CS 378 – Big Data Programming

Lecture 17

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
  - When output of multiple jobs is input to one job

- Implemented in the `run()` method

Job Chaining

- Data pipelines often produce temporary or intermediate files
- Output from one job that is input to another
  - As part of the pipeline, these files should be cleaned up
  - But you may want to keep them until the pipeline completes
  - Once complete, temp files can be deleted

Job Chaining - Scripting

- Another approach to managing job flow
  - Scripting languages
  - Shell scripts, python, …
- Benefits
  - Changing the job flow does not require compilation
  - Script can use services and systems that are not Java
  - Easy to build flows between existing jobs

Chain Folding

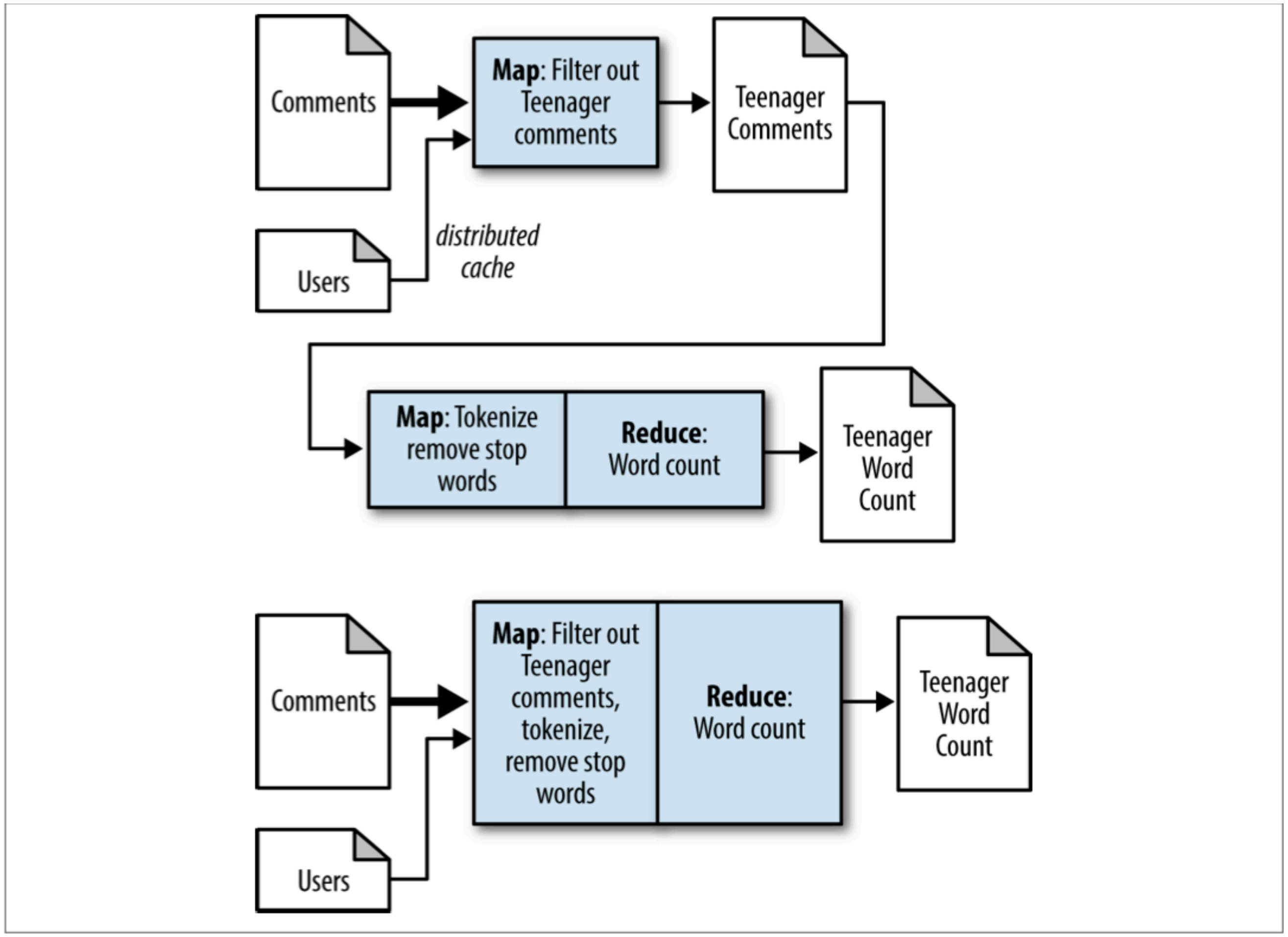
- Basic patterns that can be "folded":
  - Each record is submitted to multiple mappers
    - Combine these multiple map phases
  - Or to a reducer, then to a mapper
    - Push the map logic "upstream"
- Major benefit – reduce the amount of data moving through a data pipeline
  - Reduce disk I/O
  - Reduce data transfer (shuffle) over the network

