MapReduce: Recap

Juliana Freire & Cláudio Silva

Some slides borrowed from Jimmy Lin, Jeff Ullman, Jerome Simeon, and Jure Leskovec
MapReduce: Recap

• Sequentially read a lot of data
  Why?
• Map: extract something we care about
  \( \text{map} \ (k, v) \rightarrow <k’, v’>* \)
  \( There \ is \ one \ Map \ call \ for \ each \ (k,v) \ pair \)
• Group by key: Sort and Shuffle
• Reduce: aggregate, summarize, filter, or transform
  \( \text{reduce} \ (k’, <v’>*) \rightarrow <k’, v’”>* \)
  \( There \ is \ one \ Reduce \ per \ unique \ key \ k’ \)
• Write the result

\( Structure \ remains \ the \ same, \ Map \ and \ Reduce \ change \ to \ fit \ the \ problem \)
The crew of the space shuttle Endeavor recently returned to Earth as ambassadors, harbingers of a new era of space exploration. Scientists at NASA are saying that the recent assembly of the Dextre bot is the first step in a long-term space-based man/machine partnership. "The work we're doing now -- the robotics we're doing - - is what we're going to need .........................
Map-Reduce: A Diagram

**MAP:**
Read input and produces a set of key-value pairs

**Intermediate:**
- k1:v k1:v k2:v
- k1:v k3:v k4:v
- k4:v k5:v
- k4:v k1:v k3:v

**Group by Key:**
- Hash merge, Shuffle, Sort, Partition

**Reduce:**
Collect all values belonging to the key and output

**Output:**
Map-Reduce: In Parallel

All phases are distributed with many tasks doing the work


Juliana Freire & Cláudio Silva
Shuffle and Sort in Hadoop

• Probably the most complex aspect of MapReduce!

• Map side
  – Map outputs are buffered in memory in a circular buffer
  – When buffer reaches threshold, contents are “spilled” to disk
  – Spills merged in a single, partitioned file (sorted within each partition): combiner runs here

• Reduce side
  – First, *map outputs are copied over to reducer machine*
  – “Sort” is a multi-pass merge of *map outputs (happens in memory and on disk)*
  – Final merge pass goes directly into reducer
Shuffle and Sort

Mapper

circular buffer (in memory)

merged spills (on disk)

Combiner

Reducer

intermediate files (on disk)

spills (on disk)

other mappers

other reducers

Combiner

Map side

Reduce side
MapReduce: Data Flow

- **Input and final output** are stored on the *distributed file system* (DFS)
  - Scheduler tries to schedule map tasks “close” to physical storage location of input data
  
  You can specify a directory where your input files reside using `MultipleInputs.addInputPath`

- **Typical usage pattern:**
  - Huge files (100s of GB to TB)
  - Mostly reads and appends to data

- **Intermediate results** are stored on *local FS* of Map and Reduce workers

- Output is often input to another MapReduce task
Chaining Multiple MapReduce Jobs

- [http://www.slideshare.net/martyhall/hadoop-tutorial-mapreduce-part-8-mapreduce-workflows](http://www.slideshare.net/martyhall/hadoop-tutorial-mapreduce-part-8-mapreduce-workflows)
- MapReduce workflow systems:
  - Cascading ([http://www.cascading.org](http://www.cascading.org))
  - Oozie ([http://oozie.apache.org](http://oozie.apache.org)) - supports construction of complex DAG workflows
Distributed File System

- Reliable distributed file system
- Data kept in “chunks” spread across machines
- Each chunk **replicated** on different machines
  - Seamless recovery from disk or machine failure

Bring computation directly to the data!

Chunk servers also serve as compute servers
How many Map (M) and Reduce (R) tasks?

- **Rule of thumb:**
  - Make $M$ much larger than the number of nodes in the cluster
  - Improves dynamic load balancing and speeds up recovery from worker failures
  - The number of maps is usually driven by the number of DFS blocks in the input files
    - One DFS chunk per map is common
    - You can adjust the DFS block size…

Note: You can provide a hint about the number of Map tasks by modifying JobConf's `conf.setNumMapTasks(int num)`

- Usually $R$ is smaller than $M$: Output is spread across $R$ files
Reinements: Backup Tasks

• Problem: Slow workers significantly lengthen the job completion time:
  – Other jobs on the machine
  – Bad disks
• Solution: Near end of phase, spawn backup copies of tasks -- whichever one finishes first “wins”
• Effect: Dramatically shortens job completion time
• In Hadoop, speculative tasks are on by default

Refinement: Combiners

- Often a Map task will produce many pairs of the form \((k, v_1), (k, v_2), \ldots\) for the same key \(k\)
  - E.g., popular words in the word count example

- Can save network time by **pre-aggregating values in the mapper**:
  - \(\text{combine}(k, \text{list}(v_1)) \rightarrow v_2\)
  - Combiner is usually same as the reduce function

- Works only if reduce function is commutative and associative
Refinement: Combiners

• Back to word counting example:
  – Combiner combines the values of all keys of a single mapper (single machine):

  – Much less data needs to be copied and shuffled!
Repartition: Partition Function

- Control how keys get partitioned
  - Inputs to map tasks are created by contiguous splits of input file
  - Reduce needs to ensure that records with the same intermediate key end up at the same worker
- System uses a default partition function:
  - hash(key) mod R, where R is the number of reducers
- Sometimes useful to override the hash function:
  - E.g., hash(hostname(URL)) mod R ensures URLs from a host end up in the same output file
Some Constraints and Unknowns

• Limited control over data and execution flow
  – All algorithms must expressed in m, r, c, p

• You don’t know:
  – Where mappers and reducers run
  – When a mapper or reducer begins or finishes
  – Which input a particular mapper is processing
  – Which intermediate key a particular reducer is processing