

# Big Data Processing

## MapReduce

Albert Bifet(@abifet)



# Who am I

- Professor at Telecom ParisTech
- I work on data stream mining algorithms and systems
  - MOA: Massive Online Analytics
  - Apache SAMOA: Scalable Advanced Massive Online Analytics
- PhD: UPC BarcelonaTech, 2009
- Previous affiliations:
  - University of Waikato (New Zealand)
  - Yahoo! Labs (Barcelona)
  - Huawei (Hong Kong)

# Big Data

**BIG DATA** are data sets so large or complex that traditional data processing applications can not deal with.

# Big Data

**BIG DATA** are data sets so large or complex that traditional data processing applications can not deal with.

**BIG DATA** is an OPEN SOURCE Software Revolution.

# Digital Universe

EMC Digital Universe with Research &  
Analysis by IDC

**The Digital Universe of Opportunities:  
Rich Data and the Increasing Value of the  
Internet of Things**  
April 2014

# Digital Universe



Figure: EMC Digital Universe, 2014

# Digital Universe

1blue!20blue!5

Memory unit	Size	Binary size
kilobyte (kB/KB)	$10^3$	$2^{10}$
megabyte (MB)	$10^6$	$2^{20}$
gigabyte (GB)	$10^9$	$2^{30}$
terabyte (TB)	$10^{12}$	$2^{40}$
petabyte (PB)	$10^{15}$	$2^{50}$
exabyte (EB)	$10^{18}$	$2^{60}$
zettabyte (ZB)	$10^{21}$	$2^{70}$
yottabyte (YB)	$10^{24}$	$2^{80}$

