CS 199 ACC
MapReduce

Prof. Robert J. Brunner

Ben Congdon
Tyler Kim
Bhuvan Venkatesh

MP 1

How was it?

This Week

- Distributed Systems
- What MapReduce is (according to the original paper)
- The MapReduce programming paradigm
- Hadoop

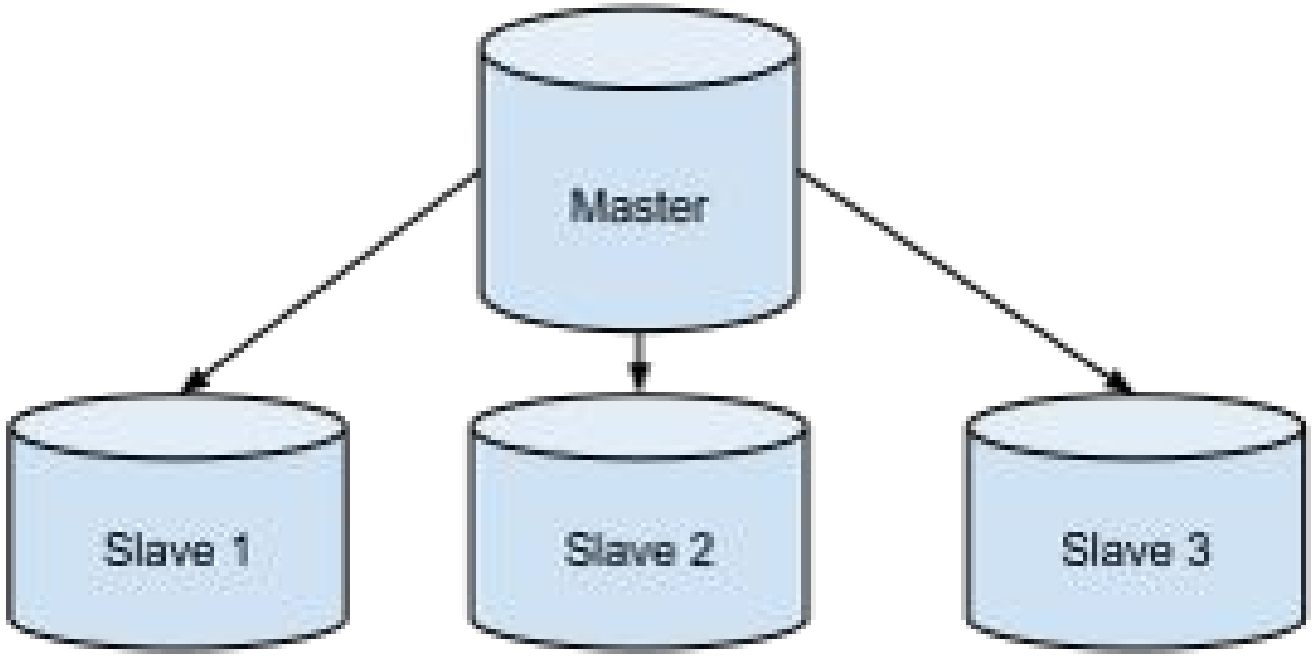
Quick Recap: Reasons

- Processors aren’t getting that much faster and multithreading isn’t helping
- Let’s distribute the computation, but coordination is hard.
- How do we handle different operating systems, network topologies, load balancing and failure?
- Enter MapReduce

