CS 378 – Big Data Programming

Lecture 12
User Sessions from Logs
Review

• Assignment 5 – Avro Objects

• We’ll look at implementation details of:
  – Mapper
  – Combiner
    • Should we use one? Can we use one?
  – Reducer
  – Avro generated Java code
Other Issues

- Running MRUnit tests with Avro objects
  - Codehaus jackson version consistency
    - jackson-mapper-asl
    - jackson-core-asl
  - Avro serialization

- “shaded” JAR file – all dependencies included
  - Except Hadoop JAR – Why?

- Avro field definitions
  - Unions, defaults, ...
Review - Design Pattern

• Structured to hierarchical design pattern

• Data sources linked by some foreign key

• Data is structured and row based
  – For example, from databases

• Data is semi-structured and event based
  – Web logs
Sessionizing Web Logs

• Create user sessions from web logs

• Represents all the actions by a user

• Allows later analysis to “replay” the user actions

• Collect measures and metrics about user behavior
  – Pages viewed, time on page, clicks
  – Path through the site, entry to the site (from a search engine?)
Sessionizing Web Logs

• To start (this or any “big data” application)
• We need to understand the data
  – Fields, values
  – Data size

• We need to define our goal
  – What do we want to end up with
Web Logs

• Let’s look at some data
• Logs saved in database
  – Log entries already have structure
  – Tab separated values
  – Easily parsed (lots of work has been done for us)
Web Logs

• Our goal is to aggregate user actions into sessions, so we can better understand
  – User behavior
  – The impact changes have on user behavior

• So what should a session look like?
User Session

• Data about the session as a whole

• List of events (pages viewed, actions taken)
  – Ordered in time

• In our logs, what data is session-wide

• What data is impression/action specific
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
Assignment 6

• Define an Avro object for user session
  – One user session for each unique userID
  – Session will include an array of events
  – Events ordered by timestamp

• Identify data associated with the session as a whole
• Identify data associated with individual events
• Include all the fields in the log entries
• Create enums where requested
Assignment 6

• Run WordCount on dataSet6.tsv – see what’s in it
  – Modify WordCount to output values for each field:
    • fieldname:value
  – Ignore these fields (they have lots of values):
    • event_timestamp, image_count, initial_price, mileage, referrer, user_id, vin

• event_type
  – Break this into two fields in your schema:
  – Type (enum): use the first word of this field value
    • change, click, contact, edit, share, show, submit, visit
  – Subtype (enum): use remainder of the string
    • Example: ContactForm
Assignment 6

Recommendations

• **Use `dataset6Small.tsv` for development and testing**

• Get your app working with just a few fields populated
  – Session with no events
  – Add events, but just a few fields first
  – Extend the schema
  – Populate the new field(s) in your schema

• Write some unit tests as you go