# Digital Universe

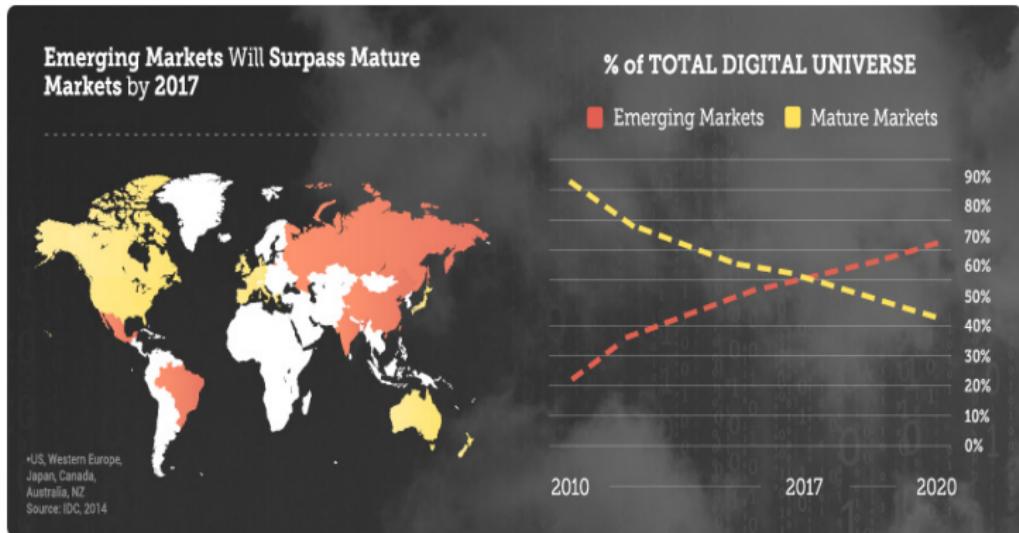


Figure: EMC Digital Universe, 2014

# Digital Universe

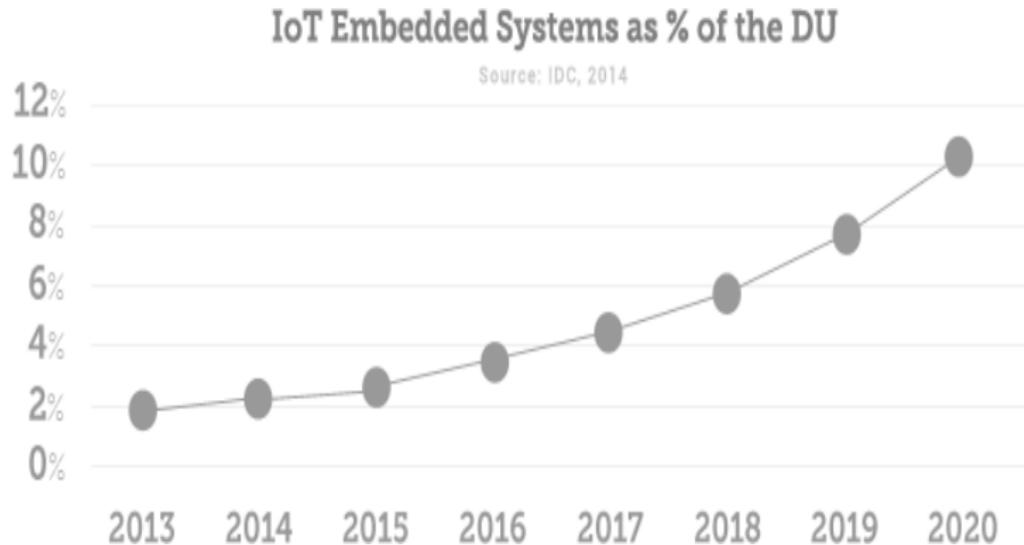


Figure: EMC Digital Universe, 2014

# Digital Universe

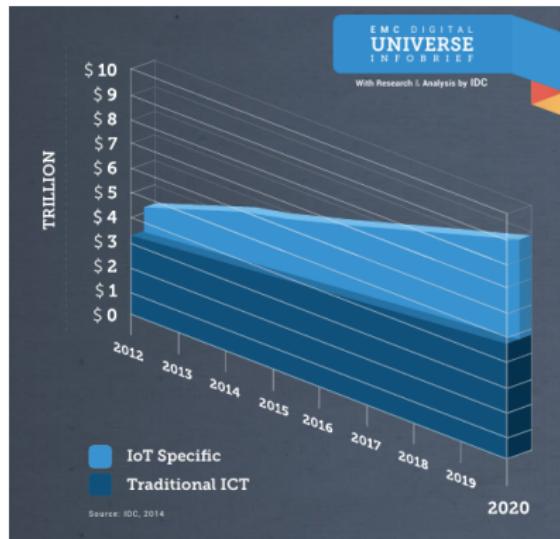


Figure: EMC Digital Universe, 2014

# Digital Universe

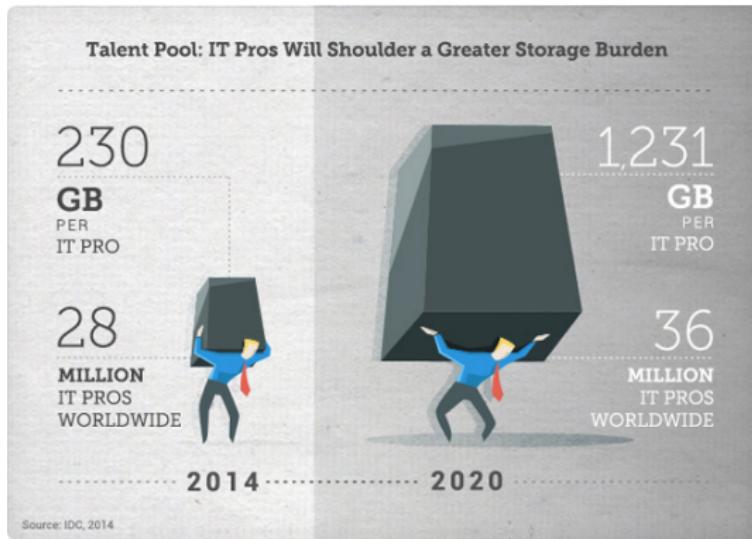


Figure: EMC Digital Universe, 2014

# Big Data 6V's

- Volume
- Variety
- Velocity
- Value
- Variability
- Veracity

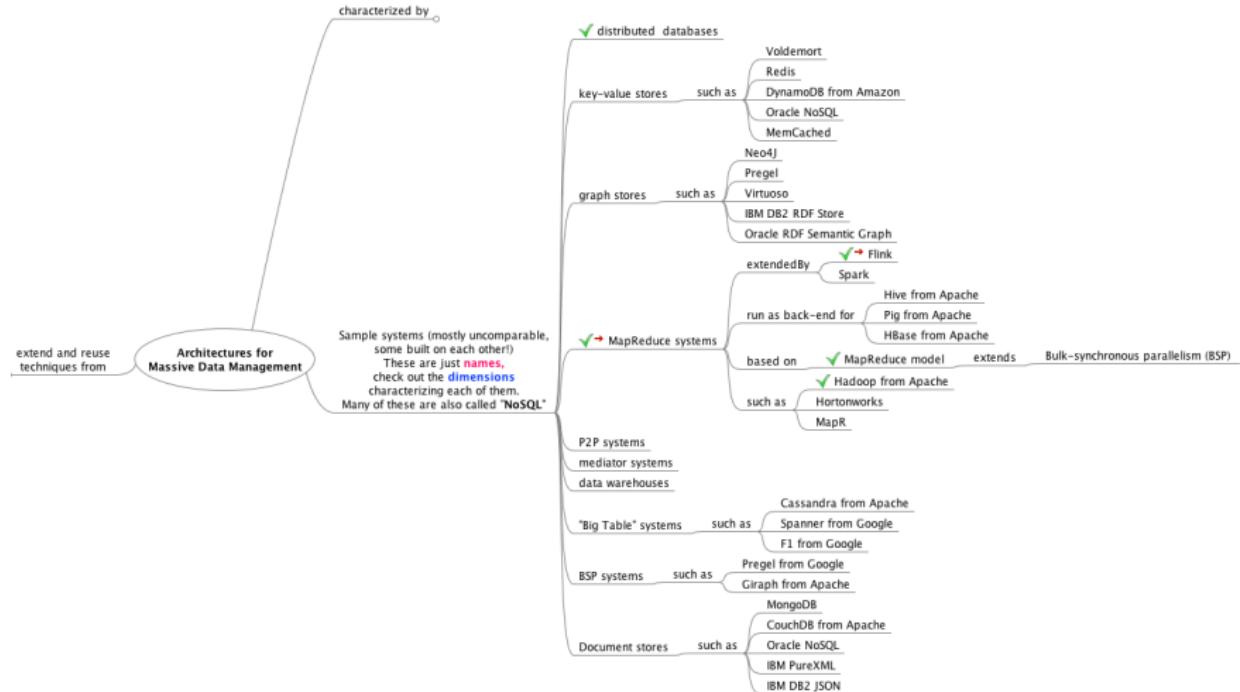
# Controversy of Big Data

- All data is BIG now
- Hype to sell Hadoop based systems
- Ethical concerns about accessibility
- Limited access to Big Data creates new digital divides
- Statistical Significance:
  - When the number of variables grow, the number of fake correlations also grow Leinweber: S&P 500 stock index correlated with butter production in Bangladesh

# Future Challenges for Big Data

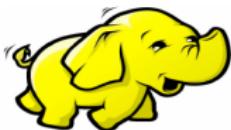
- Evaluation
- Time evolving data
- Distributed mining
- Compression
- Visualization
- Hidden Big Data

# Big Data Ecosystem



# Batch and Streaming Engines

**Batch only**



**Streaming only**



**Hybrid**



Figure: Batch, streaming and hybrid data processing engines.

