Grep

• Find all lines matching some pattern
• No need to combine anything
  – Reduce is not needed, i.e., just identity function
• Map takes line and outputs it if it matches the pattern
• Map could also take an entire document and emit all matching lines
  – Not a good idea if there is a single large document, but works well if there are many documents

URL Access Frequency

• Web log shows individual URL accesses
• Essentially the same Word Count
• Map can work with individual URL access records, or with an entire log file
  – Word Count analogy: work with individual words or with documents
• Reduce combines the partial counts for each URL
Reverse Web-Link Graph

• For each URL, find all pages (URLs) pointing to it (incoming links)
• Problem: Web page has only outgoing links
• Need all (anySource, P) links for each page P
  – Suggests Reduce with P as the key, source as value
• Map: for page source, create all (target, source) pairs for each link to a target found in page
• Reduce: since target is key, will receive all sources pointing to that target

Inverted Index

• For each word, create list of documents (document IDs) containing it
• Same as reverse Web-link graph problem
  – “Source URL” is now “document ID”
  – “Target URL” is now “word”
• Can augment this to create list of (document ID, position) pairs for each word
  – Map emits (word, (document ID, position)) while parsing a document
Distributed Sorting

• Does not look like a good match for MapReduce
• Send arbitrary data subset to reduce task?
  – How to merge them? Need another MapReduce phase.
• Can Map do pre-sorting and Reduce the merging?
  – Use set of input records as Map input
  – Map pre-sorts it and single reducer merges them
  – Does not scale!
• We need to get multiple reducers involved
  – What should we use as the intermediate key?

Distributed Sorting, Revisited

• MapReduce environment guarantees that for each reduce task the assigned set of intermediate keys is processed in key order
  – After receiving all (key2, val2) pairs from mappers, reducer sorts them by key2, then calls Reduce on each (key2, list(val2)) group
• Can leverage this guarantee for sorting
  – Map outputs (sortKey, record) for each record
  – Reduce simply emits the records unchanged
  – Make sure there is only a single reducer machine
• So far so good, but this still does not scale
Distributed Sorting, Revisited Again

• Quicksort-style partitioning
• For simplicity, consider case with 2 machines
  – Goal: each machine sorts about half of the data
• Assuming we can find the median record, assign all smaller records to machine 1, all others to machine 2
  – Can find approximate median by using random sampling
• Sort locally on each machine, then “concatenate” output

Partitioning Sort in MapReduce

• Consider 2 reducers for simplicity
• Run MapReduce job to find approximate median of data
  – Hadoop also offers InputSampler
    • Runs on client and is only useful if data is sampled from few splits, i.e., splits themselves should contain random data samples
• Map outputs (sortKey, record) for an input record
• All sortKey < median are assigned to reduce task 1, all others to reduce task 2
• Reduce just outputs the record component
Partitioning Sort in MapReduce

• Why does this work?
  • Machine 1 gets all records less than median and sorts them correctly because it sorts by key
  • Machine 2 similarly produces a sorted list of all records greater than or equal to median
• What about concatenating the output?
  – Not necessary, except for many small files (big files are broken up anyway)
• Generalizes obviously to more reducers

Handling Mapper Failures

• Master pings every worker periodically
• Workers who do not respond in time are marked as failed
• Mapper’s in-progress and completed tasks are reset to idle state
  – Can be assigned to other mapper
  – Completed tasks are re-executed because result is stored on mapper’s local disk
•Reducers are notified about mapper failure, so that they can read the data from the replacement mapper
Handling Reducer Failures

- Failed reducers identified through ping as well
- Reducer’s in-progress tasks are reset to idle state
  - Can be assigned to other reducer
  - No need to restart completed reduce tasks, because result is written to distributed file system

Handling Master Failure

- Failure unlikely, because it is just a single machine
- Can simply `abort` MapReduce computation
  - Users re-submit aborted jobs when new master process is up
- Alternative: master writes periodic `checkpoints` of its data structures so that it can be re-started from checkpointed state