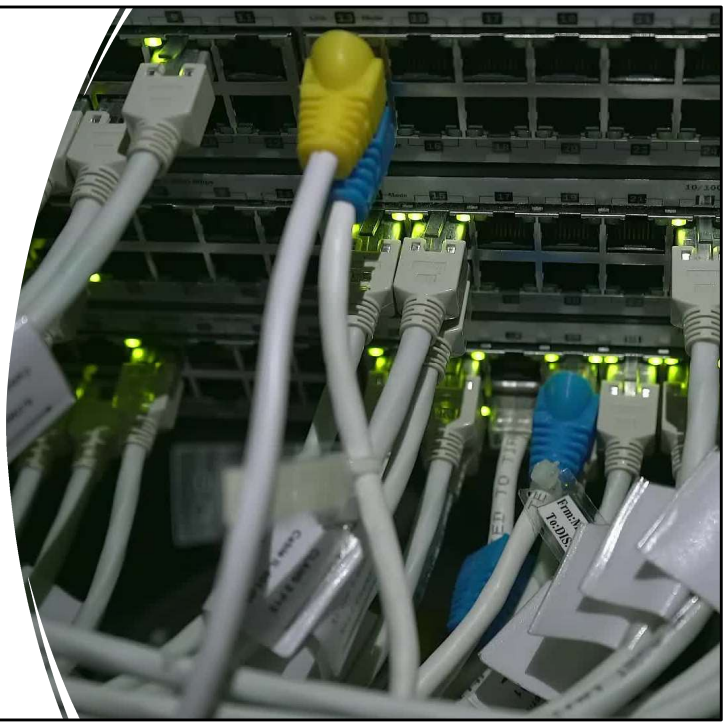
Data-Intensive Distributed Computing CS431/451/631/651

Module 2 - MapReduce



This Module's Agenda

- Computer Clusters
- Distributed Computation (MapReduce)
- Distributed Storage
- Algorithm Design

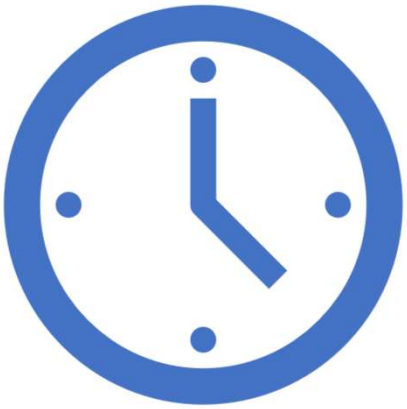
Hello, World?

- Something basic to do with a text file:
- How many times does the word “Waterloo” appear?
- We usually did this as the last tutorial in CS116!
- Read lines, Split lines, count “Waterloo”

Word Count at Scale

Assume HDD: 100MB/s sustained sequential reads

File Size	Load Time
10MB	0.1 seconds
1GB	10 seconds
10GB	1.67 minutes
100GB	16 minutes
10TB	28 hours



28 hours???

- How can we improve that time?
- NVMe Gen4.0 – 7000 MB/s sequential read
- Only 23 minutes now!
- Price / TB = $150 vs $15 for HDD

Not fast enough?

- You can make a RAID of NVMe drives
- You need an enterprise server to have the PCIe lanes for that

Horizontal vs Vertical

SUPER BEEFY SERVER - $200,000

COMMODITY SERVER - $2000

CHEAPER IS BETTER?

HORIZONTAL SCALING

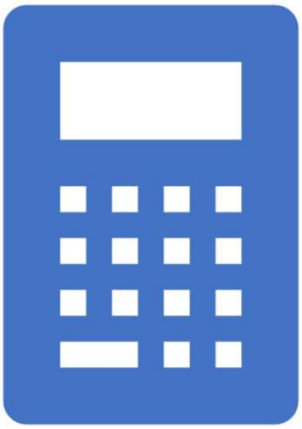
- 100x the servers, 100x the speed?

Hello World x100

Each server loads 1/100th of the file

Each server counts "Waterloo"

Add the 100 totals together



MapReduce

- Two Functions
- Map: Like* Python's / Racket's Map
- Reduce Like* Python's Reduce

* KINDA

Map is actually like map-then-flatten: each call outputs a LIST of things, and these lists get merged together.
Reduce is similarly structured.

People sometimes calls the phases "Classification" and "Aggregation"

Key-Value Pairs

MapReduce is based around Key-Value Pairs
This is a common way to break things down!

If the input is a text file:

Key – Position of a line
Value – Text of a line.

MapReduce

Programmer defines two functions:

$\texttt{map: } (k_1, v_1) \rightarrow \texttt{List}[(k_2, v_2)]$

$\texttt{reduce: } (k_2, \texttt{List}[v_2]) \rightarrow \texttt{List}[(k_3, v_3)]$

Map

Input:

- key : k_1
- value : v_1

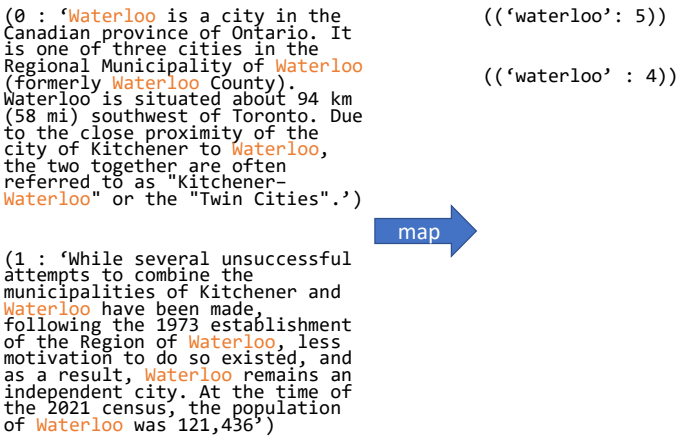
Output:

- List[(k_2, v_2)]

Note: The output key can be different than the input key!

The key will almost always be different, and often will be part of the value! Again, you can think of "Map" being "Classify this value by extracting keys from it"

Map – Counting Waterloo



K1 is an integer, V1 is a string (a line of text). K2 is a string (a word),

Reduce

Input:

- key: k_2
- **ALL** values associated with that key: List[v_2]

Output:

- List[(k_3, v_3)]

Again, the types need not be the same.

K3 is more likely to be the same as K2 here. If you're using Reduce to mean "Aggregate" what you're doing is a fold (sorry, a "reduce") that merges all of the values with the same key into a single value.

BUT, you CAN have different keys, and you can produce multiple key-value pairs here, just like Map can. If you're doing that, you're probably planning more than one MapReduce iterations though! Maybe. It depends.

Reduce – Counting Waterloo

K2=K3, V2=V3. This is doing typical aggregation.

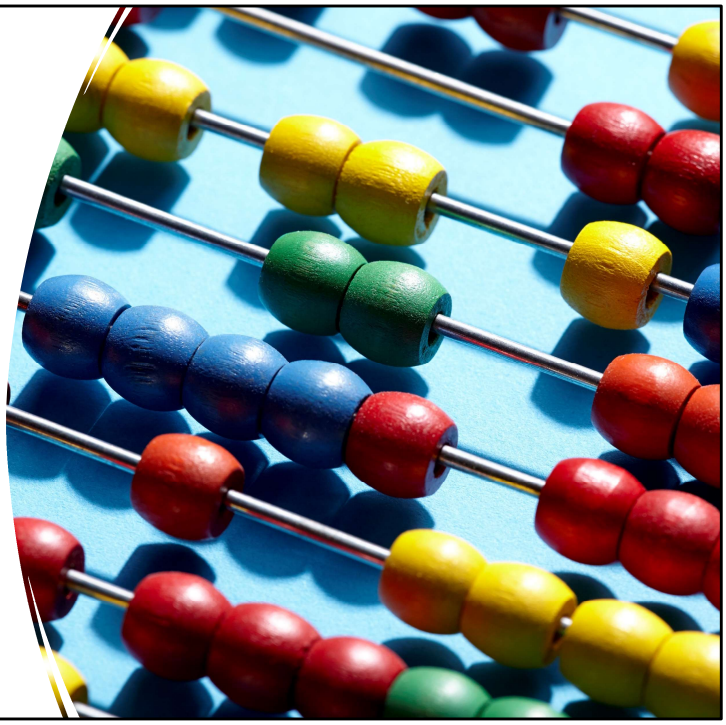
Square Peg, Round Hole?

MapReduce requires a key, even though we only need a single integer (the count)

It's also crazy overkill to use MapReduce to look for just one word's frequency. Let's make it more general shall we?

All Word Counts

- From Counter to Map
- Keys are Words, Values are Counts
- Reducer is now non-trivial
  - (and having a key makes sense)



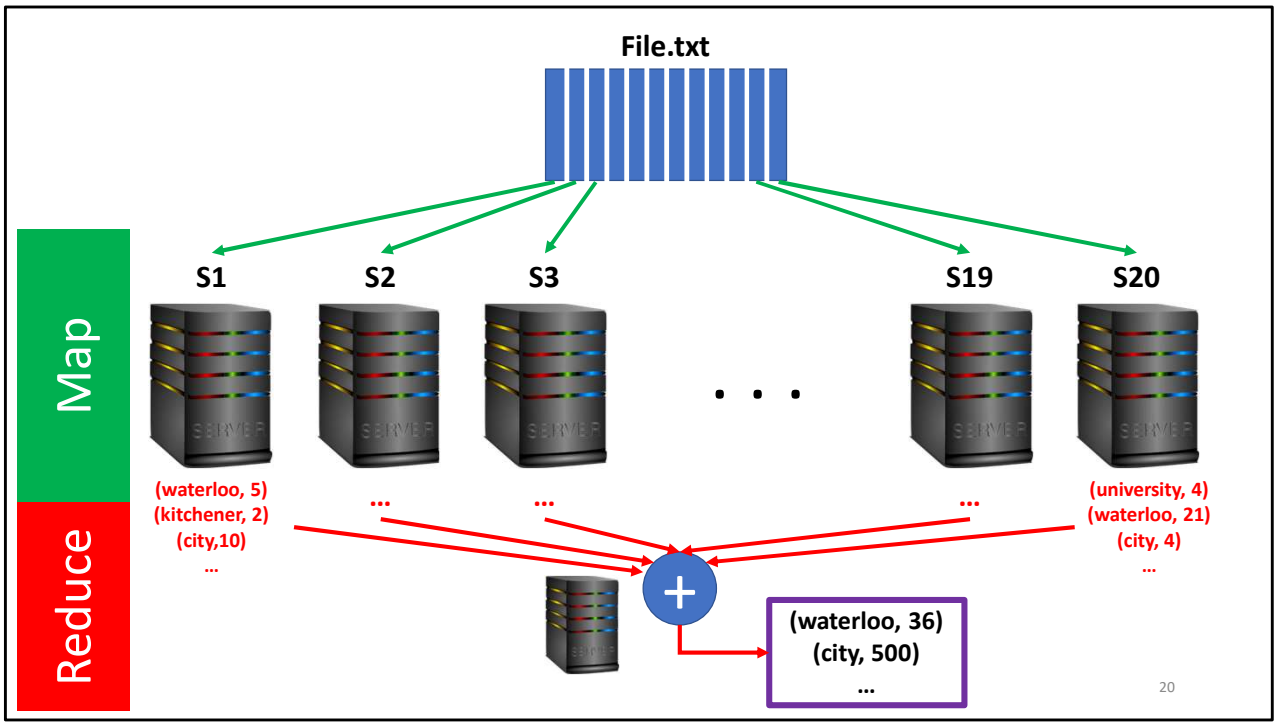
Nothing changes here with the keys and values, we're just going to emit counts for each word, not just one. (Good thing we already used the word as a key!)

The expected output is …

- For each word in the input file, count how many times it appears in the file.

Word	Count
Waterloo	36
Kitchener	27
City	512
Is	12450
The	16700
University	123
…	

19

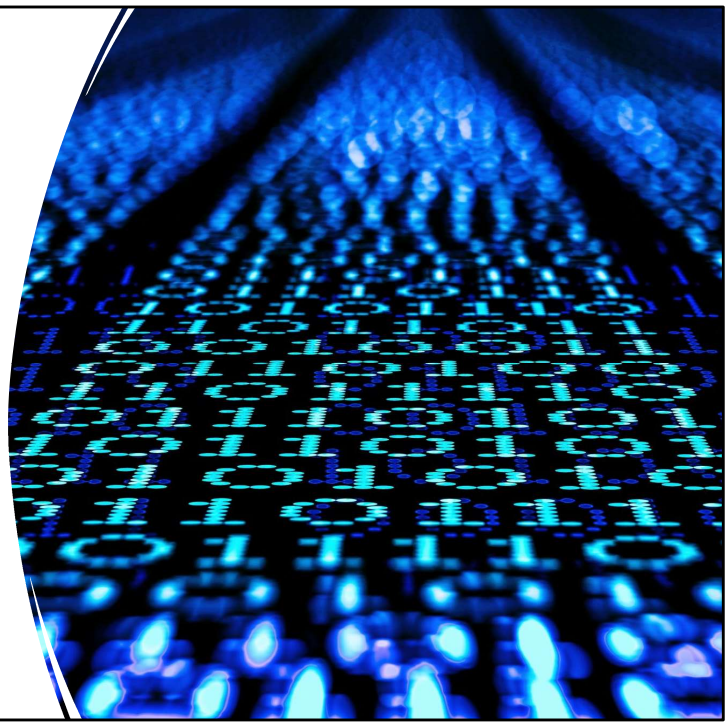


All mappers send list of (key, value) pairs to the reducer, where the key is word and value is its count.
The reducer adds up all intermediate results. But it can now be a bottleneck.