A little bit about distributed systems

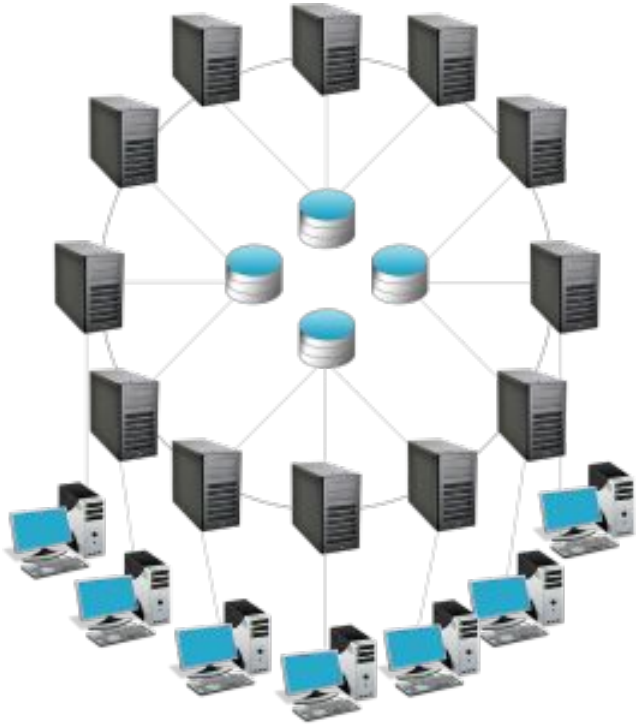
- We aren’t talking about supercomputers. Supercomputers are big computers that you typically use software like MPI (Message Passing Interface) to code up software
- We are talking about multiple computers connected together
- There are two key differences: there is no idea of actions happening “first” (meaning someone cannot declare that the value happens first)
- Computers can fail at any point which is a problem

Simple solution - Not Resistant to Failure



http://erlycoder.com/themes/site4fast/images/master-slave1.png

More Complicated Solution - True Distributed



http://sourcedelica.com/wordpress/wp-content/uploads/2015/09/pg-distrib-logo.png

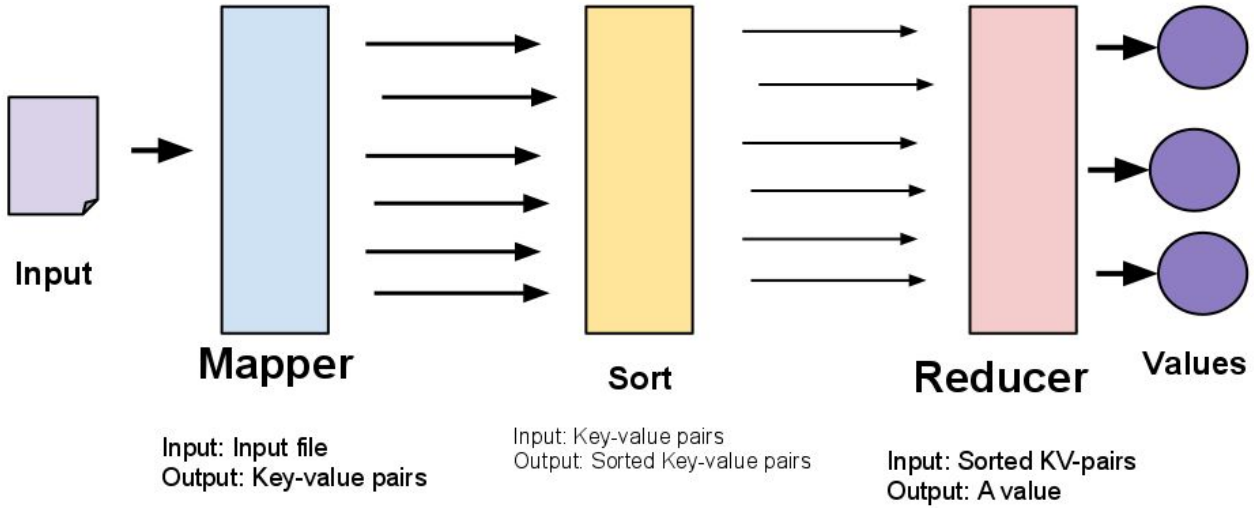
MapReduce Benefit

- MapReduce can handle everything in a truly distributed fashion
- Meaning, all you have to do is write the application code and the framework will do the load balancing, failure recapturing, scheduling, and network partitions should they arise
- One drawback is that it limits the applications we can write, but if your use case has a MapReduce analog, then you are in the clear.

