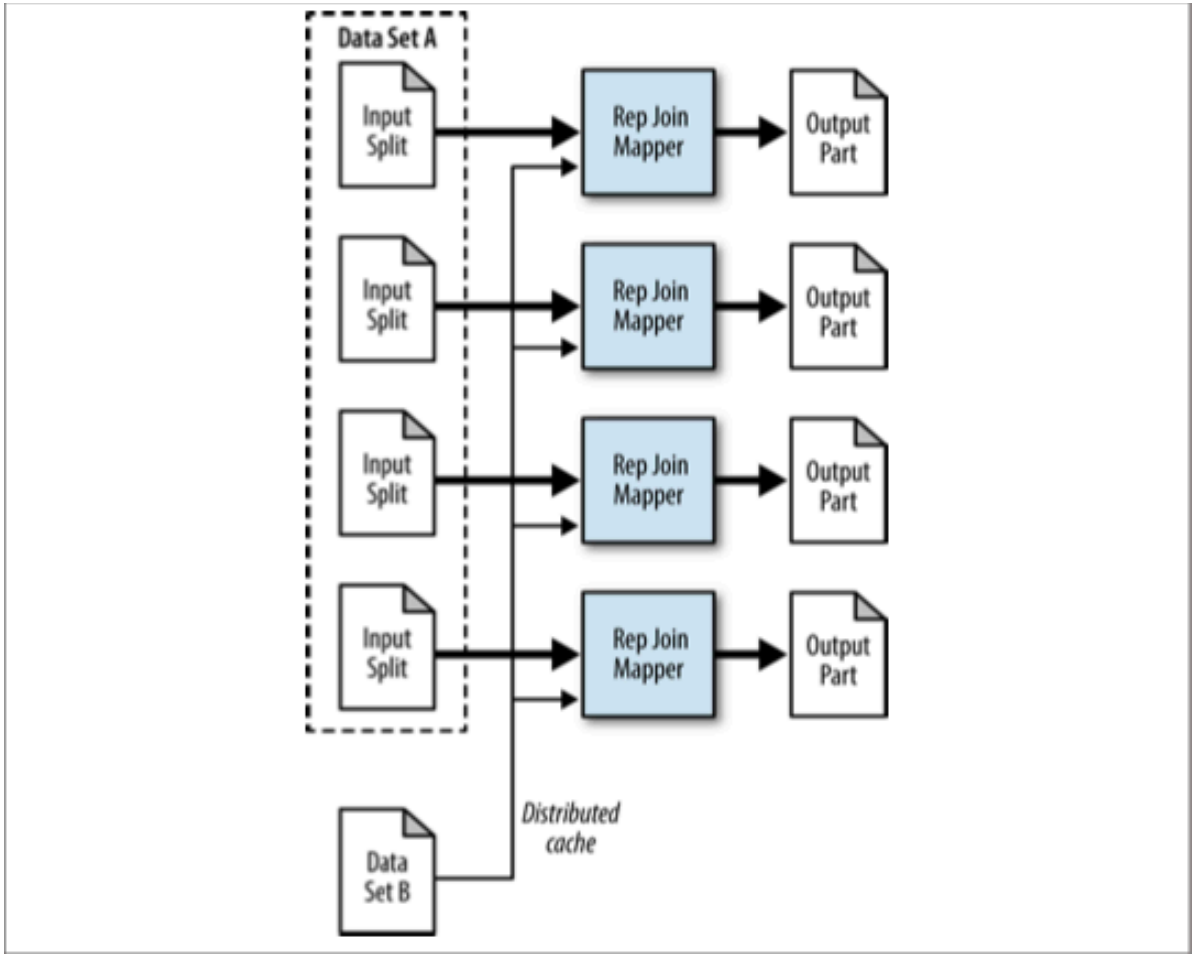
Review - Replicated Join

- Can be done completely in mappers
  - No need for sort, shuffle, or reduce
  - Only one of all the files can be “large”
  - Files are replicated with `DistributedCache`

- Restrictions:
  - All but one of the inputs must fit in memory
  - Can only accomplish an inner join, or
  - A left outer join where the large data source is “left” part

Replicated Join - Data Flow

Figure 5-2 from MapReduce Design Patterns



DistributedCache

- In the driver code (`run()` method)
  - Get the file name from the command line
  - Tell Hadoop about this file
  - File(s) conveyed in the configuration object

```
Path cacheFilePath = new Path(args[3]);
DistributedCache.addCacheFile(
    cacheFilePath.toUri(), conf);
```

DistributedCache

- In the mapper code (`setup()` method)
  - Get the file names from the configuration object
  - Load the data

```
Path[] paths = DistributedCache.getLocalCacheFiles(
      context.getConfiguration());
```

For each entry in `paths`, input the data:

```
Scanner scanner = new Scanner(
   new File(path[i].toString()));
```

Review - Multiple Outputs

- Hadoop class `MultipleOutputs`

- We saw this before with binning
  - Map-only pattern

- Since we have out user sessions completed in reduce
- Can we do the same thing (binning) in reduce output?
  - Suppose we want sessions to be “binned” or “partitioned” by some characteristic of the session

Session Categories

- Consider the following categories of sessions:
- Levels of user engagement
  - “Bouncer” – only one impression in the session
  - “Browser” – only SRP (search results page) views
  - “Searcher” – at least one “click through”
  - “Submitter” – submitted a lead

- In the `reduce()` method, categorize the user session
- Output the session to the corresponding name

MultipleOutputs Setup

- In the `run()` method, specify the named output

```
MultipleOutputs.addNamedOutput(job, "sessionType",
    TextOutputFormat.class, Text.class, Text.class);
```

- Enable counters for the multiple outputs

```
MultipleOutputs.setCountersEnabled(job, true);
```

MultipleOutputs Setup

- In the reduce class, define an instance variable

```
private MultipleOutputs multipleOutputs;
```

- In the `setup()` method of reducer

```
public void setup(Context context) {
    multipleOutputs = new MultipleOutputs(context);
}
```

- In `reduce()` method:

```
multipleOutputs.write("sessionType", key, value, category);
```

- In the `cleanup()` method of reducer

```
public void cleanup(Context context)
        throws InterruptedException, IOException{
    multipleOutputs.close();
}
```