Architectures for massive data management

MapReduce

Albert Bifet
albert.bifet@telecom-paristech.fr

September 22, 2015

Who am I

- Associate Professor at Télécom 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)

Course Goals

1. Discuss the main characteristics (dimensions) of massive data management plaforms
   - Big Data
2. Present the main classes of such systems, according to the above dimensions
3. Analyze advantage/disadvantage trade-offs
4. Introduce some open research issues

Today's course plan

1. Motivation MapReduce
2. MapReduce
3. Apache Hadoop
4. MapReduce Algorithms

Motivation MapReduce

How Many Servers Does Google Have?

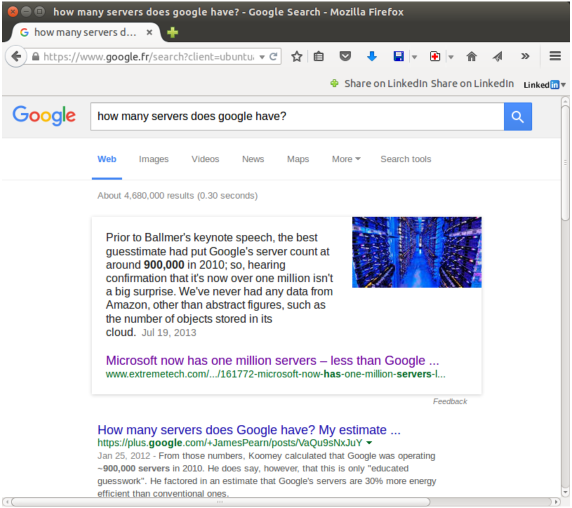


Figure: Asking Google

A Google Server Room

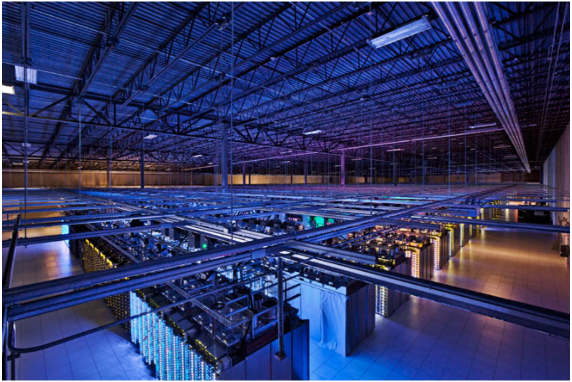


Figure: https://www.youtube.com/watch?t=3&v=avP5d16wEp0

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?

Google 2004

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

Google 2004

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.”

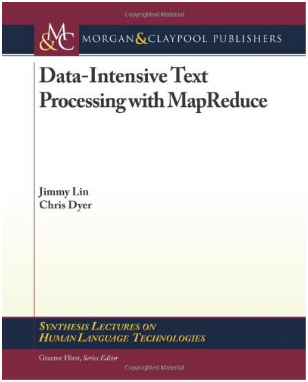
Jeff Dean Facts

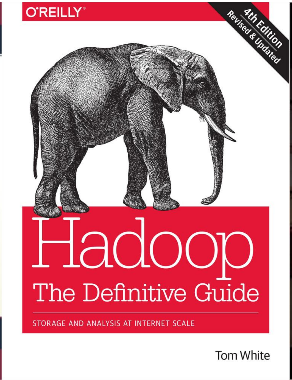
Google Culture Facts

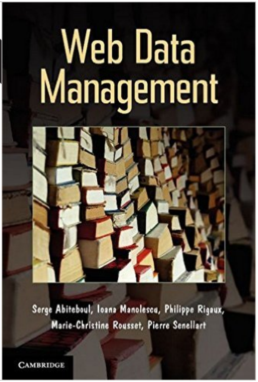
“The rate at which Jeff Dean produces code jumped by a factor of 40 in late 2000 when he upgraded his keyboard to USB2.0.”
”The speed of light in a vacuum used to be about 35 mph. Then Jeff Dean spent a weekend optimizing physics.”