# Motivation MapReduce

# How Many Servers Does Google Have?

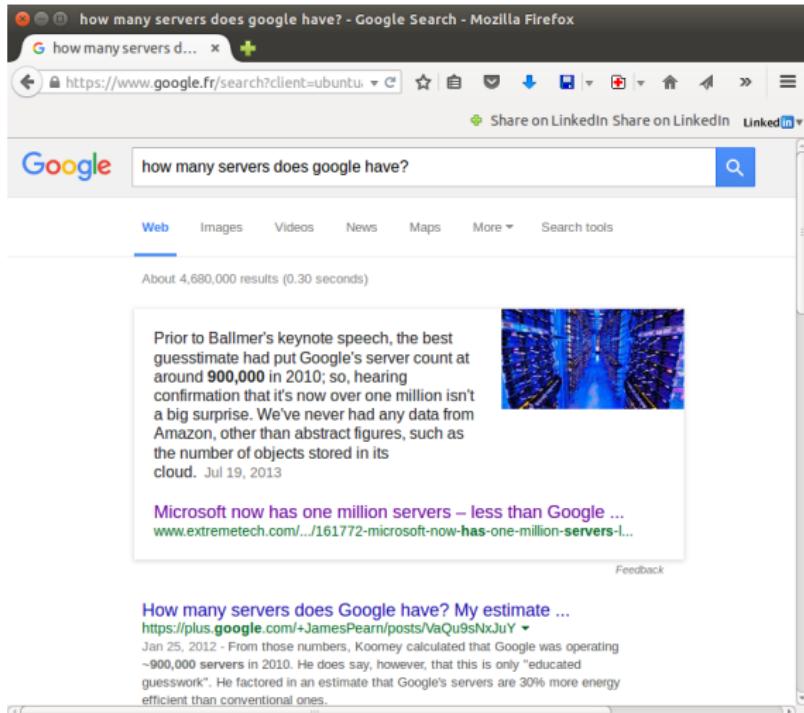


Figure: Asking Google

# A Google Server Room

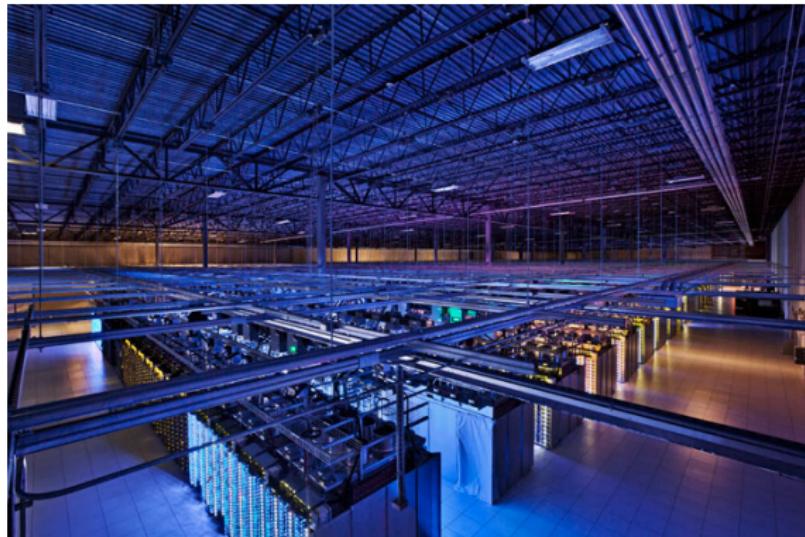


Figure: <https://www.youtube.com/watch?t=3&v=avP5d16wEp0>

# Typical Big Data Challenges

- How do we break up a large problem into smaller tasks that can be executed in parallel?
- How do we assign tasks to workers distributed across a potentially large number of machines?
- How do we ensure that the workers get the data they need?
- How do we coordinate synchronization among the different workers?
- How do we share partial results from one worker that is needed by another?
- How do we accomplish all of the above in the face of software errors and hardware faults?

There was need for an abstraction that hides many system-level details from the programmer.

There was need for an abstraction that hides many system-level details from the programmer.

**MapReduce** addresses this challenge by providing a simple abstraction for the developer, transparently handling most of the details behind the scenes in a scalable, robust, and efficient manner.

# Jeff Dean



## MapReduce, BigTable, Spanner

*MapReduce: Simplified Data Processing on Large Clusters*

Jeffrey Dean and Sanjay Ghemawat

OSDI'04: Sixth Symposium on Operating System Design and Implementation

# Jeff Dean Facts



## Google Culture Facts

"When Jeff Dean designs software, he first codes the binary and then writes the source as documentation."

"Jeff Dean compiles and runs his code before submitting, but only to check for compiler and CPU bugs."

# Jeff Dean Facts



## Google Culture Facts

"The rate at which Jeff Dean produces code jumped by a factor of 40 in late 2000 when he upgraded his keyboard to USB2.0."

"The speed of light in a vacuum used to be about 35 mph. Then Jeff Dean spent a weekend optimizing physics."

# Jeff Dean Facts

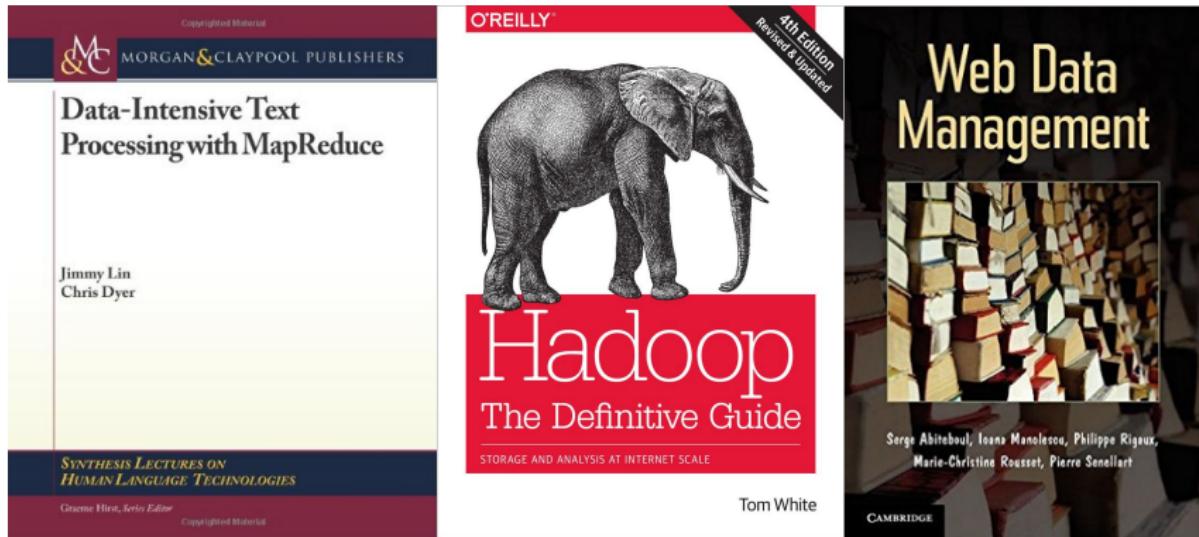


## Google Culture Facts

Compilers don't warn Jeff Dean. Jeff Dean warns compilers

# MapReduce

# References



# Numbers Everyone Should Know (Jeff Dean)

L1 cache reference	0.5 ns
Branch mispredict	5 ns
L2 cache reference	7 ns
Mutex lock/unlock	100 ns
Main memory reference	100 ns
Compress 1K bytes with Zippy	10,000 ns
Send 2K bytes over 1 Gbps network	20,000 ns
Read 1 MB sequentially from memory	250,000 ns
Round trip within same datacenter	500,000 ns
Disk seek	10,000,000 ns
Read 1 MB sequentially from network	10,000,000 ns
Read 1 MB sequentially from disk	30,000,000 ns
Send packet CA to Netherlands to CA	150,000,000 ns

# Typical Big Data Problem

- Iterate over a large number of records
- Extract something of interest from each
- Shuffle and sort intermediate results
- Aggregate intermediate results
- Generate final output

# Typical Big Data Problem

- Iterate over a large number of records
- Extract something of interest from each **-MAP-**
- Shuffle and sort intermediate results
- Aggregate intermediate results **-REDUCE-**
- Generate final output

