



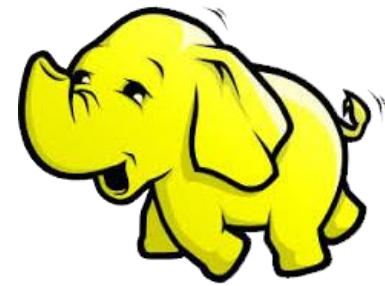
Parallel Computing - Hadoop

**Chapter 10 from “Data Science and Big Data Analytics:
Discovering, Analyzing, Visualizing and Presenting Data”**

1st Edition by [EMC Education Services](#)

Ka-Chun Wong, Department of Computer Science, City University of Hong Kong
Charles Tappert Seidenberg, School of CSIS, Pace University

Google Origins



2003

The Google File System

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung
Google*



2004

MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat
jeff@google.com, sanjay@google.com

Google, Inc.



2006

Bigtable: A Distributed Storage System for Structured Data

Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach
Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber
(fay.jeff, sanjay, wilson, kerr, zhang, tushar, fikes, gruber)@google.com

Google, Inc.



Abstract

Bigtable is a distributed storage system for managing structured data that is designed to scale to a very large number of nodes across thousands of commodity servers. Many projects at Google store data in Bigtable, including web indexing, Google Earth, and Google File Service. These applications place very different demands on Bigtable, both in terms of data size (from URLs to

achieved scalability and high performance, but Bigtable provides a different interface than such systems. Bigtable does not support a full relational data model; instead, it provides clients with a simple data model that supports dynamic control over data layout and format, and allows clients to reason about the locality properties of data represented in the underlying storage. Data is indexed using row and column names that can be arbitrary strings. Bigtable also treats data as uninterpreted strings.



Apache Hadoop

- **Hadoop:**

- an open-source software framework that supports data-intensive **distributed** applications, licensed under the Apache license.

- **Goals / Requirements:**

- Abstract and facilitate the storage and processing of large and/or rapidly growing data sets
 - Structured and non-structured data
 - Simple programming models
- High scalability and availability
- Use commodity (cheap!) hardware with little redundancy
- Fault-tolerance
- Move computation rather than data



Contents

- 10.1 Analytics for Unstructured Data
 - 10.1.1 Use Cases
 - 10.1.2 MapReduce
 - 10.1.3 Apache Hadoop
- 10.2 The Hadoop Ecosystem
 - 10.2.1 Pig
 - 10.2.2 Hive
 - 10.2.3 HBase
- Summary



10.1 Analytics for Unstructured Data

- These slides document some key technologies and tools related to the **Apache Hadoop** software library
 - Hadoop stores data in a distributed system
 - Hadoop implements a parallel programming model (or paradigm) known as MapReduce
 - Hadoop ecosystem tools
 - Pig
 - Hive
 - Hbase
 -



10.1 Analytics for Unstructured Data

10.1.1 Use Cases

- IBM Watson – Jeopardy playing machine
 - To educate Watson, Hadoop was utilized to process data sources
 - Encyclopedias, dictionaries, news wire feeds, literature, Wikipedia, etc.
- LinkedIn – network of over 250 million users in 200 countries
 - Hadoop is used to process daily transaction logs, examine users' activities, feed extracted data back to production systems, restructure the data, develop and test analytic models
- Yahoo! – large Hadoop deployment
 - Search index creation and maintenance, Webpage content optimization, spam filters, etc.



Hadoop clusters

- We have ~20,000 machines running Hadoop
- Our largest clusters are currently 2000 nodes
- Several petabytes of user data (compressed, unreplicated)
- We run hundreds of thousands of jobs every month



10.1 Analytics for Unstructured Data

10.1.2 MapReduce

- The MapReduce model (or paradigm) breaks a large task into smaller tasks, runs the tasks in parallel, and consolidates the outputs of the individual tasks into the final output
- Map
 - Applies an operation to a piece of data
 - Provides some intermediate output
- Reduce
 - Consolidates the intermediate outputs from the map step
 - Provides the final output
- Each step uses key/value pairs (i.e. $\langle \text{key}, \text{value} \rangle$) as the data format between input and output



10.1 Analytics for Unstructured Data

10.1.2 MapReduce

- MapReduce provides
 - Automatic parallelization and distribution
 - I/O scheduling
 - Load balancing
 - Network and data transfer optimization
 - Fault tolerance
 - Handling of machine failures
- Main Philosophy: **Scale out, not up!**
 - Large number of **commodity servers** as opposed to some high end specialized servers

Apache Hadoop:

Open source implementation of MapReduce

10.1 Analytics for Unstructured Data

10.1.2 MapReduce

MapReduce word count example

<1234, "For each word in each string">



Map

<For, 1> <each, 1> <word, 1> <in, 1> <each, 1> <string, 1>

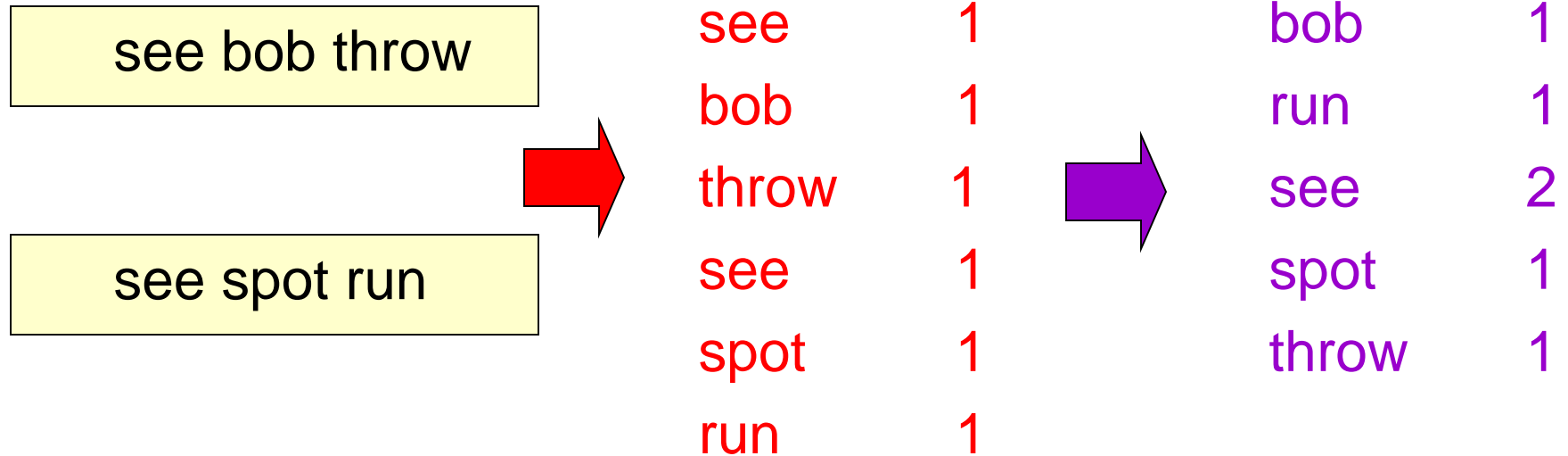
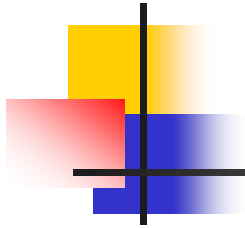