Chain Folding

- Patterns that can benefit from folding

- In the data pipeline
  - Adjacent map phases might be merged
- Example:
  - Map only job, like a replicated join
  - Followed by map and reduce job
- Avoid writing the output of job one by joining the map logic of job one and two

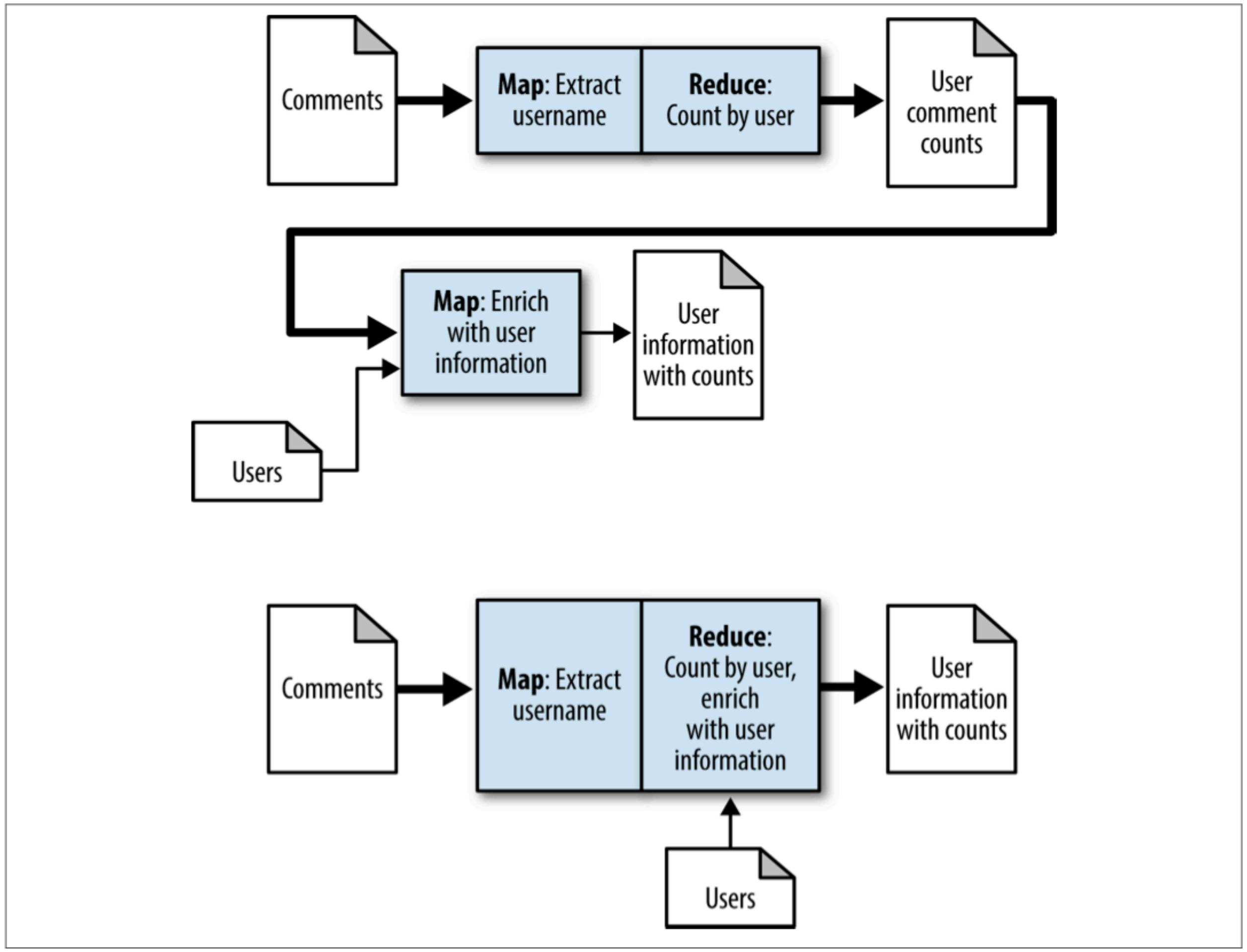
Chain Folding



Chain Folding

- Patterns that can benefit from folding
- A data pipeline ends with a map-only job
- Avoid reading the output of the penultimate job by merging the map logic of the final job into the previous reduce step

Chain Folding


Comments
Map: Extract username
Reduce: Count by user
User comment counts