Can we have multiple reducers like mappers?

Memory?

- The Counter used 8 bytes max
- How much does the Dictionary use?
- O(n) if there are n unique words.
- In 10TB of data…what's n?



It's probably small? Thinking at scale means being very careful about every single time you've said "This number is probably small". Assumptions are the enemy here!
IF it's a text file then the lines are "probably" small. This might be a safe assumption even in 10TB of text. But what if there are very long lines?

Map – Counting Waterloo, Alternative

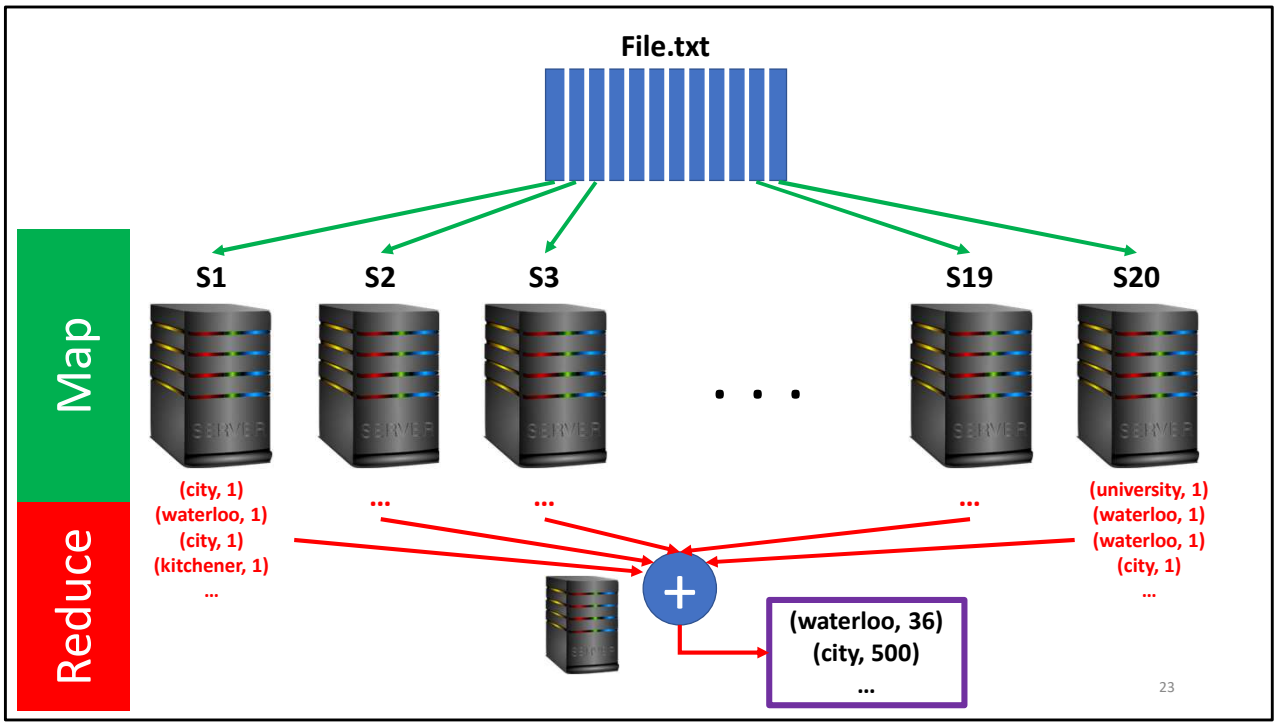
(0 : ‘Waterloo is a city in the Canadian province of Ontario. It is one of three cities in the Regional Municipality of Waterloo (formerly Waterloo County). Waterloo is situated about 94 km (58 mi) southwest of Toronto. Due to the close proximity of the city of Kitchener to Waterloo, the two together are often referred to as "Kitchener–Waterloo" or the "Twin Cities".’)

(1 : ‘While several unsuccessful attempts to combine the municipalities of Kitchener and Waterloo have been made, following the 1973 establishment of the Region of Waterloo, less motivation to do so existed, and as a result, Waterloo remains an independent city. At the time of the 2021 census, the population of Waterloo was 121,436’)

((‘waterloo’: 1), (‘waterloo’: 1), (‘waterloo’: 1) (‘waterloo’: 1), (‘waterloo’: 1))

((‘waterloo’ : 1), (‘waterloo’: 1), (‘waterloo’: 1), (‘waterloo’: 1))

Now we do not need a dictionary for each row, we can emit words as soon as we see them.



All mappers send list of (key, value) pairs to the reducer, where the key is word and value is its count.
The reducer adds up all intermediate results. But it can now be a bottleneck.

Can we have multiple reducers like mappers?

Word Count in MapReduce

```
def map(line):
  for word in line:
    emit(word, 1)

def reduce(key, values):
  sum = 0
  for v in values:
    sum += v
  emit(key, sum)
```

The textbook calls it emit so I'm doing the same. In MapReduce code it's "context.write"

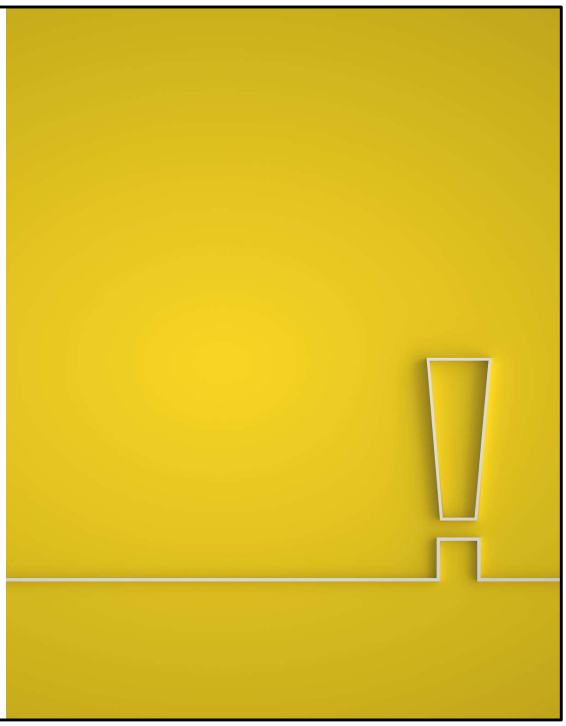
Emit / write means "this is the output of the function" – but it's not returned, it's output asynchronously (i.e. the framework can handle the key-value pair in another thread while your map function continues to process the rest of the line)

Problem

The Reduce server is getting too much data! If the file was 10TB, then more than 10TB will arrive!

Why? “some text” => (some,1) (text,1)

Slightly larger!



Distribution

What if you have multiple reducers?

Each reducer gets ALL pairs for a given Key

MapReduce

Programmer defines ~~two~~ three functions:

$\texttt{map: } (k_1, v_1) \rightarrow \texttt{List}[(k_2, v_2)]$
$\texttt{reduce: } (k_2, \texttt{List}[v_2]) \rightarrow \texttt{List}[(k_3, v_3)]$
$\texttt{partition: } (k_2, v_2, n \in \mathbb{N}) \rightarrow [0, n)$

Partition will default to a hash function that hashes the key and ignores the value

In other words, partition takes a key-value pair plus the number of reducers (n) and assigns it to one of the reducers (which are numbered starting from 0).
Although it is given the value as well as the key, you NORMALLY want to decide based only on the key. Otherwise you're kinda defeating the purpose, which ensuring that each reducer gets ALL values for a given key.

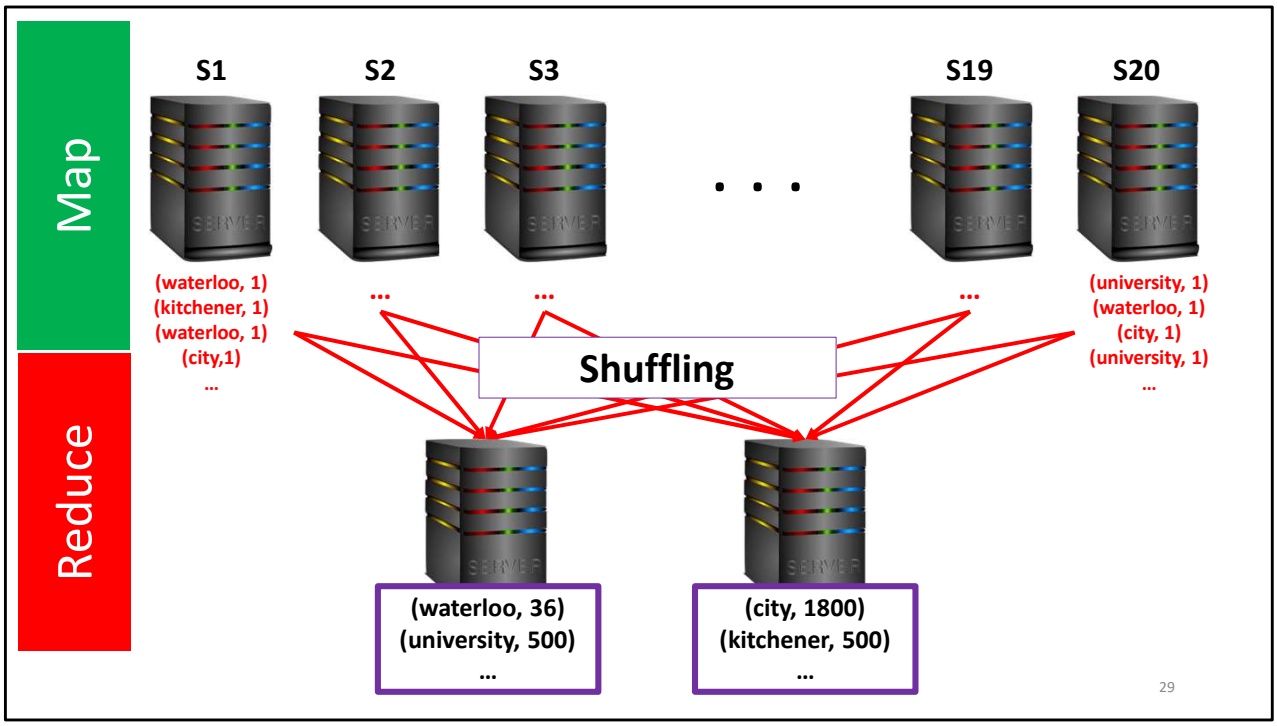
It CAN make sense, if you want to split the key up depending on the values…but there's a better way to do that usually (secondary sort pattern, coming up soon)

Word Count in MapReduce, Less Pseudo, More Code

```
def map(pos : Long, text: String):
  for word in tokenize(text):
    emit(word, 1)

def reduce(key: String, values: Collection of Ints):
  sum = 0
  for v in values:
    sum += v
  emit(key, sum)

def partition(key : String, reducer_count: Nat):
  return hashcode(key) % reducer_count
```



The jargon for "sending KVP to different reducers depending on what partition tells us" is called "shuffling". To me shuffling implies random and the partition is deterministic (that's the ENTIRE POINT) but guess what: nobody asked me what it should be named

Apache Hadoop is the most famous open-source implementation of MapReduce. The logo is an elephant. Probably because they're big, powerful, hard to control, and might rip you in half if angered. Hadoop will probably not do the last one but you never know…CSCF set up datasci for us and there's a lot of configuration options I haven't looked at

MapReduce Implementations

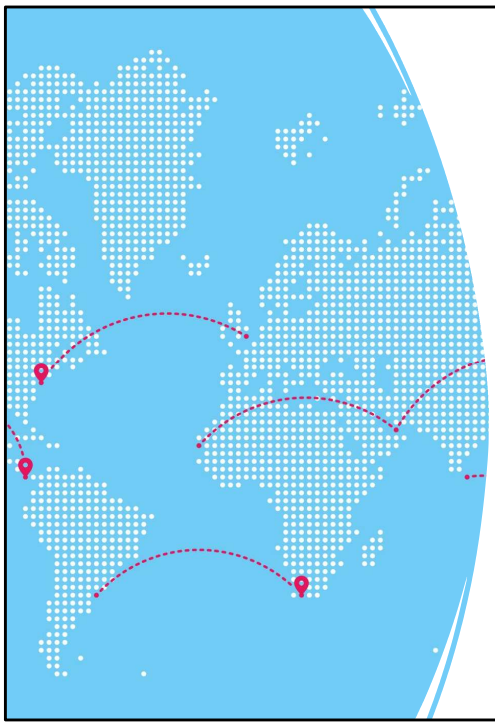
Google has a proprietary implementation in C++

Bindings in Java, Python

Hadoop provides an open-source implementation in Java

Development begun by Yahoo, later an Apache project
Used in production at Facebook, Twitter, LinkedIn, Netflix, …
Large and expanding software ecosystem
Potential point of confusion: Hadoop is more than MapReduce today

