Apache Hadoop and MapReduce Part 1

What's it all about?

- Data-parallel programming model for clusters of commodity computers
- Pioneered by Google (2004)
  - Used by Google to process over 20 PB of data per day
- Popularized by open-source Hadoop project led by Yahoo!
  - used by Yahoo!, Facebook, Amazon, …

Used for ???

- At Google:
  - Index building for Google Search
  - Article clustering for Google News
  - Machine translation
- At Yahoo!:
  - Index building for Yahoo! Search
  - Spam detection for Yahoo! Mail
- At Facebook:
  - Data mining
  - Ad optimization
  - Space detection

Application

https://books.google.com/ngrams

MapReduce Goals

1. Scalability to extreme data volumes:
   - To scan 100TB on 1 node @ 50 MB/s = 24 days
   - To scan 100TB on 1000-node cluster = 35 minutes

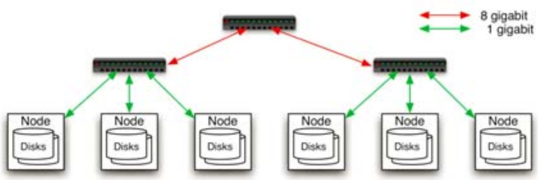
[LSST will collect 30TB astronomical data/night]

2. Cost efficiency:
   - Use commodity nodes (cheap but not reliable)
   - Commodity network
   - Automatic fault-tolerance (reduce admin costs)
   - Easy to use (reduce software development cost)

Non-Goals

- Serve as the only model for parallel computing
- Solve extremely hard problems that require complex algorithms (NP-hard)
- Complex scientific computing problems that are processing intensive

Typical Hadoop Cluster



- 40 nodes/rack, 1000-4000 nodes in cluster
- 1 GBps bandwidth in rack, 8 GBps out of rack
- Node :
  8 x 2.0 GHz cores, 8 GB RAM, 4 disks (= 4 TB?)

Challenges

- **Cheap nodes fail, especially if you have a lot**
  – If MTBF for 1 node = 3 years
  then MTBF for 1000 nodes on order of 1 day
  - If MTBF for 1 node = 1 year
  then MTBF for 10000 nodes on order of 1 hour
  – Solution: Build fault-tolerance into system
- **Commodity network = low bandwidth**
  – Solution: Minimize data transfer; do computations where the data resides
- **Programming distributed systems is hard**
  – Solution: MapReduce. Users write data-parallel "map" and "reduce" functions, system handles work distribution and faults

Two Basic Components

- Distributed file system (HDFS)
  - Modeled after GFS
  - Single namespace for entire cluster
  - Replicated data (3x) for fault-tolerance
  - provides fault-tolerance and scalability

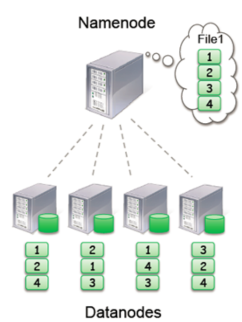
- MapReduce framework
  - Executes user jobs identified as “map” and “reduce” functions
  - Manages work distribution & fault-tolerance

HDFS

- Files split into *blocks, 32MB to 128MB (usually)*
- Blocks replicated across several *datanodes* (usually 3)

- *Namenode* stores metadata (file names, locations, etc)

- Optimized for large files, sequential reads
- Files are append-only



MapReduce

- A programming model based on functional programming concepts
- For large scale parallel data processing
- Implemented in Java but …
- Not a programming language
- Many different languages can be used for development

- No data model other than the manipulation of (key, value) records

Figure 2.2 suggests this computation.