Reduce

<For, 1>
<each, 2>
<word, 1>
<in, 1>
<string, 1>

10.1 Analytics for Unstructured Data

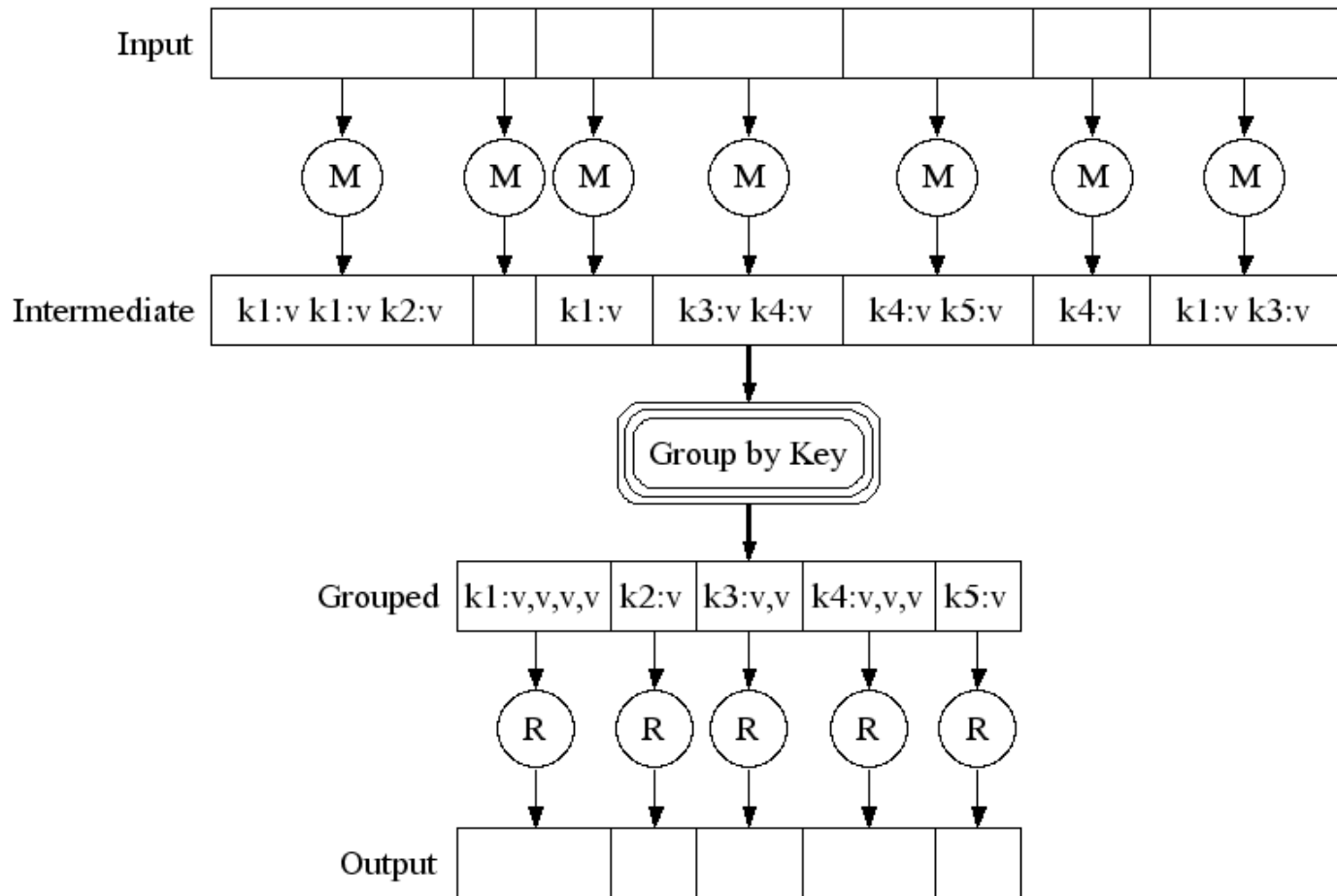
10.1.2 MapReduce



Can we do word count in parallel?

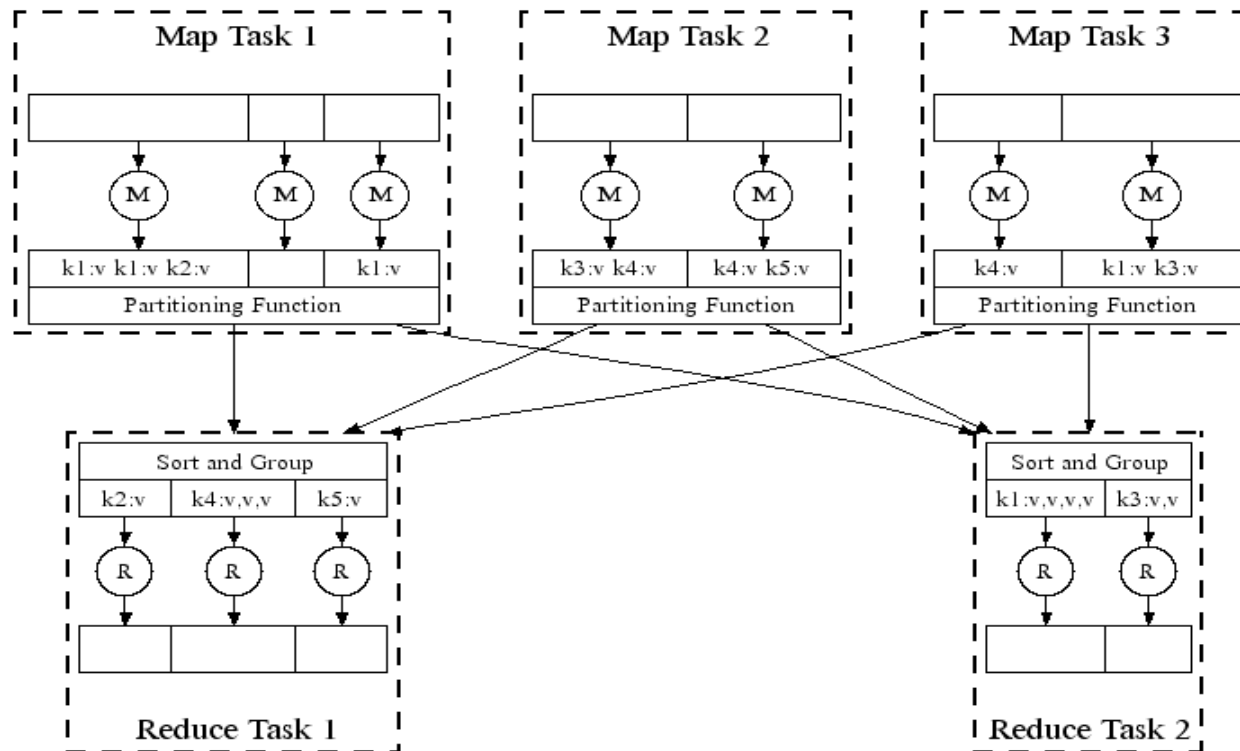
10.1 Analytics for Unstructured Data

10.1.2 MapReduce



10.1 Analytics for Unstructured Data

10.1.2 MapReduce



Handles failures automatically, e.g., restarts tasks if a node fails; runs multiples copies of the same task to avoid a slow task slowing down the whole job



10.1 Analytics for Unstructured Data

10.1.2 MapReduce

- **Failures** are **norm** in commodity hardware
 - **Worker** failure
 - Detect failure via periodic heartbeats
 - Re-execute in-progress map/reduce tasks
 - **Master** failure
 - Single point of failure; Resume from Execution Log
- **Robust**
 - Google's experience: lost 1600 of 1800 machines once!, but finished fine

10.1 Analytics for Unstructured Data

10.1.3 Apache Hadoop

- MapReduce is a distributed programming model (or paradigm) which is not easy to implement
- Executing a MapReduce job requires
 - Job scheduling based on system's workload (i.e. load balancing)
 - Input data spread across cluster of machines (i.e. IO handling)
 - Map step spread across distributed system (i.e. communications)
 - Intermediate outputs collected and distributed for reduce step
 - Final output made available to another user, another application, or another MapReduce job (i.e. IO handling)
- Next few slides present how the Hadoop environment can implement the model.