Map: Enrich with user information
User information with counts
Users
Comments
Map: Extract username
Reduce: Count by user, enrich with user information
User information with counts
Users

Classes for Chaining

- `ChainMapper`
  - Specify a sequence of mappers
  - Output of one is input to the next
  - Arbitrary number can be “chained”
- `ChainReducer`
  - Specify the reducer
  - Specify a sequence of mappers
  - Arbitrary number of mappers can be “chained”

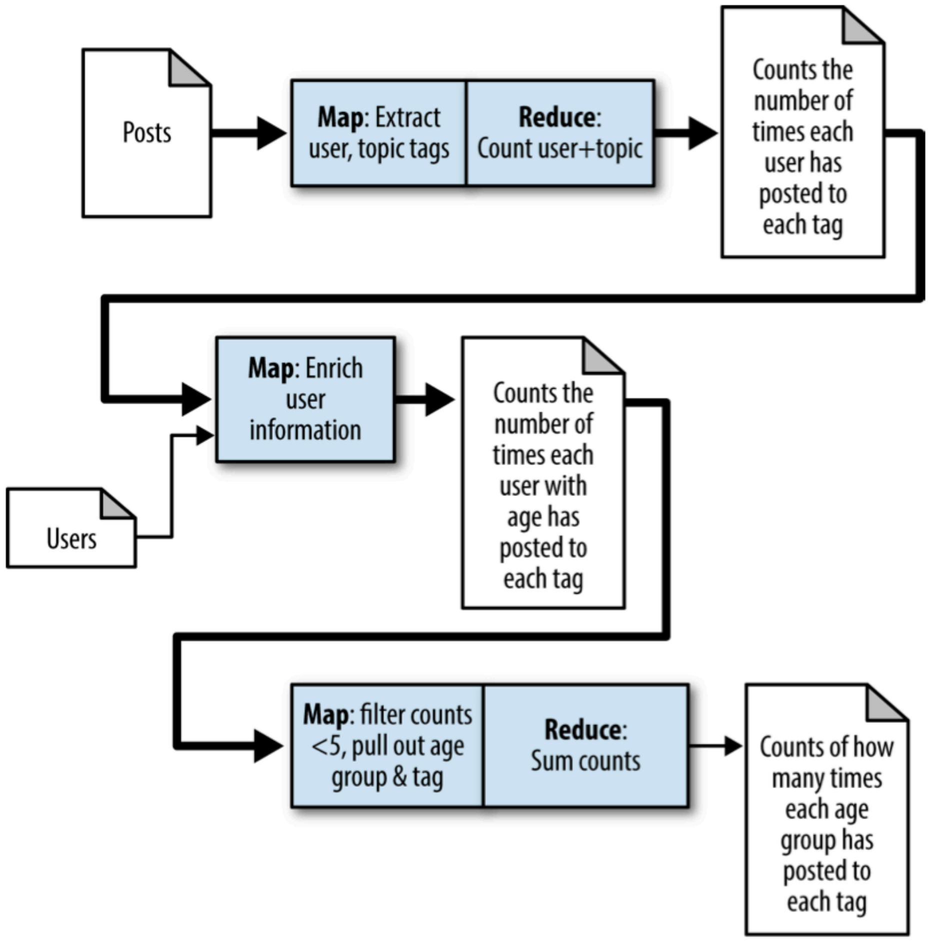
Chain Folding

- Split map phases between operations that
  - Decrease the amount of data (filtering)
  - Increase the amount of data (enrichment)