# Functional Programming

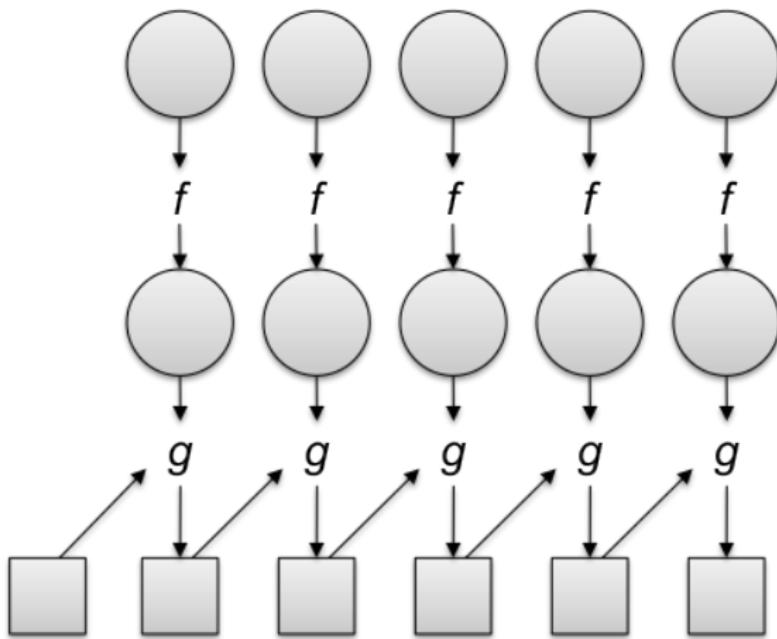


Figure: **Map** as a transformation function and **Fold** as an aggregation function

# Map and Reduce functions

- In MapReduce, the programmer defines the program logic as two functions:
  - map:  $(k_1, v_1) \rightarrow list[(k_2, v_2)]$ 
    - Map transforms the input into key-value pairs to process
  - reduce:  $(k_2, list[v_2]) \rightarrow list[(k_3, v_3)]$ 
    - Reduce aggregates the list of values for each key
- The MapReduce environment takes in charge distribution aspects.
- A complex program can be decomposed as a succession of Map and Reduce tasks

# Simplified view of MapReduce

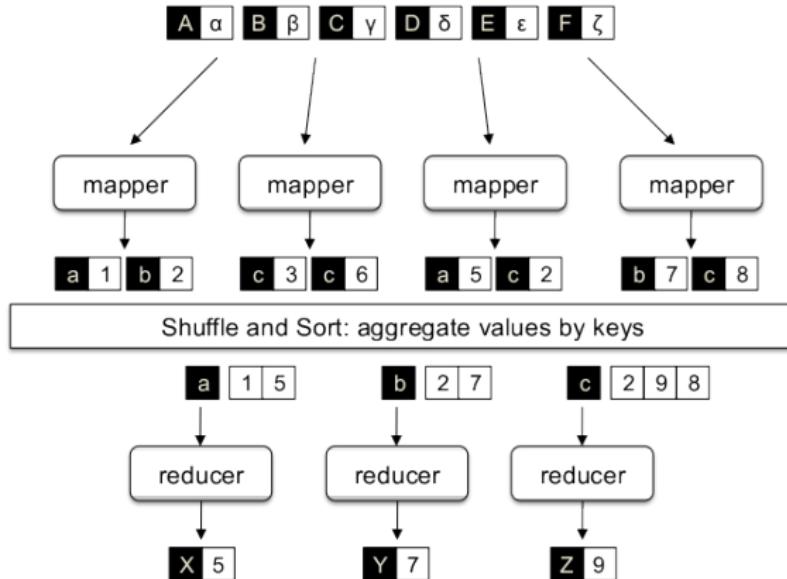


Figure: Two-stage processing structure

# An Example Application: Word Count

## Input Data

```
foo.txt: Sweet, this is the foo file  
bar.txt: This is the bar file
```

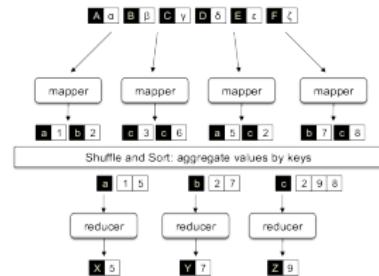
## Output Data

```
sweet 1  
this 2  
is 2  
the 2  
foo 1  
bar 1  
file 2
```

# WordCount Example

```
1: class MAPPER  
2:     method MAP(docid a, doc d)  
3:         for all term t ∈ doc d do  
4:             EMIT(term t, count 1)  
5:         end for  
6:     end method  
7: end class
```

```
1: class REDUCER  
2:     method REDUCE(term t, counts [c1, c2, ...])  
3:         sum ← 0  
4:         for all count c ∈ counts [c1, c2, ...] do  
5:             sum ← sum + c  
6:         end for  
7:         EMIT(term t, count sum)  
8:     end method  
9: end class
```



# Simple MapReduce Variations

## No Reducers

# Simple MapReduce Variations

## No Reducers

Each mapper output is directly written to a file disk

# Simple MapReduce Variations

## No Reducers

Each mapper output is directly written to a file disk

## No Mappers

# Simple MapReduce Variations

## No Reducers

Each mapper output is directly written to a file disk

## No Mappers

Not possible!

## Identity Function Mappers

Sorting and regrouping the input data

# Simple MapReduce Variations

## No Reducers

Each mapper output is directly written to a file disk

## No Mappers

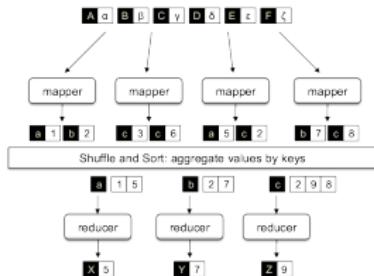
Not possible!