10.1 Analytics for Unstructured Data

10.1.3 Apache Hadoop

- Hadoop Distributed File System (HDFS)
 - File system that distributes data across a cluster to take advantage of the parallel processing of MapReduce
 - HDFS uses three Java daemons (background processors)
 1. NameNode – determines and tracks where various blocks of data are stored
 2. DataNode – manages the data stored on each machine
 3. Secondary NameNode – performs some of the NameNode tasks to reduce the load on NameNode

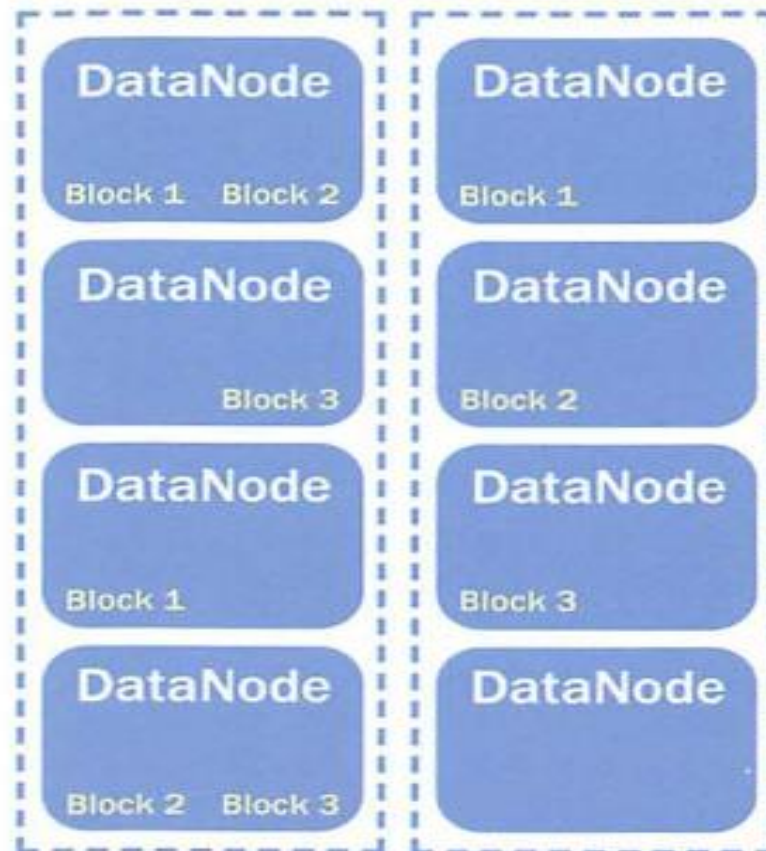
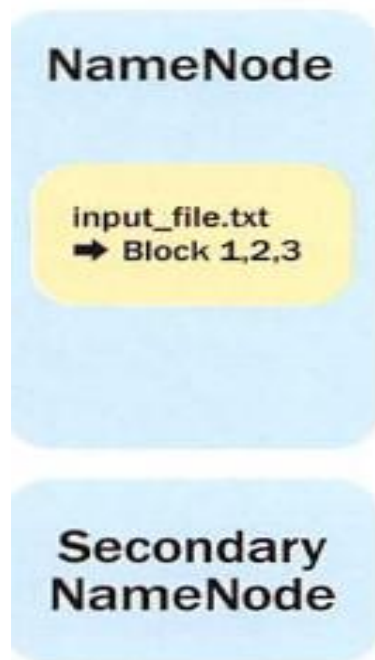
10.1 Analytics for Unstructured Data

10.1.3 Apache Hadoop

- Structuring a MapReduce Job in Hadoop

Master Nodes

8 Worker Nodes across 2 Racks



A file stored in HDFS



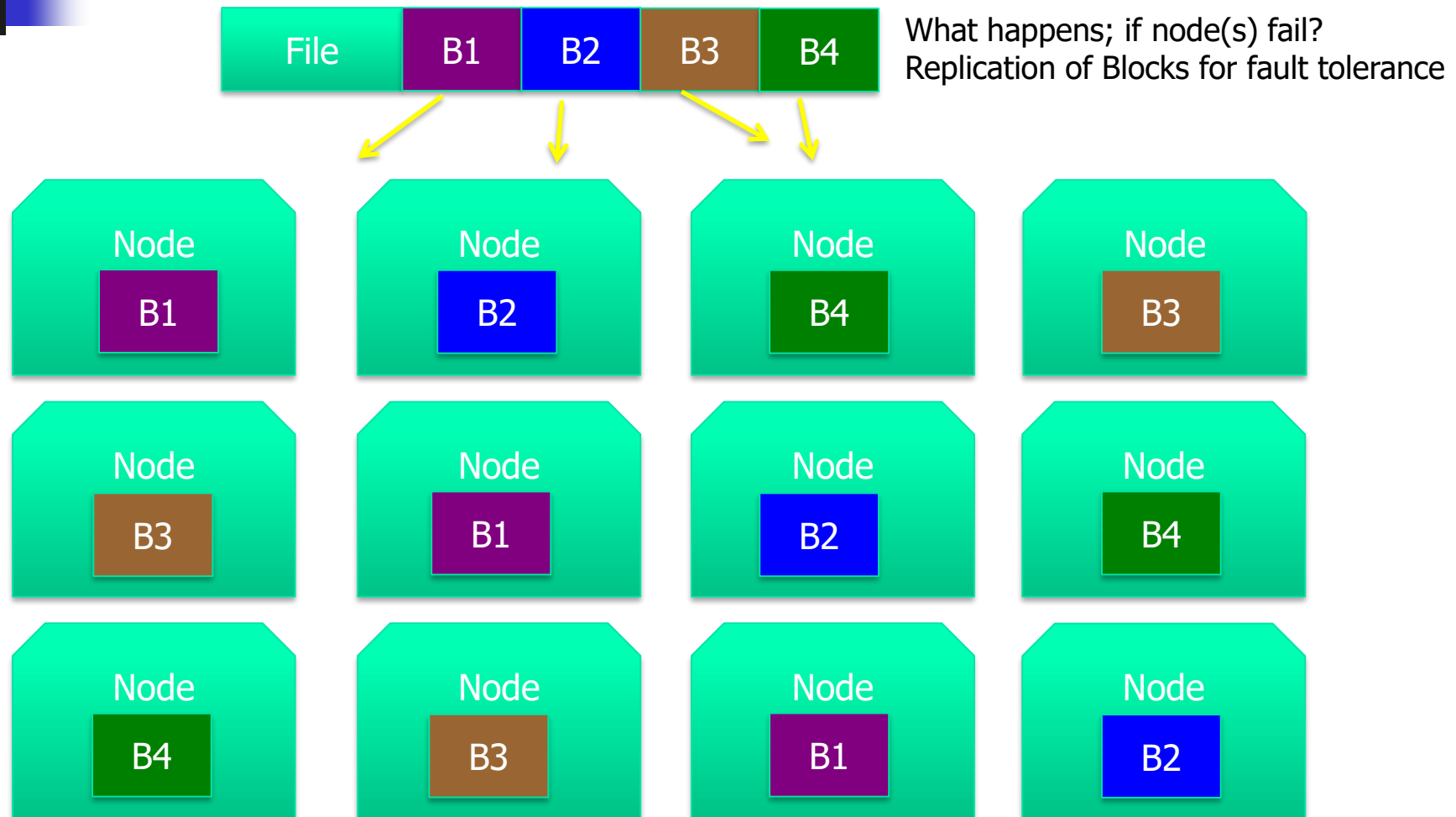
10.1 Analytics for Unstructured Data

10.1.3 Apache Hadoop

- HDFS files are divided into blocks
 - Block is the basic unit of read/write
 - Its default size is 64MB
 - The size could be adjusted (128MB)
 - It makes HDFS good for storing big data files
- HDFS blocks are replicated multiple times
 - One block stored at multiple locations (also at different racks) for usually 3 times
 - This makes HDFS storage fault tolerant and faster to read

