MAPREDUCE RELIABILITY AND PERFORMANCE

CLOUD COMPUTING

Dr. Atm Shafiul Alam

a.alam@qmul.ac.uk

Queen Mary University of London

School of Electronic Engineering and Computer Science

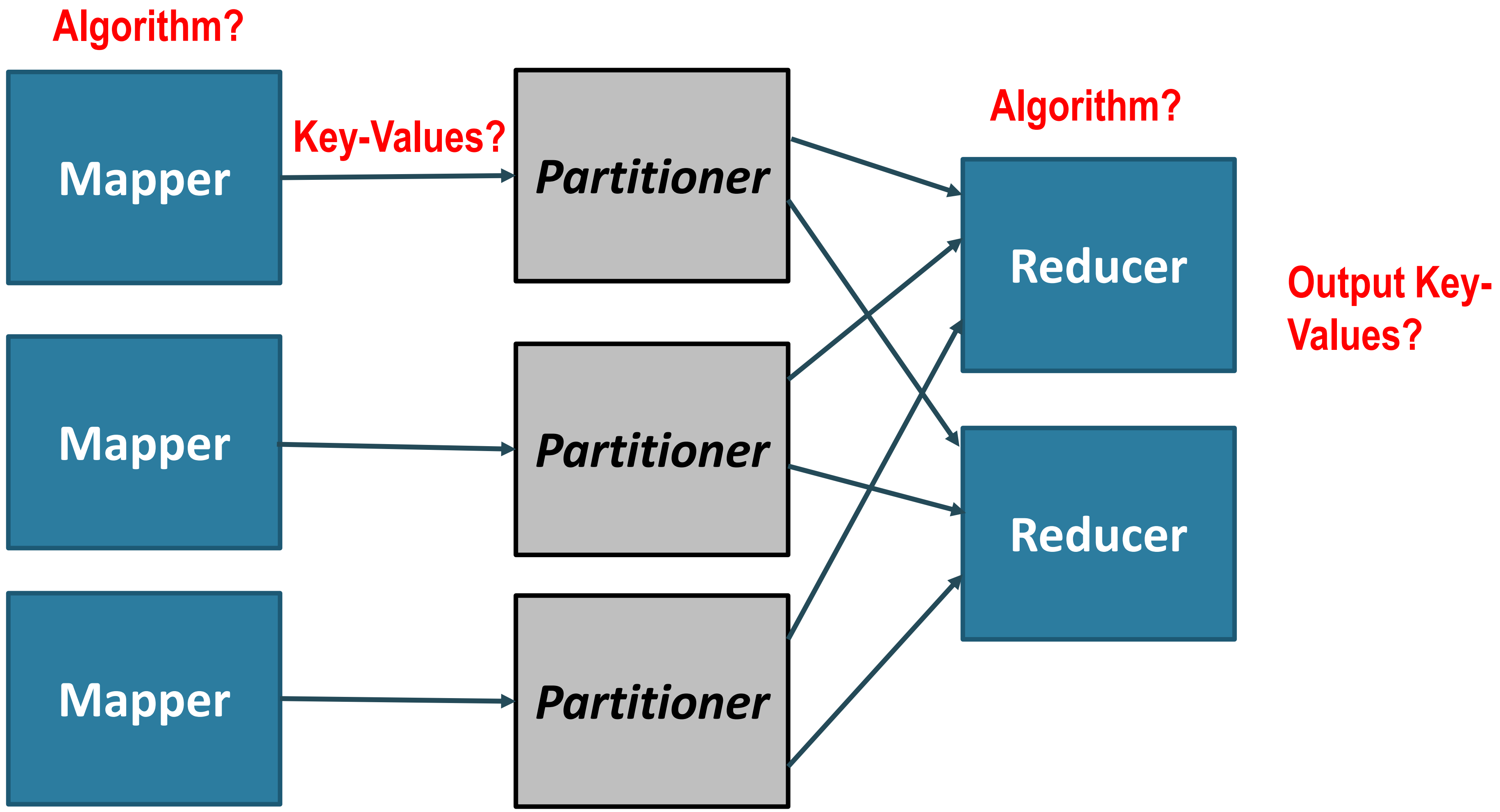
Thanks to Dr. Gareth Tyson and Dr. Félix Cuadrado who created the original version of these slides

QUICK RECAP…

Last time, we...

- Discussed some of the complexities of Map Reduce
- Described several examples
  - Computing averages
  - Generating indexes
  - Data filtering
  - Data joins

Decision points...



Numerical Summarization

- **Goal:** Calculate aggregate statistical values over a dataset

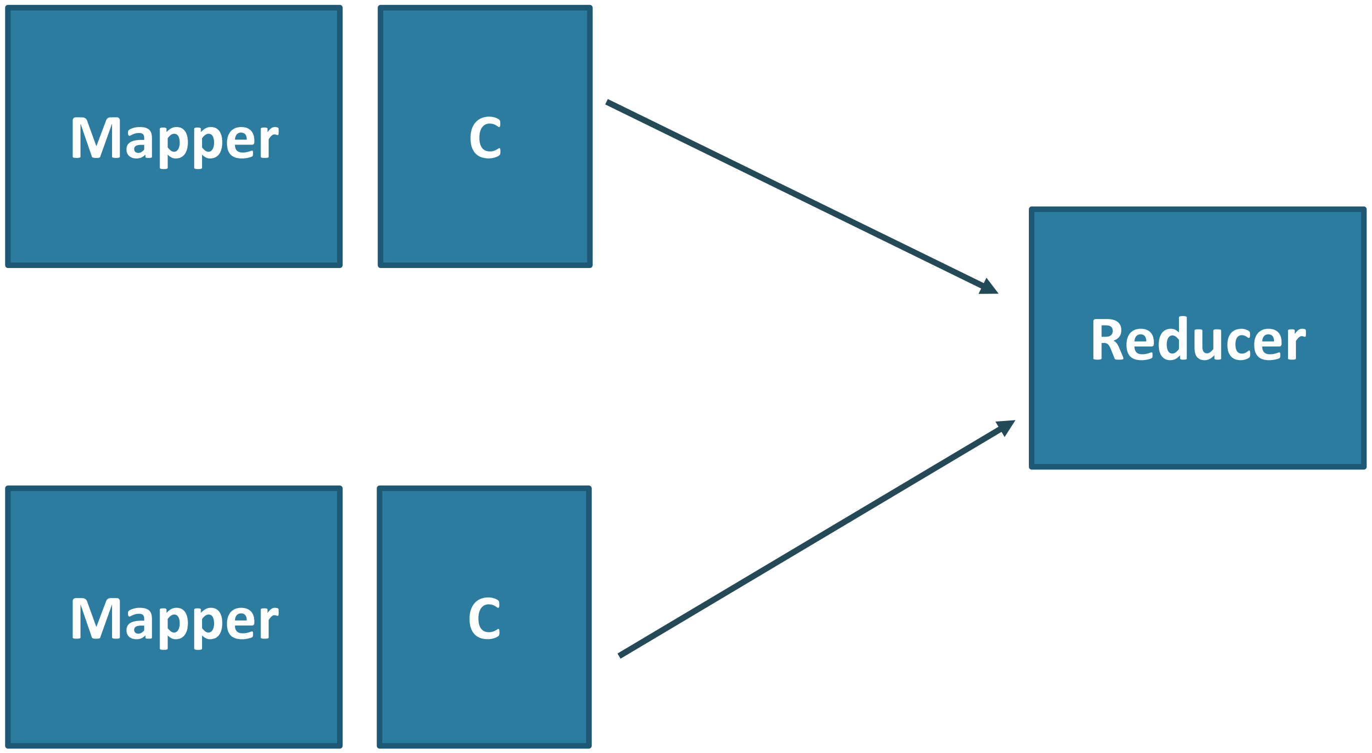
- Examples:
  - Count occurrences
  - Maximum / minimum values
  - Average / median / standard deviation

Computing averages

Input: Row with student information, grade

Goal: Compute module average

ec03847293847	100
ec29347298347	100
ec23894283472	100
ec23489209348	100
ec23492834343	100
ec34948758493	0
ec56456456545	100
ec73453435434	100

