Hadoop

Introduction to Distributed Systems and MapReduce

Cary Goltermann

Galvanize

2016

Overview

Big Data

- What is it and Why is it Important?

Distributed Systems

- Distributed Filesystems and Processing
- Distributed Systems Architecture

Hadoop

- HDFS
- MapReduce

Overview

Big Data

- What is it and Why is it Important?

Distributed Systems

- Distributed Filesystems and Processing
- Distributed Systems Architecture

Hadoop

- HDFS
- MapReduce

The Three Vs

"Big data is **high volume**, **high velocity**, and/or **high variety** information assets that require new forms of processing to enable enhanced decision making, insight discovery and process optimization." — Gartner Inc.

3 Vs of Big Data

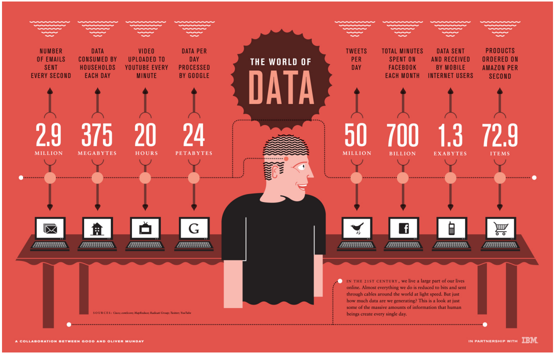
High Volume Data so large that it can't be worked with on a single computer.

High Velocity Data input/output too quick for it to be processed by a single computer.

High Variety Data in many different, disparately or not at all, structured formats. E.g. text, log file, video, etc.

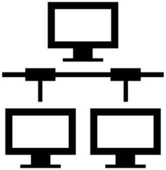
How Big is BIG?

Class	Size	Tools	Storage	Examples
Small	$< 10GB$	R / Python	Fits in a single machine's memory	Thousands of sales figures
Medium	$10GB - 1TB$	Python w/ indexed files, large database	Fits on a single machine's disk	Millions of web pages
Large	$> 1TB$	Hadoop, Spark, distributed databases	Stored across multiple machines	Billion of web clicks



- As more data is generated, and inevitably collected, the size of data you'll likely work with is going to increase.
- Tools to store and process these larger data sets are going to be increasingly required.

Local vs. Distributed



Local Use resources of one machine. Does not need to communicate with any others.

Distributed Uses the resources, processing and memory, of multiple machines. However, they need to be able to communicate with one another.

Overview

Big Data

- What is it and Why is it Important?

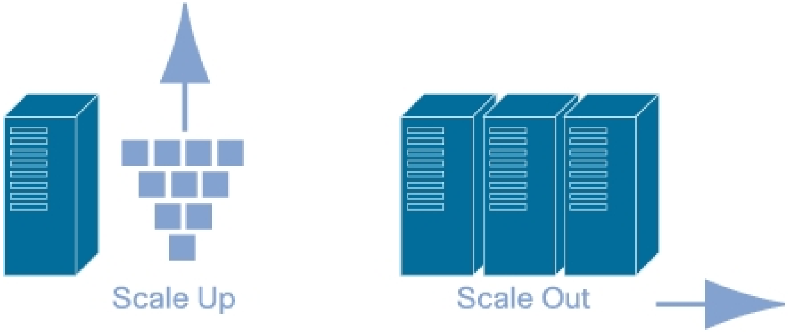
Distributed Systems

- Distributed Filesystems and Processing
- Distributed Systems Architecture

Hadoop

- HDFS
- MapReduce

Scaling



Make the computer bigger:
disk, RAM, CPU cores.

Add more computers:
take advantage of parallelizable
algorithms.

Local vs. Distributed: Pros & Cons

Local

Pros Simple, fast when computations are small enough.

Cons Physical limits to memory, disk space and CPU power.

Distributed

Pros Easily linearly scalable, can designed to be fault-tolerant.

Cons Slow to communicate over network, need to solve problems with parallelizable techniques.

Should I Use Distributed Computing Solutions?