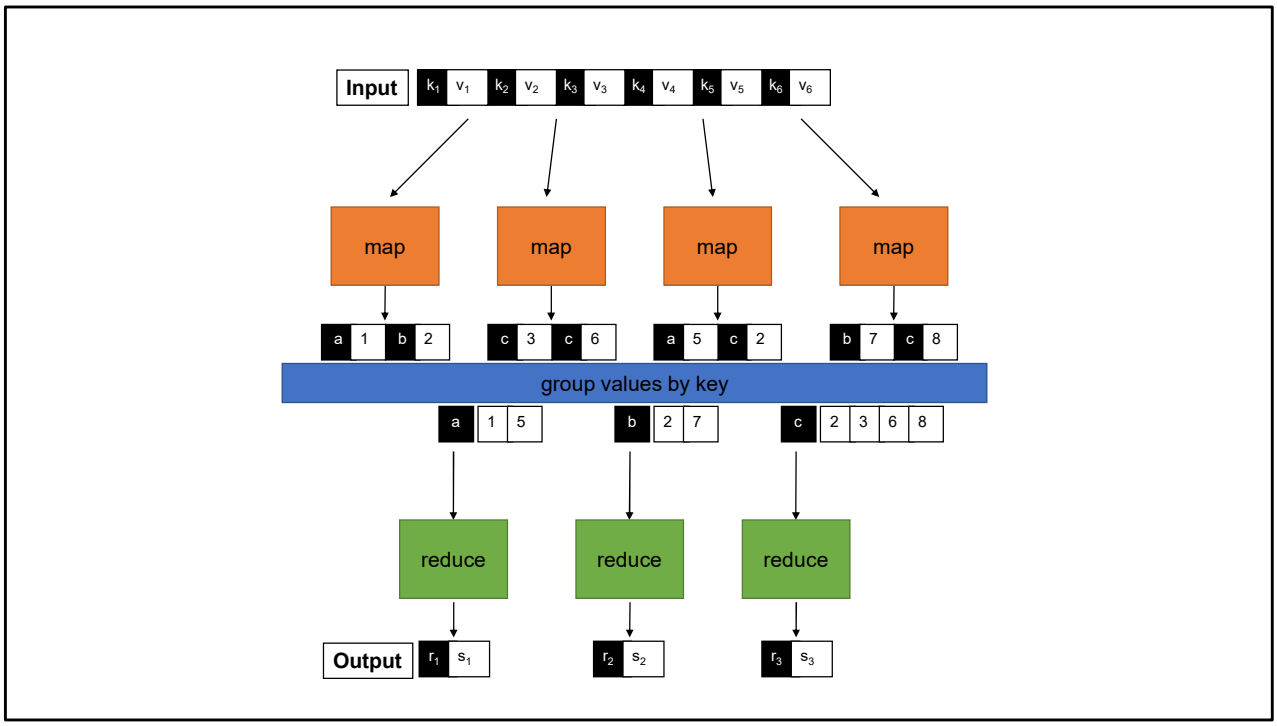
Lots of custom research implementations



Framework

- Assigns workers to map and reduce tasks
- Divides data between map workers*
- Groups intermediate values
  - Sorting pairs by key, determining which pairs go to which reduce worker
- Handles errors
  - What if a worker fails / crashes?

All things the programmer doesn't need to think about…except at a high level, potentially? The * means "there's more to talk about here, lets stick a pin in that" – Lets say the image is relevant, it's Hadoop moving data to workers in different data centers. We want to minimize this!



Q: What's the slowest operation here?
A: sending intermediate results from the mappers to the reducers

Follow-up Q: But not sending the data to the mappers? Why not?
A: Patience, young padawan.

Faster???

- How about only one value per key per mapper?

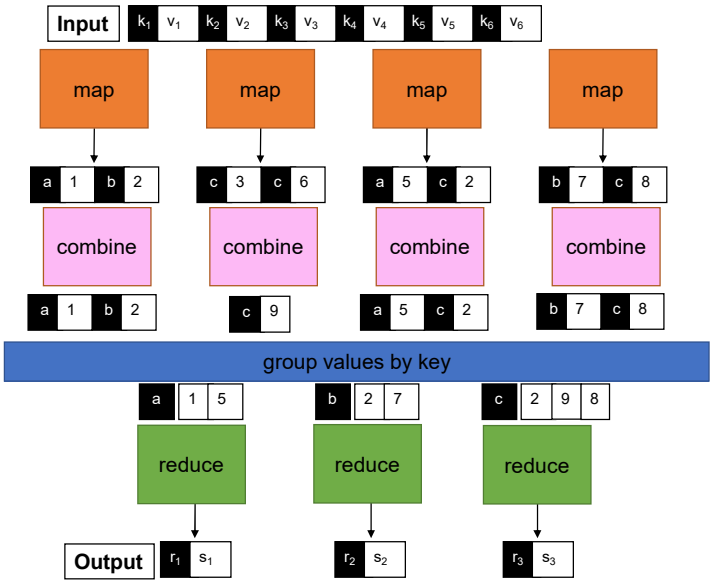
```
def combine(key, values):
  sum = 0
  for v in values:
    sum += v
  emit(key, sum)
```

MapReduce

Programmer defines ~~three~~ four functions:

$map: (k_1, v_1) \rightarrow List[(k_2, v_2)]$
$combine: (k_2, List[v_2]) \rightarrow List[(k_2, v_2)]$
$reduce: (k_2, List[v_2]) \rightarrow List[(k_3, v_3)]$
$partition: (k_2, \mathbb{N}) \rightarrow \mathbb{N}$

Combine is an OPTIONAL thing the mapper / reducer MIGHT do when idle. Note that the signature is the same as reduce, EXCEPT: input and output types are NOT allowed to be different.
Conceptually it should always be producing ONE key-value pair, since the whole point is to combine many values into one for the same key. The signature allows shenanigans and/or malarky. Keep this to a minimum, or avoid entirely.



This is just “liberated” from the textbook, with powerpoint assigning random colours. I had to remove some arrows to make it all fit and I’m very sad about that.

Note: The flow isn’t quite so linear. “combine” happens during map (or doesn’t happen at all) – grouping by values also happens DURING map, and then the values are shuffled to reducers to finish the grouping. The reducers can also use combine when merging intermediate files.

Combine

- Combine MIGHT be the same as reduce
  - **if** $k_2 = k_3$, $v_2 = v_3$ then it would be legal to do
- It also might not
  - Even if legal, it might be inappropriate! Meaning, it runs but gives the wrong answer

Averages

- Combine can't be the same as Reduce
- Why?
  - Mean(2, 3, 4) => 3
  - Mean((Mean(2, 3), 4) => 3.25

We'll circle back to this after a brief detour into all the stars I was putting beside things

Averages

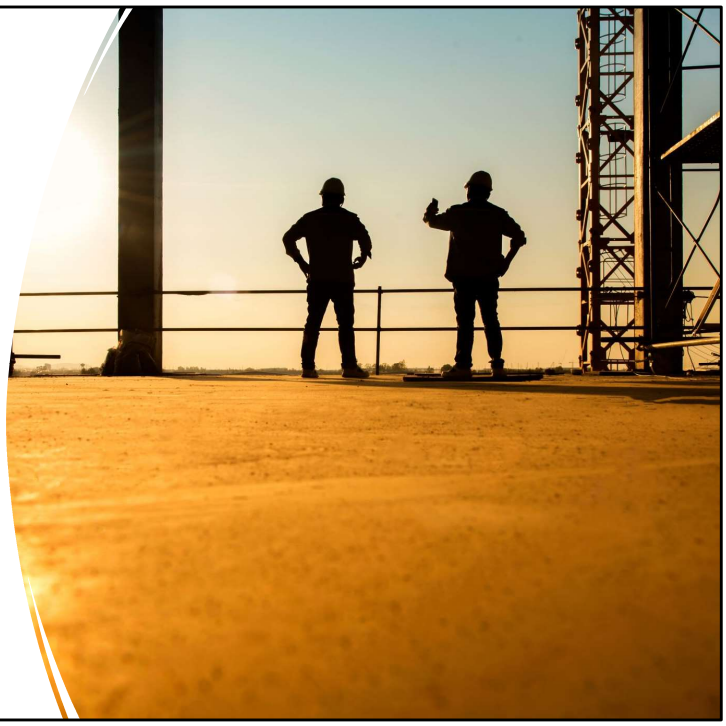
```
def map(k, v):
  emit(k, (v, 1))

def combine(k, vals):
  sum = 0
  count = 0
  for (v in vals):
    sum += v[0]
    count += v[1]
  emit(k, (sum, count))
```

```
def reduce(k, values):
  sum = 0
  count = 0
  for ((s, c) in values):
    sum += s
    count += c
  emit(k, sum / count)
```

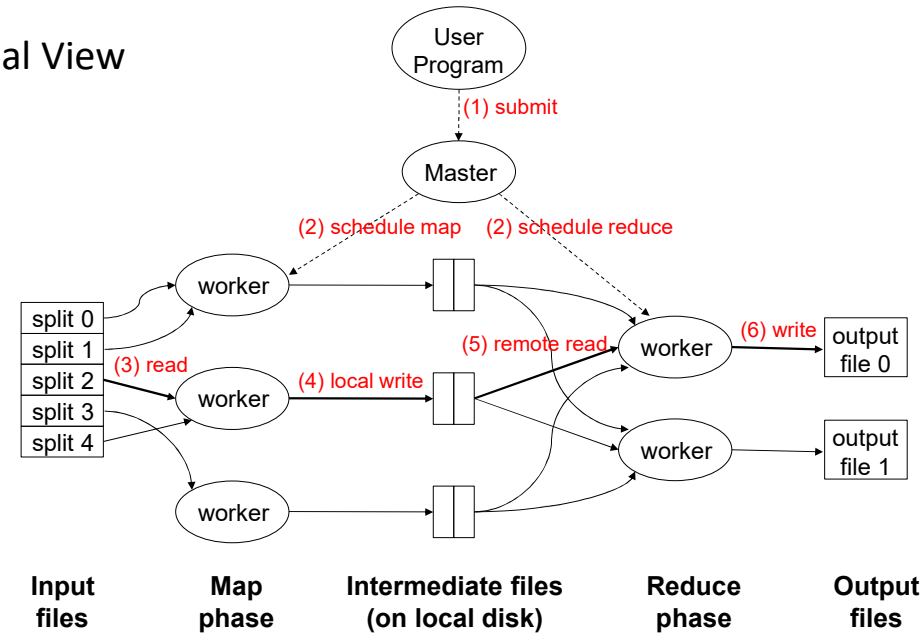
Physical View

What's Hadoop doing behind the scenes?



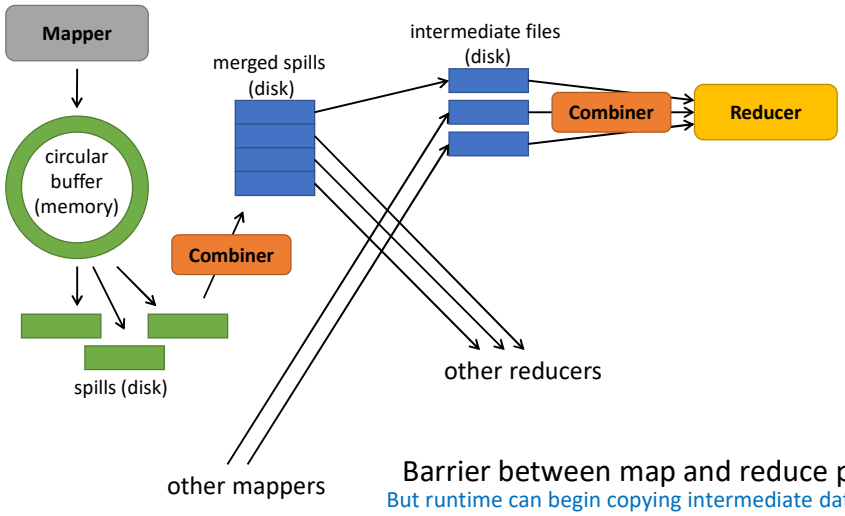
Right: Two worker nodes watching the sun rise while waiting for data

Physical View



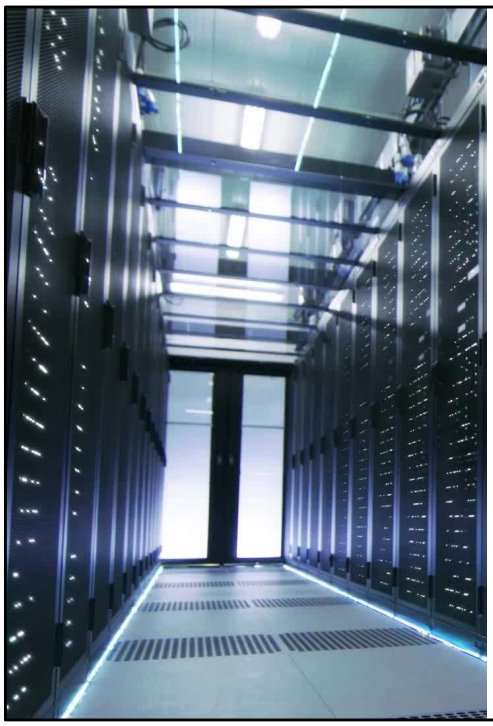
Adapted from (Dean and Ghemawat, OSDI 2004)