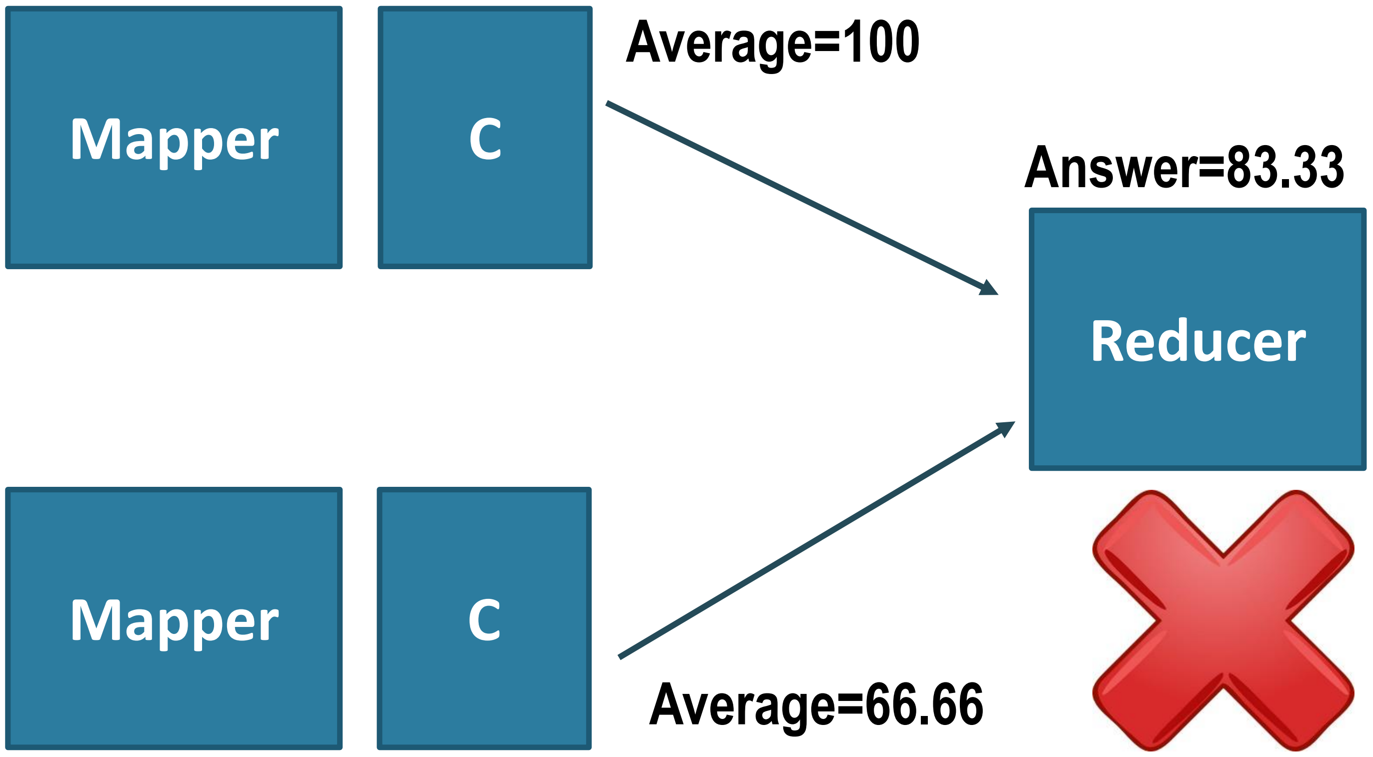


Computing averages

Input: Row with student information, grade

Goal: Compute module average

ec03847293847	100
ec29347298347	100
ec23894283472	100
ec23489209348	100
ec23492834343	100
ec34948758493	0
ec56456456545	100
ec73453435434	100

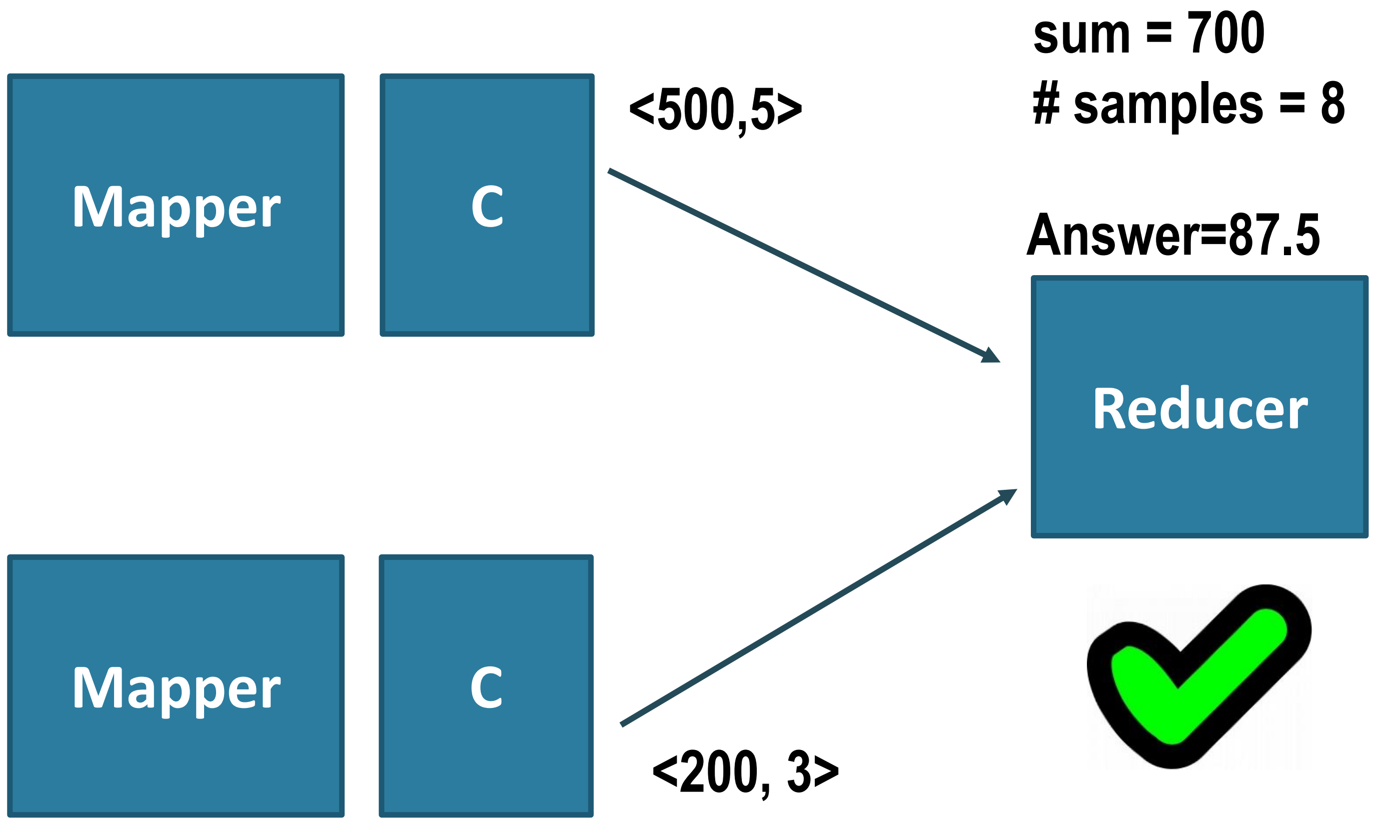


Computing averages

Input: Row with student information, grade

Goal: Compute module average

ec03847293847	100
ec29347298347	100
ec23894283472	100
ec23489209348	100
ec23492834343	100
ec34948758493	0
ec56456456545	100
ec73453435434	100



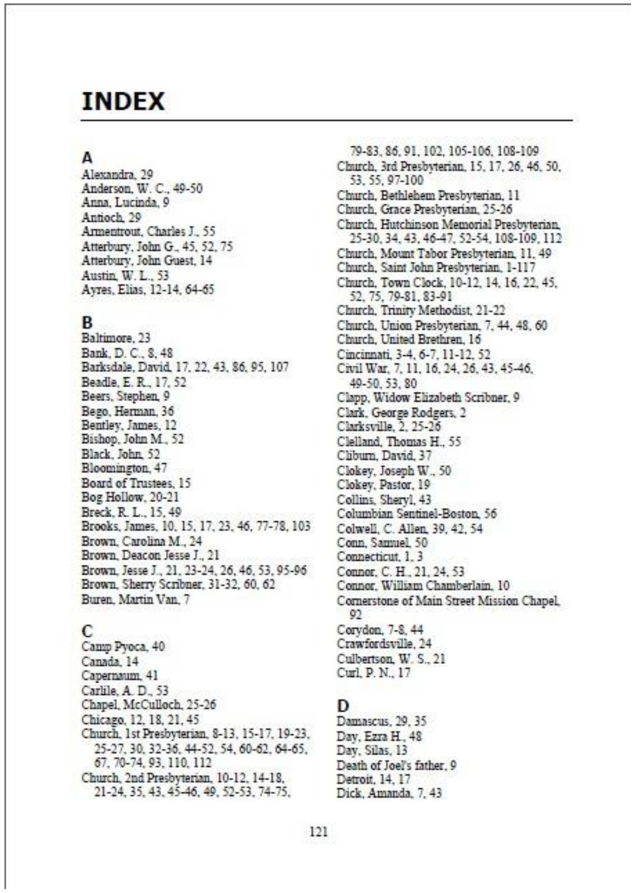
Some questions…

- How many key-value pairs will be fed into each mapper?
  - Each mapper will get a roughly even share of the input keys
- How many key-value pairs will be emitted by each mapper?
  - Based on the algorithm - one key will be emitted per “feature” identified
  - Divided by #mappers (e.g. /10 if there are 10 mappers)
- How many keys will be fed into each reducer?
  - Each reducer will get a roughly even share of the keys