MapReduce

References


Copyrighted Material
MORGAN&CLAYPOOL PUBLISHERS
Data-Intensive Text Processing with MapReduce
Jimmy Lin
Chris Dyer
SYNTHESIS LECTURES ON HUMAN LANGUAGE TECHNOLOGIES
Graeme Hirst, Series Editor
Copyrighted Material


O'REILLY
4th Edition
Revised & Updated
Hadoop
The Definitive Guide
STORAGE AND ANALYSIS AT INTERNET SCALE
Tom White


Web Data Management
Serge Abiteboul, Ioana Manolescu, Philippe Rigaux, Marie-Christine Rousset, Pierre Senellart
CAMBRIDGE

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

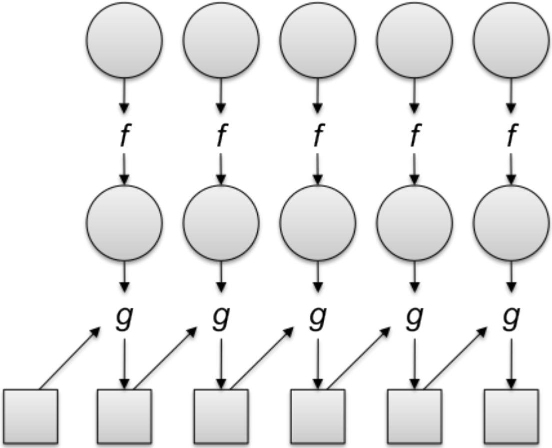


Figure: Map as a transformation function and Fold as an aggregation function

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

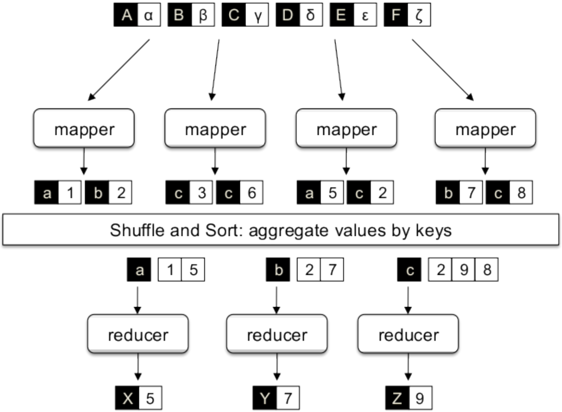


Figure: Two-stage processing structure

An Example Application: Word Count

Input Data

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

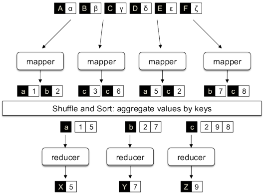
Output Data

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

WordCount Example

```
class Mapper
    method Map(docid a, doc d)
        for all term t ∈ doc d do
            Emit(term t, count 1)
        end for
    end method
end class
```

```
class Reducer
    method Reduce(term t, counts [c1, c2, ...])
        sum ← 0
        for all count c ∈ counts [c1, c2, ...] do
            sum ← sum + c
        end for
        Emit(term t, count sum)
    end method
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

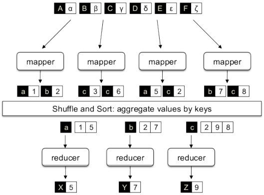
Not possible!

Identity Function Mappers

Sorting and regrouping the input data

Identity Function Reducers

Sorting and regrouping the data from mappers



MapReduce Framework

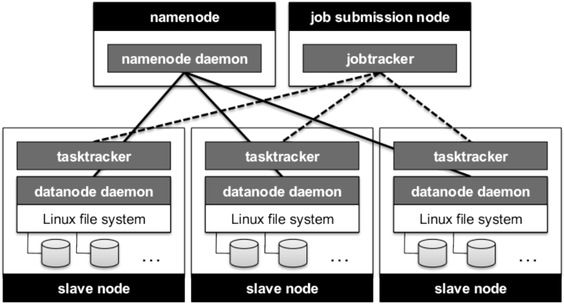


Figure: Runtime Framework

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