## Identity Function Mappers

Sorting and regrouping the input data

## Identity Function Reducers

Sorting and regrouping the data from mappers



# MapReduce Framework

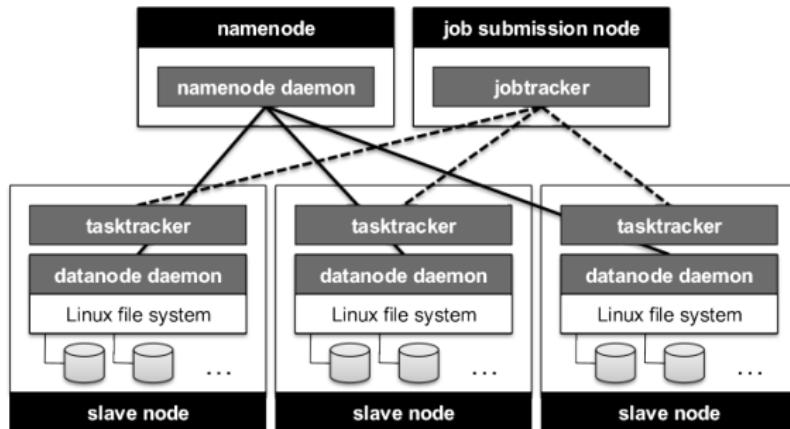


Figure: Runtime Framework

# MapReduce Framework

- Handles scheduling
  - Assigns workers to map and reduce tasks
- Handles “data distribution”
  - Moves processes to data
- Handles synchronization
  - Gathers, sorts, and shuffles intermediate data
- Handles errors and faults
  - Detects worker failures and restarts
- Everything happens on top of a distributed filesystem

# Fault Tolerance

The Master periodically checks the availability and reachability of the tasktrackers (heartbeats) and whether map or reduce jobs make any progress

- if a mapper fails, its task is reassigned to another tasktracker
- if a reducer fails, its task is reassigned to another tasktracker; this usually require restarting mapper tasks as well (to produce intermediate groups)
- if the jobtracker fails, the whole job should be re-initiated

Speculative execution: schedule redundant copies of the remaining tasks across several nodes

# Complete MapReduce Framework

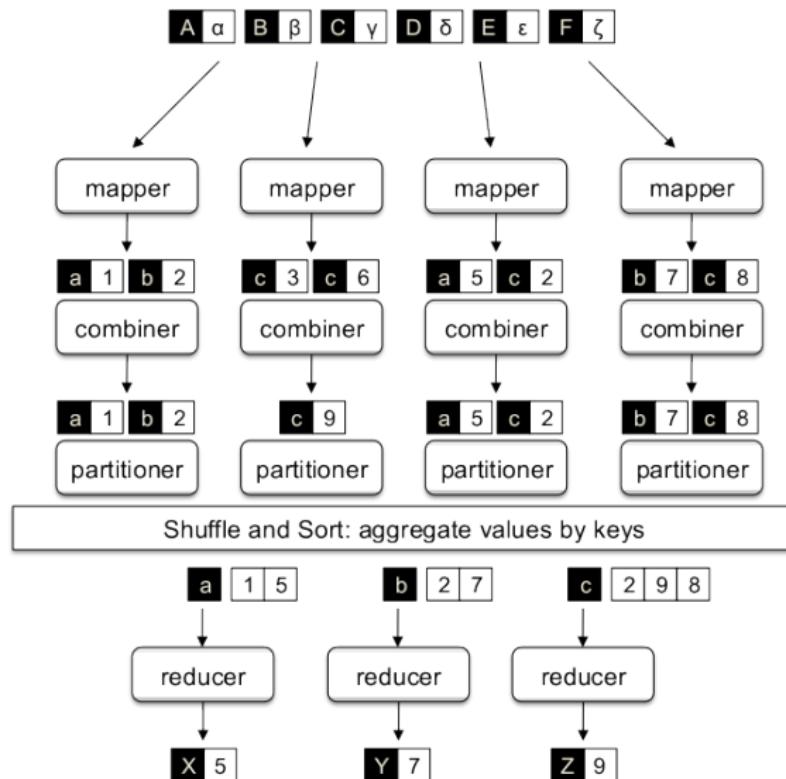


Figure: Partitioners and Combiners

# Partitioners and Combiners

## Partitioners

Divide up the intermediate key space and assign intermediate key-value pairs to reducers: “**simple hash of the key**”

*partition:  $(k, \text{number of partitions}) \rightarrow \text{partition for } k$*

## Combiners

Optimization in MapReduce that allow for local aggregation before the shuffle and sort phase: “**mini-reducers**”

*combine:  $(k_2, \text{list}[v_2]) \rightarrow \text{list}[(k_3, v_3)]$*

Run in memory, and their goal is to reduce network traffic.

# Apache Hadoop

# Origins of Apache Hadoop



- Hadoop was created by Doug Cutting (Apache Lucene) when he was building Apache Nutch, an open source web search engine.
- Cutting was an employee of **Yahoo!**, where he led the Hadoop project.
- The name comes from a favorite stuffed elephant of his son.

# Initial Differences between Hadoop MapReduce and Google MapReduce

- In Hadoop MapReduce, the list of values that arrive to the reducers are not ordered. In Google MapReduce it is possible to specify a secondary sort key for ordering the values.
- In Google MapReduce reducers, the output key should be the same as the input key. Hadoop MapReduce reducers can output different key-value pairs (with different keys to the input key)
- In Google MapReduce mappers output to combiners, and in Hadoop MapReduce mappers output to partitioners.

# What Is Apache Hadoop?



The Apache Hadoop project develops open-source software for reliable, scalable, distributed computing.

It includes these modules:

- Hadoop Common: The common utilities that support the other Hadoop modules.
- Hadoop Distributed File System (HDFS): A distributed file system that provides high-throughput access to application data.
- Hadoop YARN: A framework for job scheduling and cluster resource management.
- Hadoop MapReduce: A YARN-based system for parallel processing of large data sets

# Hadoop v2

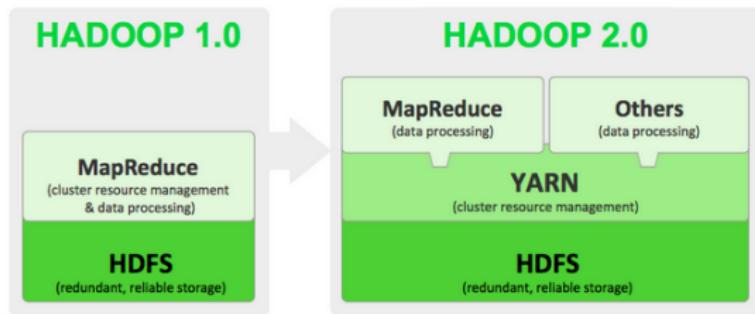


Figure: Apache Hadoop NextGen MapReduce (YARN)

# Apache Hadoop NextGen MapReduce (YARN)

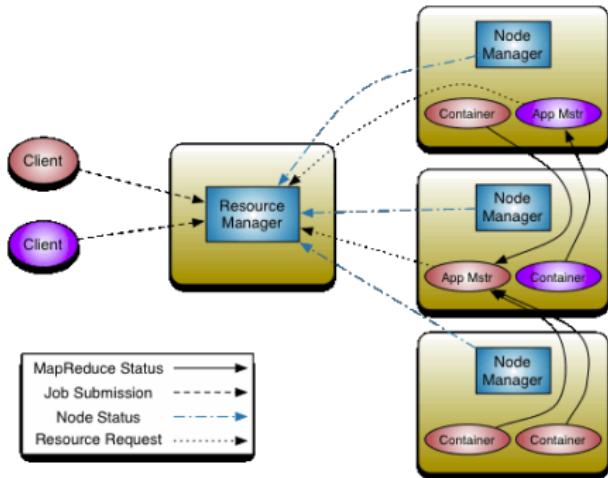


Figure: MRv2 splits up the two major functionalities of the JobTracker, resource management and job scheduling/monitoring, into separate daemons. An application is either a single job in the classical sense of Map-Reduce jobs or a DAG of jobs.

