

0010010110101111001111110101010101000
1010100010101110111000111010110011011
0010010110101111001111110101010101000

NoSQL

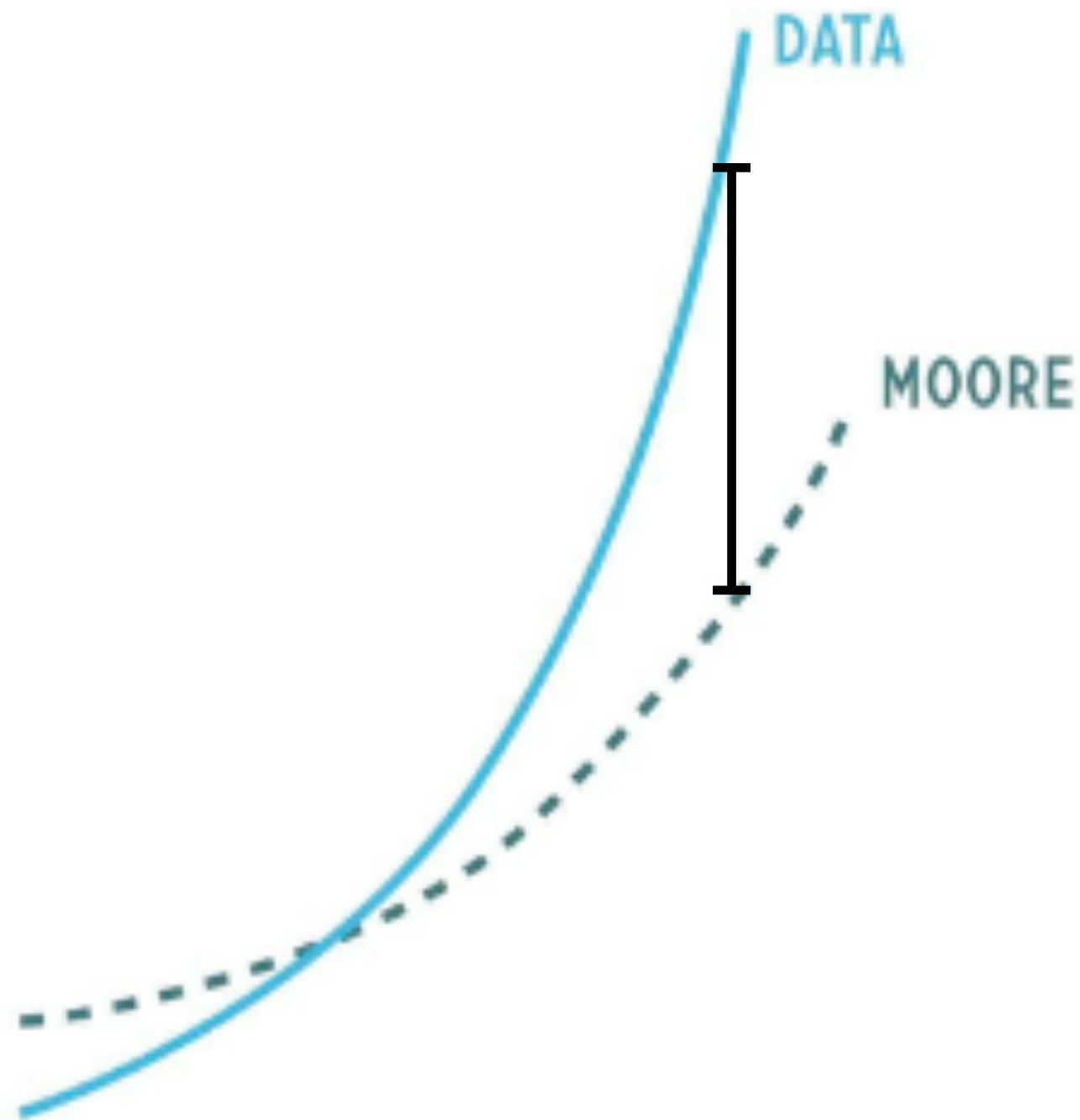
Luiz Celso Gomes Jr - André Santanchè
MC536 2013/2