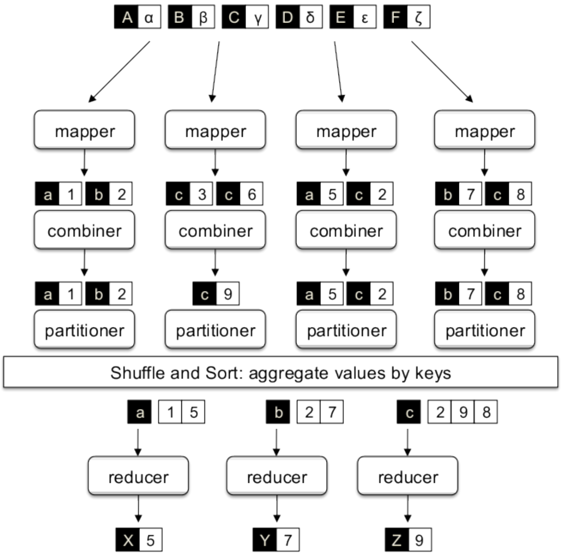


Figure: Partitioners and Combiners

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, number of partitions) $\rightarrow$ *partition for k*

Combiners

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

combine: $(k_2, list[v_2]) \rightarrow list[(k_3, v_3)]$

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 ouput 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

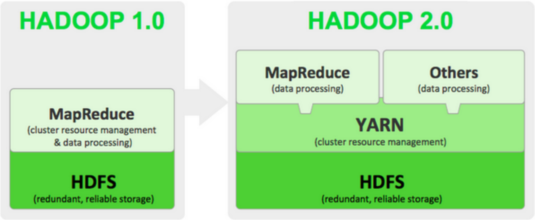


Figure: Apache Hadoop NextGen MapReduce (YARN)

Apache Hadoop NextGen MapReduce (YARN)

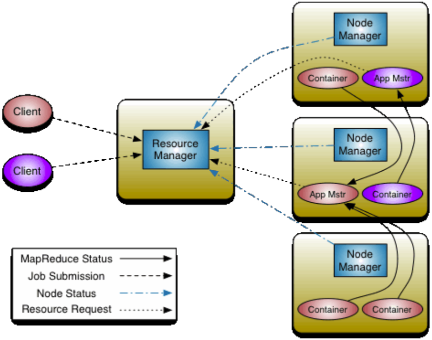


Figure: MRv2 splits up the two major functionalities of the JobTracker, resource management and job scheduling/monitoring, into separate daemons. An application is either a single job in the classical sense of Map-Reduce jobs or a DAG of jobs.

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

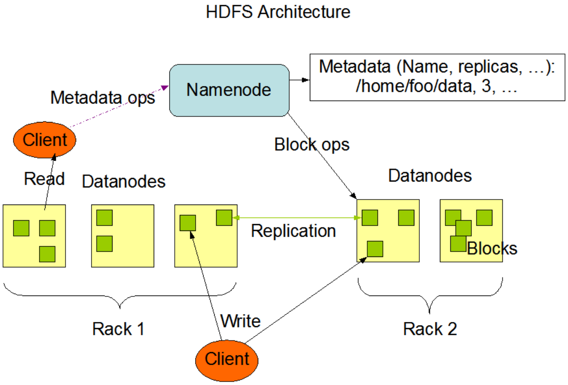


Figure: Distributed File System Architecture

The Distributed File System

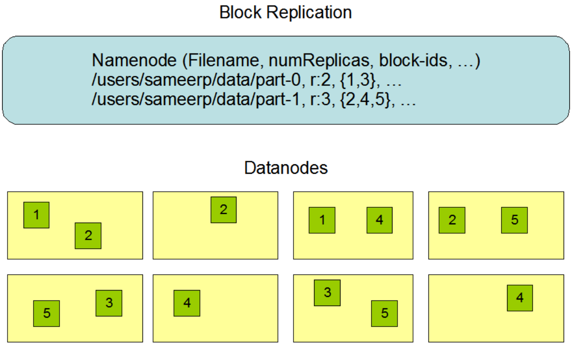


Figure: Block Replication

An Example Application: Word Count

Input Data

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

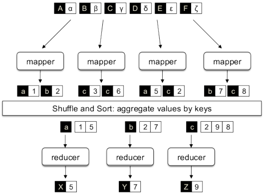
Output Data

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

WordCount Example

class Mapper
    method Map(docid a, doc d)
        for all term $t \in$ doc d **do**
            Emit(term t, count 1)
        end for
    end method
end class

class Reducer
    method Reduce(term t, counts $[c_1, c_2, \ldots]$)
        $sum \leftarrow 0$
        for all count $c \in$ counts $[c_1, c_2, \ldots]$ **do**
            $sum \leftarrow sum + c$
        end for
        Emit(term t, count sum)
    end method
end class



Mapper Java Code