Distributed Group By in MapReduce



Map side:
Map outputs are buffered in memory in a circular buffer
When buffer reaches threshold, contents are "spilled" to disk
Spills are merged into a single, partitioned file (sorted within each partition)
Combiner runs during the merges

First, map outputs are copied over to reducer machine
"Sort" is a multi-pass merge of map outputs (happens in memory and on disk)
Combiner runs during the merges
Final merge pass goes directly into reducer

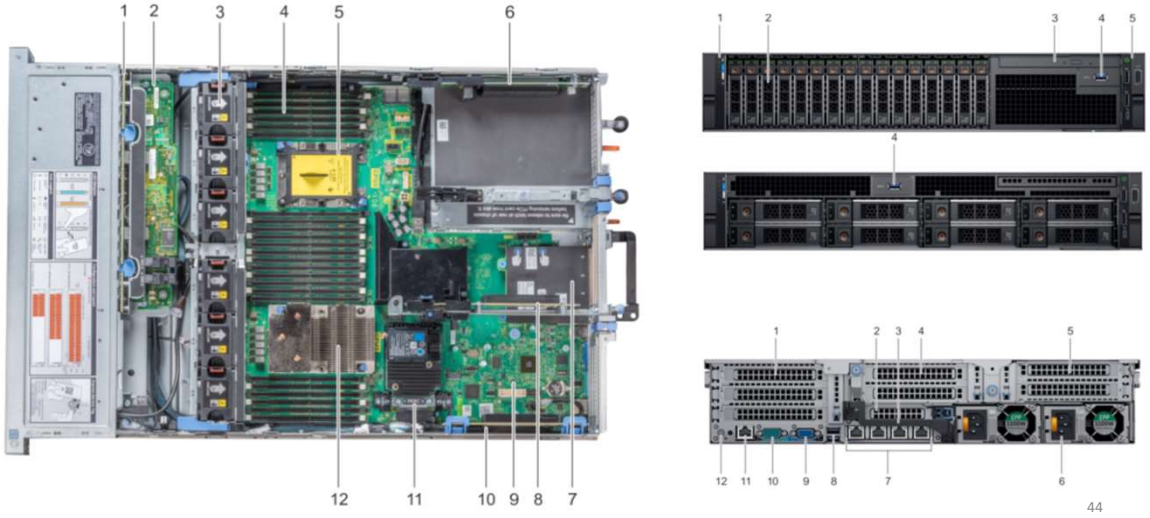


Let's Get (More) Physical

What does a data center really look like?

Really.

The anatomy of a server

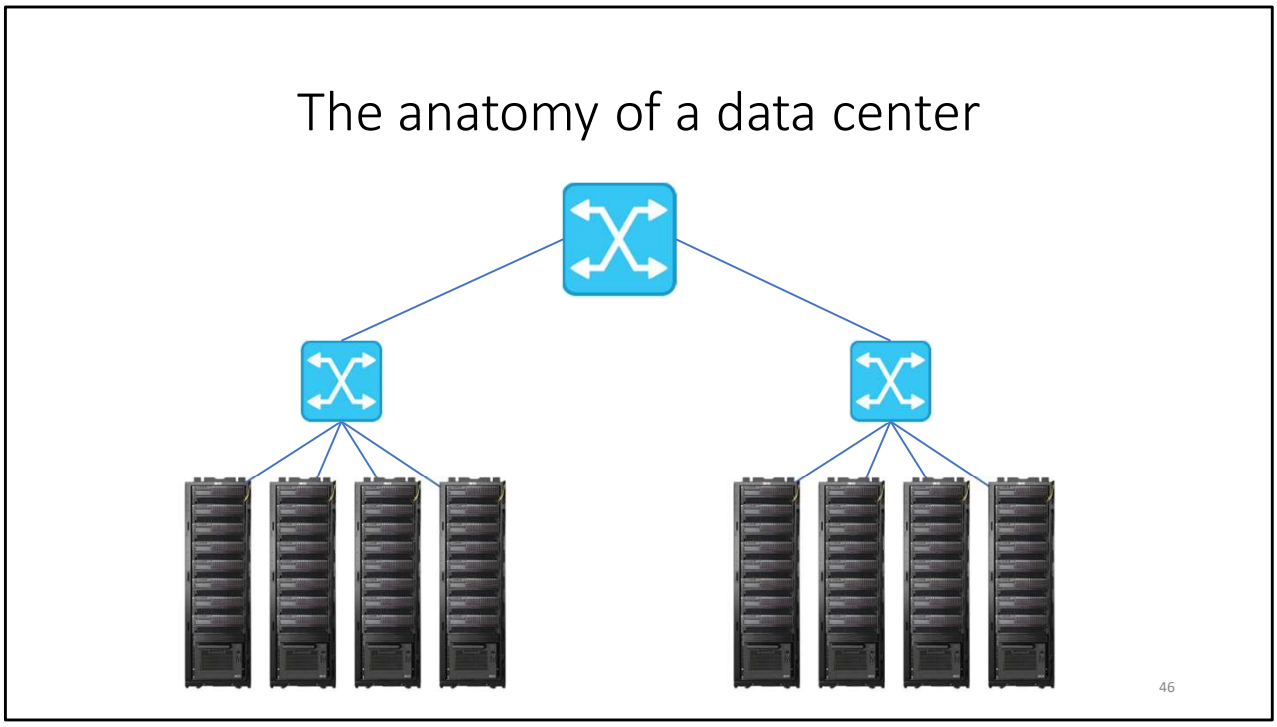


Left: Top view of a server
Right: the two top figures are the front of the server with two storage configurations: 1)16 2.5 inch drives 2) 8 3.5 inch drivers
Right: bottom is the back of the server. We can see network interfaces (7)
(11 is a network port too, this is an IPMI port for OOB (out-of-band) management)

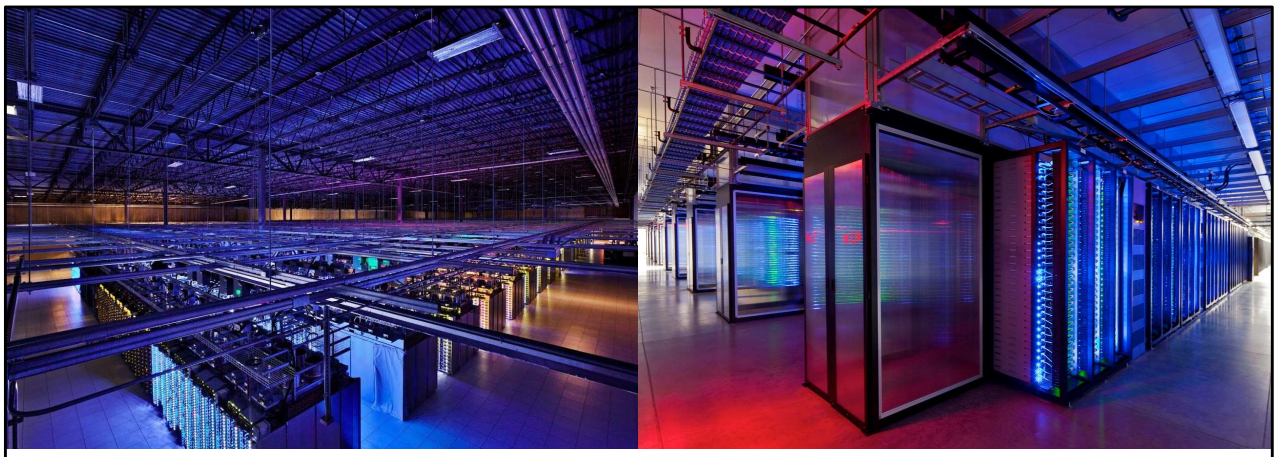
The anatomy of a server rack

45

We put multiple servers in a server rack. There is a network switch that connects the servers in a rack. This switch also connects the rack to other racks.
How Embarassing: this rack clearly has different servers! These look to be 3U chassises in a 5 foot rack. Bottom seems to be a ventilation unit? IDK, I'm not a sysop.



Clusters of racks of servers build a data center. This is a very simplistic view of a data center.

