CS 378 – Big Data Programming

Lecture 21
MetaPatterns
Review

- Assignment 9 – Job Chaining
  - Filter and bin sessions (same as assignment 8)
  - 3 jobs that process submitter, sharer, clicker bins
    - Can use the same map class
    - Compute stats for click types – over all sessions, not just sessions containing the click
  - Fourth job – aggregate click stats
    - Across the 3 session types
    - Across all click types (extra credit)
MetaPatterns

• We’ve discussed: Job chaining
  – Multiple jobs solving a multi-stage problem
  – When processing cannot be done in one job
  – When one output is input to multiple jobs

• Chain folding
  – Merging multiple mappers
  – Merging map logic with reducer
Job Merging

• Two jobs that read the same data
• But otherwise are unrelated

• If loading and parsing the data is expensive
• Let’s do this only once
Job Merging
Job Merging

• In effect we make the mappers read same data
  – Already the case

• And we make the reducers read same data
  – Presumably the two mappers output different data
  – How?

• Note: We’re not limited to merging two jobs
Job Merging

• What will it take?

• Both jobs must have the same map output key/value
  – Is there a way to avoid this?
  – How about a union type for key, or value, or both?

• Best applied to existing, frequently run jobs
• Requires the code to be merged
Job Merging

• Basic idea

• New mapper does work of both old mappers
  – For each input record
  – Do the work of first “old” mapper
  – Do the work of second “old” mapper
  – Might need to write two output values

• Add data to the key to distinguish the two
Job Merging

• Merge the mapper code:
  – Does the work of both “old” mappers
  – Adds data to any output indicating the origin

• Reducer code:
  – Identify input type based on extra data in the key
  – Separate the output with MultipleOutputs
Job Merging

• This pattern can be simplified by implementing a custom class for the new intermediate key
• Combines the old key with the tag
• Need a custom `ComparableWritable`
  – Why?
  – Isn’t `Writable` enough?
• Example (from the textbook)
Job Merging

• Using the TaggedText class

• Reduce signature (of the merged reducer):
  
  – reduce(TaggedText key, Iterable<XX> values, Context context)

• Original reducers had signature:
  
  – Reduce(Text key, Iterable<XX> values, Context context)

• What does the “merged” reducer do?
Job Merging

• Can we generalize the TaggedText class?

• Handle any key type?