Before MapReduce, Questions?

Introduce a new paradigm

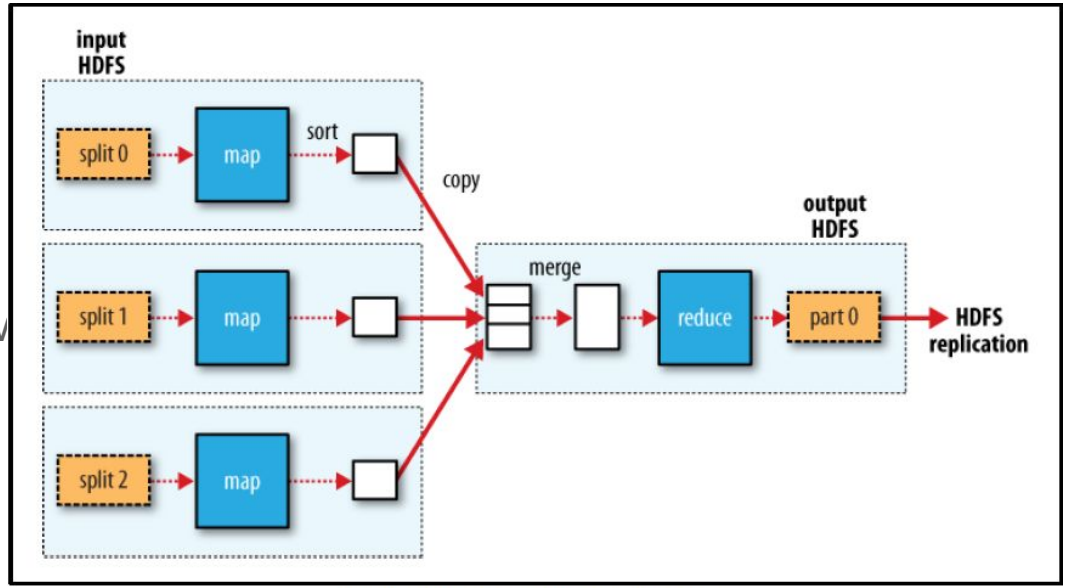
- There are ways to limit all of this, using the mapreduce paradigm
- Like you saw last week, we are going to limit to input pairs. We are going to limit the intermediate output to key/value pairs. The output will be aggregated by keys.



https://liamgavinmurray.files.wordpress.com/2014/12/mapreduce_diag3.png?w=639

Benefits

- We can have the map and reducing on different machines!
- If a machine fails, one can retrace the steps and restart that part of the process
- We can run this in master Slave, or we can make this “Truly distributed”
- We can schedule tasks To reduce network traffic Because the network is slow
- Avoid side effects



MapReduce data flow with a single reduce task

http://upload-images.jianshu.io/upload_images/563632-3cf142bf4c818bab.png?imageMogr2/auto-orient/strip%7CimageView2/2/w/1240/q/100

Drawbacks

- It’s slooow for small tasks - worse than non-distributed (Can you think why?)
- When the data is not sequential, MapReduce induces lots of disk reads which slows down the computation.
- No levels of abstraction, meaning you have to hand code everything
- Batch processing, meaning that you have to wait for the task to be done before you can interpret the results. If you forget to include some output information, you have to re-run the job which could cost thousands.

Hadoop/Zookeeper

An Aside: Zookeeper

- Zookeeper is a *coordination service*. Meaning it takes a bunch of computers and let's them communicate and agree on information. What does that mean?
- You can create "nodes". One can connect to these nodes through a path (a zookeeper path). Then, one can watch that node for changes or write to the node if a change needs to occur.
- Seems pretty simple, but it allows you to build some pretty powerful services off of it. Hadoop will mainly be using it for "leader election"