10.1 Analytics for Unstructured Data

10.1.3 Apache Hadoop





10.1 Analytics for Unstructured Data

10.1.3 Apache Hadoop

- Structuring a MapReduce Job in Hadoop
 - A typical MapReduce Java program has three classes
 - Driver
 - provides details such as input file locations, names of mapper and reducer classes, location of reduce class output, etc.
 - Mapper
 - provides computing logic to process each data block
 - Reducer
 - aggregates the data provided by the mapper

In Java

Map function

Reduce function

Run this program as a
MapReduce job

File Edit Options Buffers Tools Java Help



```
public class WordCount {

    public static class Map extends MapReduceBase implements
        Mapper<LongWritable, Text, Text, IntWritable> {
        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();

        public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable>
            output, Reporter reporter) throws IOException {
            String line = value.toString();
            StringTokenizer tokenizer = new StringTokenizer(line);
            while (tokenizer.hasMoreTokens()) {
                word.set(tokenizer.nextToken());
                output.collect(word, one);
            }
        }

        public static class Reduce extends MapReduceBase implements
            Reducer<Text, IntWritable, Text, IntWritable> {
            public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text,
                IntWritable> output, Reporter reporter) throws IOException {
                int sum = 0;
                while (values.hasNext()) { sum += values.next().get(); }
                output.collect(key, new IntWritable(sum));
            }
        }

        public static void main(String[] args) throws Exception {
            JobConf conf = new JobConf(WordCount.class);
            conf.setJobName("wordcount");
            conf.setOutputKeyClass(Text.class);
            conf.setOutputValueClass(IntWritable.class);
            conf.setMapperClass(Map.class);
            conf.setCombinerClass(Reduce.class);
            conf.setReducerClass(Reduce.class);
            conf.setInputFormat(TextInputFormat.class);
            conf.setOutputFormat(TextOutputFormat.class);
            FileInputFormat.setInputPaths(conf, new Path(args[0]));
            FileOutputFormat.setOutputPath(conf, new Path(args[1]));

            JobClient.runJob(conf);
        }
    }
}
```

mapreduce.java All L9 (Java/l Abbrev)