Outline

- The rise of NoSQL
- MapReduce & Hadoop
- NoSQL Databases
- Cloud Computing

Promiscuous

Small computers vs ~~Smart~~ programmers



Web Application - stairway to hell

- LAMP implementation
- Hardware (Performance, Costs?)
- Slave Databases (Availability, Write Performance?)
- Memcached (Read Performance, Consistency?)
- Denormalize (Consistency?)
- Materialize views, drop stored proc, drop secondary indexes...
- Sharding
- What now?

NoSQL – New generation of DBMSs

- Commodity Hardware (Performance X Costs)
- Replication (Availability, Write Performance?)
- Eventual Consistency (Availability, Write Performance?)
- Sharding (Performance)
- Scalability
- Flexible Schema
- (mainly) Explicit optimization
- – New generation of DBMSs

NoSQL - Definition

- NoWayNeverNotAChanceGetOutOfHereSQL
- NotOnlySQL
- NoACID
- NoTraditionalDatabaseResearchMentality
- NoBoundsNoRulesNoSense

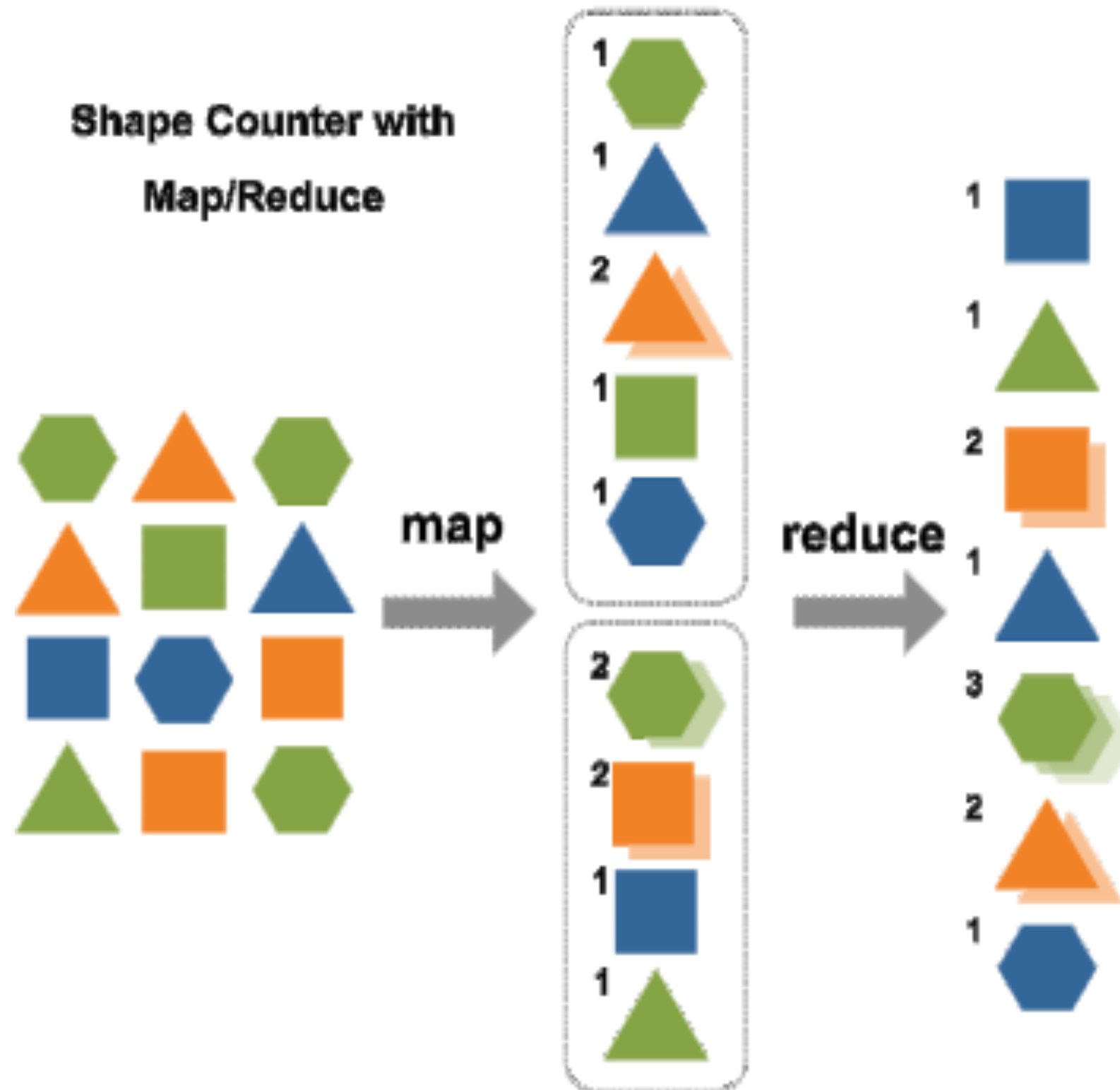
NoSQL – Influences

- Memcached/Key Stores
- Column Databases
- CAP theorem (Consistency, Availability, or Partition Tolerance)
- MapReduce