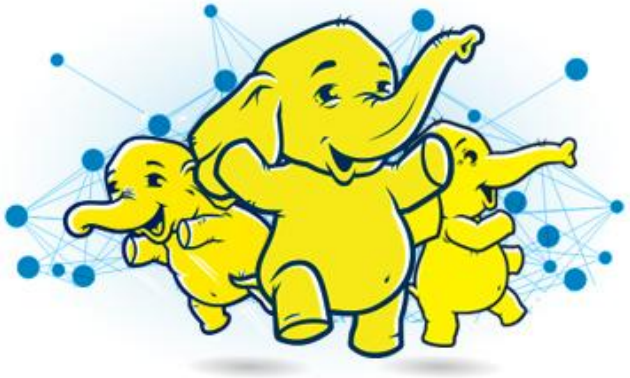
https://zookeeper.apache.org/images/zookeeper_small.gif

What That Means

- Hadoop is going to be using a hybrid model. Instead of having a static master, we are going to dynamically elect a master and if it goes down
- This means that Zookeeper is going to start up and Hadoop is going to elect a leader, which will coordinate reads and writes. If the leader fails at any time, a new leader can be chosen.
- “But what if the leader and the node that is slated to be the leader fail at the same time?”. Hadoop has you covered
- https://wiki.apache.org/hadoop/ZooKeeper/FailureScenarios

Hadoooop!

- So, you don’t want to write all that other code right?
- The most popular implementation of MapReduce Framework is Hadoop’s MR
- Hadoop also serves as a distributed file system, meaning that it can store very large files over multiple computers
- We usually load our data into hadoop through a command line utility, web interface, or program, and then process the data



http://bigdataanalyticsnews.com/wp-content/uploads/2014/09/Hadoop_elephants.jpg

Writing a Job

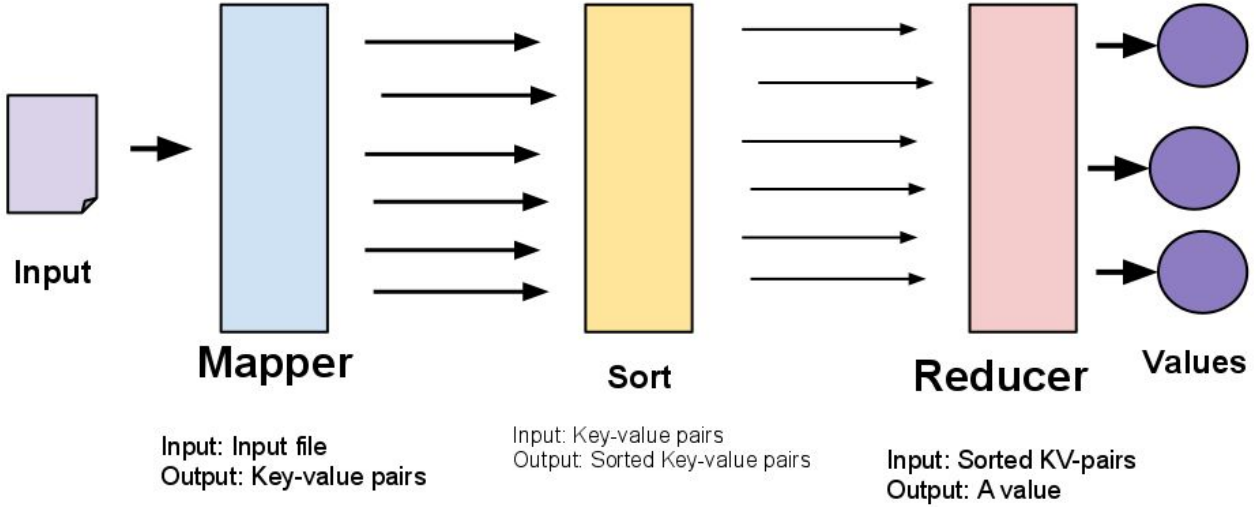
- To write a hadoop job, you usually specify a java class that has instructions for what to do when you get the input file, and what to do for the intermediate keys, including outputting the file.
- Lots of yucky boilerplate
- Ties the code to the file format, so if you need to change file formats, you have to compile (you tie your mapper to your reducer essentially)
- https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html
- For those interested

We aren’t going to do that!