Wrote /home/shivnath/Desktop/mapreduce.java

By Shivnath Babu

10.1 Analytics for Unstructured Data

10.1.3 Apache Hadoop

- Structuring a MapReduce Job

<1234, "For each word in each string">



<For, 1> <each, 1> <word, 1> <in, 1> <each, 1> <string, 1>



<each, (1, 1)> <For, (1)> <in, (1)> <string, (1)> <word, (1)>

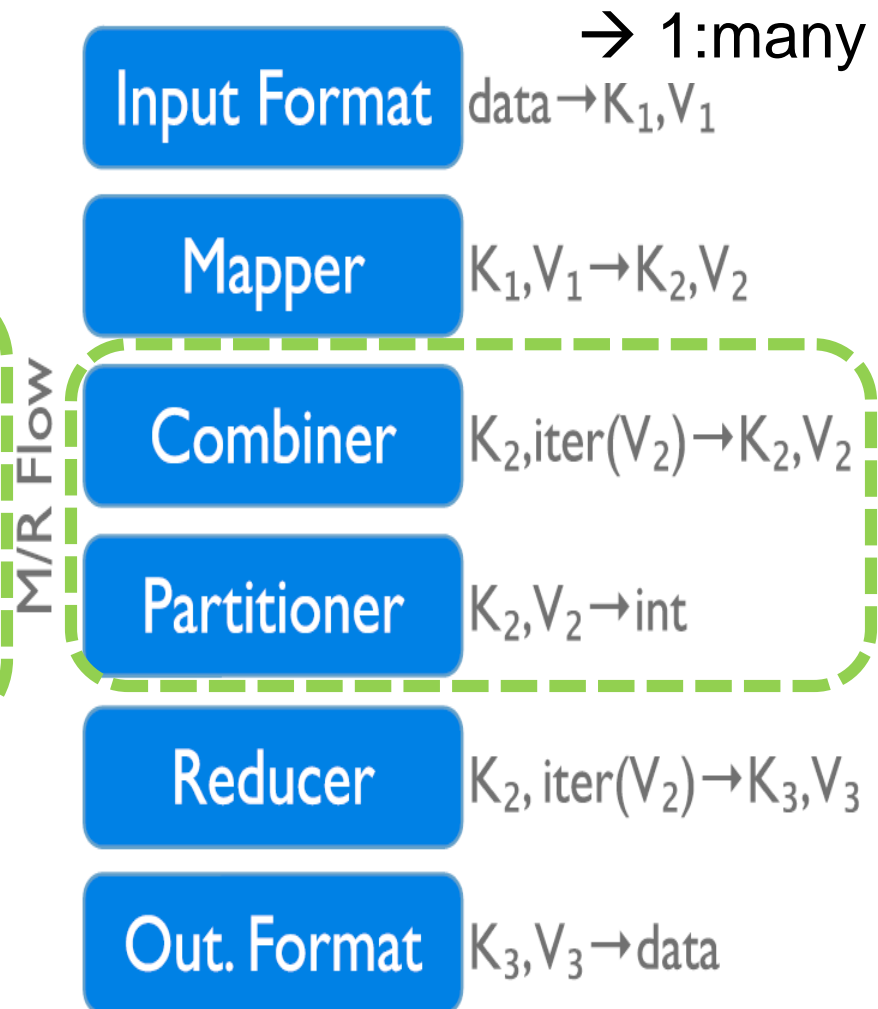


<each, 2>
<For, 1>
<in, 1>
<string, 1>
<word, 1>

Shuffle and Sort

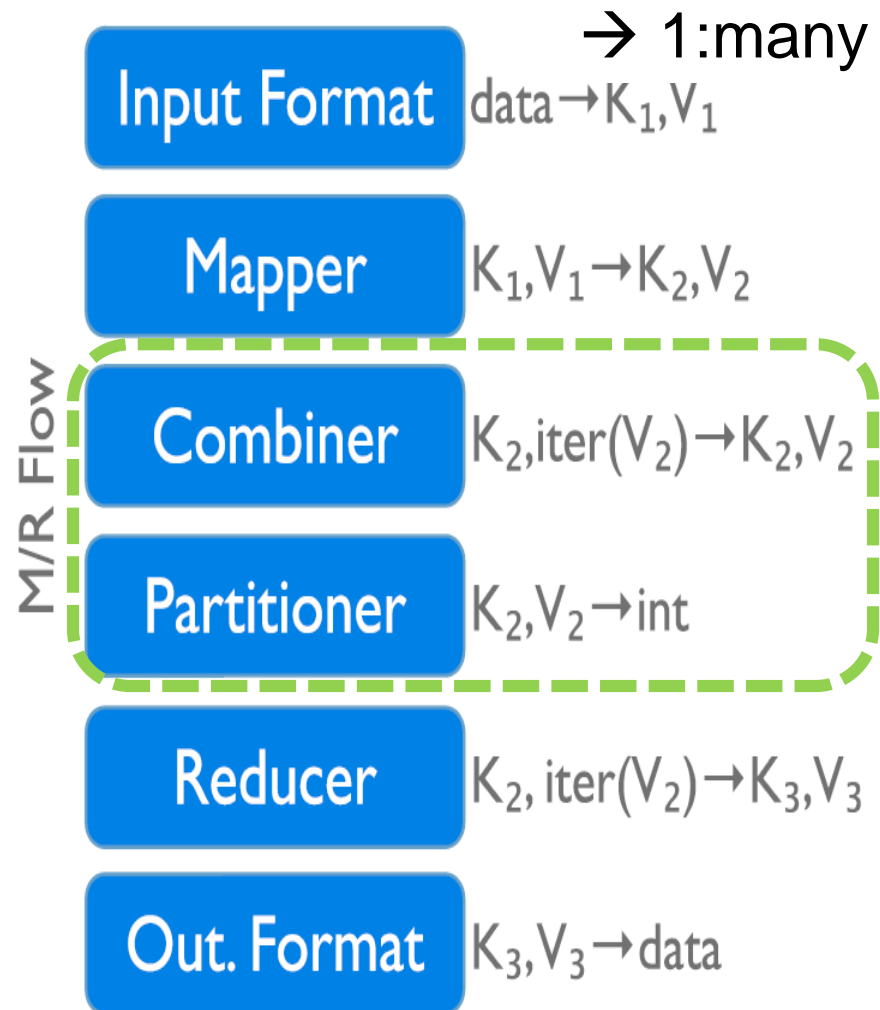
MapReduce Functions in Hadoop

- InputFormat
- Map function
- Combiner
 - Sorting & Merging
- Partitioner
 - Shuffling
- Reduce function
- OutputFormat



MapReduce Functions in Hadoop

- Optional Functions:
- Combiner:
 - “Mini-reducers” that run in memory after the map phase.
 - Used as an optimization to reduce network traffic.
- Partitioner
 - Often a simple hash of the key, e.g., $\text{hash}(k') \bmod n$.
 - Divides up key space for parallel reduce operations.



10.1 Analytics for Unstructured Data

10.1.3 Apache Hadoop

<1234, "For each word in each string">



<For, 1> <each, 1> <word, 1> <in, 1> <each, 1> <string, 1>



<For, 1> <each, 2> <word, 1> <in, 1> <string, 1>



using a
combiner

10.1 Analytics for Unstructured Data

10.1.3 Apache Hadoop

<1234, "For each word in each string">



<For, 1> <each, 1> <word, 1> <in, 1> <each, 1> <string, 1>



using a
partitioner

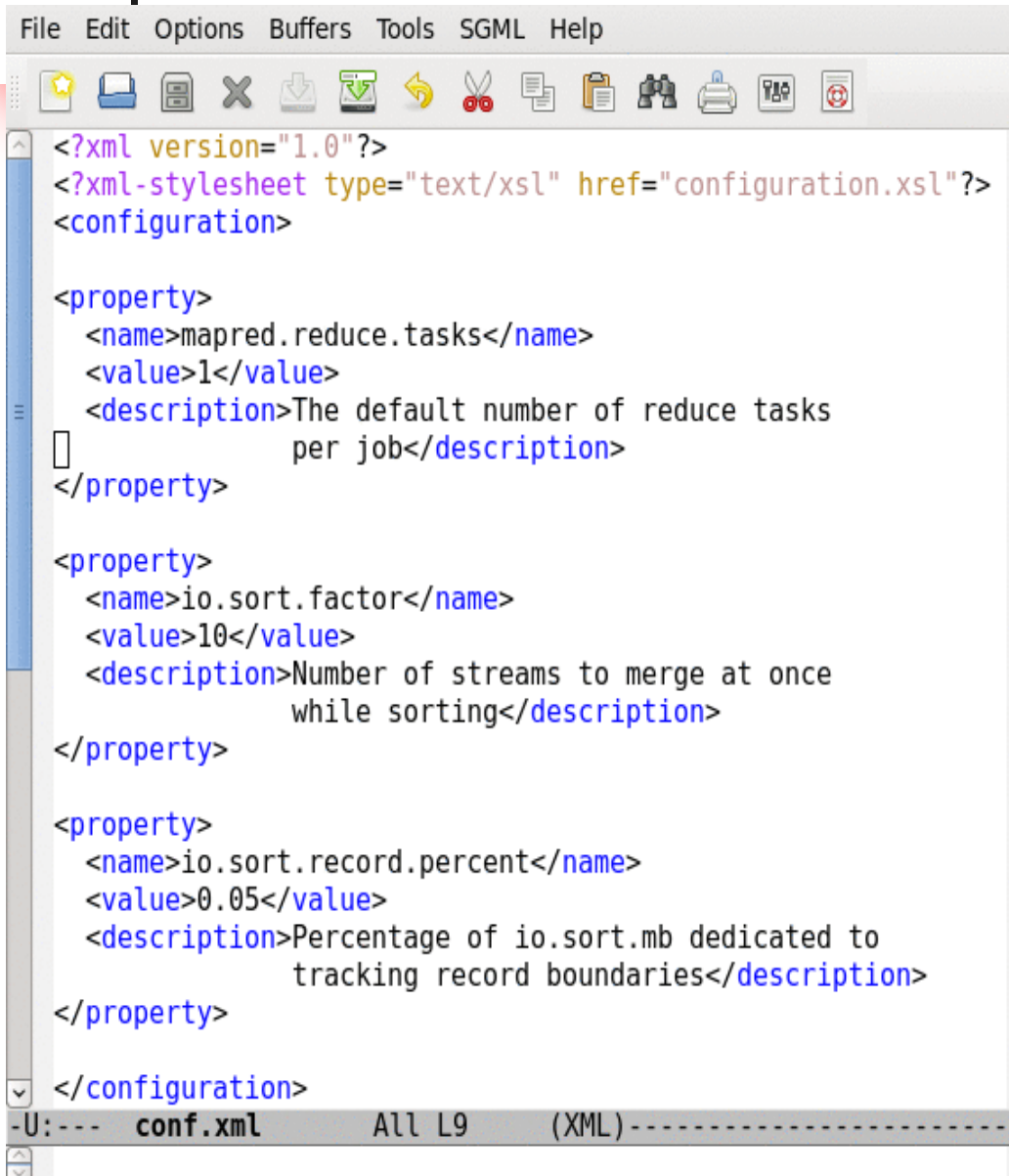
<each, (1, 1) <in, (1)>



<For, (1)> <string, (1)> <word, (1)>



Job Configuration Parameters

A screenshot of an XML editor window. The title bar shows 'File Edit Options Buffers Tools SGML Help'. The toolbar contains icons for opening, saving, undo, redo, cut, copy, paste, and other standard editing functions. The main text area displays XML code for a Hadoop configuration file. The code starts with an XML declaration and a stylesheet reference, followed by a root element 'configuration'. Inside 'configuration', there are three 'property' elements. Each 'property' element has 'name', 'value', and 'description' sub-elements. The first property is 'mapred.reduce.tasks' with value '1'. The second is 'io.sort.factor' with value '10'. The third is 'io.sort.record.percent' with value '0.05'. The code ends with the closing 'configuration' tag. The status bar at the bottom shows '-U:--- conf.xml All L9 (XML)-----'.