# Apache Hadoop NextGen MapReduce (YARN)

In YARN, the ResourceManager has two main components:

- The **Scheduler**: responsible for allocating resources to the various running applications subject to familiar constraints of capacities, queues etc.
- The **ApplicationsManager**: responsible for accepting job-submissions, negotiating the first container for executing the application specific ApplicationMaster and provides the service for restarting the ApplicationMaster container on failure.

# The Hadoop Distributed File System HDFS

## Assumptions and Goals

- Hardware Failure
- Streaming Data Access
- Large Data Sets
- Simple Coherency Model ( write-once-read-many access model)
- “Moving Computation is Cheaper than Moving Data”
- Portability Across Heterogeneous Hardware and Software Platforms

# The Distributed File System

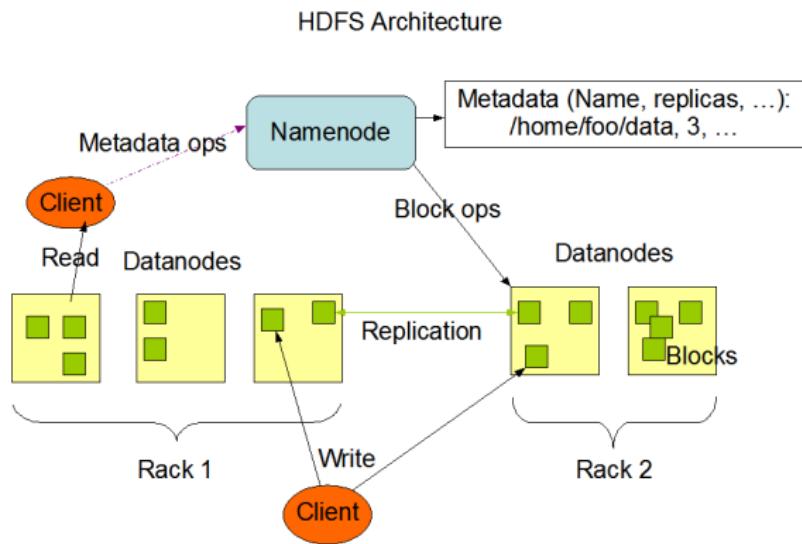


Figure: Distributed File System Architecture

# The Distributed File System

## Block Replication

Namenode (Filename, numReplicas, block-ids, ...)  
/users/sameerp/data/part-0, r:2, {1,3}, ...  
/users/sameerp/data/part-1, r:3, {2,4,5}, ...

## Datanodes

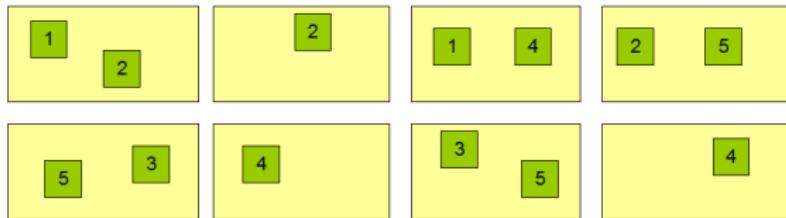


Figure: Block Replication

# An Example Application: Word Count

## Input Data

```
foo.txt: Sweet, this is the foo file  
bar.txt: This is the bar file
```

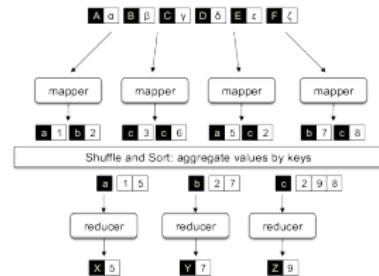
## Output Data

```
sweet 1  
this 2  
is 2  
the 2  
foo 1  
bar 1  
file 2
```

# WordCount Example

```
1: class MAPPER  
2:     method MAP(docid a, doc d)  
3:         for all term t ∈ doc d do  
4:             EMIT(term t, count 1)  
5:         end for  
6:     end method  
7: end class
```

```
1: class REDUCER  
2:     method REDUCE(term t, counts [c1, c2, ...])  
3:         sum ← 0  
4:         for all count c ∈ counts [c1, c2, ...] do  
5:             sum ← sum + c  
6:         end for  
7:         EMIT(term t, count sum)  
8:     end method  
9: end class
```



# Mapper Java Code

```
public static class TokenizerMapper  
    extends Mapper<Object, Text, Text, IntWritable> {  
  
    private final static IntWritable one = new IntWritable(1);  
    private Text word = new Text();  
  
    public void map(Object key, Text value, Context context  
                    ) throws IOException, InterruptedException {  
        StringTokenizer itr = new StringTokenizer(value.toString());  
        while (itr.hasMoreTokens()) {  
            word.set(itr.nextToken());  
            context.write(word, one);  
        }  
    }  
}
```

# Reducer Java Code

```
public static class IntSumReducer
    extends Reducer<Text, IntWritable, Text, IntWritable> {
    private IntWritable result = new IntWritable();

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

# Driver Java Code