- We are going to use Hadoop’s MR streaming
- We will read from STDIN and write to STDOUT. We will have a mapper program and a reducer program which solves the coupling problem
- This does make the implementation a bit slower because we are relying on the OS to make reads and writes fast instead of Hadoop handling it itself.
- But, it makes the code simpler!

One more thing - Hadoop Sorts

- After the map operation completes, hadoop will sort the map outputs by their values
- Why does it sort?



https://liamgavinmurray.files.wordpress.com/2014/12/mapreduce_diag3.png?w=639

Mapper Example

```
import sys
# input comes from STDIN (standard input)
for line in sys.stdin:
    # remove leading and trailing whitespace
    line = line.strip()
    # split the line into words
    words = line.split()
    # increase counters
    for word in words:
        # tab-delimited; the trivial word count is 1
        print("{}\t{}".format(word, 1))
```

Reducer Example

```
current_word, word = None, None
current_count = 0
for line in sys.stdin:
    word, count = line.strip().split('\t', 1)
    if current_word == word: # Why does this line work?
        current_count += count
    else:
        if current_word: # Initialize current_word
            print('{}\t{}'.format(current_word, current_count))
        current_count = count
        current_word = word
```

Hadoop Extras

- Hadoop is also a distributed file system; we can store files across multiple computers
- Hadoop has many native extensions for languages
- There are also service-level extensions that run on Hadoop like Hive or Pig
- Hadoop also has rebalancing algorithms, fault tolerance, and partition tolerance through things like replication down pat (though this can break sometimes).

Hadoop Drawbacks

- You have to manage your own cluster
- Things can break in very technical ways.
  - If one data node goes down, the cluster rebalances. But since the cluster is rebalancing, the machines are under a higher load. This means that they are more prone to failure
  - If there is a network partition (meaning that the computers get split up into two separate groups) then zookeeper will do the right thing, but hadoop may not
  - https://issues.apache.org/jira/browse/MAPREDUCE-1800. (plenty of problems that aren't getting addressed in the open source community)
- Lots of legacy (compatibility) constraints

Working on the Cluster

HDFS

- Hadoop provides a CLI interface to the HDFS, but it’s not pretty
- Commands are of the form:
  - hdfs dfs -<command> <arguments>
  - Example: hdfs dfs -ls /shared/myFolder
- Copying to/from HDFS:
  - Use -copyFromLocal to copy from the normal filesystem to HDFS
  - Use -copyToLocal to copy from HDFS to the normal filesystem
- The easy way (for small accesses): HDFS NFS Gateway
  - NFS Gateway allows HDFS to be mounted similar to a remote drive
  - Good for exploring directories, and looking at (small) output of jobs
  - **Slow** for large data transfers. (Use HDFS CLI directly)

Hadoop Jobs

- Input/Output **must** reside in HDFS
- Program code can live in the normal filesystem
  - Programs can be copied to all nodes upon job startup, because they’re small
- Submitting a streaming job to Hadoop:

```
$ hadoop jar $STREAMING_JAR \
    -files mapper.py,reducer.py
    -mapper mapper.py -reducer reducer.py \
    -input /shared/my_cool_dataset -output my_job_out
```

Hadoop Jobs

- We made things a bit easier for you:

```
$ mapreduce_streaming mapper.py reducer.py \
    /shared/my_cool_dataset my_job_out
```

MP 2

Due in one week (9/20) at 11:55pm

Introduces how to run MapReduce using on the cluster

> Office Hours!

> Check Piazza for Q&A and Announcements

Warning!

- We **don’t guarantee** the cluster uptime
- Even though Hadoop is scalable, reliable, and fault tolerant (all those buzzwords), imagine what would happen if all thirty of you tried to log on to the cluster and submit a huge mapreduce job at the deadline
  - Either the cluster crashes or it runs at a snail’s pace.
- As with course policy, if it’s late it is late.

Attendance

bit.ly/199Attendance0