CS 378 – Big Data Programming

Lecture 17

Join Patterns
Review

• Assignment 7 – User Session
  – Reduce side join (impressions and leads)

• Questions/issues:
  – null vs. “null” in Avro object output
  – vdp_index

  – Where do these logs come from?
Join Patterns

• Suppose we to join many sources, only one of which is large
  – User sessions (large)
  – Map from apikey to site names
  – Map from ZIP codes to DMA (demographic marketing area)
  – ...

• This is called a *replicated* join
  – All the small files will be replicated to all machines
Replicated Join

- Can be done completely in mappers
  - No need for sort, shuffle, or reduce
  - 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
Join Patterns

• OK, so replicated join was interesting, but more than one of my data sources is large.
• Is there a way to do a map-side join in this case?
• Or is reduce-side join my only option?

• If we organize the input data in a specific way,
• We can do this on the map-side.
Composite Join

• **Hadoop class** CompositeInputFormat

• Restricted to inner, or full outer join
• Input data sets must have the same #of partitions
  – Each input partition must be sorted by key
  – All records for a particular key must be in the same partition

• Seems pretty restrictive ...
Composite Join

• These conditions might exist for data from other mapReduce jobs where:
  
  • The jobs had the same # of reducers
    – Recall that input data sets must be partitioned in same way
  
  • The jobs had the same foreign key
  
  • Output files aren’t splittable
Composite Join

• If all those conditions are true, this join works
  – Map-side only, so it’s efficient if we can use it.

• If you find that you are preparing and formatting the data only to be able to use composite join
• It’s probably not worth it.
• Just use a reduce-side join.
Composite Join – Data
Composite Join – Data Flow
CompositeJoinInput

• In the driver code (**run()** method)
  – Get the file names from the command line
  – Specify the input format, join type, and files

```java
conf.setInputFormat(CompositeInputFormat.class);

conf.set("mapred.join.expr",
    CompositeInputFormat.compose("inner",
        KeyValueTextInputFormat.class, file1, file2));
```
CompositeJoinInput

• How might this implement inner join?

• Outer join?

• Could we do any other join type?
  – Left outer? Anti-join?
Cartesian Product

• Pairs every record with every other record
  – No keys needed
  – $N \times M$ results, for datasets of size $N$, $M$

• Map-only job

• But still expensive to compute

• Hadoop class: CartesianInputFormat
Cartesian Product – Data Flow