- Because of the overhead involved with having multiple computers in a network communicate with each other, and the restrained set of problems that can readily be solved in a parallelizable way, it's not a good enough reason to choose a distributed solution just because things are "taking awhile" on your local machine.
- However, if you'd like to get practice with the Big Data tools so that you feel comfortable with them and can have them on your resume, then it may be worthwhile to use these solutions, even on data this isn't "Big".

Overview

Big Data

- What is it and Why is it Important?

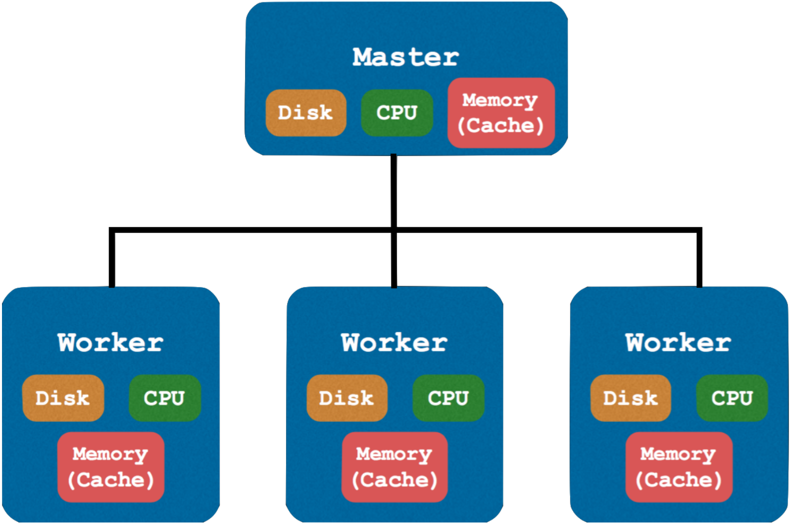
Distributed Systems

- Distributed Filesystems and Processing
- Distributed Systems Architecture

Hadoop

- HDFS
- MapReduce

High Level Architecture



Overview

Big Data

- What is it and Why is it Important?

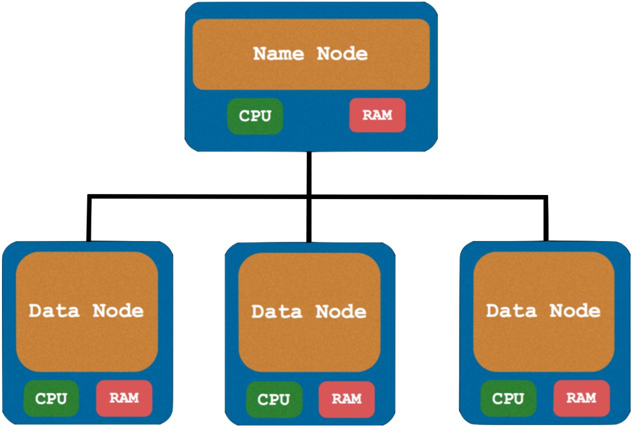
Distributed Systems

- Distributed Filesystems and Processing
- Distributed Systems Architecture

Hadoop

- HDFS
- MapReduce

Hadoop Distributed File System Architecture



Data nodes store the data.

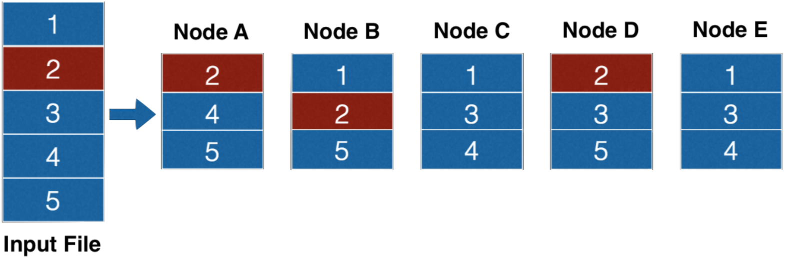
The name node keeps track of where the data is stored.