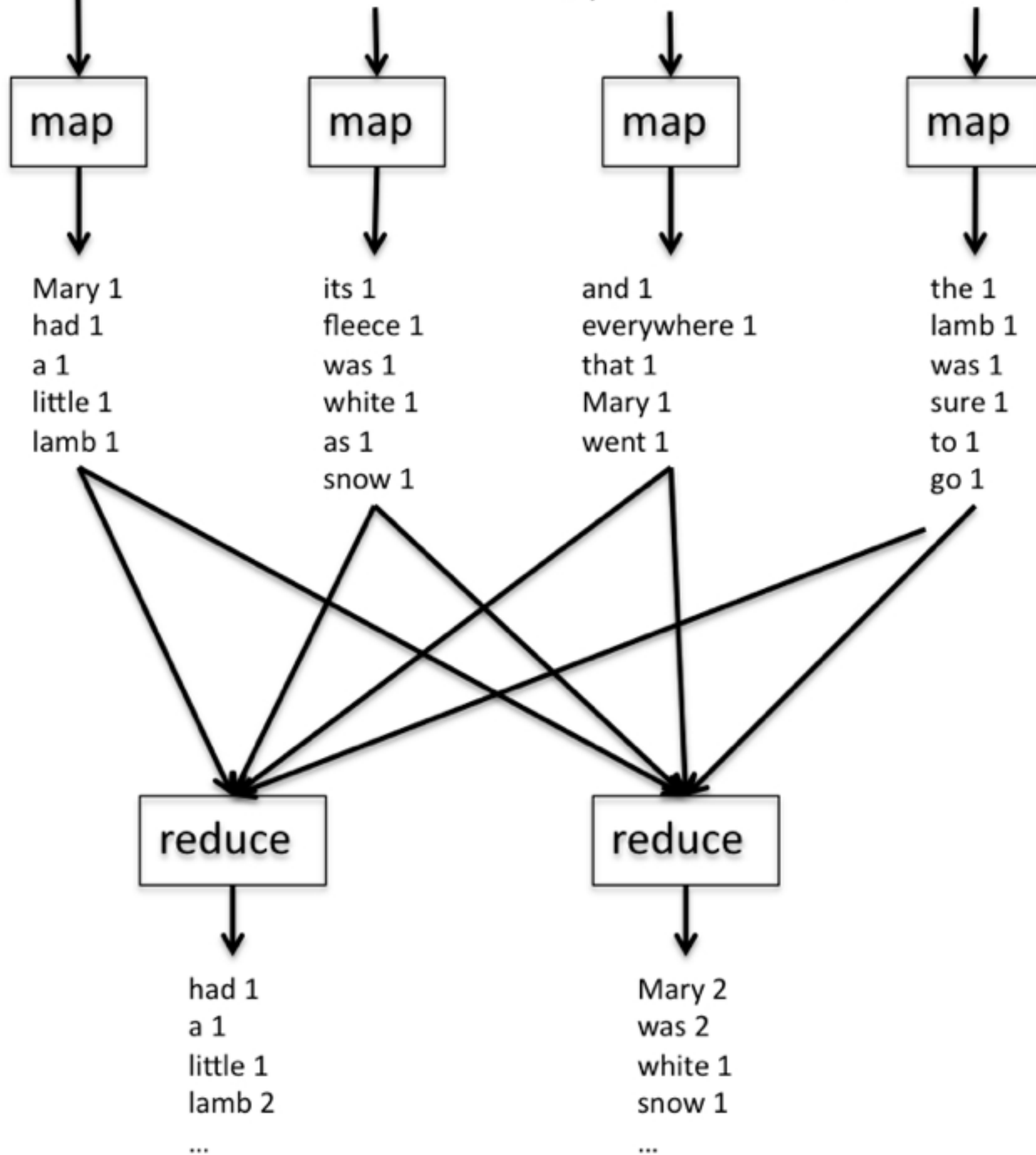
MapReduce Functional Model

- $\text{Map}(k_1, v_1) \rightarrow \text{list}(k_2, v_2)$
- $\text{Reduce}(k_2, \text{list}(v_2)) \rightarrow \text{list}([k_3,] v_3)$

MapReduce Example



Mary had a little lamb its fleece was white as snow and everywhere that Mary went the lamb was sure to go



Hadoop Java MapReduce – WordCount (1)


```
1 package org.myorg;
2
3 import java.io.IOException;
4 import java.util.*;
5
6 import org.apache.hadoop.fs.Path;
7 import org.apache.hadoop.conf.*;
8 import org.apache.hadoop.io.*;
9 import org.apache.hadoop.mapreduce.*;
10 import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
11 import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
12 import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
13 import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
14
15 public class WordCount {
16
17     public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {
18         private final static IntWritable one = new IntWritable(1);
19         private Text word = new Text();
20
21         public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
22             String line = value.toString();
23             StringTokenizer tokenizer = new StringTokenizer(line);
24             while (tokenizer.hasMoreTokens()) {
25                 word.set(tokenizer.nextToken());
26                 context.write(word, one);
27             }
28         }
29     }
30 }
```

Hadoop Java MapReduce – WordCount (2)

```
31 public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {