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
<configuration>

  <property>
    <name>mapred.reduce.tasks</name>
    <value>1</value>
    <description>The default number of reduce tasks
      per job</description>
  </property>

  <property>
    <name>io.sort.factor</name>
    <value>10</value>
    <description>Number of streams to merge at once
      while sorting</description>
  </property>

  <property>
    <name>io.sort.record.percent</name>
    <value>0.05</value>
    <description>Percentage of io.sort.mb dedicated to
      tracking record boundaries</description>
  </property>

</configuration>
```

- 190+ parameters in Hadoop
- Set manually or defaults are used
- <http://hadoop.apache.org/docs/r1.2.1/api/org/apache/hadoop/conf/Configuration.html>

10.1 Analytics for Unstructured Data

10.1.3 Apache Hadoop

- Developing and Executing a Hadoop MapReduce Program
- Common practice is to use an IDE tool such as Eclipse
- The MapReduce program consists of three Java files
 - Driver code, mapper code, and reducer code
- Java code is compiled and stored in a JAR file and executed against the specified HDFS input files



10.1 Analytics for Unstructured Data

10.1.3 Apache Hadoop

```
% tar xzf hadoop-1.0.4.tar.gz
% export HADOOP_HOME=$(pwd)/hadoop-1.0.4
% $HADOOP_HOME/bin/hadoop version
Hadoop 1.0.4
```

In another directory, checkout the Git repository that accompanies this example:

```
% git clone git://github.com/tomwhite/hadoop-drdobbs.git
% cd hadoop-drdobbs
% mvn install
```

The repository contains a small amount of sample data for testing:

```
% cat data/*.tsv
dobbs      2007      20        18        15
dobbs      2008      22        20        12
doctor     2007      545525    366136    57313
doctor     2008      668666    446034    72694
```

10.1 Analytics for Unstructured Data

10.1.3 Apache Hadoop

```
public class ProjectionMapper extends Mapper<LongWritable, Text, Text, LongWritable> {
    private Text word = new Text();
    private LongWritable count = new LongWritable();

    @Override
    protected void map(LongWritable key, Text value, Context context)
        throws IOException, InterruptedException {
        // value is tab separated values: word, year, occurrences, #books, #pages
        // we project out (word, occurrences) so we can sum over all years
        String[] split = value.toString().split("\t+");
        word.set(split[0]);
        if (split.length > 2) {
            try {
                count.set(Long.parseLong(split[2]));
                context.write(word, count);
            } catch (NumberFormatException e) {
                // cannot parse - ignore
            }
        }
    }
}
```

("dobbs", 20)
("dobbs", 22)
("doctor", 545525)
("doctor", 668666)

```
package com.tom_e_white.drdobbs.mapreduce;

import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;
```

10.1 Analytics for Unstructured Data

10.1.3 Apache Hadoop

```
public class LongSumReducer<KEY> extends Reducer<KEY, LongWritable, KEY, LongWritable> {  
    private LongWritable result = new LongWritable();  
  
    public void reduce(KEY key, Iterable<LongWritable> values,  
        Context context) throws IOException, InterruptedException {  
        long sum = 0;  
        for (LongWritable val : values) {  
            sum += val.get();  
        }  
        result.set(sum);  
        context.write(key, result);  
    }  
}
```

("dobbs", 42)
("doctor", 1214191)

```
package org.apache.hadoop.mapreduce.lib.reduce;  
  
import java.io.IOException;  
import org.apache.hadoop.io.LongWritable;  
import org.apache.hadoop.mapreduce.Reducer;
```

10.1 Analytics for Unstructured Data

10.1.3 Apache Hadoop

```
public class AggregateJob extends Configured implements Tool {
```

```
    @Override
```

```
    public int run(String[] args) throws Exception {  
        Job job = new Job(getConf());  
        job.setJarByClass(getClass());  
        job.setJobName(getClass().getSimpleName());
```

```
        FileInputFormat.addInputPath(job, new Path(args[0]));  
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
```

```
        job.setMapperClass(ProjectionMapper.class);  
job.setCombinerClass(LongSumReducer.class);  
job.setReducerClass(LongSumReducer.class);
```

```
        job.setOutputKeyClass(Text.class);  
        job.setOutputValueClass(LongWritable.class);
```

```
        return job.waitForCompletion(true) ? 0 : 1;  
    }
```

```
    public static void main(String[] args) throws Exception {  
        int rc = ToolRunner.run(new AggregateJob(), args);  
        System.exit(rc);  
    }
```

```
package com.tom_e_white.drdobbs.mapreduce;
```

```
import org.apache.hadoop.conf.Configured;  
import org.apache.hadoop.fs.Path;  
import org.apache.hadoop.io.LongWritable;  
import org.apache.hadoop.io.Text;  
import org.apache.hadoop.mapreduce.Job;  
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;  
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  
import org.apache.hadoop.mapreduce.lib.reduce.LongSumReducer;  
import org.apache.hadoop.util.Tool;  
import org.apache.hadoop.util.ToolRunner;
```


10.1 Analytics for Unstructured Data

10.1.3 Apache Hadoop

https://hadoop.apache.org/docs/r1.0.4/commands_manual.html#jar

```
% $HADOOP_HOME/bin/hadoop jar target/hadoop-drdobbs-1.0.0.jar \  
com.tom_e_white.drdobbs.mapreduce.AggregateJob data output
```

```
% cat output/part-r-00000  
dobbs    42  
doctor   1214191
```

```
% hadoop fs -copyFromLocal data data
```

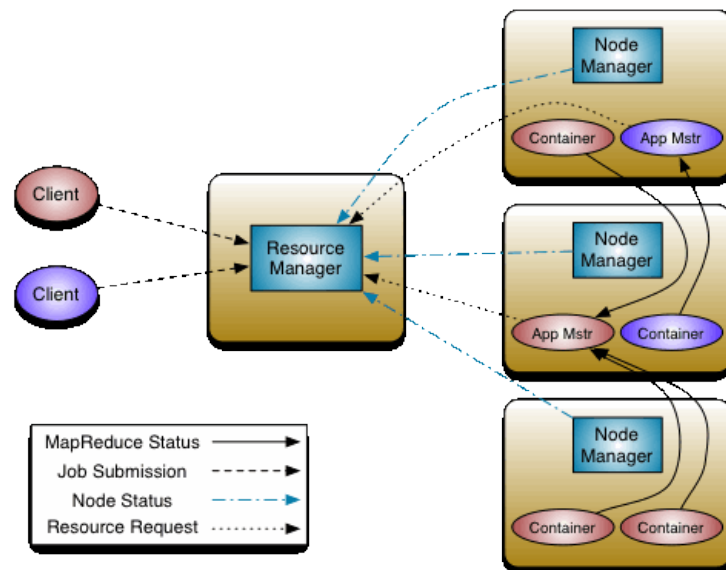
```
% hadoop jar target/hadoop-drdobbs-1.0.0.jar \  
com.tom_e_white.drdobbs.mapreduce.AggregateJob \  
-D fs.default.name=hdfs://localhost:8020 \  
-D mapred.job.tracker=localhost:8021 \  
data output
```

