Hadoop

Scalable Distributed Computing

Claire Jaja, Julian Chan

October 8, 2013

What is Hadoop?

- A general-purpose storage and data-analysis platform
- Open source Apache software, implemented in Java
- Enables easier parallel processing
- Designed to run on clusters built on cheap commodity hardware
- Consists mainly of:
  - A distributed file system
  - MapReduce programming framework

http://hadoop.apache.org

What is Parallel Processing?

- What is sequential processing?
  - When your program only has a single code path from beginning to finish.
  - Other than simple data analysis tasks, rarely can you get away with just one thread in the real world.

- Parallel Processing
  - When your program executes multiple code paths simultaneously
  - Almost all apps you use regularly are multi-threaded.
  - Examples:
    - Email app displaying mail while downloading new ones in the background.
    - Using multiple machines to do data analysis in parallel (SETI@home, bitcoin mining worms, etc.)

Why do you need parallel processing?

- End of Moore's law
  - No more free lunch.
  - CPUs are getting more cores but clock speed stays the same
  - SOLUTION: Do processing on multiple cores across multiple machines simultaneously
  - PROBLEM: Writing multi-threaded code is HARD

- Big Data™ Slow Harddrives
  - Data, data everywhere!
  - Hard drives are really cheap and voluminous these days, but they are still painfully slow to read from and write to.
  - One way we can mitigate the problem is to split up our data into chunks, put them on multiple hard drives, and read from them simultaneously.
  - PROBLEM: Once you have a lot of machines connected together, things tend to go wrong (as your friendly sysadmin can probably tell you). You code would need to deal with these failures.

You will need to do Parallel Processing, but it is hard.

Hey, I’ve been doing parallel processing since Nixon was president. Get off my lawn!

- The High Performance Computing (HPC) and Grid Computing communities have been doing large-scale data processing for years
- Broadly, the approach in HPC is to distribute the work across a cluster of machines, which access a shared filesystem, hosted by a Storage Area Network (SAN).
- Works well for predominantly compute-intensive jobs, but it becomes a problem when nodes need to access larger data volumes (hundreds of gigabytes, the point at which MapReduce really starts to shine).
- Network bandwidth becomes the bottleneck and compute nodes go idle.

From O’Rilley Hadoop the definitive guide 3rd ed.

A Simple Hadoop example – Word Counts

- Input: large text file(s) with white space separated tokens
- Task: count the number of occurrences of each token
- Output: text file with each type and its count

(Sounds familiar? We did this task in HW1 for LING 570 - make_voc)

Why use Hadoop for this task?

- Significant time savings for large (100s of GBs) files
- Lots of I/O
- Easy to parallelize (counts on one piece of file are independent of counts on other pieces)
- Lends itself to the MapReduce paradigm (output is key-value pairs)

Word count: sequential solution

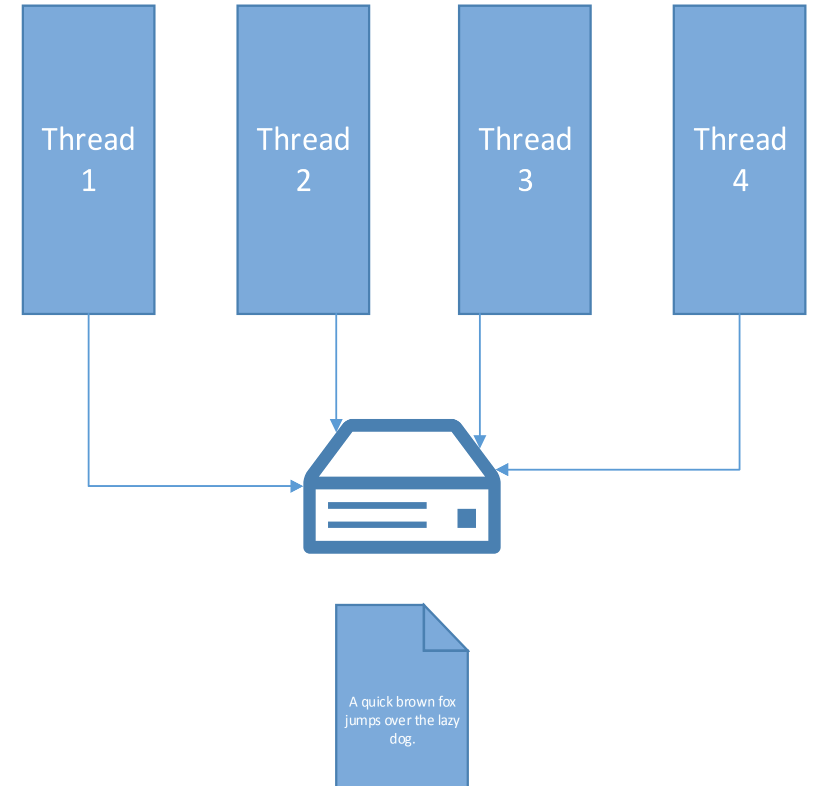
- Read a line
- Tokenize the line
- Update hash table
- Repeat