Fault-tolerance Through Replication

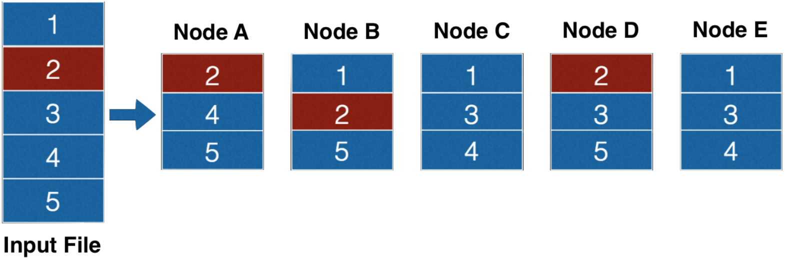
- Each file is broken up into blocks, default size is 64/128 MB.
- Each of these blocks is replicated on 3, by default, of the data nodes in the cluster.

Fault-tolerance Through Replication

- Each file is broken up into blocks, default size is 64/128 MB.
- Each of these blocks is replicated on 3, by default, of the data nodes in the cluster.

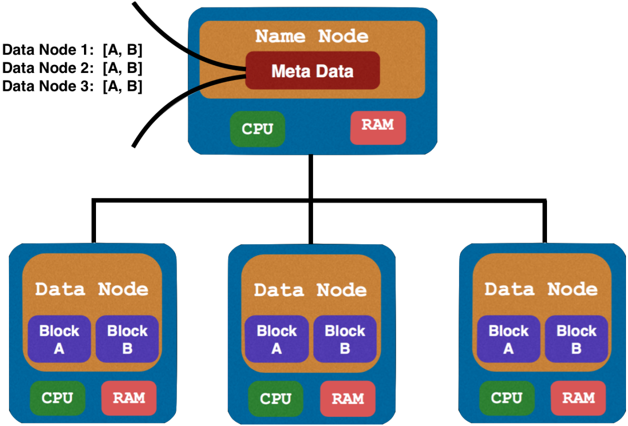


Question



How many nodes can be lost before the original file isn't recoverable?

HDFS



Since the name node knows about where all the copies of each block is it can automatically make new ones if some are lost.

Overview

Big Data

- What is it and Why is it Important?

Distributed Systems

- Distributed Filesystems and Processing
- Distributed Systems Architecture

Hadoop

- HDFS
- MapReduce

Processing in Hadoop

If HDFS is the storage in our distributed system, then how are would we design a means by which we can process all of our distributed data?

Processing in Hadoop

If HDFS is the storage in our distributed system, then how are would we design a means by which we can process all of our distributed data?

We can move the computation portion of our endeavors to the computer that is storing each part of our data in HDFS.

Processing in Hadoop

If HDFS is the storage in our distributed system, then how are would we design a means by which we can process all of our distributed data?

We can move the computation portion of our endeavors to the computer that is storing each part of our data in HDFS.

This requires us to have a processing framework which can natively work on data that isn't all stored in the same location.

Enter MapReduce (duce...duce...duce).

Advantages of MapReduce

There are a lot of details that need to be taken care of when doing distributed processing:

- Splitting up data
- Moving data between nodes
- Managing resources, computational and memory
- Status and monitoring
- Fault-tolerance

Advantages of MapReduce

There are a lot of details that need to be taken care of when doing distributed processing:

- Splitting up data
- Moving data between nodes
- Managing resources, computational and memory
- Status and monitoring
- Fault-tolerance

MapReduce is automatically going to take care almost all the complications associated with these. All we have to do is play by its rules.

MapReduce Strategy

The intuition of the MapReduce framework boils down to divide and conquer.

1. Split a task into smaller subtasks.
2. Solve these independently of one another (in parallel).
3. Recombine the output of each subtask into a final result.

The Real Map & Reduce

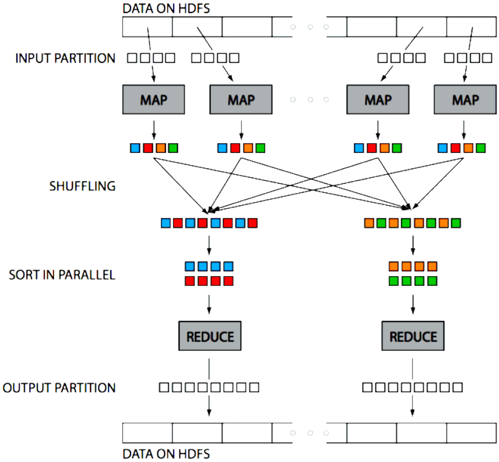
Mapping and reducing are concepts that belong to the functional programming paradigm. They are composed of the:

Map Applies a function to each of the elements of a data structure.