```
% hadoop fs -cat output/part-r-00000  
dobbs    42  
doctor   1214191
```

10.1 Analytics for Unstructured Data

10.1.3 Apache Hadoop

- Yet Another Resource Negotiator (YARN)
 - YARN is designed to separate the MapReduce functionality from the management of running jobs and the distributed environment (Scheduler and ApplicationsManager).



Details:

<https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html>



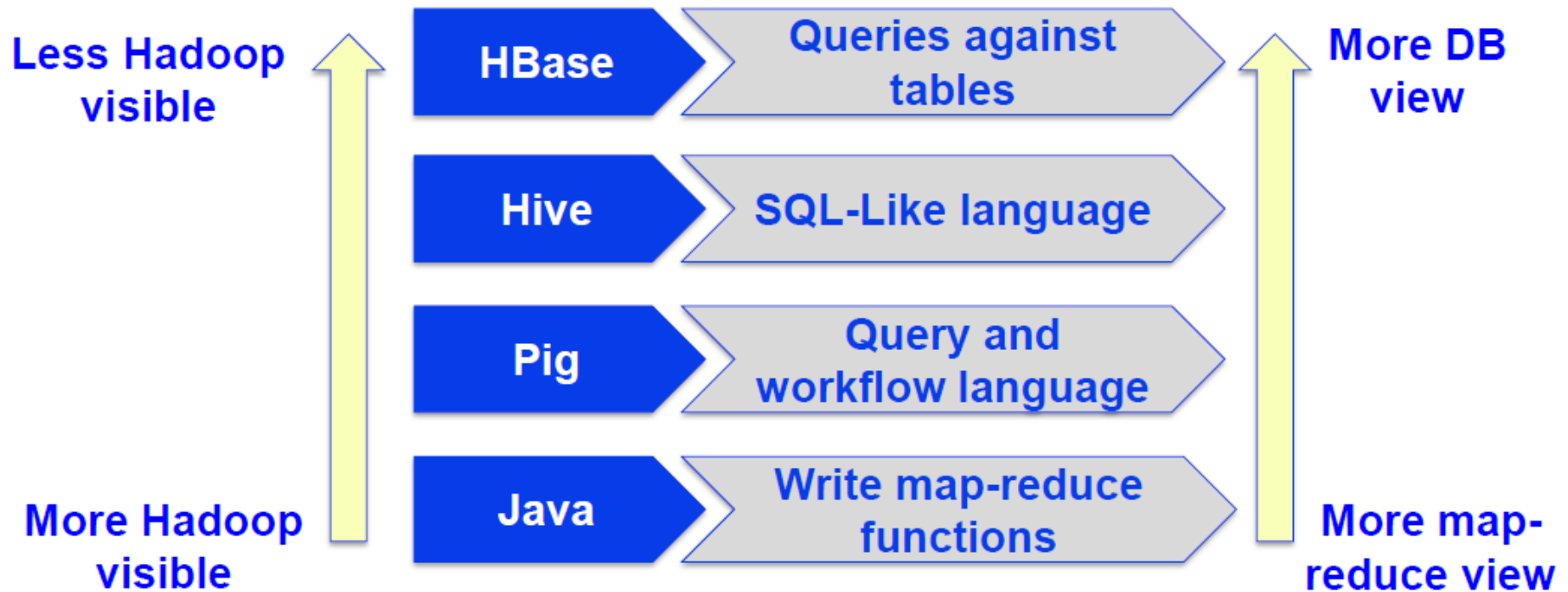
Lecture Break



10.2 The Hadoop Ecosystem

- Tools have been developed to make Hadoop easier to use and provide additional functionalities and features
 - Pig – provides a high-level data-flow programming language
 - Hive – provides SQL-like access
 - HBase – provides real-time reads and writes
 -

10.2 The Hadoop Ecosystem



10.2 The Hadoop Ecosystem

10.2.1 Pig

- Pig consists of
 - A data flow language called **Pig Latin**
 - An environment to execute the Pig code
- Example of Pig commands (in Hadoop)

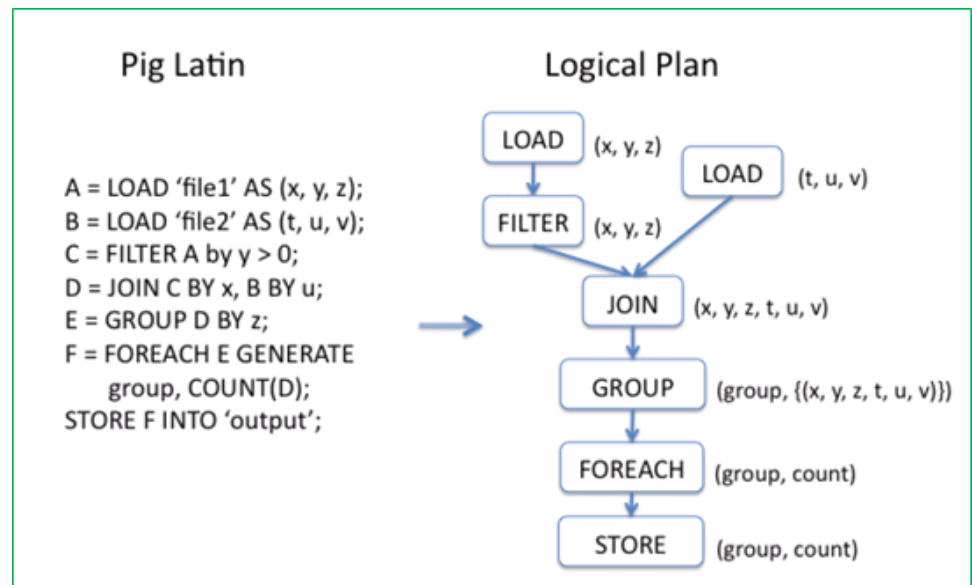
```
$ pig
grunt> records = LOAD '/user/customer.txt' AS
                    (cust_id:INT, first_name:CHARARRAY,
                     last_name:CHARARRAY,
                     email_address:CHARARRAY);
grunt> filtered_records = FILTER records
                    BY email_address matches '.*@isp.com';
grunt> STORE filtered_records INTO '/user/isp_customers';
grunt> quit
$
```

10.2 The Hadoop Ecosystem

10.2.1 Pig

- Framework for analyzing large un-structured and semi-structured data on top of Hadoop.

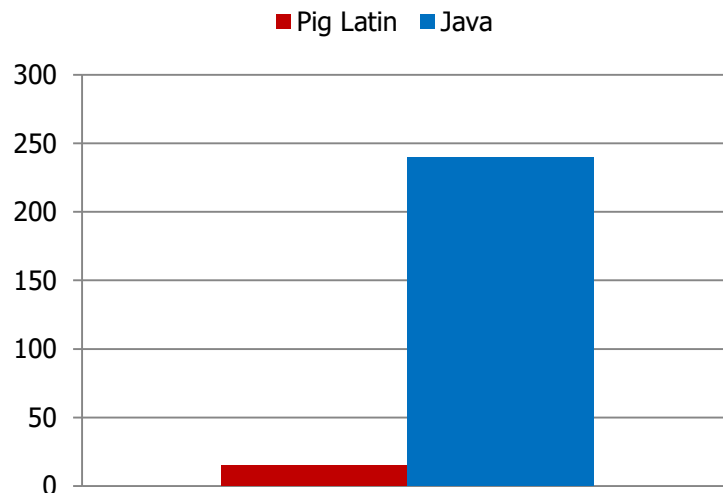
The Pig engine parses and compiles Pig Latin scripts into MapReduce jobs run on top of Hadoop. Pig Latin is a data processing language; the high level language interface for Hadoop.



10.2 The Hadoop Ecosystem

10.2.1 Pig

- Faster development
 - Fewer lines of code (Writing map reduce like writing SQL queries)
 - Re-use others' code (e.g. PiggyBank)
- One test: Find the top 5 words with highest frequencies
 - 10 lines of Pig Latin V.S 200 lines in Java
 - 15 minutes in Pig Latin V.S 4 hours in Java



10.2 The Hadoop Ecosystem

10.2.1 Pig

■ Word Count using MapReduce

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobClient;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

public class WordCount extends Configured implements Tool {

    public static class MapClass extends MapReduceBase
        implements Mapper<LongWritable, Text, Text, IntWritable> {

        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();

        public void map(LongWritable key, Text value,
            OutputCollector<Text, IntWritable> output,
            Reporter reporter) throws IOException {
            String line = value.toString();
            StringTokenizer itr = new StringTokenizer(line);
            while (itr.hasMoreTokens()) {
                word.set(itr.nextToken());
                output.collect(word, one);
            }
        }
    }

    public static class Reduce extends MapReduceBase
        implements Reducer<Text, IntWritable, Text, IntWritable> {

        public void reduce(Text key, Iterator<IntWritable> values,
            OutputCollector<Text, IntWritable> output,
            Reporter reporter) throws IOException {
            int sum = 0;
            while (values.hasNext()) {
                sum += values.next().get();
            }
            output.collect(key, new IntWritable(sum));
        }
    }
}
```

```
public int run(String[] args) throws Exception {
    JobConf conf = new JobConf(getConf(), WordCount.class);
    conf.setJobName("wordcount");

    conf.setOutputKeyClass(Text.class);
    conf.setOutputValueClass(IntWritable.class);

    conf.setMapperClass(MapClass.class);
    conf.setCombinerClass(Reduce.class);
    conf.setReducerClass(Reduce.class);

    List<String> other_args = new ArrayList<String>();
    for(int i=0; i < args.length; ++i) {
        try {
            if ("-m".equals(args[i])) {
                conf.setNumMapTasks(Integer.parseInt(args[++i]));
            } else if ("-r".equals(args[i])) {
                conf.setNumReduceTasks(Integer.parseInt(args[++i]));
            } else {
                other_args.add(args[i]);
            }
        } catch (NumberFormatException except) {
            System.out.println("ERROR: Integer expected instead of " + args[i]);
            return printUsage();
        } catch (ArrayIndexOutOfBoundsException except) {
            System.out.println("ERROR: Required parameter missing from " +
                args[i-1]);
            return printUsage();
        }
    }
    // Make sure there are exactly 2 parameters left.
    if (other_args.size() != 2) {
        System.out.println("ERROR: Wrong number of parameters: " +
            other_args.size() + " instead of 2.");
        return printUsage();
    }
    FileInputFormat.setInputPaths(conf, other_args.get(0));
    FileOutputFormat.setOutputPath(conf, new Path(other_args.get(1)));
    JobClient.runJob(conf);
    return 0;
}

public static void main(String[] args) throws Exception {
    int res = ToolRunner.run(new Configuration(), new WordCount(), args);
    System.exit(res);
}
```

