MapReduce and Spark

MIE451
Scott Sanner

Slides are taken from Liping Liu and Ch. 2 of the book Mining of Massive Datasets: http://www.mmds.org/

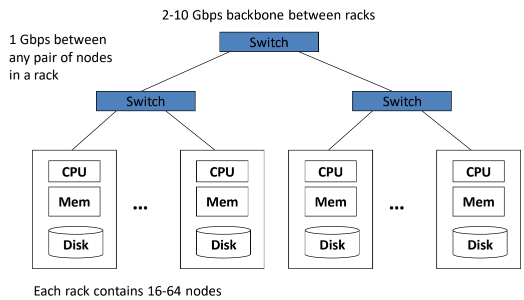
The big data challenge

- Google Example
  - 20+ billion web pages x 20KB = 400+ TB
  - 1 computer reads 30-35 MB/sec from disk
    - ~4 months to read the web
  - ~1,000 hard drives to store the web
  - Takes even more to **do** something useful with the data!
- With ``small data'', we still want to make the program faster with parallel computing

Distributed computation: overview

- **Today, a standard architecture for such problems is emerging:**
  - Cluster of commodity Linux nodes
  - Commodity network (ethernet) to connect them
- **Challenges:**
  - How to distribute computation?
  - Distributed/parallel programming is hard
  - Machines fail:
    - One server may stay up 3 years (1,000 days)
    - If you have 1,000 servers, expect to loose 1/day
- **Map-reduce** addresses all of the above

Cluster Architecture



In 2011 it was guestimated that Google had 1M machines, http://bit.ly/Shh0RO

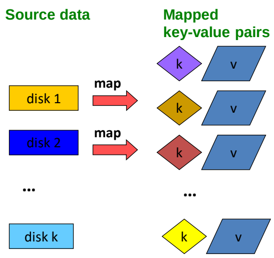
Distribute computation

- **Issue: Copying data over a network takes time**
- **Idea:**
  - Bring computation close to the data
  - Store files multiple times for reliability
- **Map-reduce** addresses these problems
  - Elegant way to work with big data
  - **Programming model**
    - Map-Reduce
  - **Storage Infrastructure – File system**
    - Google: GFS. Hadoop: HDFS

MapReduce Programming

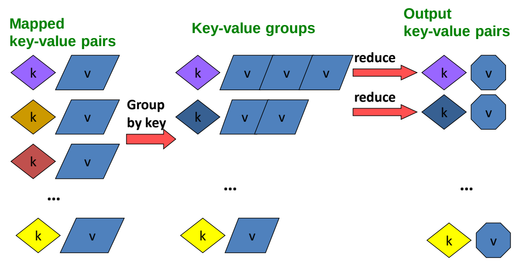
- Read data
- **Map:**
  - Extract something you care about
- **Sort and Shuffle**: Group by key
- **Reduce:**
  - Aggregate, summarize, filter or transform
- Write the result

MapReduce: The Map Step



7

MapReduce: The Reduce Step



8

More Specifically

- **Input:** a set of key-value pairs
- Programmer specifies two methods:
  - **Map(line in data file)** → <k', v'>*
    - Takes a line in data file and outputs **set** of key-value pairs
    - There is one Map call for every data file line
  - **Reduce(k', <v'>*)** → <k', v''>
    - **All values *v'* with same key *k'* are reduced together and processed in *v'* order**
    - There is one Reduce function call per unique key *k'*

9

Word count example

- We have a huge text document
- Count the number of times each distinct word appears in the file
  - File too large for memory, but all <word, count> pairs fit in memory
- **Sample application:**
  - Analyze web server logs to find popular URLs

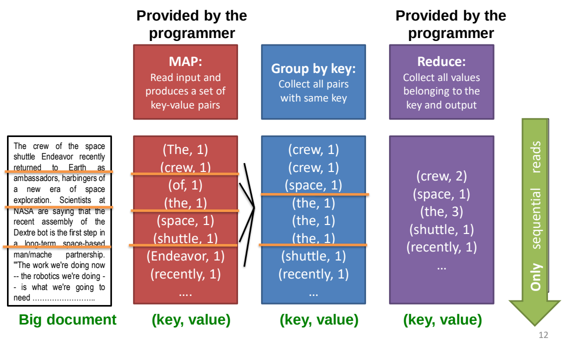
10

Task: Word Count

- A method that works on a small document
  - `words(doc.txt) | sort | uniq -c`
    - where `words` takes a file and outputs the words in it, one per a line
- This method captures the essence of **MapReduce**
  - Great thing is that it is naturally **parallelizable**