Computing inverted index

- **Goal:** Generate index from a dataset to allow faster searches for specific features

- Examples:
  - Building index from a textbook.
  - Finding all websites that match a search term



INDEX

A
Alexandra, 29
Anderson, W. C., 49-50
Anna, Lucinda, 9
Antioch, 29
Armentrout, Charles J., 55
Atterbury, John G., 45, 52, 75
Atterbury, John Guest, 14
Austin, W. L., 53
Ayres, Elias, 12-14, 64-65

B
Baltimore, 23
Bank, D. C., 8, 48
Barksdale, David, 17, 22, 43, 86, 95, 107
Beadle, E. R., 17, 52
Beers, Stephen, 9
Bego, Herman, 36
Bentley, James, 12
Bishop, John M., 52
Black, John, 52
Bloomington, 47
Board of Trustees, 15
Bog Hollow, 20-21
Breck, R. L., 15, 49
Brooks, James, 10, 15, 17, 23, 46, 77-78, 103
Brown, Carolina M., 24
Brown, Deacon Jesse J., 21
Brown, Jesse J., 21, 23-24, 26, 46, 53, 95-96
Brown, Sherry Scribner, 31-32, 60, 62
Buren, Martin Van, 7

C
Camp Pyoca, 40
Canada, 14
Capernaum, 41
Carlile, A. D., 53
Chapel, McCulloch, 25-26
Chicago, 12, 18, 21, 45
Church, 1st Presbyterian, 8-13, 15-17, 19-23, 25-27, 30, 32-36, 44-52, 54, 60-62, 64-65, 67, 70-74, 93, 110, 112
Church, 2nd Presbyterian, 10-12, 14-18, 21-24, 35, 43, 45-46, 49, 52-53, 74-75, 79-83, 86, 91, 102, 105-106, 108-109
Church, 3rd Presbyterian, 15, 17, 26, 46, 50, 53, 55, 97-100
Church, Bethlehem Presbyterian, 11
Church, Grace Presbyterian, 25-26
Church, Hutchinson Memorial Presbyterian, 25-30, 34, 43, 46-47, 52-54, 108-109, 112
Church, Mount Tabor Presbyterian, 11, 49
Church, Saint John Presbyterian, 1-117
Church, Town Clock, 10-12, 14, 16, 22, 45, 52, 75, 79-81, 83-91
Church, Trinity Methodist, 21-22
Church, Union Presbyterian, 7, 44, 48, 60
Church, United Brethren, 16
Cincinnati, 3-4, 6-7, 11-12, 52
Civil War, 7, 11, 16, 24, 26, 43, 45-46, 49-50, 53, 80
Clapp, Widow Elizabeth Scribner, 9
Clark, George Rodgers, 2
Clarksville, 2, 25-26
Clelland, Thomas H., 55
Cliburn, David, 37
Clokey, Joseph W., 50
Clokey, Pastor, 19
Collins, Sheryl, 43
Columbian Sentinel-Boston, 56
Colwell, C. Allen, 39, 42, 54
Conn, Samuel, 50
Connecticut, 1, 3
Connor, C. H., 21, 24, 53
Connor, William Chamberlain, 10
Cornerstone of Main Street Mission Chapel, 92
Corydon, 7-8, 44
Crawfordsville, 24
Culbertson, W. S., 21
Curl, P. N., 17

D
Damascus, 29, 35
Day, Ezra H., 48
Day, Silas, 13
Death of Joel's father, 9
Detroit, 14, 17
Dick, Amanda, 7, 43

121

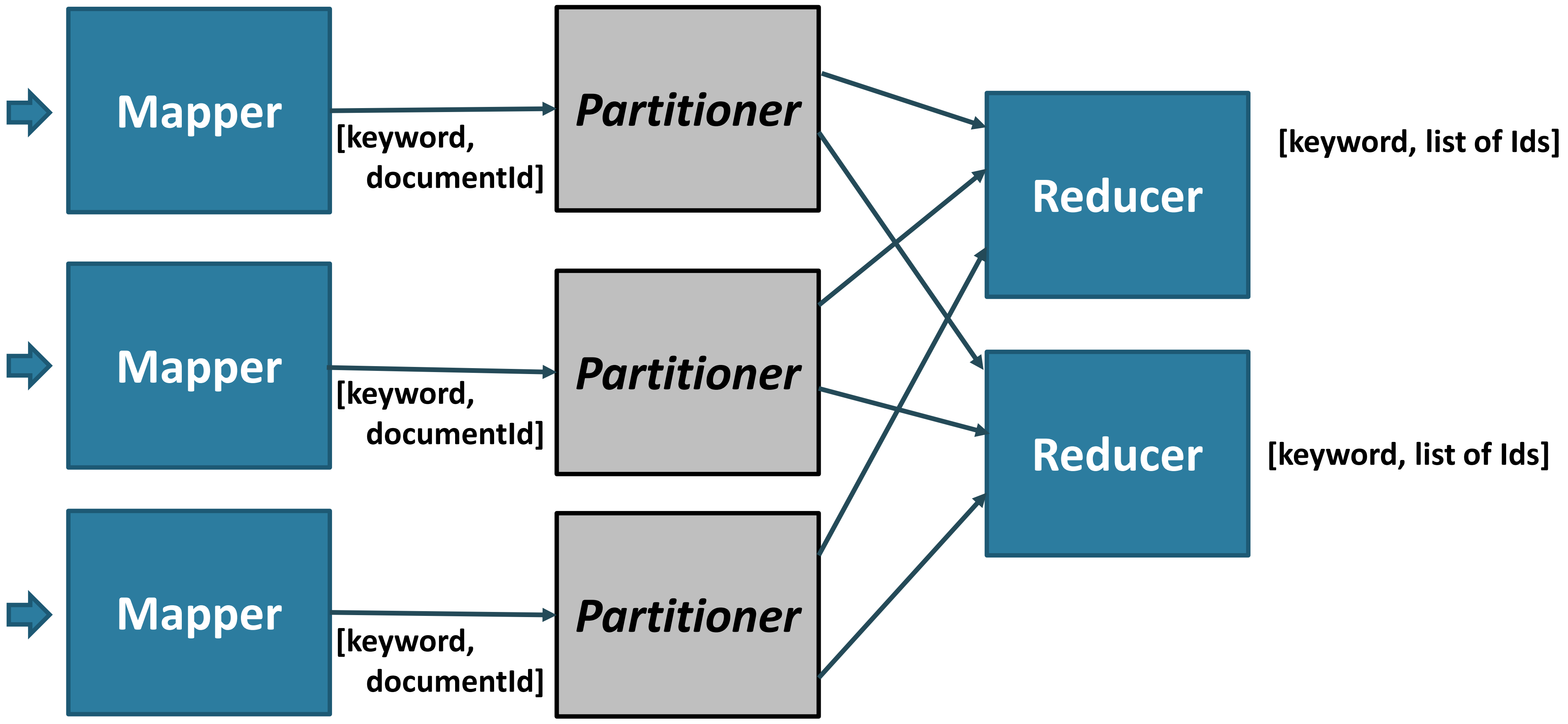
Inverted Index Mapper

```
public void map (String docId, String text){

   String[] features = findFeatures(text);

      for(String feature: features){
           emit(feature, docId);
      }
}
```