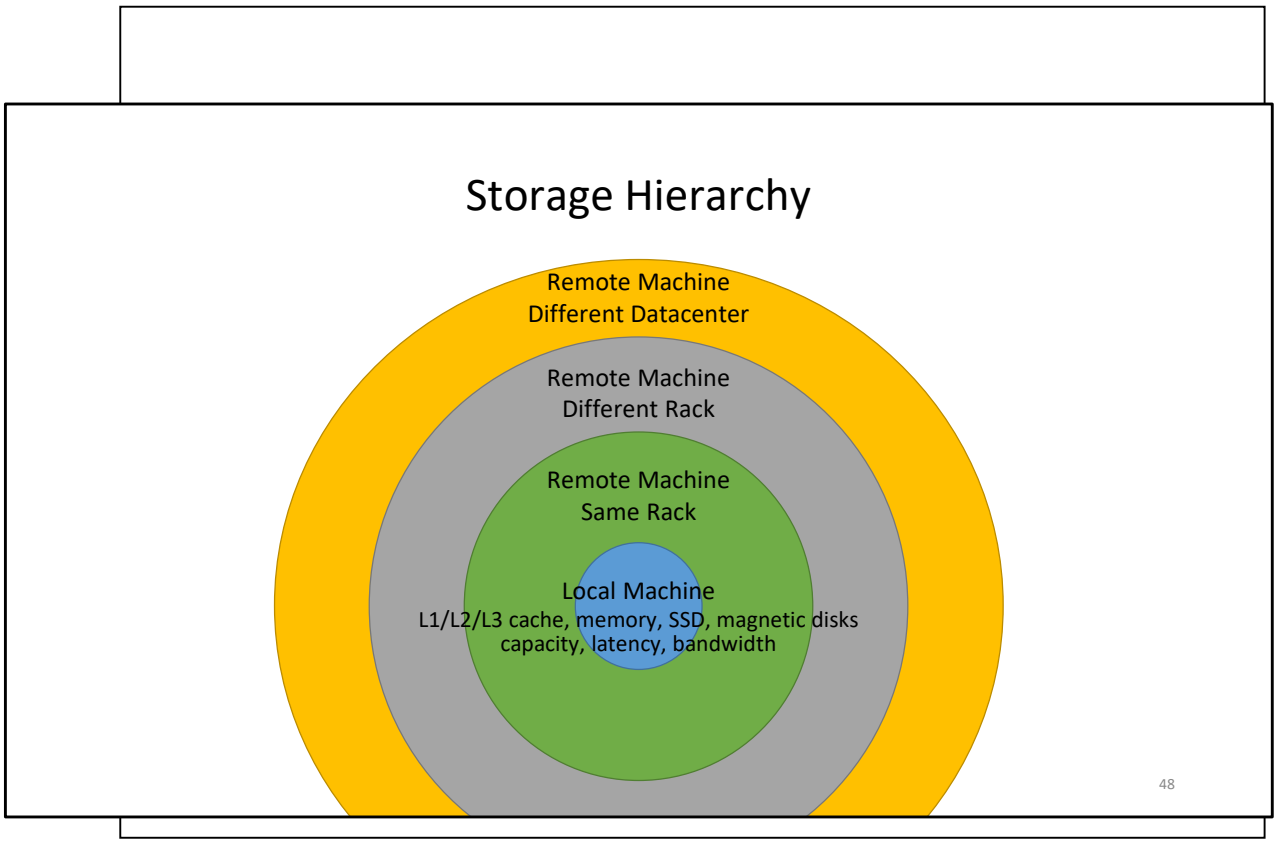


The anatomy of a data center

Google's data center video

47

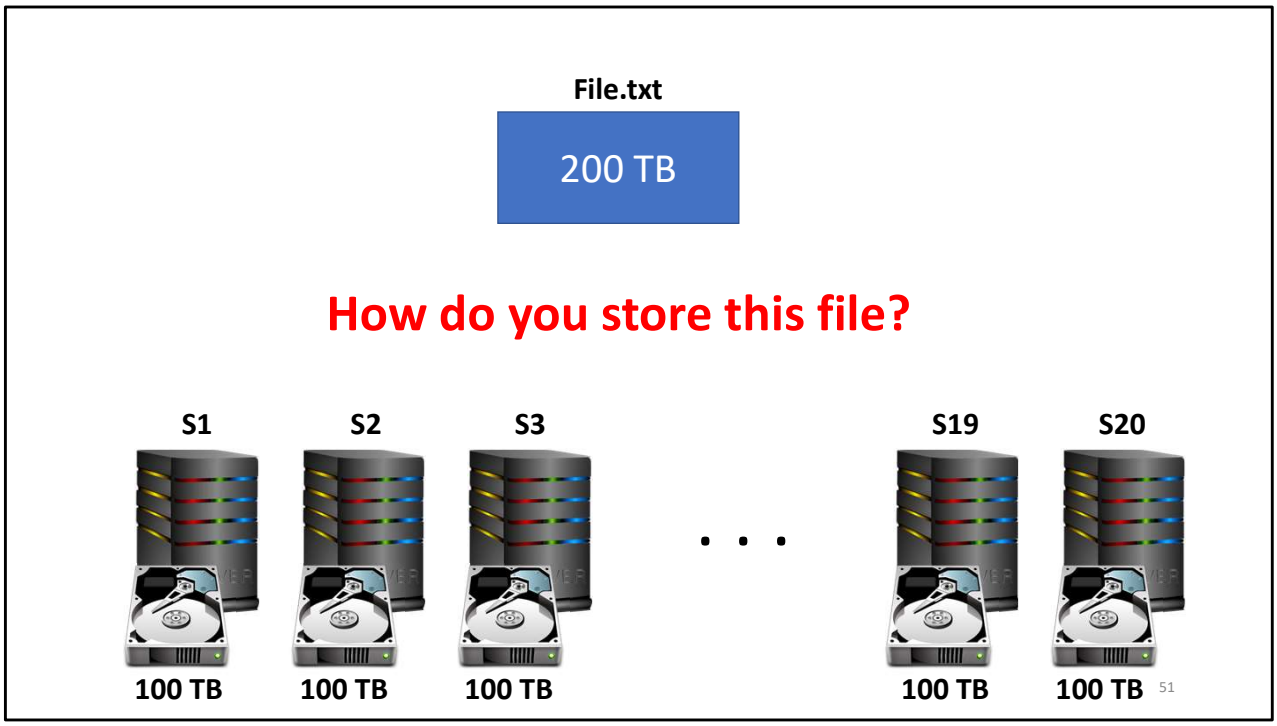
https://youtu.be/XZmGGAbHqa0



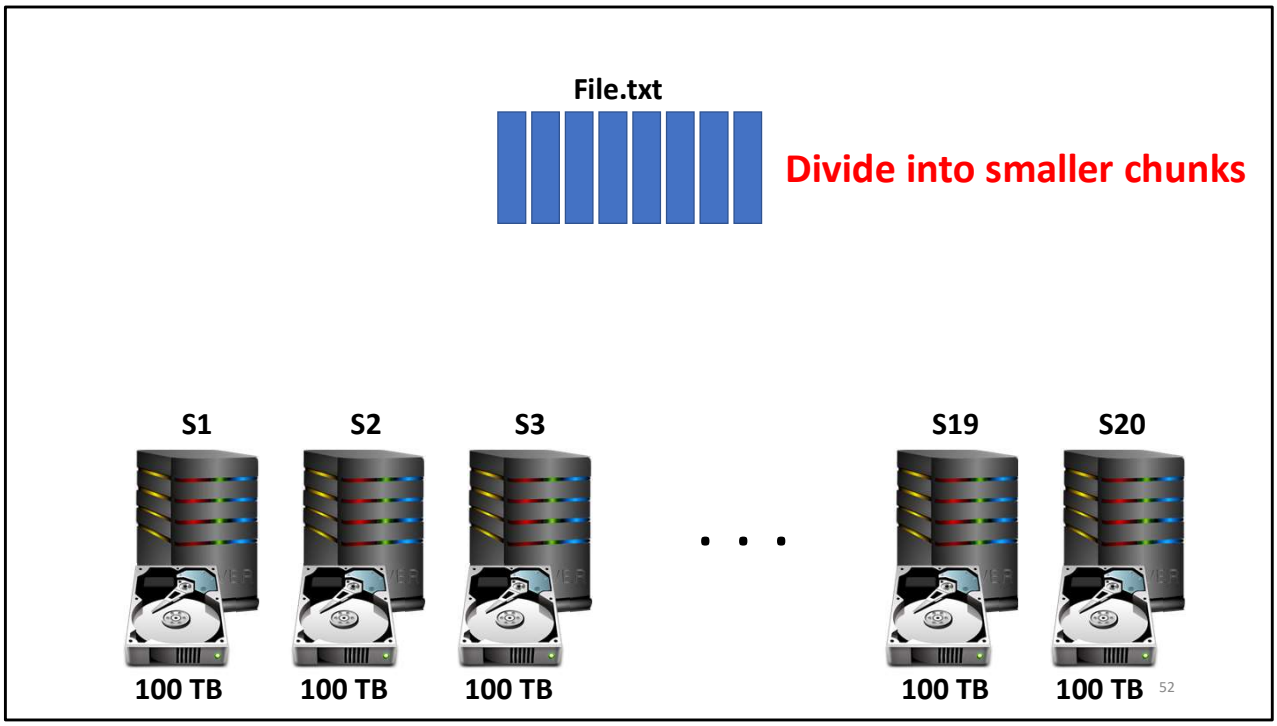
Capacity, latency, and bandwidth for reading data change depending on where the data is. The lowest latency and highest bandwidth is achieved when the data we need is on our local server.
We can increase capacity by utilizing other servers but at the cost of higher latency and lower bandwidth.

https://colin-scott.github.io/personal_website/research/interactive_latency.html

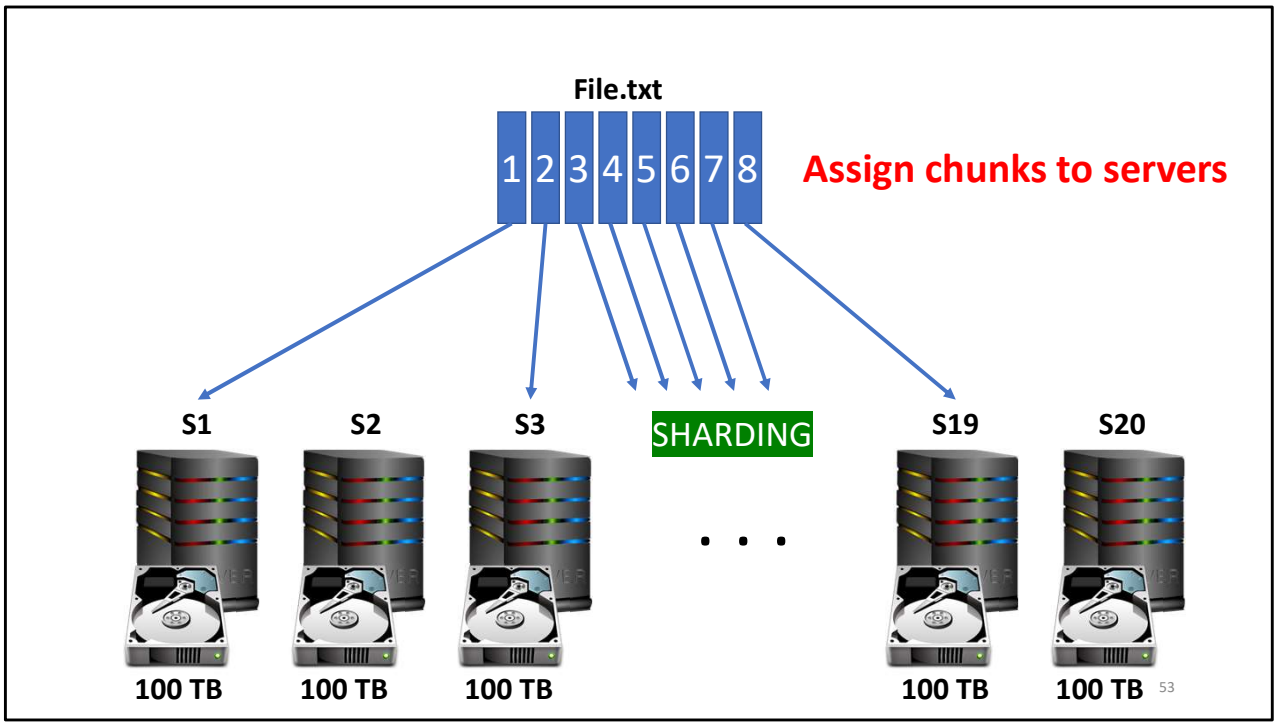

Distributed File System
How can we store a large file on a distributed system?
50



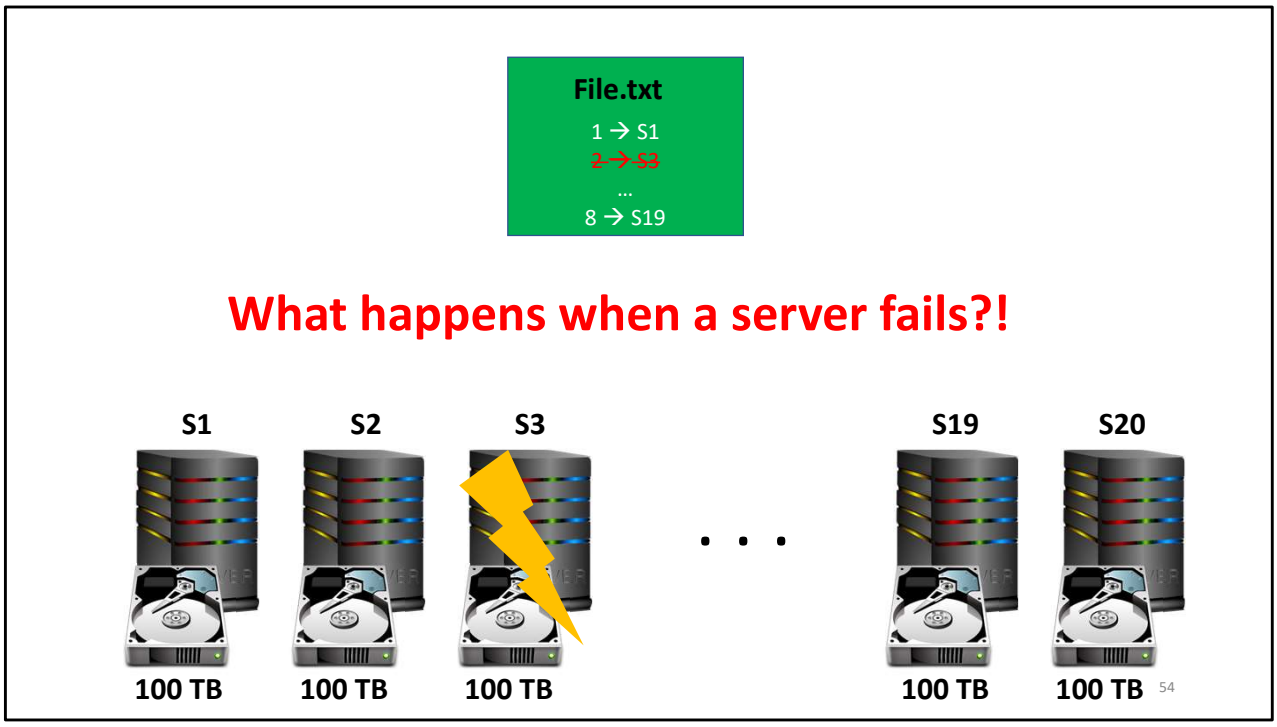
Assume that we have 20 identical networked servers each with 100 TB of disk space. How would you store a file on these server? This is the fundamental question in distributed file systems.



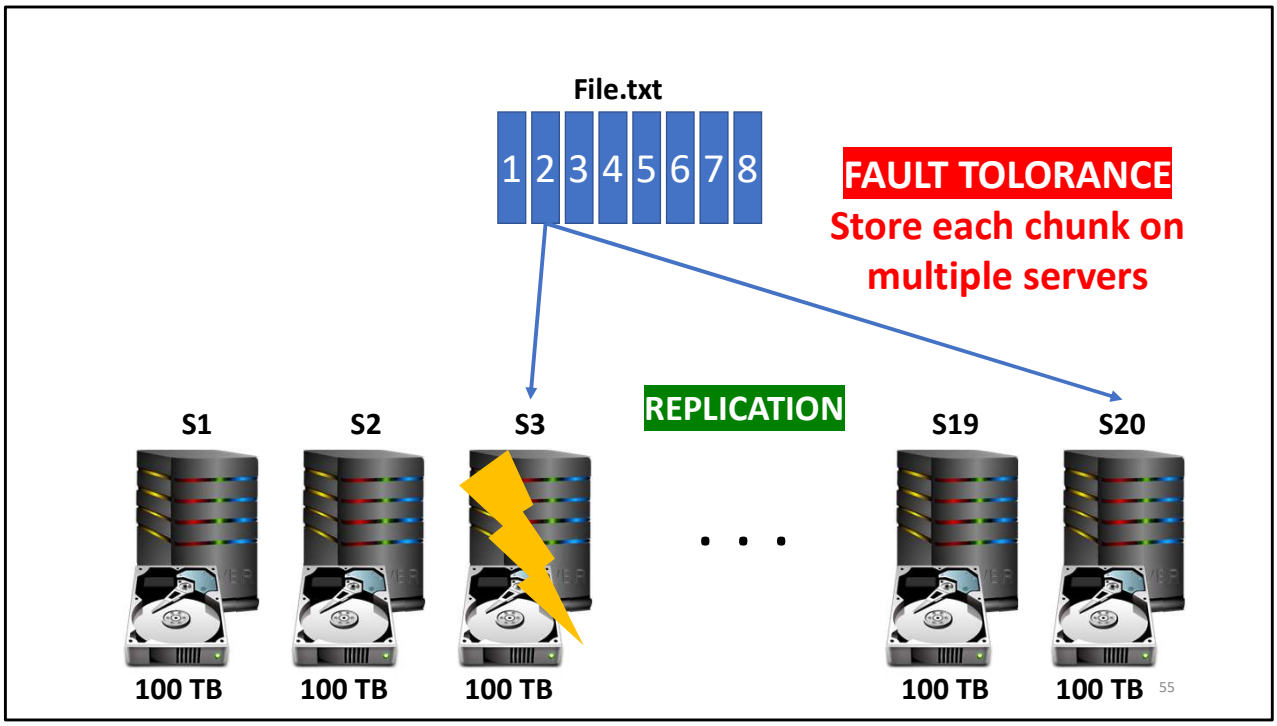
We can split the file into smaller chunks.