- Works, but performance is limited by disk read speed.
- You could also run out of ram for the hash table.

Parallelized Word Count – first try

- Let's use multiple threads!
- Each thread will process one part of the text.
- Once they are done, we will combine the results.

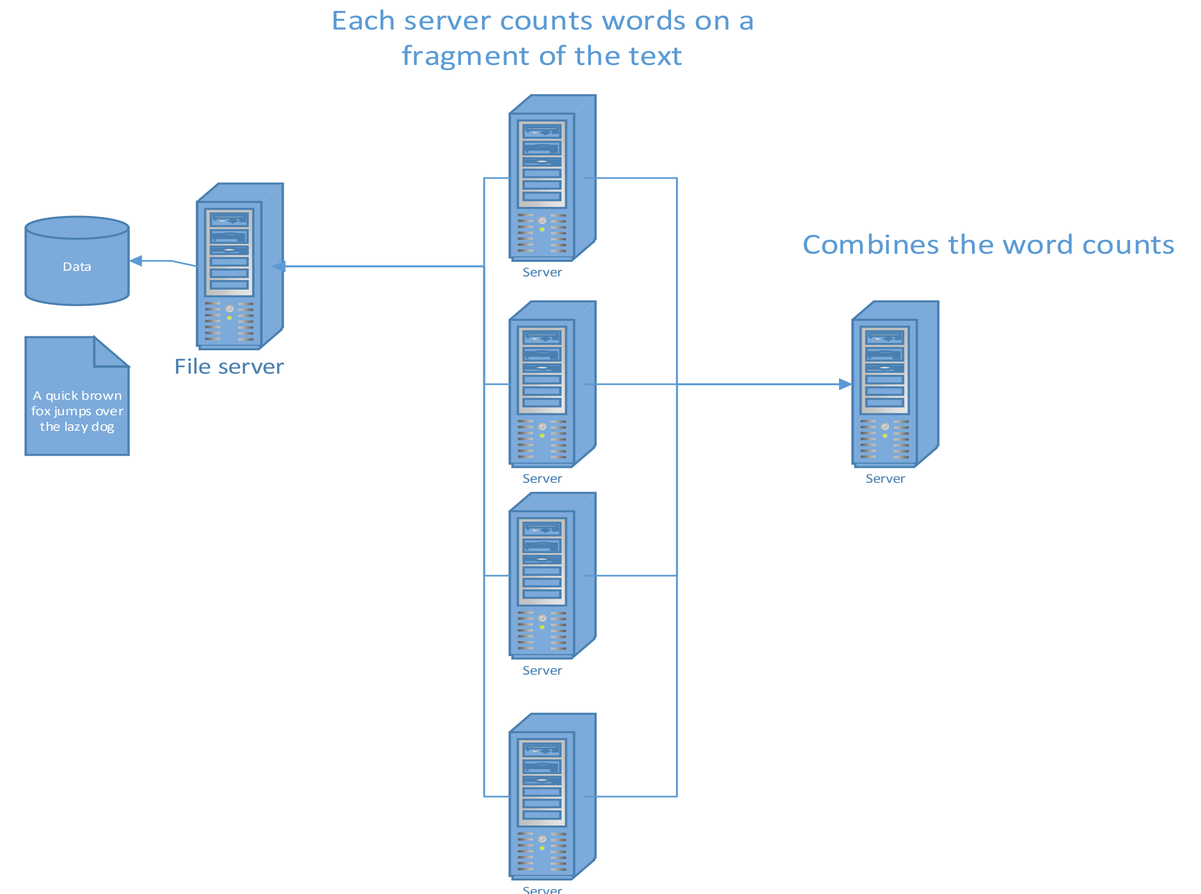
- Problem: performance is still limited by the read speed of the drive.
- In practice, it may even be slower than the sequential solution because of poor data access locality.
- Adding more machines still won't help you.