11

MapReduce: Word Counting



12

Word Count Using MapReduce

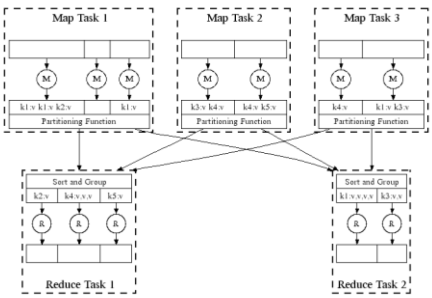
```
map(key, value):
// key: document name; value: text of the document
   for each word w in value:
        emit(w, 1)


reduce(key, values):
// key: a word; value: an iterator over counts
        result = 0
        for each count v in values:
                result += v
        emit(key, result)
```


Map-Reduce: Environment

Map-Reduce environment takes care of:

- Partitioning the input data
- Scheduling the program's execution across a set of machines
- Performing the **group by key** step
- Handling machine failures
- Managing required inter-machine communication

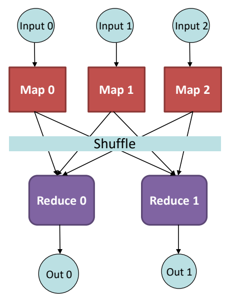
Map-Reduce: In Parallel



All phases are distributed with many tasks doing the work

Map-Reduce

- Programmer specifies:
  - Map and Reduce and input files
- **Workflow:**
  - Read file lines as key-value-pairs
  - Map transforms input kv-pairs into a new set of k'v'-pairs
  - Sorts & Shuffles the k'v'-pairs
  - All k'v'-pairs with a given k' are sent to the same reduce
  - Reduce processes all k'v'-pairs grouped by key into new k''v''-pairs
  - Write the resulting pairs to files
- All phases are distributed with many tasks doing the work

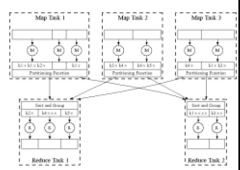


Data Flow

- **Input and final output are stored on a distributed file system (FS):**
  - Scheduler tries to schedule map tasks "close" to physical storage location of input data
- **Intermediate results are stored on local FS of Map and Reduce workers**
- **Output is often input to another MapReduce task**

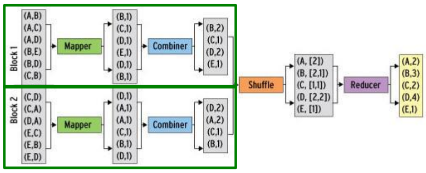
Refinement: Combiners

- Often a Map task will produce many pairs of the form (k,v_1), (k,v_2), … 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:**
  - combine(k, list(v_1)) → v_2
  - Combiner is usually same as the reduce function
- Works only if reduce function is commutative and associative,
  - Consider sum, max, mean and stddev



Refinement: Combiners

- **Back to our 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!

19

Refinements: Backup Tasks

- **Problem**
  - Slow workers significantly lengthen the job completion time:
    - Other jobs on the machine
    - Bad disks
    - Weird things
- **Solution**
  - Near end of phase, spawn backup copies of tasks
    - Whichever one finishes first "wins"
- **Effect**
  - Dramatically shortens job completion time

20

Coordination: Master

- **Master node takes care of coordination:**
  - **Task status:** (idle, in-progress, completed)
  - **Waiting tasks** get scheduled as workers become available
  - When a map task completes, it sends the master the location and sizes of its *R* intermediate files, one for each reducer
  - Master pushes this info to reducers
- Master pings workers periodically to detect failures

21

Storage Infrastructure

- **Problem:**
  - If nodes fail, how to store data persistently?
- **Answer:**
  - **Distributed File System:**
    - Provides global file namespace
- **Typical usage pattern**
  - Huge files (100s of GB to TB)
  - Data is rarely updated in place
  - Reads and appends are common

22

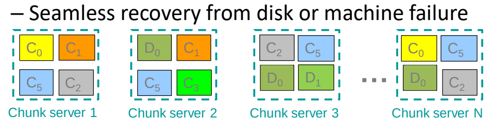
Distributed File System

- **Chunk servers**
  - File is split into contiguous chunks
  - Typically each chunk is 16-64MB
  - Each chunk replicated (usually 2x or 3x)
  - Try to keep replicas in different racks
- **Master node**
  - a.k.a. Name Node in Hadoop's HDFS
  - Stores metadata about where files are stored
  - Might be replicated
- **Client library for file access**
  - Talks to master to find chunk servers
  - Connects directly to chunk servers to access data

23

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

24