And assign the chunks (e.g., randomly) to the servers.



If a server that contains one of the chunks fails, the files become corrupted. Since failure rate is high on commodity servers, we need to figure out a solution.



If each chunk is stored on multiple server, if a server fails there is a backup. The number of copies determines how much resilience we want.

Hadoop Distributed File System (HDFS)

Adapted from Erik Jonsson (UT Dallas)

56

Goals of HDFS

- Very Large Distributed File System
  - 10K nodes, 100 million files, 10PB
- Assumes Commodity Hardware
  - Files are replicated to handle hardware failure
  - Detect failures and recover from them
- Optimized for Batch Processing
  - Provides very high aggregate bandwidth

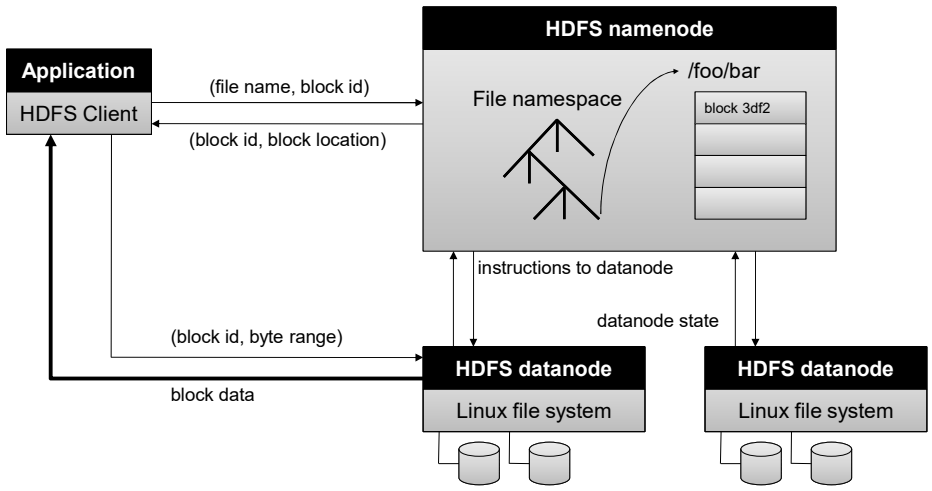
Distributed File System

- Data Coherency
  - Write-once-read-many access model
  - Client can only append to existing files
- Files are broken up into blocks
  - Typically 64MB block size
  - Each block replicated on multiple DataNodes
- Intelligent Client
  - Client can find location of blocks
  - Client accesses data directly from DataNode

58

HDFS is not like a typical file system you use on Windows or Linux. It was specifically designed for Hadoop. It cannot perform some of the typical operations that other file systems can do like random write. Instead it is optimized for large sequential reads and append only writes.

HDFS Architecture



Adapted from (Ghemawat et al., SOSP 2003)

59

Note that the namenode is relatively lightweight, it's just storing where the data is located on datanodes not the actual data.

May still have a redundant namenode in the background if the primary one fails

HDFS client gets data information from namenode and then interacts with datanodes to get that data

Note that namenode has to communicate with datanodes to ensure consistency and redundancy of data (e.g., if a new clone of the data needs to be created)

Functions of a NameNode

- Manages File System Namespace
  - Maps a file name to a set of blocks
  - Maps a block to the DataNodes where it resides
- Cluster Configuration Management
- Replication Engine for Blocks

NameNode Metadata

- Metadata in Memory
  - The entire metadata is in main memory
  - No demand paging of metadata
- Types of metadata
  - List of files
  - List of Blocks for each file
  - List of DataNodes for each block
  - File attributes, e.g. creation time, replication factor
- A Transaction Log
  - Records file creations, file deletions etc

DataNode

- A Block Server
  - Stores data in the local file system (e.g. ext3)
  - Stores metadata of a block (e.g. CRC)
  - Serves data and metadata to Clients
- Block Report
  - Periodically sends a report of all existing blocks to the NameNode
- Facilitates Pipelining of Data
  - Forwards data to other specified DataNodes

Block Placement Policy

- Current Policy: 3 replicas will be stored on at least 2 racks
  - One replica on local node
  - Second replica on a remote rack
  - Third replica on same remote rack
    - Rebalance **might** later move this to a third rack
- Clients read from nearest replicas

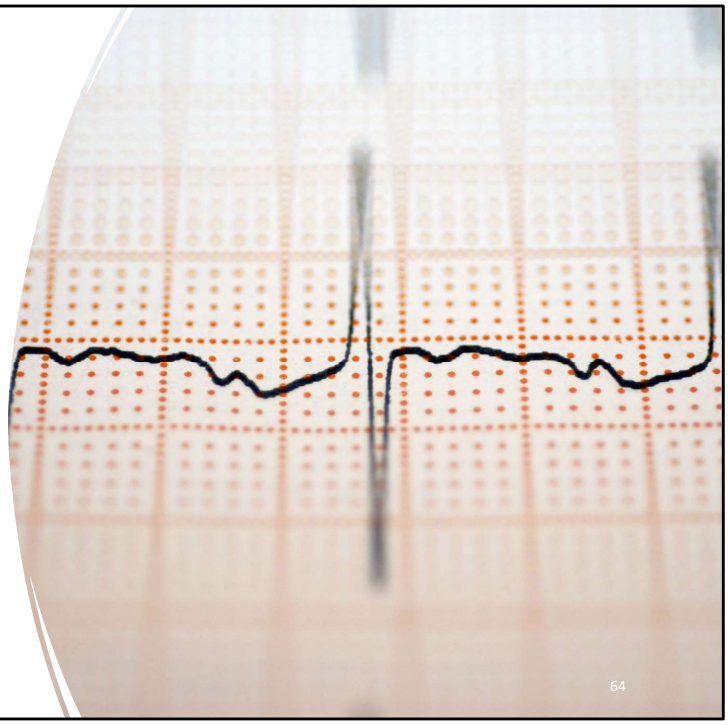
63

Compromise between safety and efficiency. If all your data is on one rack, all it takes is one little fire to lose it all! But, inter-rack communication has higher latency, lower bandwidth than intra-rack, so the remote replica is sent to one rack, and assigned to two nodes there. Load balancing, resharding, etc. might cause the third replica to move to a third rack. Will never have all 3 replicas in one rack.

This is the DEFAULT! You can change this policy if you want to. You can have a replica factor > 3 if you want. You can have a replica factor of 2, for that matter…but shouldn't.

Heartbeats

- DataNodes send heartbeat to the NameNode
  - Once every 3 seconds
- NameNode uses heartbeats to detect DataNode failure



64

Replication Engine

- NameNode detects DataNode failures
  - Chooses new DataNodes for new replicas
  - Balances disk usage
  - Balances communication traffic to DataNodes

65

Balance Disk Usage – Each HDD should have approximately the same usage
Balance Traffic – If a node / rack is currently quite busy with traffic, don't assign it too many reshards. (BUT, it might be perfectly OK to handle an intra-rack resharding)

HDFS Demo

- Dan – open PuTTY and show them how to do some stuff?

- Students viewing this on the webpage –
  - Ummm, google “HDFS Demo”, the first one on Google is good I think

Google File System (GFS)

Terminology differences:

GFS master = Hadoop namenode

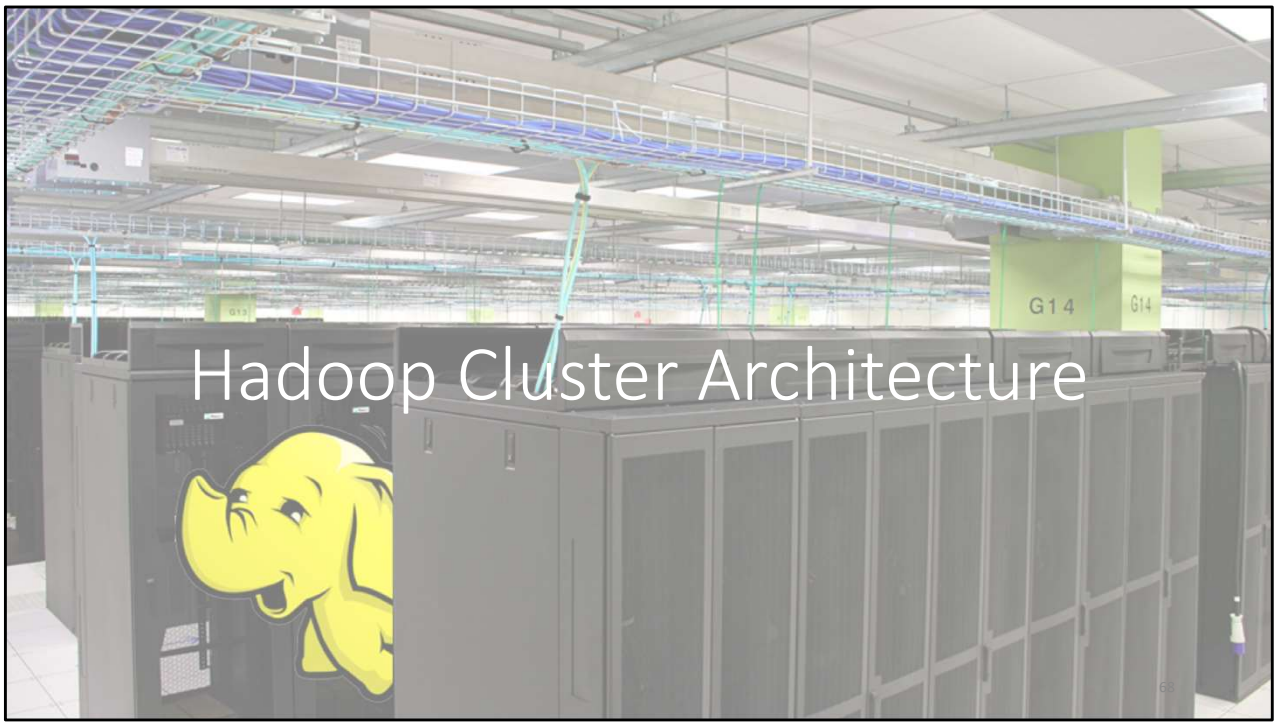
GFS chunkservers = Hadoop datanodes

Implementation differences:

Different consistency model for file appends

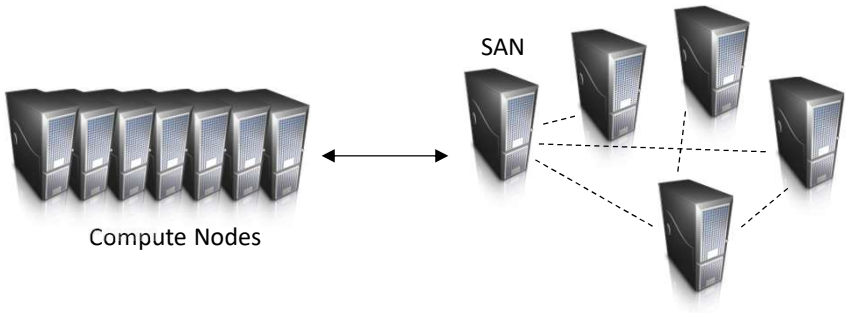
Implementation language

Performance


Hadoop Cluster Architecture

How do we get data to the workers?

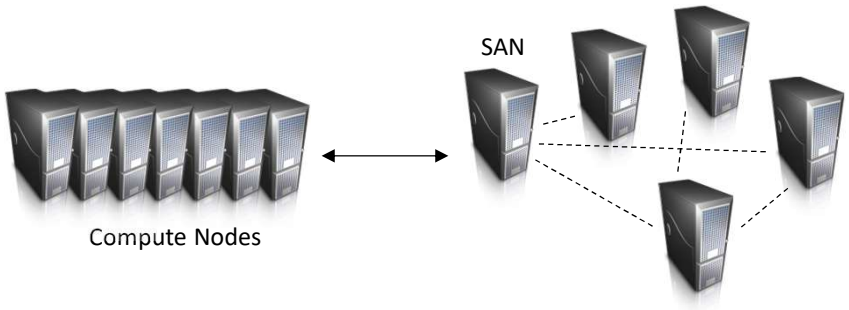
Let's consider a typical supercomputer…



69

SAN: Storage Area Network

Compute-Intensive vs. Data-Intensive



Why does this make sense for compute-intensive tasks?
What's the issue for data-intensive tasks?