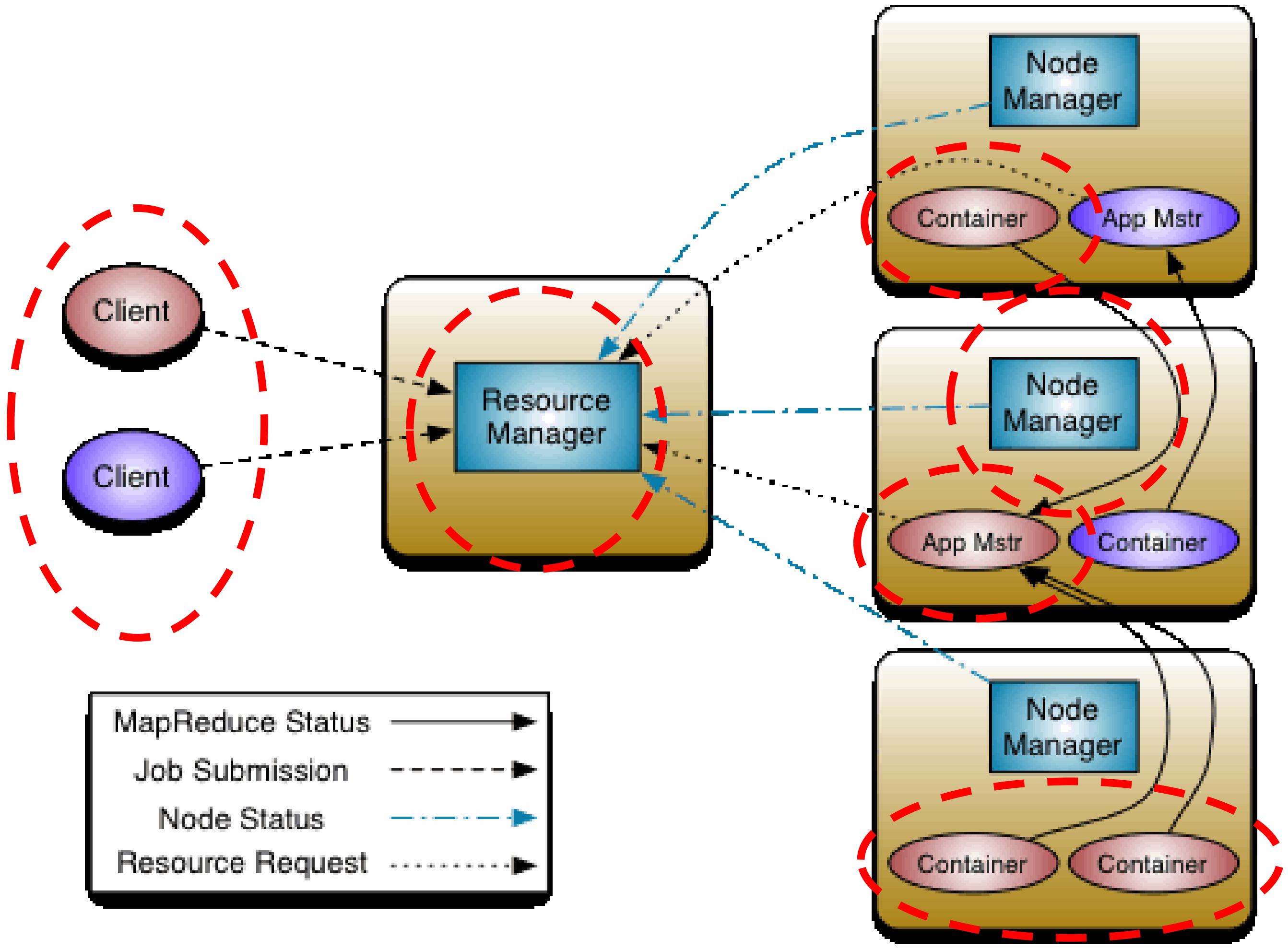
Inverted Index Reducer

```
public void reduce (String feature,
                                       String[] docIds){

      emit(feature, formatNicely(docIds))
        //formatNicely() combines indexes into list
        //that is easy to read
}
```

Inverted Index Structure

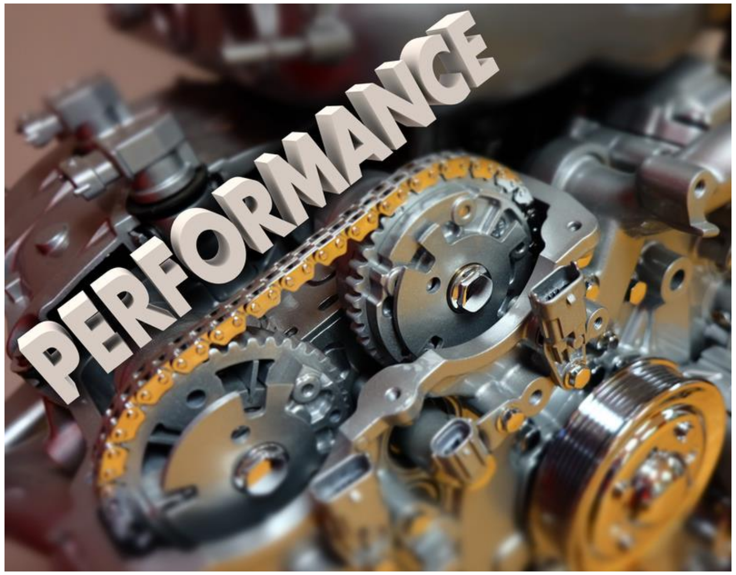


Job Execution Architecture (YARN)



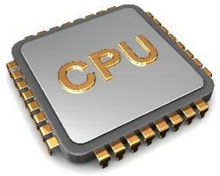
Today's contents

- **Hadoop Performance**
- Hadoop reliability



Speedup concept

- General concept of speedup in parallel computing:
  - How much faster an application runs on parallel computer?
  - What benefits derive from the use of parallelism?
- Speedup of a parallel processing system is a function of n (the number of processors):



$$s(n) = \frac{time\_taken\_with\_1\_processor}{time\_taken\_with\_n\_processors}$$

- Speedup is problem-dependent as well as architecture-dependent
- A 100% embarrassingly parallel job can be fully parallelized.

$$s_{emb}(n) = n$$

Speedup concept

- Imagine it takes 10 minutes with 1 processor

- OR 5 minutes with 2 processors

$$\frac{time\_taken\_with\_1\_processor}{time\_taken\_with\_n\_processors} \quad \frac{\mathbf{10}}{\mathbf{5}} = \mathbf{2}$$

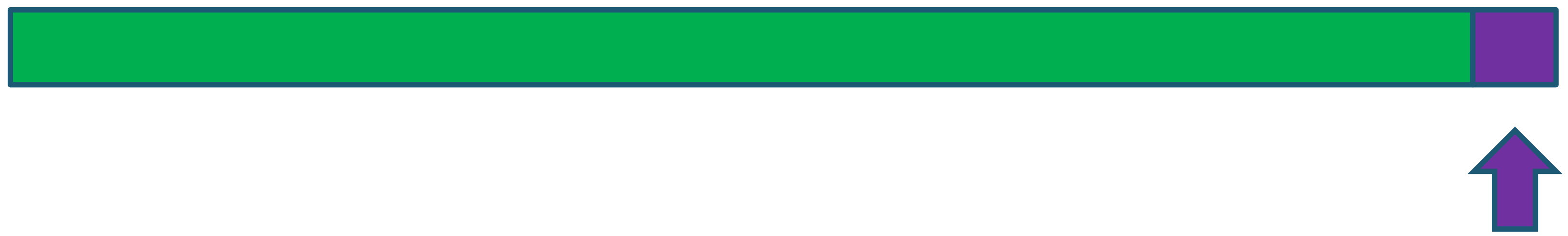
Speedup is 2

Amdahl's Law

- But in many jobs, some parts (*f*) of the computation can **only** be executed sequentially on one processor.
  - Load data & pre-process
  - Split data
  - Extract statistics
  - Aggregate

Sequential

Parallel



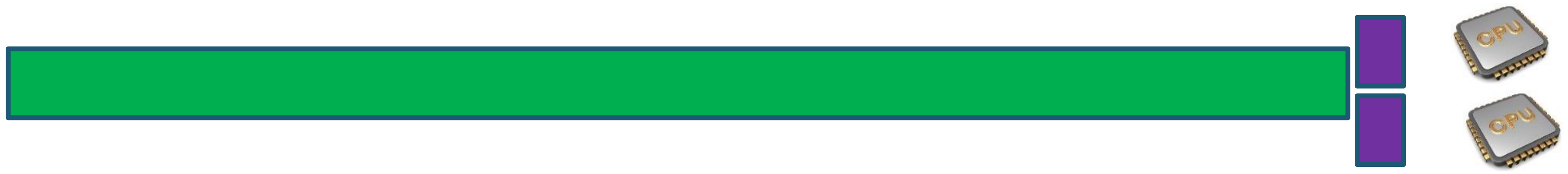
Can only speed up code that can be parallelised…

Amdahl's Law

- But in many jobs, some parts (*f*) of the computation can **only** be executed sequentially on one processor.
  - Load data & pre-process
  - Split data
  - Extract statistics
  - Aggregate

$\left.\begin{matrix}\text{Load data \& pre-process}\\ \text{Split data}\end{matrix}\right\}$ **Sequential**

$\left.\begin{matrix}\text{Extract statistics}\\ \text{Aggregate}\end{matrix}\right\}$ **Parallel**