Parallelized Word Count – second try

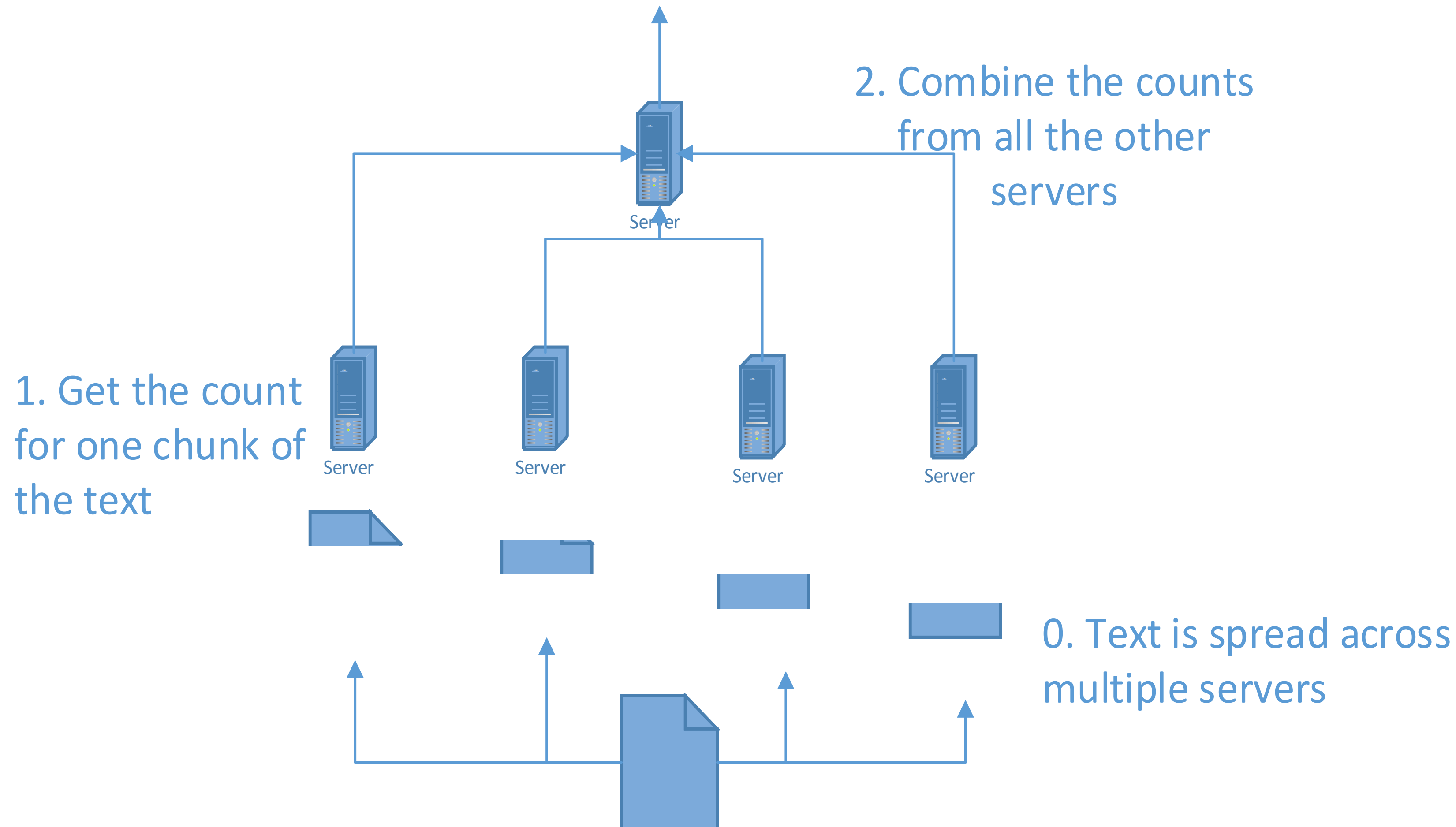
- The text file is put on a fileserver
- Each server accesses a part of the text on the file server and does the word count
- Depending on how good the file server is, this will be faster than sequential.