70

This makes sense for compute-intensive tasks as the computations (for some chunk of data) are likely to take a long while even on such sophisticated hardware, so the communication costs are greatly outweighed by the computation costs. For data-intensive tasks, the computations (for some chunk of data) aren't likely to take nearly as long, so the computation costs are greatly outweighed by the communication costs. Likely to experience latency and bottleneck even with high speed transfer.

What's the solution?

Don't move data to workers… move workers to the data!

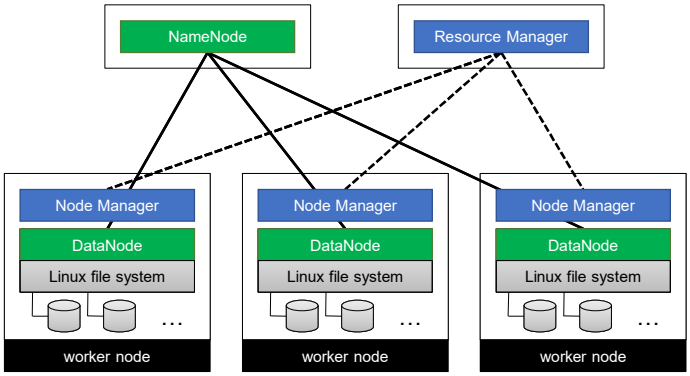
Key idea: co-locate storage and compute

Start up worker on nodes that hold the data

71

If a server is responsible for both data storage and processing, Hadoop can do a lot of optimization. For example, when assigning mapreduce tasks to servers, Hadoop considers which servers contain what part of the file locally to minimize copy over network. If all of the data can be process locally where it is stored there will be no need to move the data.

Putting everything together…

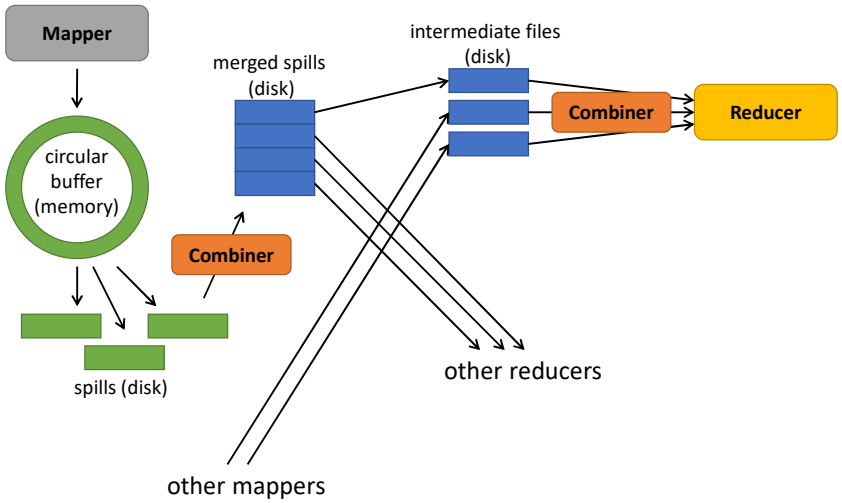


72

This figure shows how computation and storage is co-located on a Hadoop cluster.
Node manager manages running tasks on a node (e.g., if we have spare resources, do the next job assigned to us)
Resource manager is responsible for managing available resources in the cluster

73

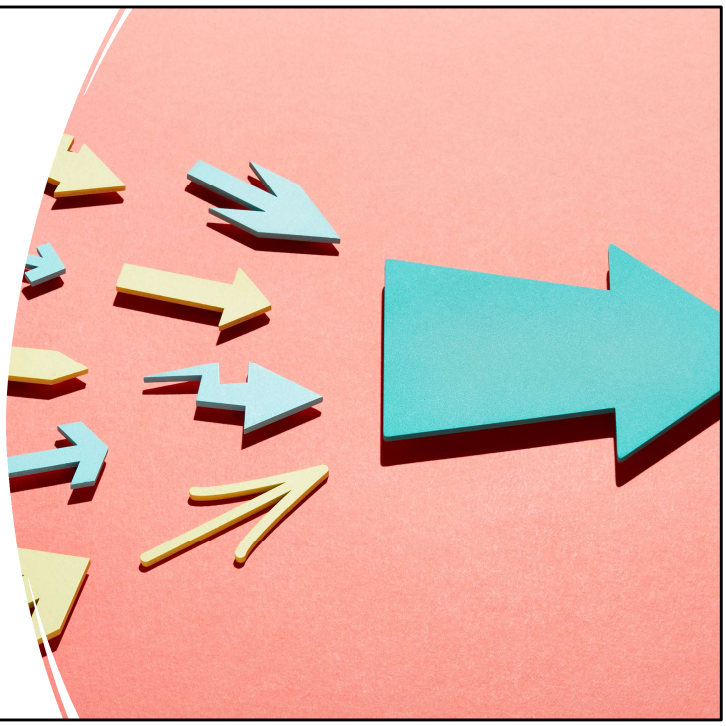
Back to Combiners in MapReduce



The combiner may or may not run while merging spills on the mapper side. It also may or may not run when merging partitions on the reducer side. The framework will decide this as part of optimizing the job schedule.

Combiner Design

- Combiners are like Reducers – they have the same signature
  - A reducer can have different key types
- Combiners are **optional**
  - May not be run
  - May run once
  - May run many times

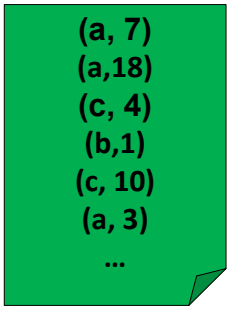


Reminder: If the reducer has k2 = k3, v2 = v3, then it MIGHT work as a combiner. But it also might not!

Computing the mean

```
def map(key : String, value: Int):
  emit(key, value)

def reduce(key: String, values: List[Int]):
  sum = 0
  count = 0
for value in values:
   sum += value
   count += 1
   emit(key, sum / count)
```



Note that we cannot have a combiner here! The reducer won't work (why?) and there's not really a way to create a different function that will work, either.

Computing the mean (v2)

```
def map(key : String, value: Int):
  emit(key, value)

def combine(key: String, values: List[Int]):
  for value in values:
    sum += value
    count += 1
   emit(key, (sum, count))

def reduce(key: String, values: List[(Int, Int)]):
  for (v, c) in values:
    sum += v
    count += c
  emit(key, sum / count)
```

INVALID

(a, 7)
(a,18)
(c, 4)
(b,1)
(c, 10)
(a, 3)
…

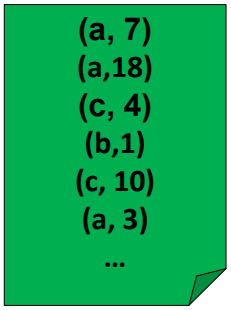
This isn't valid. Combine is OPTIONAL. It MUST have the same input and output types! This design incorrectly assumes that combiners are always run.

Computing the mean (v3)

```
def map(key : String, value: Int):
  emit(key, (value, 1))

def combine(key: String, values: List[(Int, Int)]):
  for (v, c) in values:
    sum += v
    count += c
   emit(key, (sum, count))

def reduce(key: String, values: List[(Int, Int)]):
  for (v, c) in values:
    sum += v
    count += c
  emit(key, sum / count)
```



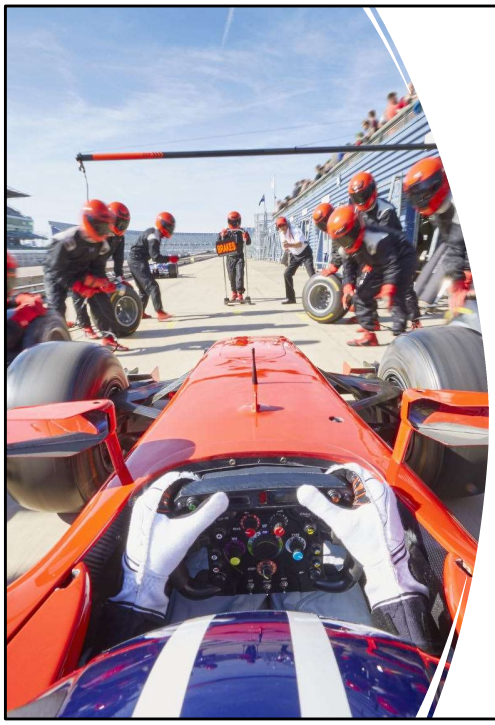
The fix is to change the mapper to emit the same type as the combiner will.

Performance

Input size: 200m integers, 3 unique keys

V1 (baseline) ~120 seconds

V3 (combiner) ~90 seconds



I wanna go fast

Combiners improve performance by reducing network traffic

Combiners work during file merges.

- Local filesystem is faster than network access

But memory is faster than the filesystem

Computing the mean (v4)

```
class mapper:
    def setup(self):
        self.sums = Map()
        self.counts = Map()
    def map(self, key, value):
        self.sums[key] += value
        self.counts[key] += 1
    def cleanup(self):
        for (key, count) in counts:
        emit(key, (sums[key], count))
```

Didn't you say not to do this???

(a, 7)
(a,18)
(c, 4)
(b,1)
(c, 10)
(a, 3)
…

Yes, you should avoid remembering things because you might end up trying to remember too much. However, if you're sure there won't be too many things, then you can! Functional programming isn't a prison. You can deviate, you should just be careful when you do so.

Think at scale: How many keys are there? Can the mapper hold a count and sum in memory for every single key??? If it can, this is OK. If it can't…then it's not OK. That's all there is to it.
Remember to always ask these questions! Remember "probably fine" means "not fine". Be certain.

In-Mapper Combine

Preserve state across calls to map

Advantage: Speed

Disadvantage: Requires memory management

I prefer to think of “IMC” as meaning “In-memory combiner” since that’s how it works, and you can do the same technique on a reducer, too.

That might seem strange because everything is already grouped by key, but remember, the reducer is allowed to change the key-types and can emit whatever you want it to, so it can often make sense to use IMC in your reducer, too!

Performance

Input size: 200m integers, 3 unique keys

V1 (baseline) ~120 seconds

V3 (combiner) ~90 seconds

V4 (IMC) ~60 seconds

Discussion: Can we do this for word frequency?

```
class mapper:
  def setup(self):
    counts = HashMap()
  def map(self, key: Long, value: String):
    for word in tokenize(value):
      counts[word] += 1
  def map_cleanup():
    for (key, count) in counts:
      emit(key, count)
```

Probably? It's usually safe to assume less than 1M unique words. If your counter is int, that's 4MB for the counts, Maybe another 8MB for the keys??? Assume 50% storage inefficiency and 24MB should be enough. Famous last words.

Once again note that this is python-like pseudocode, not actual python. (It's close though, if there were MapReduce python bindings. Replace HashMap with counter, replace counts: with counts.items() in the cleanup loop)

New Problem: Term Co-Occurrence

M_{ij}: number of times word i and word j coöccur in some context

E.g. how many times is i followed immediately b j in a sentence

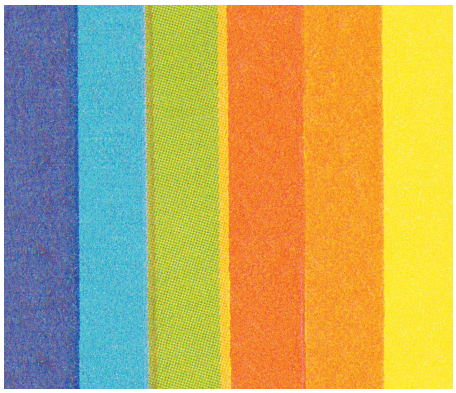
M is N x N, where N is the vocabulary

This is just one possible definition for what "context" means.
Note the umlat over the second o. This is actually a **diaeresis,** the New Yorker does this to indicate a syllable break. Most people use a dash, e.g. co-op but coöp is just the kind of pretentiousness I can get behind!

Two Approaches

Pairs

Stripes



Sorry, the PowerPoint stock photo engine failed to find a good picture for “pairs” so I made do…

Pair – We’ll be computing individual Cells
Stripe – We’ll be computing individual Rows

Pairs

Mapper

Input: Sentence

Output: ((a, b), 1), for all pairs of words a, b in the sentence.

Reducer

Input: pair of words, list of counts

Output: Pair of words, count

In this case the reducer function can also serve as the combiner.

Pairs, In Pseudocode

```
def map(key : Long, value: String):
  for u in tokenize(value):
    for each v that coöccurs with u in value:
      emit((u, v), 1)

def reduce(key: (String, String), values: List[Int]):
   for value in values:
     sum += value
   emit(key, sum)
```

Note that we can pick whatever definition of cooccurrence we want…it might just mean “is at the next index”. That’s the power of pseudocode! Best language

Pairs Analysis

- Easy to implement
- Easy to understand
- That's a lot of pairs!
- Combiner won't do much. Why?

The combiner won't do much because there are N x N potential keys. Most keys will have few entries, so there will be few cases where the combiner reduces the number of pairs.

Stripes

Mapper

Input: Sentence

Output: (a, {b_1:c_1, b_2:c_2, …, b_m:c_m}), where:

a is a word from the input

b_1 … b_m are all words that coöccur with a

c_i is the number of times (a, b_i) coöccur

{} means a map (aka a dictionary, associative array, etc)

In this case the reducer function can also serve as the combiner.