Amdahl's Law

- Best case is sequential processing time + parallel processing time
- **Amdahl's law:** if the remaining (1–*f)* work can be perfectly parallelized, then the **speedup** with ***p*** processors is:

$$\psi \le \frac{\underline{1}}{\underline{f} + \underline{(1-f)/p}}$$

Time to run *everything* in sequence

Fraction of sequential code + an even share of parallel code

Amdahl's Law Example

- ***f*** is the fraction of inherently sequential computations, e.g. 0.05 if it is 5%
  - I.e. 95% of code can be executed in parallel
- ***p*** is the number of processors, *e.g.* number of compute nodes

f = 5% (0.05)

p = 8 processors

$$\psi \leq \frac{1}{f + (1 - f)/p}$$

$$\frac{1}{0.16875}$$

$$\psi \leq \frac{1}{\quad + (1 - \quad)/\quad}$$

Amdahl's Law Example

- **95%** of a parallel program's execution time is spent within inherently **sequential** code. What is the maximum speedup achievable by this program?
  - ***f*** = 0.95
  - ***p*** = 100…1000…..1000000?
- Terrible speedup:
  - Numerator is always 1
  - Denominator is (**0.95** + *roughly* **0**)
  - This will always result in answer of just over **1**
  - We only get high speedup when we have a low fraction of sequential code

$$\psi \le \frac{1}{\underline{f} + \underline{(1-f)/p}}$$

Amdahl's Law Explained

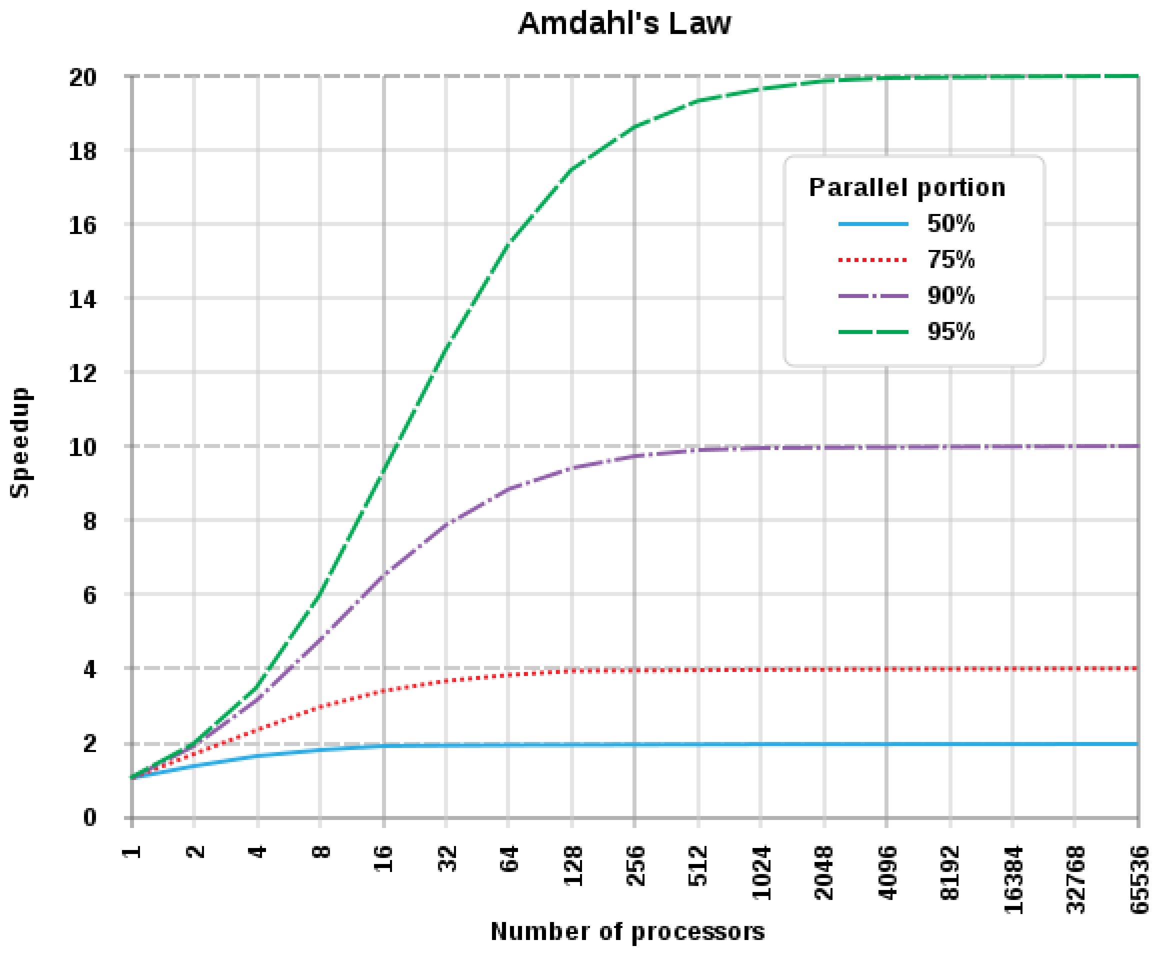
Numerator is always **1** – represents sequential execution on a single node

$$\psi \leq \frac{1}{f + (1-f)/p}$$

If code **can** be parallelised, the denominator is **small (near 0)**
⇨ **Large speedup!**

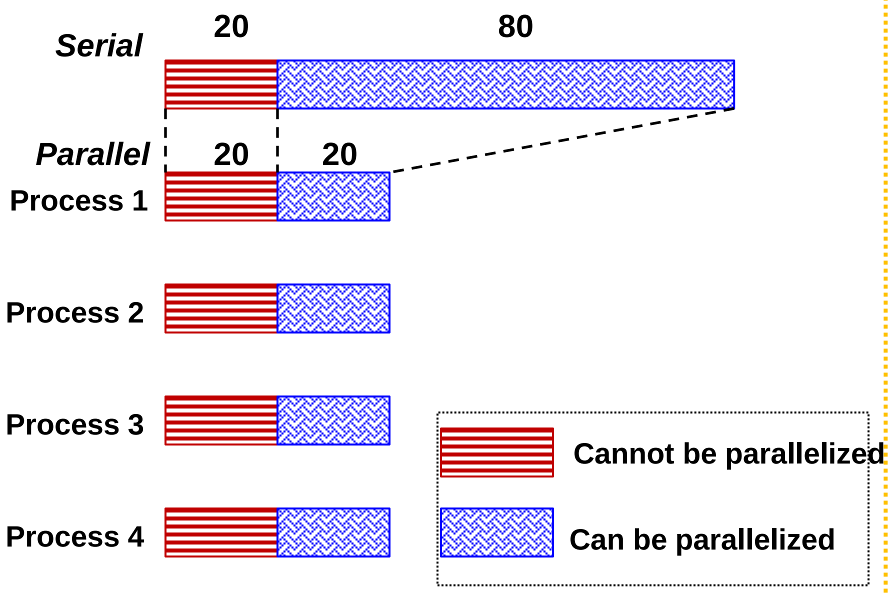
If code **cannot** be parallelised, the denominator is **large (near 1)**
⇨ **Tiny speedup!**

The benefits of higher *p* decreases...

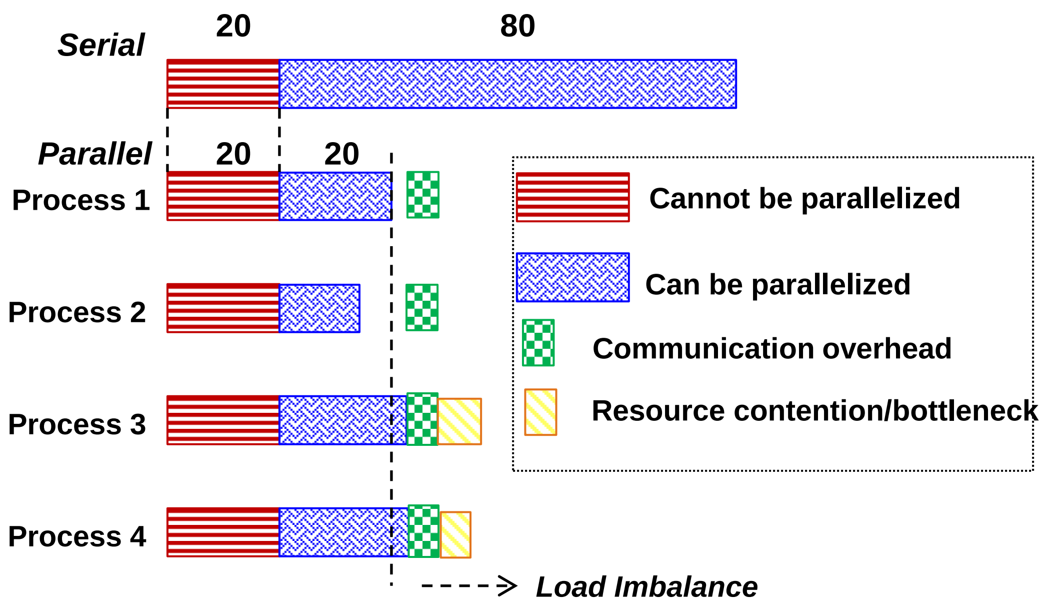


Speedup: Real vs. actual cases

- Amdahl's argument is too simplified to be applied to real cases
- When we run a parallel program, there are a communication overheads and a workload imbalance among processes in general