- Problem: Network speed becomes the bottleneck.



Parallelized Word Count – distributed data

- Instead of having a centralized fileserver, we will spread the data over the servers doing the processing



Parallelized Word Count - Distributed Data cont.

- First, you'll need to write code to distribute the chunks out to individual machines.
  - You need to keep track of which chunk is not processed, pending, and processed.
- Now what if one of those machines goes down while the count is happening?
  - How do you detect it?
  - Your boss wouldn't appreciate - "meh, just restart the whole thing"
- Then you need to write code to combine all the results.
  - What if two servers are done at the same time?

- After you've taken care of all these potential problems…

Congratulations!

- **You have just reinvented Hadoop!**

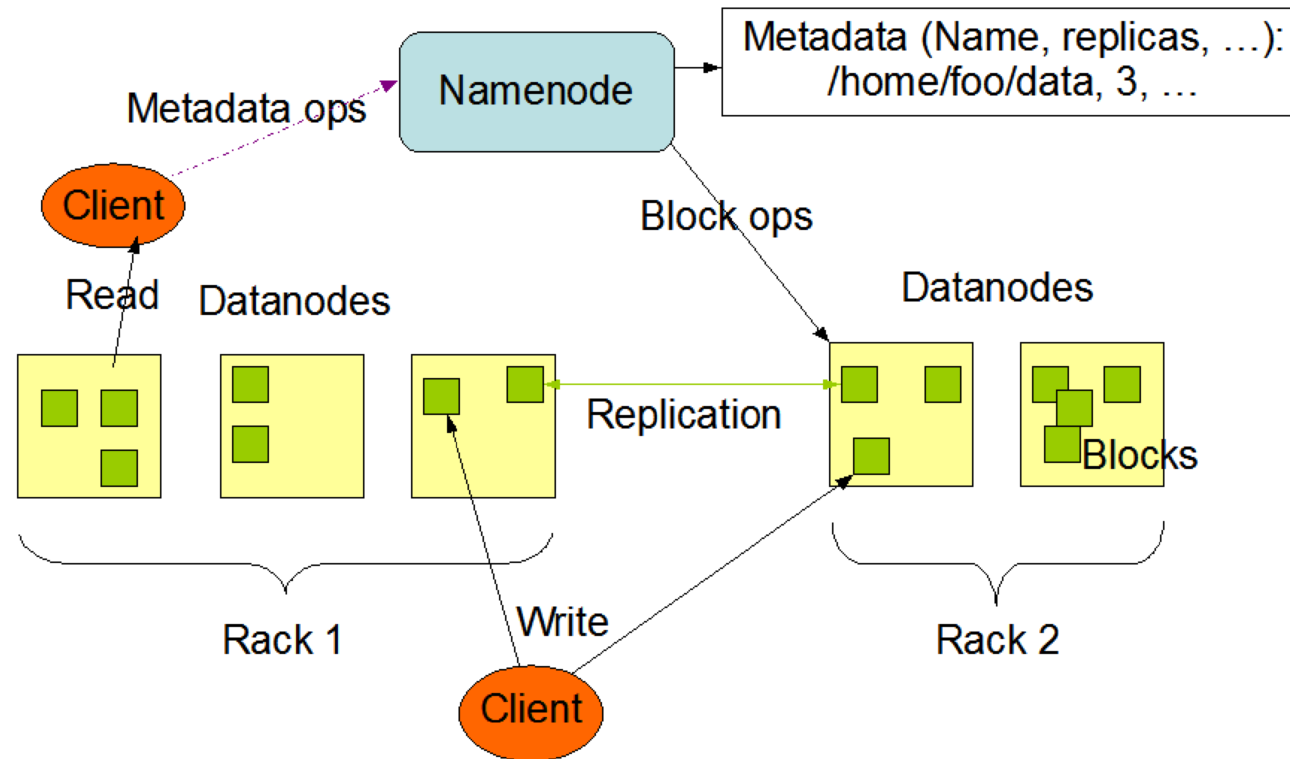
Hadoop Distributed File System (HDFS) motivations and goals