Stripes, Pseudocode

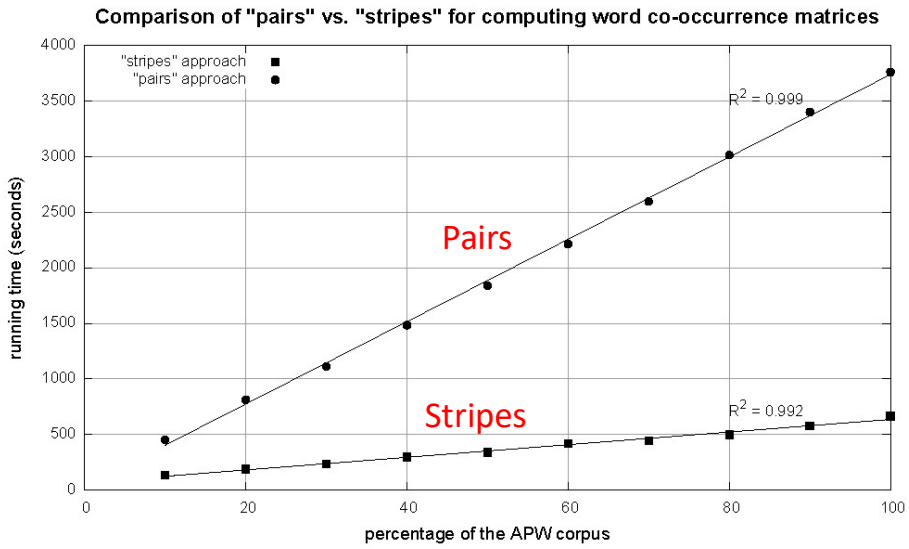
```
def map(key: Long, value: String):
   for u in tokenize(value)
     counts = {}
     for each v that coöcurs with u in value:
       counts(v) += 1
     emit(u, counts)

def reduce(key: Long, values: List[Map[String->Int]]):
  for value in values:
    sum += value
  emit(key, sum)
```

Here adding two Maps means taking the union of the keys, and setting the value to be the sum of the two values if it occurs in both Maps, otherwise taking the value from the single map that has it. You MIGHT need to write that code yourself, but that's what puts the pseudo in pseudocode

Stripes Analysis

- Fewer key-value pairs to send
- Combiners will do more work
- Map is a heavier object than a single Int
- More computationally intensive
- Will the map fit in memory???



Cluster size: 38 cores
Data Source: Associated Press Worldstream (APW) of the English Gigaword Corpus (v3), which contains 2.27 million documents (1.8 GB compressed, 5.7 GB uncompressed)

So Always Use Stripes?

No. There's a tradeoff.

"Easier to understand and implement" is NOT bad.

You'll see after A1, mwhahaha. (For CS431 this only hits you on A2, don't get complacent)

For English words and normal sentence lengths, the stripe fits in memory easily. It won't always work out that way.

Another Problem, Relative Frequencies

$$f(B|A) = \frac{N(A,B)}{N(A,*)}$$

Where N(A, B) is number of coöccurrences of A and B, and N(A,*) is the sum of N(A,x) over all x

Why do we want to do this?

How do we make it fit into MapReduce?

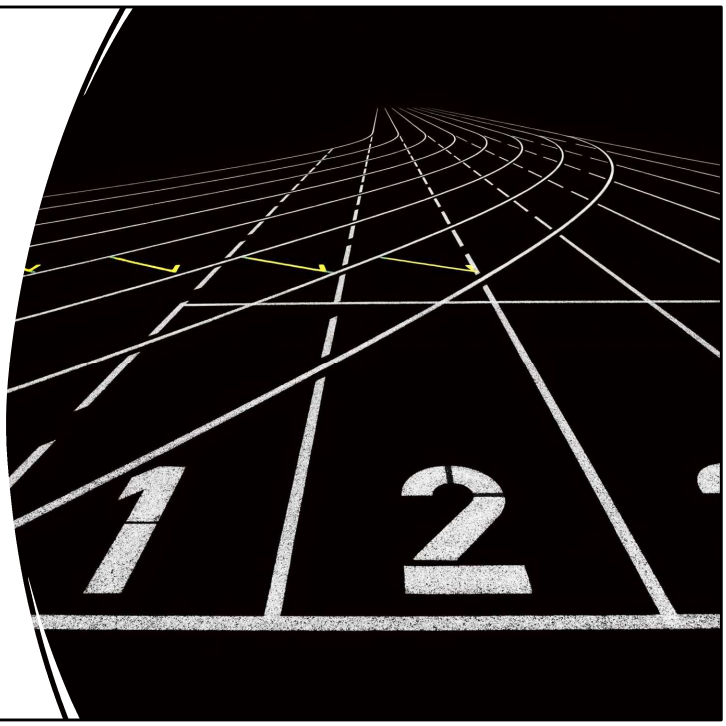
Note that N(A,*) might be “number of occurrences of A” depending on the definition of occurrence / co-occurrence. It also might not!

Stripes

A -> {B_1:C_1, B_2,C_2, ….}

Easy-Peasy. If N(A, B) = N(B, A) then N(A,*) is just C_1+C_2+…

The stripe gives us all the information we need!



Pairs?

```
def reduce(key:Pair[String], values: List[Int]):
   let (a, b) = key
   for v in values:
     sum += v
   emit((b, a), sum / freq(a))
```

Hmmm, what's freq(a)? We don't know that until we've processed all keys of the form (a, *)

- '*' Here means "everything", like it does with command line, etc.
- This is also called the "marginal sum of a" – Accountants would jot numbers in the margin of a spreadsheet (a physical one, not Excel) and add them all up at the end, so this is known as a "marginal" value. (The meaning most people are familiar with is "barely" – because if you're on the margin you're "only just" on the page)

f(B|A): "Pairs"

(a, *) → 32 Reducer holds this value in memory

(a, b_1) → 3
(a, b_2) → 12
(a, b_3) → 7
(a, b_4) → 1
…

(a, b_1) → 3 / 32
(a, b_2) → 12 / 32
(a, b_3) → 7 / 32
(a, b_4) → 1 / 32
…

For this to work:

Emit extra (a, *) for every b_n in mapper
Make sure all a's get sent to same reducer (use partitioner)
Make sure (a, *) comes first (define sort order)
Hold state in reducer across different key-value pairs

"Define sort order" means override the Compare methods for the data type…if you have to. Here we don't assuming the words are words (i.e. just letters). * is before a so lexicographically "*" comes first…the empty string might work even better!

Pairs, Mapper and Partitioner

```
def map(key: Long, value: String):
   for u in tokenize(value):
     for v in cooccurrence(u):
        emit((u, v), 1)
        emit((u, “*”), 1)

def partition(key: Pair, value: Int, N: Int):
   return hash(key.left) % N
```

Pairs, Mapper and Partitioner (improved)

```
def map(key: Long, value: String):
   for u in tokenize(value):
     for v in cooccurrence(u):
        emit((u, v), 1)
     emit((u, “*”), len(cooccurrence(u))

def partition(key: Pair, value: Int, N: Int):
   return hash(key.left) % N
```

While we can maybe hope our combiner will compact the “*” counts down, it will be very little extra work to only send one * per token per line – this obviously only makes a difference for definitions of co-ocurrnece where a token will co-occur with many other tokens on the same line. If not, then the length is always 1 so it makes no improvement – it also doesn’t need to.,

Pairs, Reducer

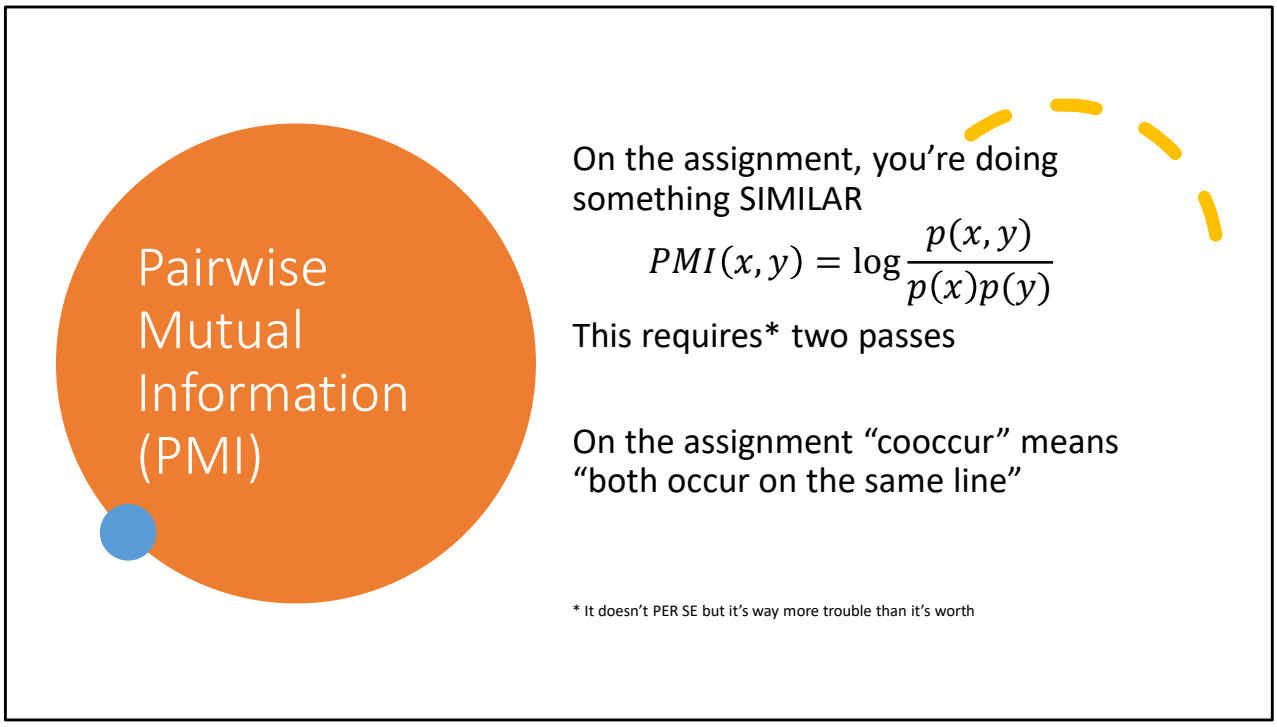
Stats term

```
marginal = 0

def reduce(key: Pair, values: List[Int]):
   let (a, b) = key
   for (v in values):
     sum += v
   if (b == “*”):
      marginal = sum
   else:
     emit((b, a), sum / marginal)
```

Marginal variable: A variable that can be found by summing all values in a column (or row) and writing the value in the margin. C(A,*) is a marginal variable because it can be found by summing all C(A,x) values. It’s not piece of paper
So there is no actual margin, but a name is a name.

Will this work? Yes, at least in Hadoop using English words. Remember, Hadoop at least sorts the keys. Pairs will sort lexicographically by the first key, and then the second. So (a,*) comes before any (a, b) because the ASCII code of * is 42, which is less than the lower-case letters…you could always use the empty string instead of “*” if you’re worried..



The reason it "requires" two passes is that you need to make sure that (x,*) and (y,*) are both available on the reducer that has (x,y), which requires that all reduces have all marginal counts.
You can do this in one pass but it's quite awkward to do and ends up not really being faster.

451:

To do it in two passes, you should have one pass doing modified word count (counting number of lines that contain the token rather than counting all occurrences of the token) – then in pass two you're not concerned with the marginal sums – in the reducer code you'll load the output files from pass 1 in the "setup" method and then put it all into a hash table (aka a Java Map object)

431

You're not using MapReduce for A1 so "two passes" isn't needed – you can put marginal sums into one dictionary and pairs into another.
(In fact you should probably use a "stripes" approach where instead of pairs[(a,b)] you'd be looking for stripes[a][b] – the reason being with the stripes approach, it's really easy to see all tokens that cooccur with a, that's just the contents of stripes[a]. To get the same thing

from pairs you need to traverse the entire thing!)

PMI, Yeah, What's It Good For?

~~Absolutely Nothing!~~

PMI is useful for establishing "semantic distance" between tokens

Tokens with similar lists of cooccurrences sorted by PMI likely have similar meaning.

Semantic distance is (usually) a [-1,1] ranged metric, where 1 means "exact same meaning, including connotations, i.e. perfect synonym" and -1 means "exact opposite meaning i.e. perfect antonym"

This is one way to generate word embeddings. More on that later! But the TL;DR is we want each word should be a unit vector with a lot of dimensions, such that dot product between two vectors (i.e. the cosine of the angle between the vectors) is approximately the same as the semantic distance between those words.

Note that A1+A2 do not involve files big enough to get us meaningful word similarities, generally speaking. But this is what things like word2vec do to generate their embeddings.

Sweet, Delicious Hints

A1 suggests multiple passes as something you might want to consider.

CONSIDER IT STRONGLY

(In other words, it's possible to do with a single pass but there's no gain to doing this. This is not a challenge)

Hints = Macarons. The fanciest, most difficult to make cookie. Yet always slightly disappointing. The PowerPoint AI made this connection, not me. Spooky