CS 378 – Big Data Programming

Lecture 13
more on
Data Organization Patterns
Review

• Assignment 6 – User Session

• Questions/Issues?
  – How to determine VDP (detail page) versus SRP (results page)
  – Count for |action:click| = 1047
  – Count for |type:action| = 867
  – What’s going on here?
Assignment 6

• Define an Avro object for user session
  – One user session for each unique userID/apikey
  – Session will include an array of impressions
  – Impressions ordered by timestamp
  – Each impression will contain an array of IDs (0 or more)

• Identify data associated with the session as a whole
• Identify data associated with individual impressions
• Include all the fields listed in the assignment
• Create enums where requested
Data Organization Patterns

• Structured to hierarchical pattern
  – User session is one such example
    • Organizing web logs by user
  – Textbook shows organizing posts and comments from StackOverflow
Data Flow

Figure 4-1 from MapReduce Design Patterns
Partitioning

• Organize “similar” records into partitions

• Why?
  – Future jobs will only focus on subsets of the data

• Partitioning schemes:
  – Time: hour, day, week, month, year
  – Geography: ZIP, DMA, state, time zone, country
  – Data source: web site
  – Data type
Partitioning

• No downside, as a mapReduce job can run over all partitions if needed

• We do need to know *a priori* how many partitions we want
  – Can run a job that scans and summarizes the data
  – Get possible values, and counts
  – Just like we did for user sessions
Partitioning

• What are some of the ways we might partition our user sessions?

• How would we do this?
MapReduce in Hadoop

Figure 2.4, Hadoop - The Definitive Guide

The number of reduce tasks is not governed by the size of the input, but instead is specified independently. In "The Default MapReduce Job" on page 227, you will see how to choose the number of reduce tasks for a given job.

When there are multiple reducers, the map tasks partition their output, each creating one partition for each reduce task. There can be many keys (and their associated values) in each partition, but the records for any given key are all in a single partition. The partitioning can be controlled by a user-defined partitioning function, but normally the default partitioner—which buckets keys using a hash function—works very well.

The data flow for the general case of multiple reduce tasks is illustrated in Figure 2-4.

This diagram makes it clear why the data flow between map and reduce tasks is colloquially known as "the shuffle," as each reduce task is fed by many map tasks. The shuffle is more complicated than this diagram suggests, and tuning it can have a big impact on job execution time, as you will see in "Shuffle and Sort" on page 208.

Finally, it's also possible to have zero reduce tasks. This can be appropriate when you don't need the shuffle because the processing can be carried out entirely in parallel (a few examples are discussed in "NLineInputFormat" on page 247). In this case, the only off-node data transfer is when the map tasks write to HDFS (see Figure 2-5).

Combiner Functions

Many MapReduce jobs are limited by the bandwidth available on the cluster, so it pays to minimize the data transferred between map and reduce tasks. Hadoop allows the user to specify a combiner function to be run on the map output, and the combiner...
Partitioning

• Define a Partitioner
• Examines each map() output pair
• Computes a partition number
Data Flow

Figure 4-2 from MapReduce Design Patterns
Recommendations – Assign 6

• Run WordCount on dataSet6.txt – see what’s in it
• Build your log entry parser and test it
  – Return a Map indexed by parameter name, with value being the parameter value
  – You can use this parser on other log types in the future
• Get you app working with just a few fields populated
  – Session with no impressions
  – Add impressions, but just the fields in the provided schema
  – Extend the schema and compile it
  – Then populate the new field(s) in your mapReduce code
• Write some unit tests as you go