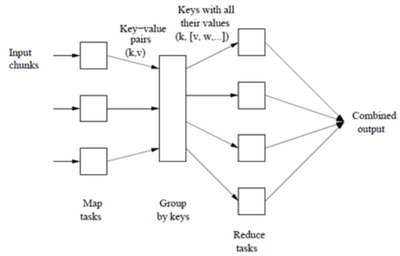


Figure 2.2: Schematic of a MapReduce computation

Consider a slightly more general program to compute the word frequency of every word in a single document

Abridged Declaration of Independence

A Declaration By the Representatives of the United States of America, in General Congress Assembled.
When in the course of human events it becomes necessary for a people to advance from that subordination in which they have hitherto remained, and to assume among powers of the earth the equal and independent station to which the laws of nature and of nature's god entitle them, a decent respect to the opinions of mankind requires that they should declare the causes which impel them to the change.
We hold these truths to be self-evident; that all men are created equal and independent; that from that equal creation they derive rights inherent and inalienable, among which are the preservation of life, and liberty, and the pursuit of happiness; that to secure these ends, governments are instituted among men, deriving their just power from the consent of the governed; that whenever any form of government shall become destructive of these ends, it is the right of the people to alter or to abolish it, and to institute new government, laying it's foundation on such principles and organizing it's power in such form, as to them shall seem most likely to effect their safety and happiness. Prudence indeed will dictate that governments long established should not be changed for light and transient causes: and accordingly all experience hath shewn that mankind are more disposed to suffer while evils are sufferable, than to right themselves by abolishing the forms to which they are accustomed. But when a long train of abuses and usurpations, begun at a distinguished period, and pursuing invariably the same object, evinces a design to reduce them to arbitrary power, it is their right, it is their duty, to throw off such government and to provide new guards for future security. Such has been the patient sufferings of the colonies; and such is now the necessity which constrains them to expunge their former systems of government. the history of his present majesty is a history of unremitting injuries and usurpations, among which no one fact stands single or solitary to contradict the uniform tenor of the rest, all of which have in direct object the establishment of an absolute tyranny over these states. To prove this, let facts be submitted to a candid world, for the truth of which we pledge a faith yet unsullied by falsehood.

(people, 2)
(government, 6)
(assume, 1)
(history, 2)
…

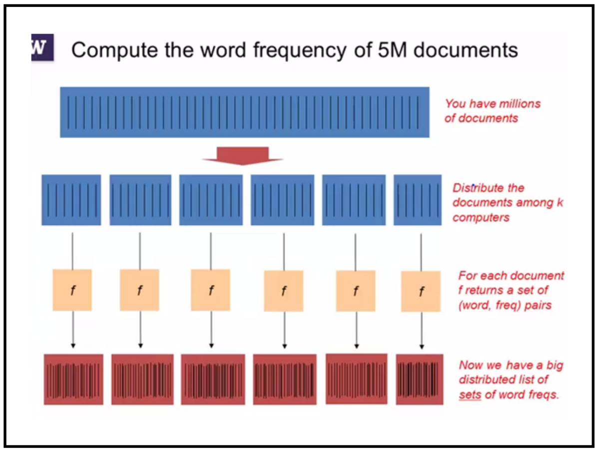
What if we want to compute the word frequency across *all* documents?