1. Speedup: Amdahl's Law Ideal Case



2. Speed-up: An Actual Case

Amdahl's Law on Map/Reduce jobs

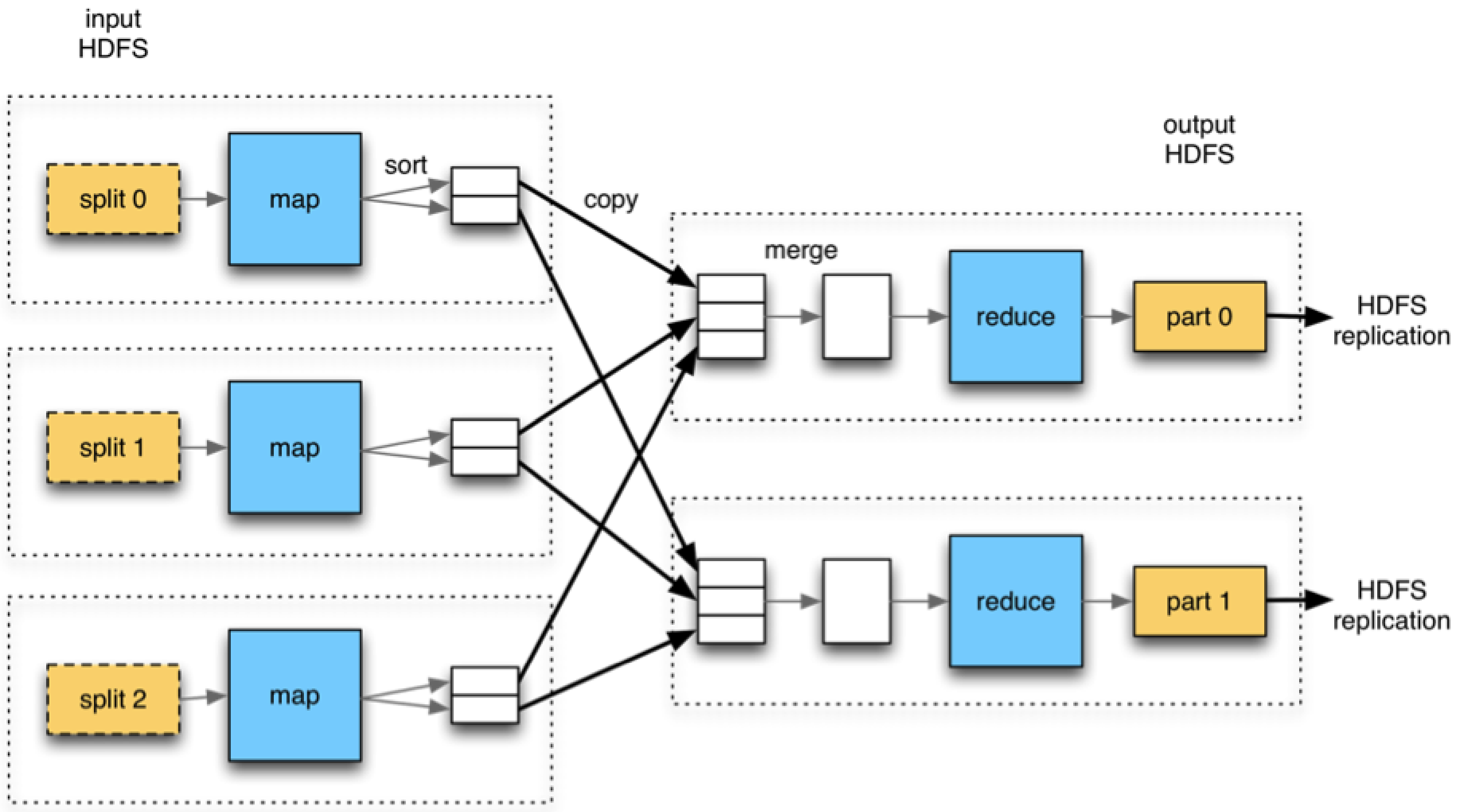
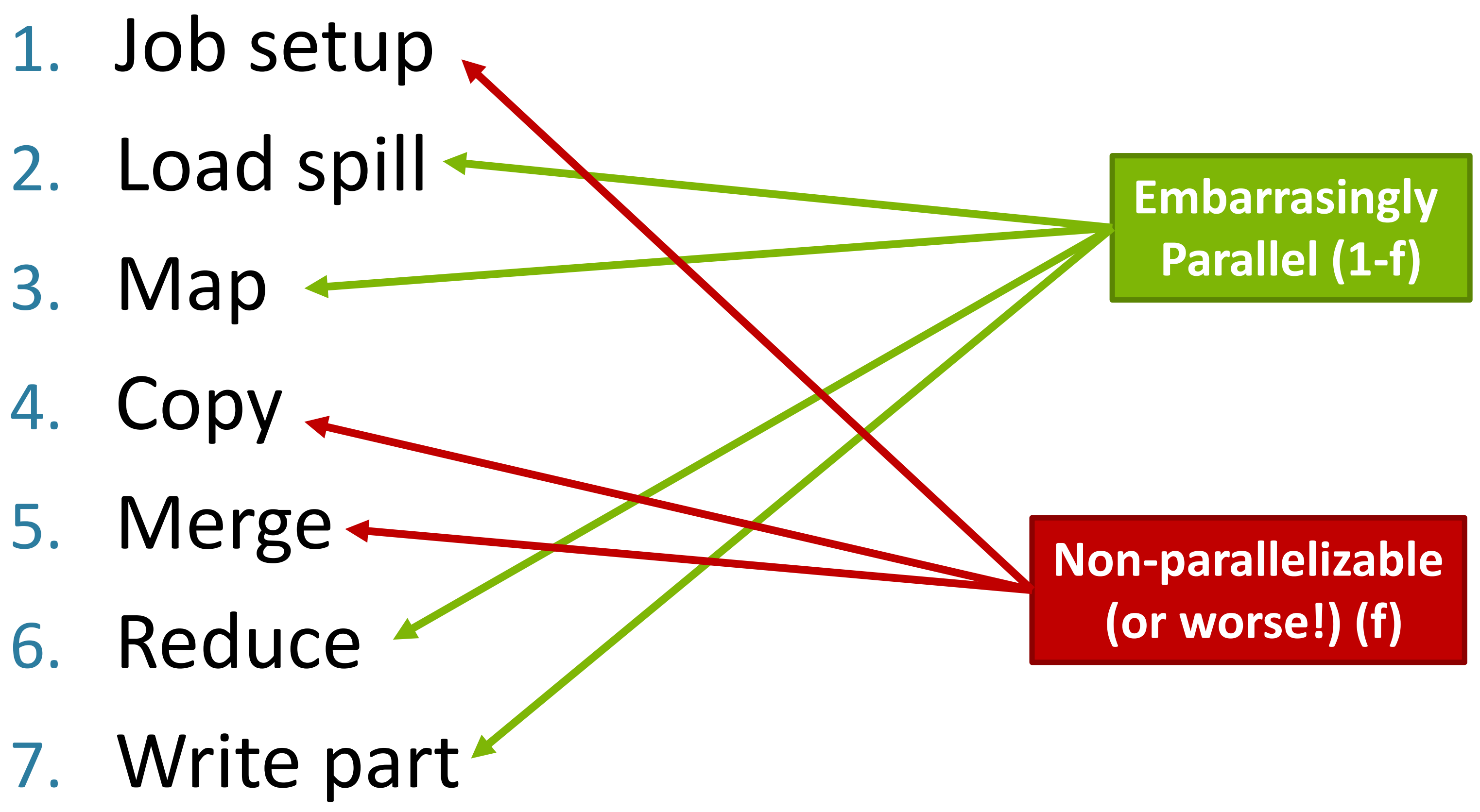


Image source: Hadoop: the definitive guide, Tom White

Amdahl's Law on Map/Reduce jobs



Indicators for Hadoop job performance

- **Latency** is the time between the start of a job and when it starts delivering output
  - In Hadoop: *total job execution time*
- **Throughput** measured in bytes/second

