

CS 378 – Big Data Programming

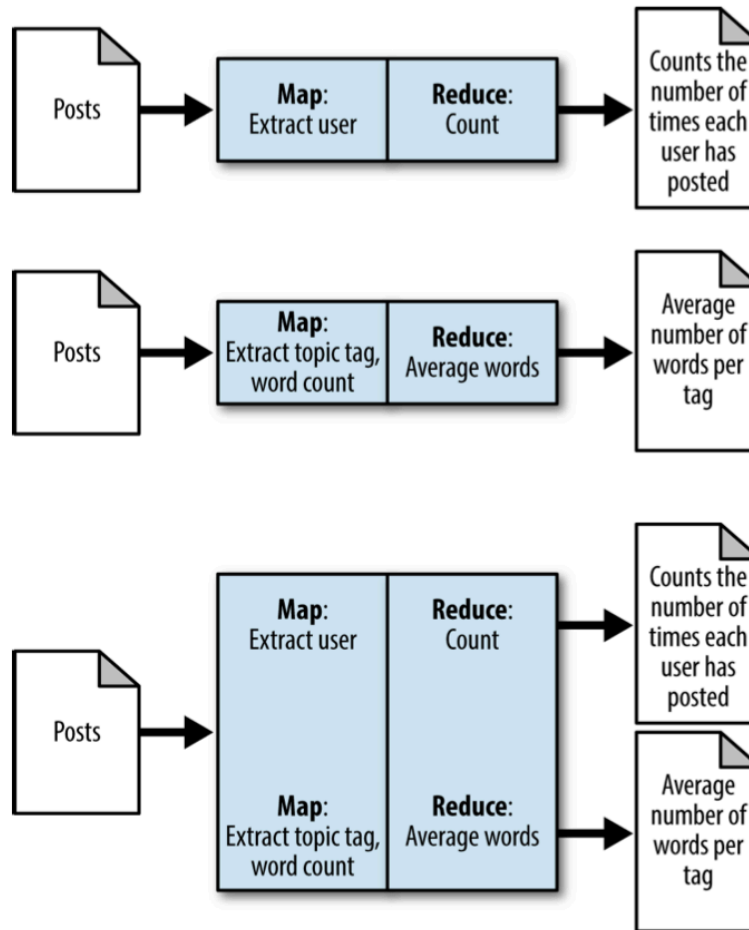
Lecture 17A
MetaPatterns
Job Merging

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
- 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

- 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

- 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?