32
33     public void reduce(Text key, Iterable<IntWritable> values, Context context)
34         throws IOException, InterruptedException {
35         int sum = 0;
36         for (IntWritable val : values) {
37             sum += val.get();
38         }
39         context.write(key, new IntWritable(sum));
40     }
41 }
42
43 public static void main(String[] args) throws Exception {
44     Configuration conf = new Configuration();
45
46     Job job = new Job(conf, "wordcount");
47
48     job.setOutputKeyClass(Text.class);
49     job.setOutputValueClass(IntWritable.class);
50
51     job.setMapperClass(Map.class);
52     job.setReducerClass(Reduce.class);
53
54     job.setInputFormatClass(TextInputFormat.class);
55     job.setOutputFormatClass(TextOutputFormat.class);
56
57     FileInputFormat.addInputPath(job, new Path(args[0]));
58     FileOutputFormat.setOutputPath(job, new Path(args[1]));
59
60     job.waitForCompletion(true);
61 }
```


Apache Hadoop Zoo

Cloudera's Distribution for Hadoop

UI Framework		<i>Hue</i>	SDK		<i>Hue SDK</i>	
Workflow	<i>Oozie</i>	Scheduling		<i>Oozie</i>	Metadata	<i>Hive</i>
Data Integration	Languages, Compilers				Fast read/write access	
						
Coordination						<i>Zookeeper</i>

Apache Hadoop Zoo

Cloudera's Distribution for Hadoop

UI Framework		<i>Hue</i>	SDK		<i>Hue SDK</i>	
Workflow	<i>Oozie</i>	Scheduling		<i>Oozie</i>	Metadata	<i>Hive</i>
Data Integration	<i>Flume, Sqoop</i>	Languages, Compilers		<i>Pig/ Hive</i>	Fast read/write access	<i>HBase</i>
						
