CS 378 – Big Data Programming

Lecture 19

MetaPatterns

Review

- Assignment 8 – Job Chaining
  - Bin sessions (as in assignment 7), filter large
  - 4 jobs that process submitter, clicker, shower, visitor
    - Can use the same map class (or should)
    - Compute stats for event subtypes – over all sessions, not just sessions containing the subtype
  - Fifth job – aggregate event subtype stats
    - Across the 4 session types
    - Across all event subtypes (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

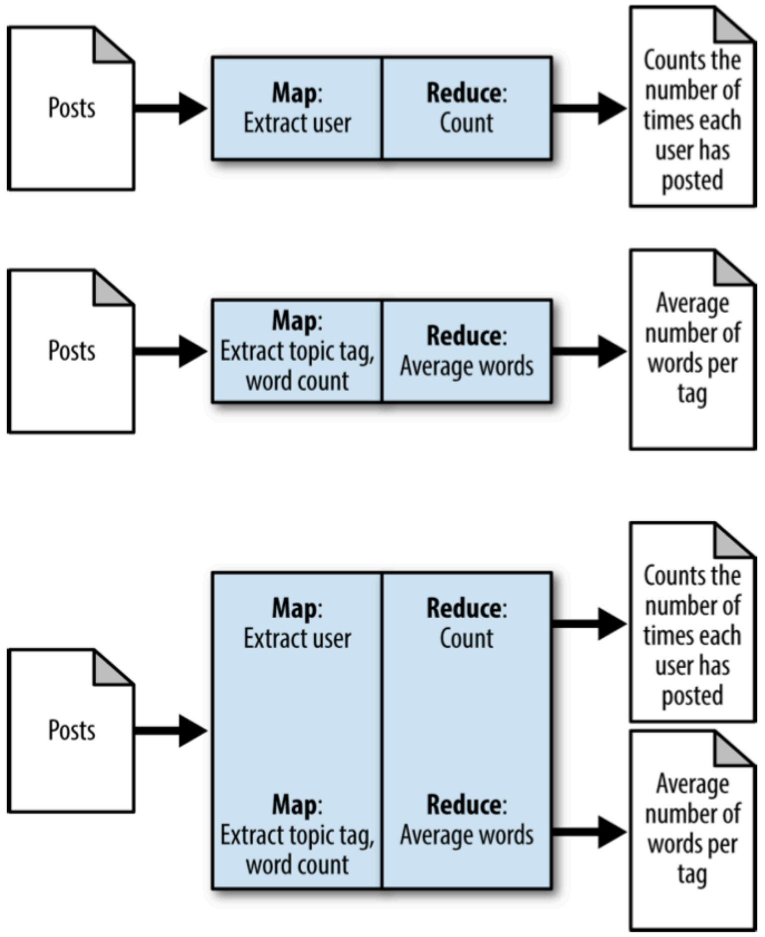
- Chain folding
  - Merging multiple mappers
  - Merging map logic with reducer

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


Posts
Map: Extract user
Reduce: Count
Counts the number of times each user has posted
Posts
Map: Extract topic tag, word count
Reduce: Average words
Average number of words per tag
Posts
Map: Extract user
Reduce: Count
Map: Extract topic tag, word count
Reduce: Average words
Counts the number of times each user has posted
Average number of words per tag

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 mappers
  - For each input record
  - Do the work of first “original” mapper
  - Do the work of second “original” mapper
  - Might need to write multiple times
    - Why?
- Add data to the key to distinguish the two

Job Merging

- Merge the mapper code:
  - Does the work of both "original" 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?