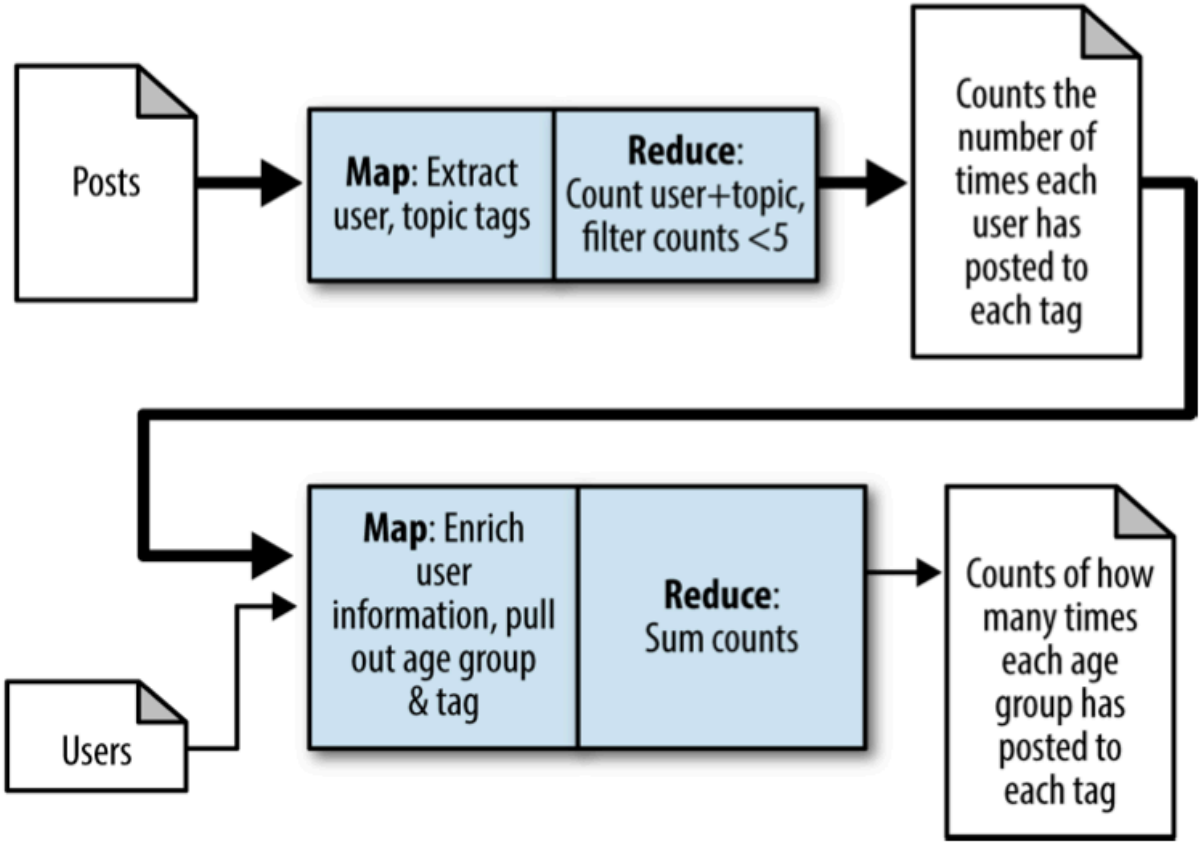
- Push the minimizing operation into previous reducer
  - This can reduce the amount of data transferred

- Generally, try to filter (minimize) data early

Chain Folding


Posts
Map: Extract user, topic tags
Reduce: Count user+topic
Counts the number of times each user has posted to each tag
Map: Enrich user information
Users
Counts the number of times each user with age has posted to each tag
Map: filter counts <5, pull out age group & tag
Reduce: Sum counts
Counts of how many times each age group has posted to each tag

Chain Folding


Posts
Map: Extract user, topic tags
Reduce: Count user+topic, filter counts <5
Counts the number of times each user has posted to each tag
Users
Map: Enrich user information, pull out age group & tag
Reduce: Sum counts
Counts of how many times each age group has posted to each tag