Coordination						<i>Zookeeper</i>

Hadoop kernel

- MapReduce
 - Execution engine - Scheduling (JobTracker, TaskTracker)
 - Zookeeper (Distributed Coordination)
- Hadoop Distributed File System (HDFS)
 - stores large files across multiple machines
 - replicated blocks
 - the job tracker schedules map or reduce jobs to task trackers with an awareness of the data location

Pig

```
INPUT = LOAD '/tmp/my-copy-of-all-pages-on-internet';

-- Extract words from each line and put them into a pig bag
-- datatype, then flatten the bag to get one word on each row
WORDS = foreach INPUT generate flatten(TOKENIZE((chararray)$0)) AS word;

-- filter out any words that are just white spaces
FILTERED_WORDS = FILTER WORDS BY word matches '\\w+';

-- create a group for each word
WORD_GROUPS = GROUP FILTERED_WORDS BY word;

-- count the entries in each group
WORD_COUNT = foreach WORD_GROUPS generate COUNT(FILTERED_WORDS) AS COUNT, GROUP AS word;

-- order the records by count
ORDERED_WORD_COUNT = ORDER WORD_COUNT BY COUNT DESC;
store ORDERED_WORD_COUNT INTO '/tmp/number-of-words-on-internet';
```


Hive

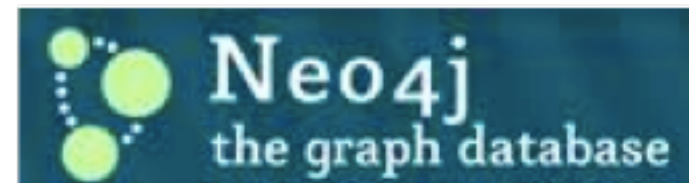
```
CREATE TABLE u_data_new (  
    userid INT,  
    movieid INT,  
    rating INT,  
    weekday INT)  
ROW FORMAT DELIMITED  
FIELDS TERMINATED BY '\t';  
  
add FILE weekday_mapper.py;  
  
INSERT OVERWRITE TABLE u_data_new  
SELECT  
    TRANSFORM (userid, movieid, rating, unixtime)  
    USING 'python weekday_mapper.py'  
    AS (userid, movieid, rating, weekday)  
FROM u_data;  
  
SELECT weekday, COUNT(*)  
FROM u_data_new  
GROUP BY weekday;
```

NoSQL Databases

APACHE
HBASE



Lucene



NoSQL database classes

- Key-value stores
- Document-oriented databases
- Graph databases

Key-value store

- schema-less
- easy to distribute/replicate
- perfect for the MapReduce model
- inspired by Google's Bigtable

HBase

- BigTable clone
- Simple Schema
- Ordered, location sensitive keys (range partitioning!)
- Column family organization
- Versioning
- Data center syncing

HBase features

- Linear and modular scalability.
- Strictly consistent reads and writes.
- Automatic and configurable sharding (horizontal fragmentation)
- Automatic failover support between RegionServers.
- Convenient base classes for backing Hadoop MapReduce jobs with HBase tables.
- Easy to use Java API for client access.
- Block cache and Bloom Filters for real-time queries.
- Thrift gateway and a REST-ful Web service

HBase DDL/DML

- create 'blogposts', 'post', 'image'
- put 'blogposts', 'post1', 'post:title', 'Hello World'
- put 'blogposts', 'post1', 'post:body', 'This is a blog post'
- put 'blogposts', 'post1', 'image:header', 'image1.jpg'
- put 'blogposts', 'post1', 'image:bodyimage', 'image2.jpg'
- scan 'blogposts'
- get 'blogposts', 'post1'

Document-oriented databases

- semi-structured schema (XML, JSON)
- easy to distribute/replicate
- may use the MapReduce model
- integrated text indexes