- "Big Data" – one disk is not enough.
- Mainframes are expensive. Clusters are cheap.
- Fault resistant - detect faults and provide quick, automatic recovery
- Move computation, not data - applications move themselves closer to where data is located
- Streaming data access - designed for batch processing with high throughput of data access
- Portability - designed to be easily portable across heterogeneous hardware and software platforms

Hadoop Distributed File System (HDFS) how it works

- NameNode: master server, manages file system namespace, regulates access to files by clients
- DataNodes: manage storage attached to nodes they run on
- files split into one or more blocks (typically 64 MB), which are then stored in a set of DataNodes
- blocks replicated for fault tolerance

Hadoop Distributed File System (HDFS) a diagram



MapReduce paradigm

"MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key."

Jeffrey Dean and Sanjay Ghemawat. 2004. MapReduce: simplified data processing on large clusters. In *Proceedings of the 6th conference on Symposium on Operating Systems Design & Implementation - Volume 6* (OSDI'04), Vol. 6. USENIX Association, Berkeley, CA, USA, 10-10
http://research.google.com/archive/mapreduce.html

MapReduce paradigm

1. Prepare data - split across servers to multiple inputs, keep track of what went where
2. Run Map() on individual inputs - run user-provided code to generate outputs in the form of key-value pairs
3. Shuffle outputs - assign each processor one (or more) key(s), send output associated with that key
4. Run Reduce() - run user-provided code to combine values for each key
5. Produce output - collect all Reduce() output, re-combine and sort by key to create final output

MapReduce Implementation in Hadoop

- JobTracker - responsible for scheduling component tasks, monitoring them, and re-executing failed tasks
- TaskTrackers - one per cluster-node, execute tasks as directed by JobTracker
- Typically, compute nodes and storage nodes are the same, i.e. MapReduce framework and HDFS run on same set of nodes - can schedule tasks on nodes where data is already present
- Must specify input/output locations and supply Map and Reduce functions
- Although Hadoop framework is implemented in Java, the Map and Reduce functions don't need to be written in Java (can be Python, Ruby, C++, etc)

Running Hadoop: an example WordCount.java - Map

```
public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {

public void map(
          LongWritable key,
          Text value,
          Context context) throws IOException, InterruptedException
{
          String line = value.toString();
  StringTokenizer tokenizer = new StringTokenizer(line);
          while (tokenizer.hasMoreTokens())
          {
             word.set(tokenizer.nextToken());
    static IntWritable one = new IntWritable(1);
             context.write(word, one);
          }
    }
```

http://wiki.apache.org/hadoop/WordCount

Running Hadoop: an example WordCount.java - Reduce

```
public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {
public void reduce(
        Text key,
        Iterable<IntWritable> values,
        Context context) throws IOException, InterruptedException
{
         int sum = 0;
         for (IntWritable val : values) {
             sum += val.get();
         }
         context.write(key, new IntWritable(sum));
    }
}
```

http://wiki.apache.org/hadoop/WordCount

Running Hadoop: an example
Command line

```
$ hadoop fs -mkdir brown
$ hadoop fs -put /corpora/ICAME/texts/brown1 brown/input
$ hadoop jar /opt/hadoop/hadoop-examples-*.jar wordcount brown/input
brown/output
```

http://depts.washington.edu/uwcl/twiki/bin/view.cgi/Main/HadoopWordCountExample

Conclusion

- Traditional HPC parallel processing is good for computation-heavy parallel processing, but is limited by the disk access and network speed bottleneck.
- Hadoop is a cheap and easy way to do parallel processing with high I/O throughput.
- NLP tends to be high in I/O, so chances are good that you'll use Hadoop at some point.
- Hadoop shields you away from having to deal with the problems that could arise in processing distributed data, so you can focus on the core algorithm.
- Hadoop is **not** the be-all end-all solution for parallel processing.
- Hadoop is not just for toy problems like Word Count. It's been deployed to index the web at Google and Yahoo.