```
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

# Hadoop MapReduce data flow

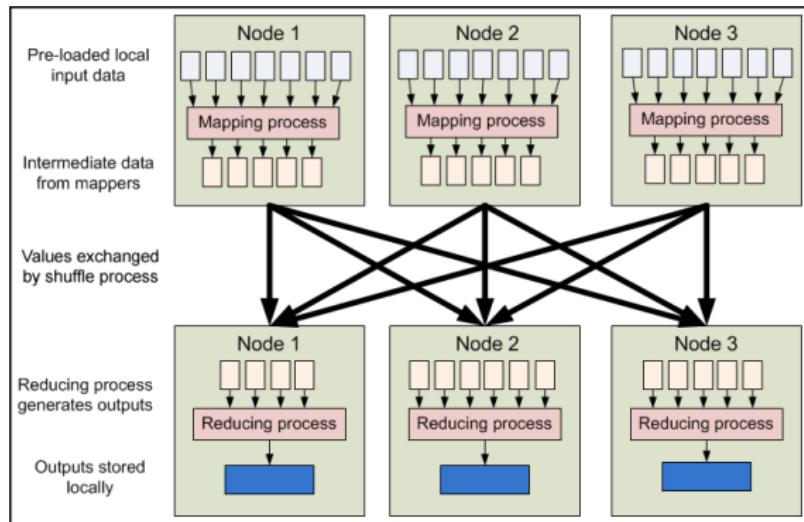


Figure: High-level MapReduce pipeline

# Hadoop MapReduce data flow

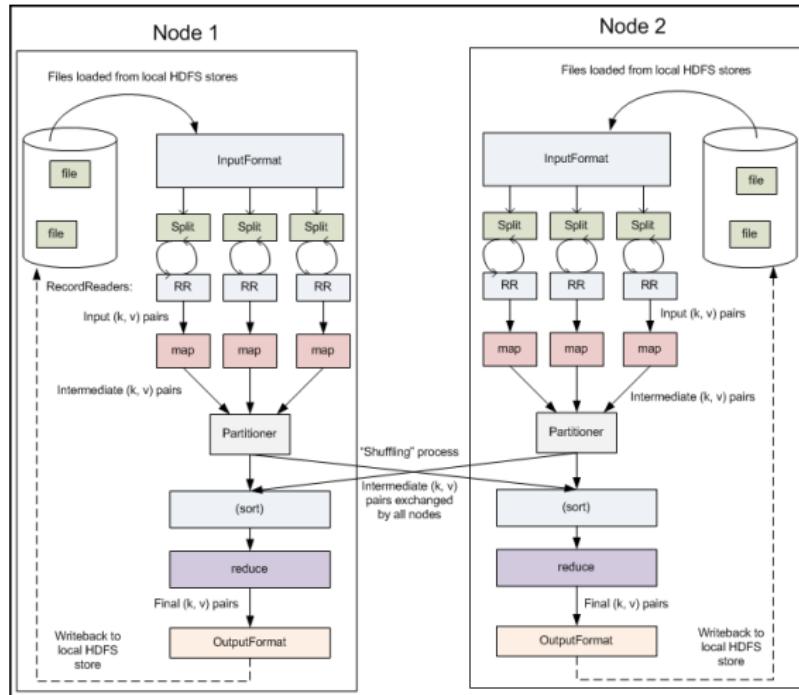


Figure: Detailed Hadoop MapReduce data flow

# Hadoop MapReduce data flow

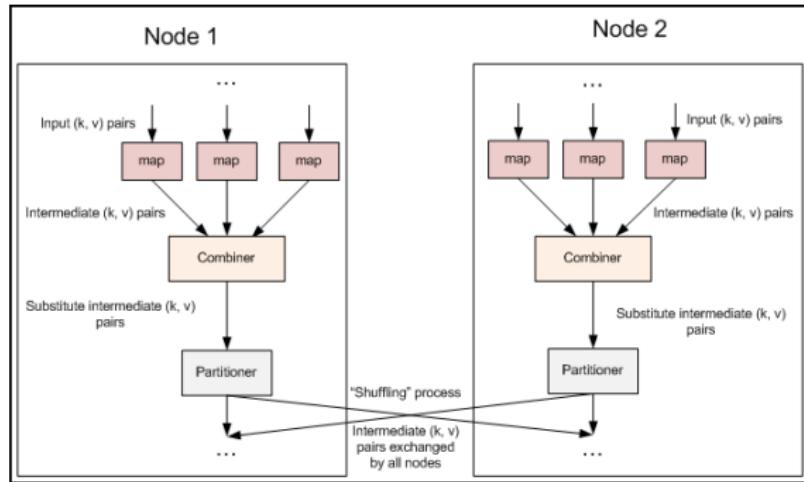


Figure: Combiner step inserted into the MapReduce data flow

# MapReduce Algorithms

# Simple MapReduce Algorithms

## Distributed Grep

- Grep: reports matching lines on input files
  - Split all files across the nodes
  - Map: emits a line if it matches the specified pattern
  - Reduce: identity function

## Count of URL Access Frequency

- Processing logs of web access
  - Map: outputs  $\langle \text{URL}, 1 \rangle$
  - Reduce: Adds together and outputs  $\langle \text{URL}, \text{Total Count} \rangle$

# Simple MapReduce Algorithms

## Reverse Web-Link Graph

- Computes source list of web pages linked to target URLs
  - Map: outputs  $\langle \text{target}, \text{source} \rangle$
  - Reduce: Concatenates together and outputs  $\langle \text{target}, \text{list(source)} \rangle$

## Inverted Index

- Build an inverted index
  - Map: emits a sequence of  $\langle \text{word}, \text{docID} \rangle$
  - Reduce: outputs  $\langle \text{word}, \text{list(docID)} \rangle$

# Joins in MapReduce

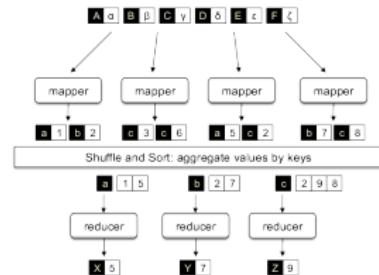
Two datasets, A and B that we need to join for a MapReduce task

- If one of the dataset is small, it can be sent over fully to each tasktracker and exploited inside the map (and possibly reduce) functions
- Otherwise, each dataset should be grouped according to the join key, and the result of the join can be computing in the reduce function

# WordCount Example Revised

```
1: class MAPPER  
2:   method MAP(docid a, doc d)  
3:     for all term t ∈ doc d do  
4:       EMIT(term t, count 1)  
5:     end for  
6:   end method  
7: end class
```

```
1: class REDUCER  
2:   method REDUCE(term t, counts [c1, c2, ...])  
3:     sum ← 0  
4:     for all count c ∈ counts [c1, c2, ...] do  
5:       sum ← sum + c  
6:     end for  
7:     EMIT(term t, count sum)  
8:   end method  
9: end class
```



# WordCount Example Revisited

```
1: class MAPPER
2:   method MAP(docid a, doc d)
3:     for all term t ∈ doc d do
4:       EMIT(term t, count 1)
5:     end for
6:   end method
7: end class
```

```
1: class MAPPER
2:   method MAP(docid a, doc d)
3:     H ← new ASSOCIATIVEARRAY
4:     for all term t ∈ doc d do
5:       H{t} ← H{t} + 1      ▷ Tally counts for entire document
6:     end for
7:     for all term t ∈ H do
8:       EMIT(term t, count H{t})
9:     end for
10:   end method
11: end class
```

# WordCount Example Revised

```
1: class MAPPER
2:   method INITIALIZE
3:      $H \leftarrow \text{new ASSOCIATIVEARRAY}$ 
4:   end method
5:   method MAP(docid a, doc d)
6:     for all term  $t \in \text{doc } d$  do
7:        $H\{t\} \leftarrow H\{t\} + 1$        $\triangleright$  Tally counts across documents
8:     end for
9:   end method
10:  method CLOSE
11:    for all term  $t \in H$  do
12:      EMIT(term  $t$ , count  $H\{t\}$ )
13:    end for
14:  end method
15: end class
```

Word count mapper using the “in-mapper combining”.