Reduce Takes a function which aggregates the elements of a data structure.

This strategy is particularly useful in distributed computing because the elements of the data structure referenced in the map step don't need to be on the same machine, they can be the partitions of a file that live on different data nodes.

MapReduce Diagram



Map Step

Mapping is simply the action of taking in some form of data and filter/transforming it into another form. As mapping step should operate on a single element of our data and output 0 or more possibly transformed versions of that data.

Map Step

Mapping is simply the action of taking in some form of data and filter/transforming it into another form. As mapping step should operate on a single element of our data and output 0 or more possibly transformed versions of that data.

Example

Takes in lines from a click through log file and outputs a tuple of the user id and the page id they clicked on from weekdays.

Map Step

Mapping is simply the action of taking in some form of data and filter/transforming it into another form. As mapping step should operate on a single element of our data and output 0 or more possibly transformed versions of that data.

Example

Takes in lines from a click through log file and outputs a tuple of the user id and the page id they clicked on from weekdays.

By default the "elements" that will be passed as single data points from your input partitions to your mapping function in Hadoop are lines from a file.

Reduce Step

Reducing is the act of taking a bunch of grouped data and combining it in some way. This grouped data will be passed to it as key-values pairs where Hadoop will automatically bundle like values by their key into an iterable with all the values associated with that key.

Reduce Step

Reducing is the act of taking a bunch of grouped data and combining it in some way. This grouped data will be passed to it as key-values pairs where Hadoop will automatically bundle like values by their key into an iterable with all the values associated with that key.

Example

Reducer takes in user ids as keys and an iterable of pages ids they've gone to and outputs the page id that a user visits most frequently.

Reduce Step

Reducing is the act of taking a bunch of grouped data and combining it in some way. This grouped data will be passed to it as key-values pairs where Hadoop will automatically bundle like values by their key into an iterable with all the values associated with that key.

Example

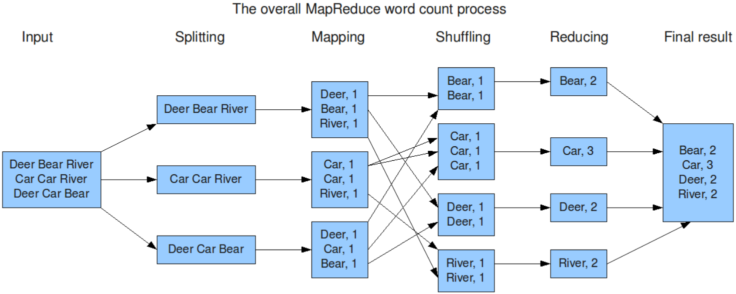
Reducer takes in user ids as keys and an iterable of pages ids they've gone to and outputs the page id that a user visits most frequently.

Frequently the input to our reducers will be coming from a mapper, though this isn't strictly necessary.

Efficiency Note

- At the end of both the mapping and reducing steps the output is written to disk into the HDFS. This can potentially have efficiency ramifications if we are performing many mapping and reducing operations in sequence since writing to disk is time consuming.
- This means that we'll want to condense our mapping and reducing operations, which may make our algorithms hard to understand, or use a different framework, e.g. Spark.

Word Count Example



Word Count Code

wordcounts.py

```python
from mrjob.job import MRJob
from string import punctuation

class MRWordCount(MRJob):

    def mapper(self, _, line):
        for word in line.split():
            yield (word.strip(punctuation).lower(), 1)

    def reducer(self, word, counts):
        yield (word, sum(counts))

if __name__ == '__main__':
    MRWordCount.run()
```

$: *python wordcounts.py file/directory* (> *counts.txt*)