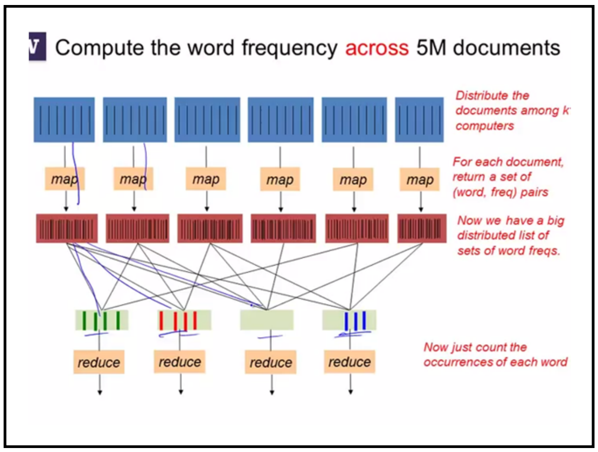
US Constitution

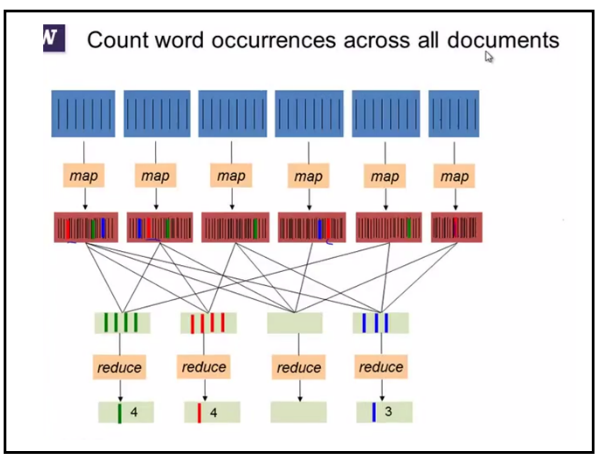
Declaration of Independence

Articles of Confederation

(people, 78)
(government, 123)
(assume, 23)
(history, 38)
…

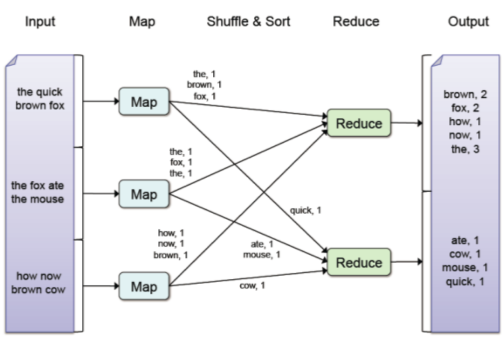

Compute the word frequency of 5M documents
You have millions of documents
Distribute the documents among k computers
f
For each document f returns a set of (word, freq) pairs
Now we have a big distributed list of sets of word freqs.


Compute the word frequency across 5M documents
Distribute the documents among k computers
map
For each document, return a set of (word, freq) pairs
Now we have a big distributed list of sets of word freqs.
reduce
Now just count the occurrences of each word


Count word occurrences across all documents
map
reduce
4
4
3

mapper, reducer

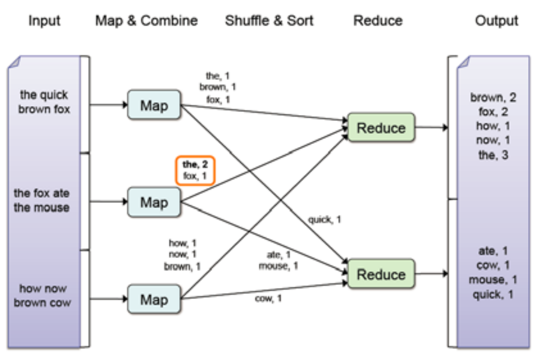
```
# A mapper in Python
def mapper(key, val):
       words = key.split()
       for word in words:
          wmr.emit(word, '1')


# A reducer in Python
def reducer(word, counts):
       total = 0
       for count in counts:
          total += int(count)
       wmr.emit(word, count)
```



Common Optimization: combiner()

- For efficiency, a mapper function is often used in conjunction with a "combiner()" on the same node
- Pushes simple operations from the reducer back to the mapper node
- Decreases size of intermediate data

```
# A combiner in Python
def combiner(word, counts):
    total = 0
    for count in counts:
       total += int(count)
    wmr.emit(word, count)
```



Fault Tolerance in MapReduce

1. If a task crashes:
   - Retry on another node
     - OK for a map because it had no dependencies
     - OK for reduce because map outputs are on disk
   - If the same task repeatedly fails, fail the job or ignore that input block

➢ Note: For fault tolerance to work, *your map and reduce tasks must be side-effect-free*

Fault Tolerance (continued)

2. If a node crashes:
   - Relaunch its current tasks on other nodes
   - Relaunch any maps the node previously ran
     - Necessary because their output files were lost along with the crashed node

Fault Tolerance (continued)

3. If a task is going slowly (straggler):
   - Launch second copy of task on another node
   - Take the output of whichever copy finishes first, and kill the other one

- Critical for performance in large clusters ("everything that can go wrong will")

Takeaways

- By providing a data-parallel programming model, MapReduce can control job execution under the hood in useful ways:
  - Automatic division of job into tasks
  - Placement of computation near data
  - Load balancing
  - Recovery from failures & stragglers

Example

- Create an inverted index of words in tweets

DATA: tweetID, tweetText

What might map() look like?
What might reduce() look like?

Example

- Web logs

DATA: userID, URL, timestamp, additional-info

Task: Count number of accesses to each domain (from URL)

What might map() look like?

What might reduce() look like?

Extensions?