- High latency is entirely **compatible** with high throughput…
- …especially in something like Hadoop where we have a lot of coordination to do at the beginning

Hadoop performance overheads

- Job setup is **costly,** becomes more complex the bigger the dataset is
- Reading from HDFS takes up some CPU cycles
  - Seconds per gigabyte
- HDFS has some latency (microseconds per block read)
- Concurrent threads give lock contention
- Disks have finite throughput (MB/sec)
- Hadoop is I/O or network bound, often not CPU bound
- More at: http://www.slideshare.net/cloudera/hadoop-world-2011-hadoop-and-performance-todd-lipcon-yanpei-chen-cloudera

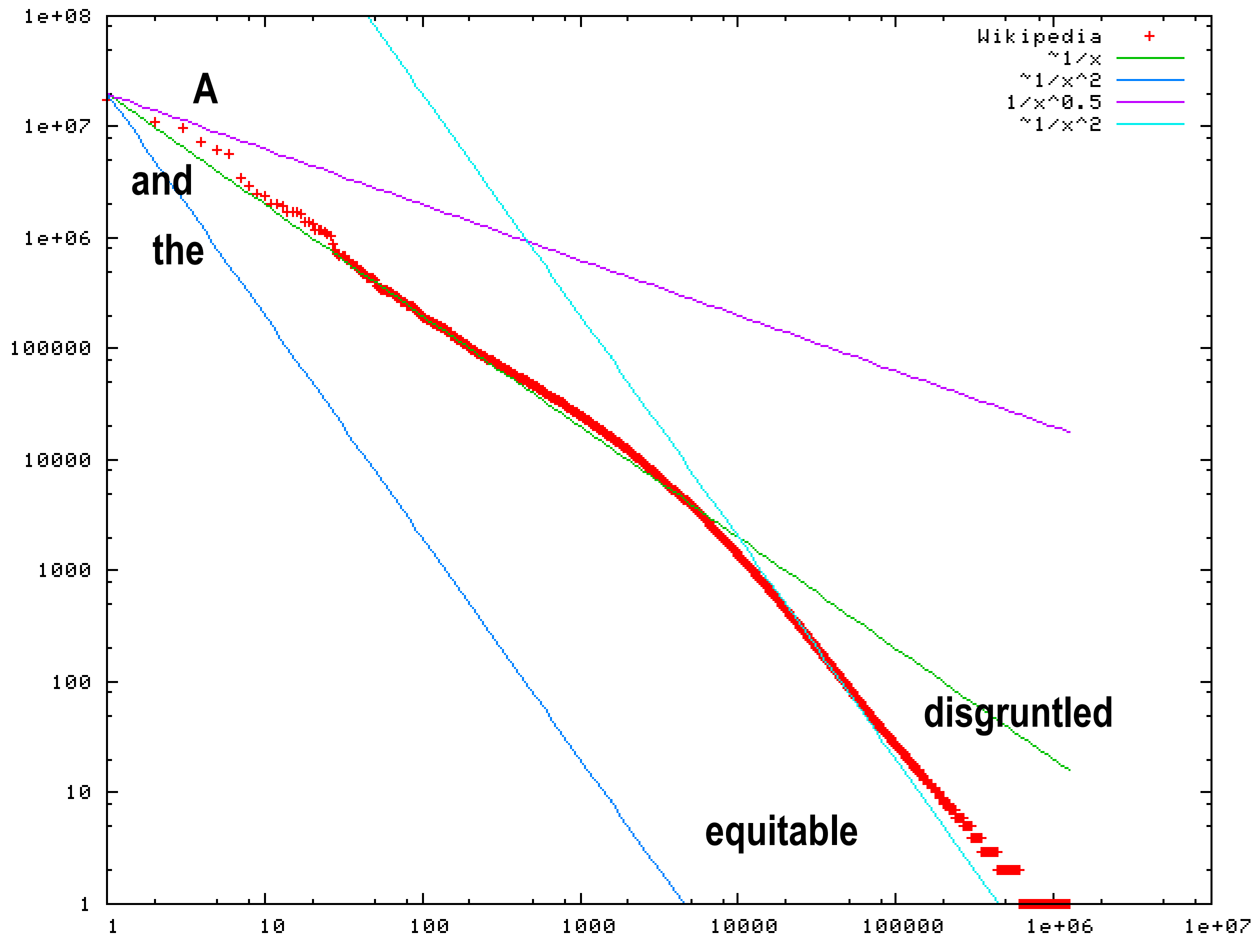
Optimization points

- For improving MapReduce job performance, focus must be on **bottlenecks**:
  - Number of keys emitted by Mappers (Combiner)
    - Reduces Map-> Reduce network transfer
    - Simplifies shuffle and sort activities at Reducer
  - Sorting in the shuffle and sort stage
    - Comparison of keys (typically with WritableComparable)
  - Avoid unnecessary Java object creation
    - E.g. reuse `Writable` objects

Load balancing problems: Data skew

- **Problem:** Not every task processes the same amount of data
  - **Mappers:** in general splits are balanced
  - **Reducers**: number of keys, number of values per key
    - E.g. Think of Word Count, partitioning by starting letter. Some are much more common than others…
- The Partitioning of Keys to Reducers can be trained to provide a balanced spread regardless of the skew
  - Initial sampling of data

Wikipedia word frequency distribution



Performance analysis of MapReduce jobs

- **Input dataset:** Size, number of records
- Average number of records generated per Mapper
  - How much information is being sent over the network
  - Effect of a Combiner?
- Number of keys/records sent to each Reducer
  - Data skew?
  - Key with too many records?

Contents

- Hadoop Performance
- **Hadoop Reliability**

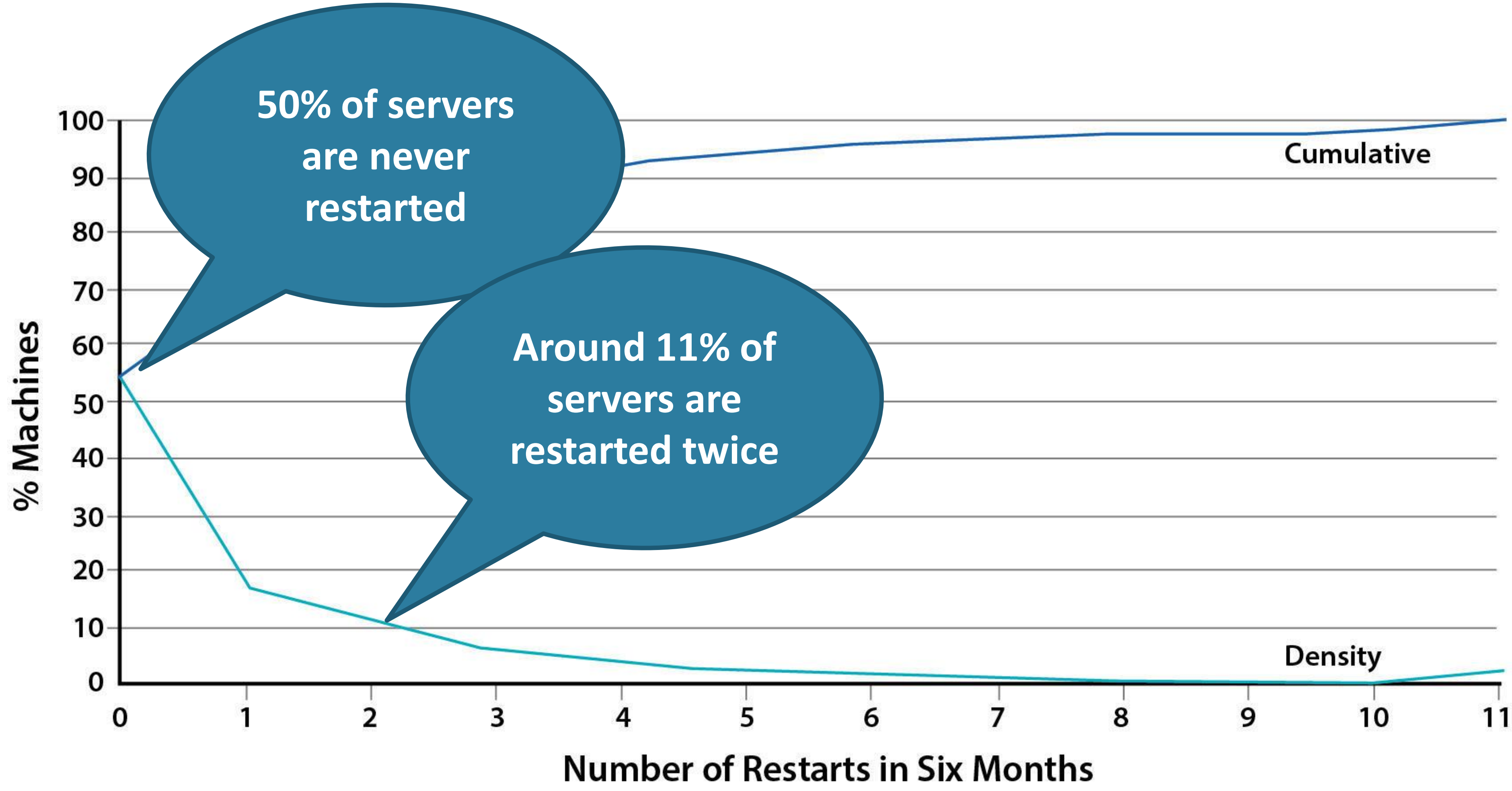
High availability

- **High availability** (HA) is a characteristic of a system
- Aims to ensure an agreed level of operational performance for a higher than normal period.
  - **Fault tolerance:** Property of a system that continues to operate correctly on the event of a failure
  - **HA** implies there are no single points of failure
  - **Graceful degradation:** When some components fail, the system temporarily works with worse performance (but still works)