```java
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

```java
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

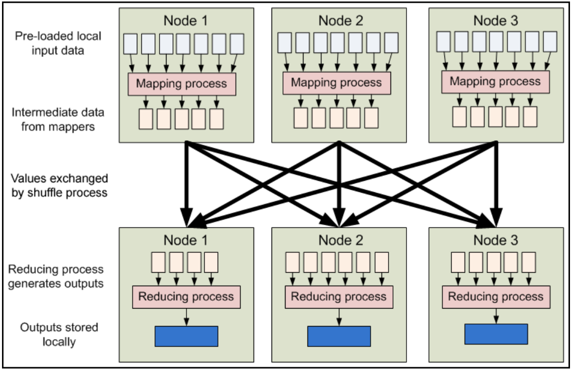


Figure: High-level MapReduce pipeline

Hadoop MapReduce data flow

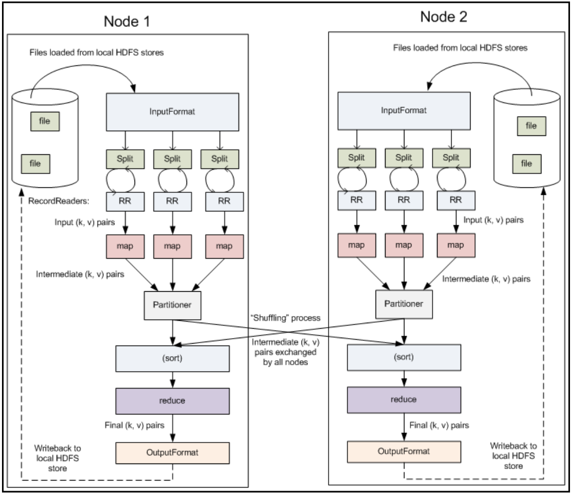


Figure: Detailed Hadoop MapReduce data flow

Hadoop MapReduce data flow

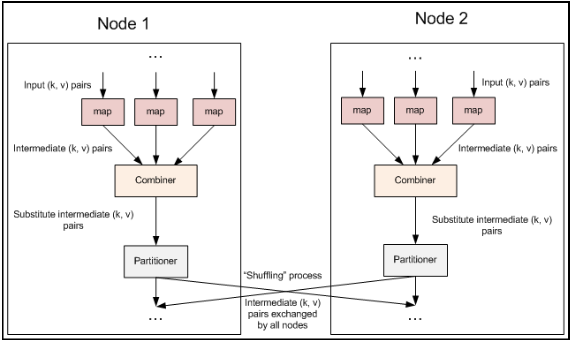


Figure: Combiner step inserted into the MapReduce data flow

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 `<URL,1>`
    - Reduce: Adds together and outputs `<URL, Total Count>`

Simple MapReduce Algorithms

Reverse Web-Link Graph

- Computes `source` list of web pages linked to `target` URLs
  - Map: outputs `<target,source>`
  - Reduce: Concatenates together and outputs `<target, list(source)>`

Inverted Index

- Build an inverted index
  - Map: emits a sequence of `<word, docID>`
  - Reduce: outputs `<word, list(docID)>`

Joins in MapReduce

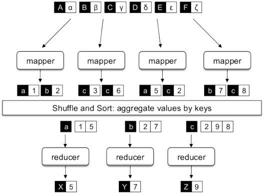
Two datasets, A and B that we need to join for a MapReduce task

- 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 Revisited

class Mapper
    method Map(docid a, doc d)
        for all term $t \in$ doc d **do**
            Emit(term t, count 1)
        end for
    end method
end class

class Reducer
    method Reduce(term t, counts $[c_1, c_2, \ldots]$)
        $sum \leftarrow 0$
        for all count $c \in$ counts $[c_1, c_2, \ldots]$ **do**
            $sum \leftarrow sum + c$
        end for
        Emit(term t, count sum)
    end method
end class



WordCount Example Revisited