MongoDB features

- Document-Oriented Storage
- Full Index Support
- Replication & High Availability
- Auto-Sharding
- Rich, document-based queries
- Fast In-Place Updates

Exercício 1

- Em um banco de dados tipo key-store os registros são distribuídos e ordenados de acordo com um id único (tipo string). Considere que um serviço de mensagens instantâneas (GTalk, Facebook Messages, MSN, etc) armazena os dados das mensagens em uma tabela de um destes bancos. Como você comporia a chave desta tabela de forma a otimizar as consultas mais frequentes da aplicação? Considere o esquema da tabela abaixo.
- Mensagem(id, sender, receiver, content, timestamp)

Exercício 2

- Considerando o cenário anterior, discuta um possível problema em se utilizar apenas timestamp nas mensagens e proponha uma solução.

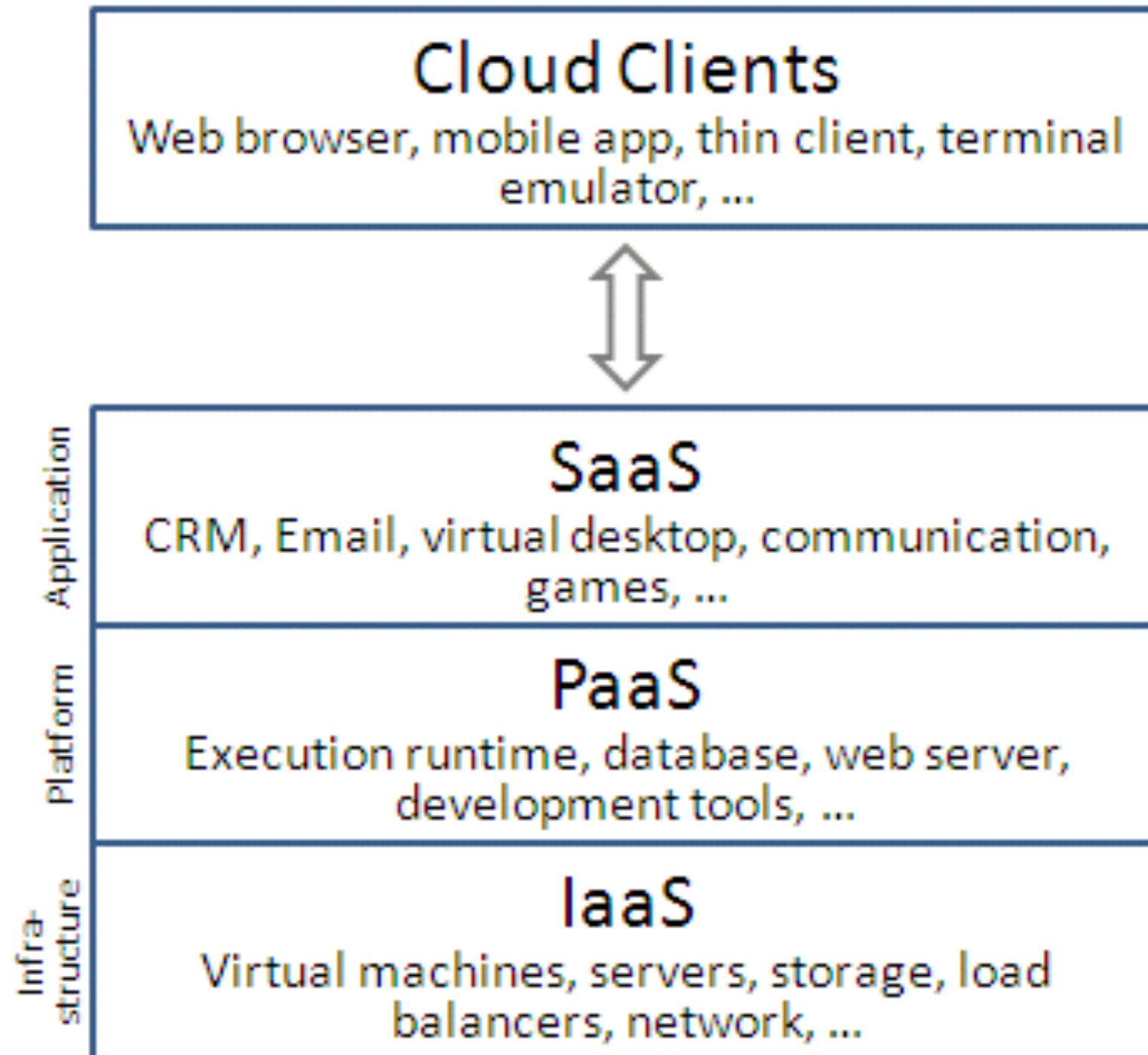
Deploying NoSQL

- Buying commodity hardware is still expensive
- Maintenance can be tricky
- Hard to find trained administrators
- Big investment on infrastructure (building, network, cooling, power, etc)
- Hard to use all the resources full time (idle resources = investment lost)

Cloud Computing

- Hard to define
- Usually refers to virtualized computing infrastructures
- Developers delegate the execution of their applications to cloud providers

Software, Platform, Infrastructure



Software, Platform, Infrastructure



Cloud computing - Advantages

- Cost
- Scalability
- Elasticity
- Reliability
- Security
- Maintenance

Cloud (infrastructure/platform) providers

- Amazon - AWS (Amazon Web Services)
- Rackspace
- Microsoft - Azure
- Google - Cloud Platform
- VMware (technology)

Exercício 3

- Cite alguns problemas (ao menos dois) associados à utilização de serviços de nuvem para gerenciamento e armazenamento de bancos de dados. O que você faria para contornar estes problemas?

Summary

- NoSQL databases are used when traditional DBMSs cannot handle the workload (e.g. BigData analysis)
- NoSQL is more scalable
- It is usually up to the developer:
 - guaranteeing consistency
 - defining an optimized (in terms of distribution) schema
 - optimizing queries

Summary (cont.)