# Average Computing Example

## Example

Given a large number of key-values pairs, where

- keys are strings
- values are integers

find all average of values by key

## Example

- Input: <‘‘a’’,1>, <‘‘b’’,2>, <‘‘c’’,10>, <‘‘b’’,4>, <‘‘a’’,7>
- Output: <‘‘a’’,4>, <‘‘b’’,3>, <‘‘c’’,10>

# Average Computing Example

```
1: class MAPPER
2:   method MAP(string t, integer r)
3:     EMIT(string t, integer r)
4:   end method
5: end class
1: class REDUCER
2:   method REDUCE(string t, integers [r1, r2, ...])
3:     sum  $\leftarrow$  0
4:     cnt  $\leftarrow$  0
5:     for all integer r  $\in$  integers [r1, r2, ...] do
6:       sum  $\leftarrow$  sum + r
7:       cnt  $\leftarrow$  cnt + 1
8:     end for
9:     ravg  $\leftarrow$  sum/cnt
10:    EMIT(string t, integer ravg)
11:  end method
12: end class
```

# Average Computing Example

## Example

Given a large number of key-values pairs, where

- keys are strings
- values are integers

find all average of values by key

**Average computing is not associative**

- $\text{average}(1,2,3,4,5) \neq \text{average}(\text{average}(1,2), \text{average}(3,4,5))$
- $3 \neq \text{average}(1.5, 4) = 2.75$

# Average Computing Example

```
1: class MAPPER
2:     method MAP(string t, integer r)
3:         EMIT(string t, pair (r, 1))
4:     end method
5: end class
6: class COMBINER
7:     method COMBINE(string t, pairs [(s1, c1), (s2, c2) . . .])
8:         sum ← 0
9:         cnt ← 0
10:        for all pair (s, c) ∈ pairs [(s1, c1), (s2, c2) . . .] do
11:            sum ← sum + s
12:            cnt ← cnt + c
13:        end for
14:        EMIT(string t, pair (sum, cnt))
15:    end method
16: end class
17: class REDUCER
18:     method REDUCE(string t, pairs [(s1, c1), (s2, c2) . . .])
19:         sum ← 0
20:         cnt ← 0
21:         for all pair (s, c) ∈ pairs [(s1, c1), (s2, c2) . . .] do
22:             sum ← sum + s
23:             cnt ← cnt + c
24:         end for
25:         ravg ← sum/cnt
26:         EMIT(string t, integer ravg)
27:     end method
28: end class
```

# Monoidify!

## Monoids as a Design Principle for Efficient MapReduce Algorithms (Jimmy Lin)

Given a set  $S$ , an operator  $\oplus$  and an identity element  $e$ , for all  $a, b, c$  in  $S$ :

- Closure:  $a \oplus b$  is also in  $S$ .
- Associativity:  $a \oplus (b \oplus c) = (a \oplus b) \oplus c$
- Identity:  $e \oplus a = a \oplus e = e$

# Average Computing Example

```
1: class MAPPER
2:   method INITIALIZE
3:      $S \leftarrow \text{new ASSOCIATIVEARRAY}$ 
4:      $C \leftarrow \text{new ASSOCIATIVEARRAY}$ 
5:   end method
6:   method MAP(string  $t$ , integer  $r$ )
7:      $S\{t\} \leftarrow S\{t\} + r$ 
8:      $C\{t\} \leftarrow C\{t\} + 1$ 
9:   end method
10:  method CLOSE
11:    for all term  $t \in S$  do
12:      EMIT(term  $t$ , pair ( $S\{t\}$ ,  $C\{t\}$ ))
13:    end for
14:  end method
15: end class
```

# Compute word co-occurrence matrices

Problem of building word co-occurrence matrices from large corpora

- The co-occurrence matrix of a corpus is a square  $n \times n$  matrix where  $n$  is the number of unique words in the corpus (i.e., the vocabulary size).
- A cell  $m_{ij}$  contains the number of times word  $w_i$  co-occurs with word  $w_j$  within a specific context
  - a sentence,
  - a paragraph
  - a document,
  - a certain window of  $m$  words (where  $m$  is an application-dependent parameter).
- Co-occurrence is a symmetric relation

# Compute word co-occurrence (“pairs” approach)

```
1: class MAPPER
2:   method MAP(docid a, doc d)
3:     for all term  $w \in$  doc d do
4:       for all term  $u \in$  NEIGHBORS( $w$ ) do
5:         EMIT(pair ( $w, u$ ), count 1)
6:       end for
7:     end for
8:   end method
9: end class

1: class REDUCER
2:   method REDUCE(pair  $p$ , counts [ $c_1, c_2, \dots$ ])
3:      $s \leftarrow 0$ 
4:     for all count  $c \in$  counts [ $c_1, c_2, \dots$ ] do
5:        $s \leftarrow s + c$ 
6:     end for
7:     EMIT(pair  $p$ , count  $s$ )
8:   end method
9: end class
```

# Compute word co-occurrence (“stripes” approach)

```
1: class MAPPER
2:   method MAP(docid a, doc d)
3:     for all term  $w \in$  doc  $d$  do
4:        $H \leftarrow$  new ASSOCIATIVEARRAY
5:       for all term  $u \in$  NEIGHBORS( $w$ ) do
6:          $H\{u\} \leftarrow H\{u\} + 1$ 
7:       end for
8:       EMIT(Term  $w$ , Stripe  $H$ )
9:     end for
10:    end method
11: end class

1: class REDUCER
2:   method REDUCE(term  $w$ , stripes [ $H_1, H_2, H_3, \dots$ ])
3:      $H_f \leftarrow$  new ASSOCIATIVEARRAY
4:     for all stripe  $H \in$  stripes [ $H_1, H_2, H_3, \dots$ ] do
5:       SUM( $H_f, H$ )
6:     end for
7:     EMIT(term  $w$ , stripe  $H_f$ )
8:   end method
```

# MapReduce Big Data Processing

A given application may have:

- A chain of map functions
  - (input processing, filtering, extraction. . . )
- A sequence of several map-reduce jobs
- No reduce task when everything can be expressed in the map  
(zero reducers, or the identity reducer function)

Prefer:

- Simple map and reduce functions
- Mapper tasks processing large data chunks (at least the size of distributed filesystem blocks)