```
class Mapper
    method Map(docid a, doc d)
        for all term t ∈ doc d do
            Emit(term t, count 1)
        end for
    end method
end class
```

```
class Mapper
    method Map(docid a, doc d)
        H ← new AssociativeArray
        for all term t ∈ doc d do
            H{t} ← H{t} + 1        ▷ Tally counts for entire document
        end for
        for all term t ∈ H do
            Emit(term t, count H{t})
        end for
    end method
end class
```

WordCount Example Revisited

```
class Mapper
    method Initialize
        H ← new AssociativeArray
    end method
    method Map(docid a, doc d)
        for all term t ∈ doc d do
            H{t} ← H{t} + 1          ▷ Tally counts across documents
        end for
    end method
    method Close
        for all term t ∈ H do
            Emit(term t, count H{t})
        end for
    end method
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

class Mapper
    method Map(string t, integer r)
        Emit(string t, integer r)
    end method
end class

class Reducer
    method Reduce(string t, integers $[r_1, r_2, \ldots]$)
        $sum \leftarrow 0$
        $cnt \leftarrow 0$
        for all integer $r \in$ integers $[r_1, r_2, \ldots]$ **do**
            $sum \leftarrow sum + r$
            $cnt \leftarrow cnt + 1$
        end for
        $r_{avg} \leftarrow sum/cnt$
        Emit(string t, integer r_{avg})
    end method
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

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

Average Computing Example

```
class Mapper
    method Map(string t, integer r)
        Emit(string t, pair (r, 1))
    end method
end class

class Combiner
    method Combine(string t, pairs [(s1, c1), (s2, c2) ...])
        sum ← 0
        cnt ← 0
        for all pair (s, c) ∈ pairs [(s1, c1), (s2, c2) ...] do
            sum ← sum + s
            cnt ← cnt + c
        end for
        Emit(string t, pair (sum, cnt))
    end method
end class

class Reducer
    method Reduce(string t, pairs [(s1, c1), (s2, c2) ...])
        sum ← 0
        cnt ← 0
        for all pair (s, c) ∈ pairs [(s1, c1), (s2, c2) ...] do
            sum ← sum + s
            cnt ← cnt + c
        end for
        r_avg ← sum/cnt
        Emit(string t, integer r_avg)
    end method
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 Competing Example

class Mapper
    method Initialize
        $S \leftarrow$ new AssociativeArray
        $C \leftarrow$ new AssociativeArray
    end method
    method Map(string t, integer r)
        $S\{t\} \leftarrow S\{t\} + r$
        $C\{t\} \leftarrow C\{t\} + 1$
    end method
    method Close
        for all term $t \in S$ **do**
            Emit(term t, pair $(S\{t\}, C\{t\})$)
        end for
    end method
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)

```
class Mapper
    method Map(docid a, doc d)
        for all term w ∈ doc d do
            for all term u ∈ Neighbors(w) do
                Emit(pair (w, u), count 1)
            end for
        end for
    end method
end class
```

```
class Reducer
    method Reduce(pair p, counts [c_1, c_2, ...])
        s ← 0
        for all count c ∈ counts [c_1, c_2, ...] do
            s ← s + c
        end for
        Emit(pair p, count s)
    end method
end class
```

Compute word co-occurrence ("stripes" approach)

```
class Mapper
    method Map(docid a, doc d)
        for all term w ∈ doc d do
            H ← new AssociativeArray
            for all term u ∈ Neighbors(w) do
                H{u} ← H{u} + 1
            end for
            Emit(Term w, Stripe H)
        end for
    end method
end class
```

```
class Reducer
    method Reduce(term w, stripes [H1, H2, H3, ...])
        Hf ← new AssociativeArray
        for all stripe H ∈ stripes [H1, H2, H3, ...] do
            Sum(Hf, H)
        end for
        Emit(term w, stripe Hf)
    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)