- There are three main categories of NoSQL databases:
 - Key Stores - for simple relational data
 - Document-oriented databases - for documents/hierarchical data
 - Graph databases - for graphs, highly interconnected data
- MapReduce is usually the parallel programming paradigm of choice

Summary (cont.)

- There are several flavors of NoSQL databases. Developers must be able to choose and optimize one (or many) adequate to the problem
- Cloud computing simplifies deployment of applications and adds scalability
- “do this and problem solved!” -- “do this” usually takes months and many burned neurons... Besides, there might be a much better “do that” around the corner.

Exercício 1

- Em um banco de dados tipo key-store os registros são distribuídos e ordenados de acordo com um id único (tipo string). Considere que um serviço de mensagens instantâneas (GTalk, Facebook Messages, MSN, etc) armazena os dados das mensagens em uma tabela de um destes bancos. Como você comporia a chave desta tabela de forma a otimizar as consultas mais frequentes da aplicação? Considere o esquema da tabela abaixo.
- Mensagem(id, sender, receiver, content, timestamp)
- Resposta: sender-receiver-timestamp (de preferencia ordenar sender-receiver)

Exercício 2

- Considerando o cenário anterior, discuta um possível problema em se utilizar apenas timestamp nas mensagens e proponha uma solução.
- Resposta: mensagens geradas em seqüência podem chegar invertidas (delays na rede, etc) no servidor. Adicionar id sequencial às mensagens em cada remetente e ordená-las no servidor. (isso garante ordem parcial, o que é suficiente neste tipo de aplicação. existem algoritmos mais complexos para ordem total.)

Exercício 3

- Cite alguns problemas (ao menos dois) associados à utilização de serviços de nuvem para gerenciamento e armazenamento de bancos de dados. O que você faria para contornar estes problemas?
- Resposta: privacidade (criptografar dados?), confiabilidade do prestador de serviço (escolher bem, contratar mais de um - complicado por falta de padrões), largura de banda do link entre empresa e prestador de serviço (contratar um ou mais bons serviços).