High Availability measurement: counting nines

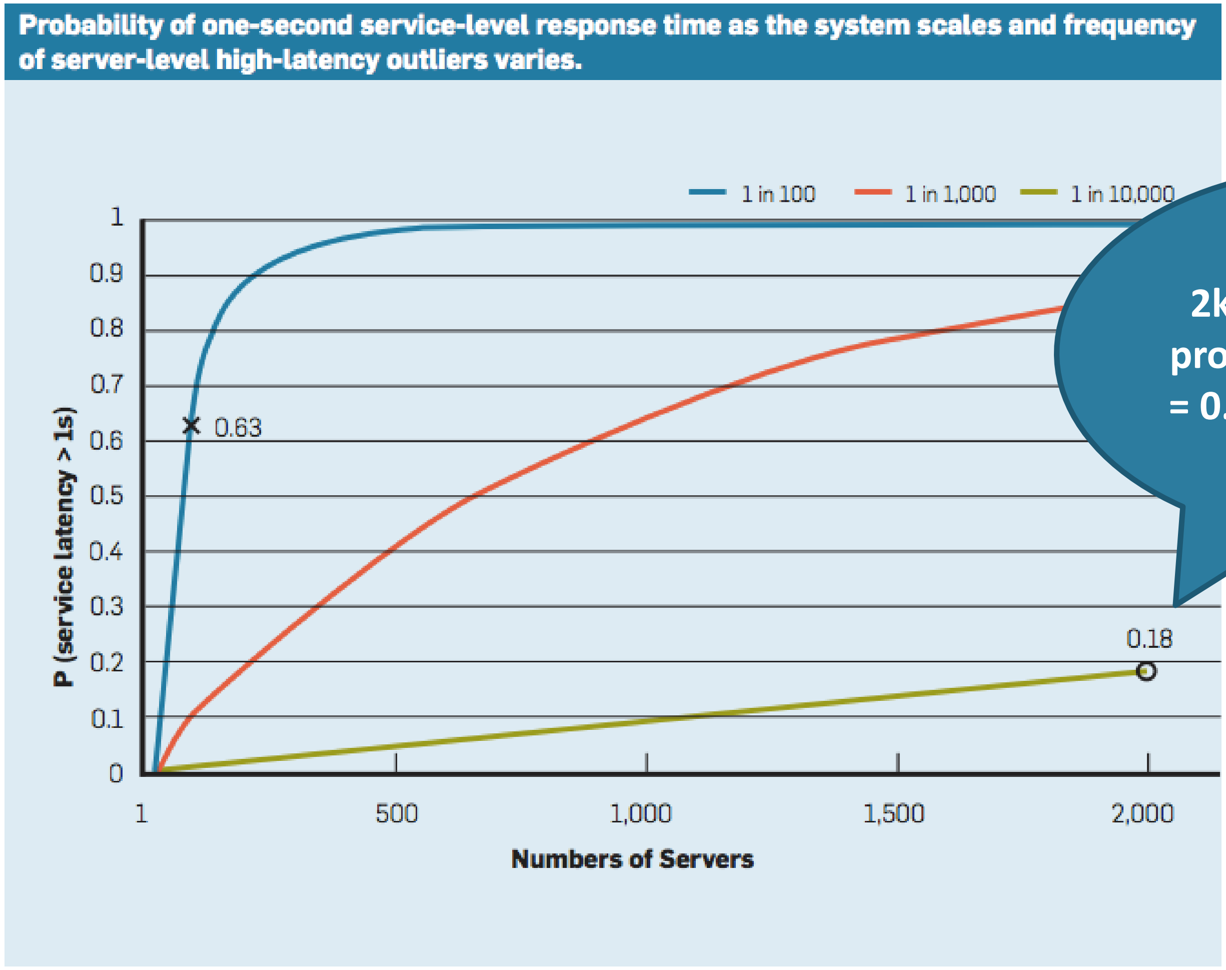
Percentage Uptime	Percentage Downtime	Downtime per year	Downtime per week
98%	2%	7.3 days	3h22m
99%	1%	3.65 days	1h41m
99.8%	0.2%	17h30m	20m10s
99.9%	0.1%	8h45m	10m5s
99.99%	0.01%	52.5m	1m
99.999%	0.001%	5.25m	6s
99.9999%	0.00001%	31.5s	0.6s

Number of machine restarts in a Google DC



From Barroso et al. Datacenter as a Computer, 2nd Ed, Morgan & Claypool, 2013

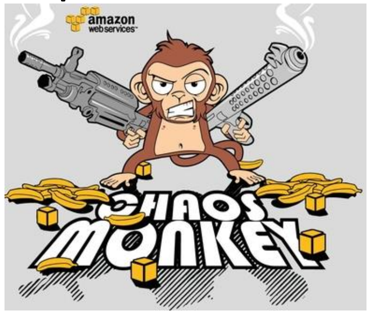
Google's Tail of Latency


Probability of one-second service-level response time as the system scales and frequency of server-level high-latency outliers varies.

2k servers + 1/10k probability of failure = 0.18 clients get >1s

Testing reliability: Netflix Chaos Monkey

- Every weekday between *9am* and *5pm*, an army of malicious programs, affectionately known as "chaos monkeys," are unleashed upon Netflix's information infrastructure.
- "Their sole purpose is to make sure that we're failing in a consistent and frequent enough way to make sure that we don't drift into overall failure," Tseitlin said. The goal is to fail often and uncover potential problems before they become actual problems.
- "The design premise there is that all of the architecture is resilient enough to retry and to begin re-serving the experience in a way that is completely transparent to the customer"



http://luckyrobot.com/netflix-chaos-monkey-keeps-movies-streaming/

Error management

- Hadoop is designed to run on commodity hardware
- **Errors are part of a job execution**
  - Data integrity error
  - Task failure
  - NodeManager failure
  - ResourceManager/ NameNode failure
- The framework is designed to detect errors and gracefully recover from them while not interrupting job execution (if possible)

Data integrity errors

- DataNodes verify checksums before writing
- HDFS clients verify checksum of read blocks
- Errors are reported to the NameNode
  1. The block is marked as corrupt
     - No more clients will read that copy of the block
  2. The client is redirected to another copy
  3. A block replica is scheduled to be copied in another node

Task failure

- Caused by software failures
    - E.g. a task requires Java 7; node has Java 6
    - E.g. the code throws a Java Exception
- Error reported back to NodeManager, task marked as failed
- Hanging tasks are detected by NodeManager and also marked as failed
- The ApplicationMaster/ResourceManager tries to reschedule the task on a different node
- There is a maximum number of retries (4 by default) before declaring job failure

NodeManager failure

- The NodeManager (NM) monitors the health of the hardware resources of the node and reports any problem to the ResourceManager (RM).
- The RM also detects NM failure by no longer receiving heartbeats from it
  - The RM marks all hosted Containers as killed, and reports the failure to the ApplicationMaster (AM)
  - The MapReduce AM will re-run all the hosted Containers in other nodes from the cluster after negotiating with RM
  - Completed Map tasks are also rescheduled to other nodes
    - Map results are not stored in the HDFS

ResourceManager/ NameNode failure

- The ResourceManager and the NameNode are the two single points of failure for Hadoop
- Currently there is no support for recovering automatically from failures on these elements if they don’t come back
- Manual reconfiguration will be required
  - The Secondary NameNode communicates periodically with NameNode and stores backup copy of index table
  - The ResourceManager stores the current state of the jobs, so it can relaunch all Containers when restarted.

Today's contents

- Hadoop Performance
- Hadoop reliability