10.2 The Hadoop Ecosystem

10.2.1 Pig

■ Word Count using Pig

```
Lines=LOAD 'input/hadoop.log' AS (line: chararray);
Words = FOREACH Lines GENERATE FLATTEN(TOKENIZE(line)) AS word;
Groups = GROUP Words BY word;
Counts = FOREACH Groups GENERATE group, COUNT(Words);
Results = ORDER Words BY Counts DESC;
Top5 = LIMIT Results 5;
STORE Top5 INTO /output/top5words;
```

10.2 The Hadoop Ecosystem

10.2.1 Pig

- User-Defined Functions (UDFs) can be written to take advantage of the combiner
- Four join implementations are built and available
- Writing load and store functions is easy if an InputFormat and OutputFormat exist
- Multi-query: pig will combine certain types of operations together in a single pipeline to reduce the number of times data is scanned.
- “Order by” provides total ordering across reducers in a balanced way
- Piggybank, a collection of user contributed UDFs

<https://cwiki.apache.org/confluence/display/PIG/PiggyBank>

By Hui Li, Judy Qiu, Adam Kawa

10.2 The Hadoop Ecosystem

10.2.1 Pig

- 70% of production jobs at Yahoo
 - 10k per day
- Twitter, LinkedIn, Ebay, AOL,...
- Used to
 - Process web logs
 - Build user behavior models
 - Process images
 - Build maps of the web
 - Do research on large data sets



10.2 The Hadoop Ecosystem

10.2.1 Pig

- Data Types
 - Scalar Types:
 - Int, long, float, double, boolean, null, chararray, bytearray;
 - Complex Types:
 - A Field is a piece of data
 - A Tuple is an ordered set of fields
 - A Bag is a collection of tuples
 - A Relation is an outer bag
 - Samples:
 - Tuple → Row in Database
 - (0002576169, Tome, 20, 4.0)
 - Relation, Bag → Table or View in Database
 - {(0002576169 , Tome, 20, 4.0),
 - (0002576170, Mike, 20, 3.6),
 - (0002576171 Lucy, 19, 4.0), }



10.2 The Hadoop Ecosystem

10.2.1 Pig

- Loading data
 - **LOAD** loads input data
 - Lines=**LOAD** 'input/access.log' AS (line: chararray);
- Projection
 - **FOREACH** ... **GENERATE** ... (similar to SELECT)
 - takes a set of expressions and applies them to every record.
- Grouping
 - **GROUP** collects together records with the same key
- Dump/Store
 - **DUMP** displays results to screen, **STORE** save results to file system
- Aggregation
 - **AVG, COUNT, MAX, MIN, SUM**

10.2 The Hadoop Ecosystem

10.2.1 Pig

- Pig Data Loader

- **PigStorage**: loads/stores relations using field-delimited text format

```
(1,John,18,4.0F)
(2,Mary,19,3.8F)
(3,Bill,20,3.9F)
```

```
students = load 'student.txt' using PigStorage('\t')
          as (studentid: int, name:chararray, age:int, gpa:double);
```

- **TextLoader**: loads relations from a plain-text format
- **BinStorage**: loads/stores relations from or to binary files
- **PigDump**: stores relations by writing the toString() representation of tuples, one per line

10.2 The Hadoop Ecosystem

10.2.1 Pig

- **Foreach ... Generate**

- The **Foreach ... Generate** statement iterates over the members of a bag

```
studentid = FOREACH students GENERATE studentid, name;
```

- The result of a **Foreach** is another bag
- Elements are named as in the input bag

10.2 The Hadoop Ecosystem

10.2.1 Pig

| Eval | Load/Store | Math | String | DateTime |
|------------|--------------|------------|-------------------|------------------|
| AVG | BinStorage() | ABS | INDEXOF | AddDuration |
| CONCAT | JsonLoader | CEIL | LAST_ INDEX_OF | CurrentTime |
| COUNT | JsonStorage | COS, ACOS | LCFORST | DaysBetween |
| COUNT_STAR | PigDump | EXP | LOWER | GetDay |
| DIFF | PigStorage | FLOOR | REGEX_ EXTRACT | GetHour |
| IsEmpty | TextLoader | LOG, LOG10 | REPLACE | GetMinute |
| MAX | HBaseStorage | RANDOM | STRSPLIT | GetMonth |
| MIN | | ROUND | SUBSTRING | GetWeek |
| SIZE | | SIN, ASIN | TRIM | GetWeekYear |
| SUM | | SQRT | UCFIRST | GetYear |
| TOKENIZE | | TAN, ATAN | UPPER | MinutesBetween |
| | | | | SubtractDuration |
| | | | | ToDate |



10.2 The Hadoop Ecosystem

10.2.1 Pig

- How to run Pig Latin scripts
 - **Local** mode
 - Local host and local file system is used
 - Neither Hadoop nor HDFS is required
 - Useful for prototyping and debugging
 - **MapReduce** mode
 - Run on a Hadoop cluster and HDFS
 - **Batch** mode - run a script directly
 - Pig -x local my_pig_script.pig
 - Pig -x mapreduce my_pig_script.pig
 - **Interactive** mode use the Pig shell to run script
 - Grunt> Lines = LOAD '/input/input.txt' AS (line:chararray);
 - Grunt> Unique = DISTINCT Lines;
 - Grunt> DUMP Unique;

10.2 The Hadoop Ecosystem

10.2.1 Pig

- Find the top 10 most visited pages in each category

Visits

| User | Url | Time |
|------|------------|-------|
| Amy | cnn.com | 8:00 |
| Amy | bbc.com | 10:00 |
| Amy | flickr.com | 10:05 |
| Fred | cnn.com | 12:00 |



Url Info

| Url | Category | PageRank |
|------------|----------|----------|
| cnn.com | News | 0.9 |
| bbc.com | News | 0.8 |
| flickr.com | Photos | 0.7 |
| espn.com | Sports | 0.9 |

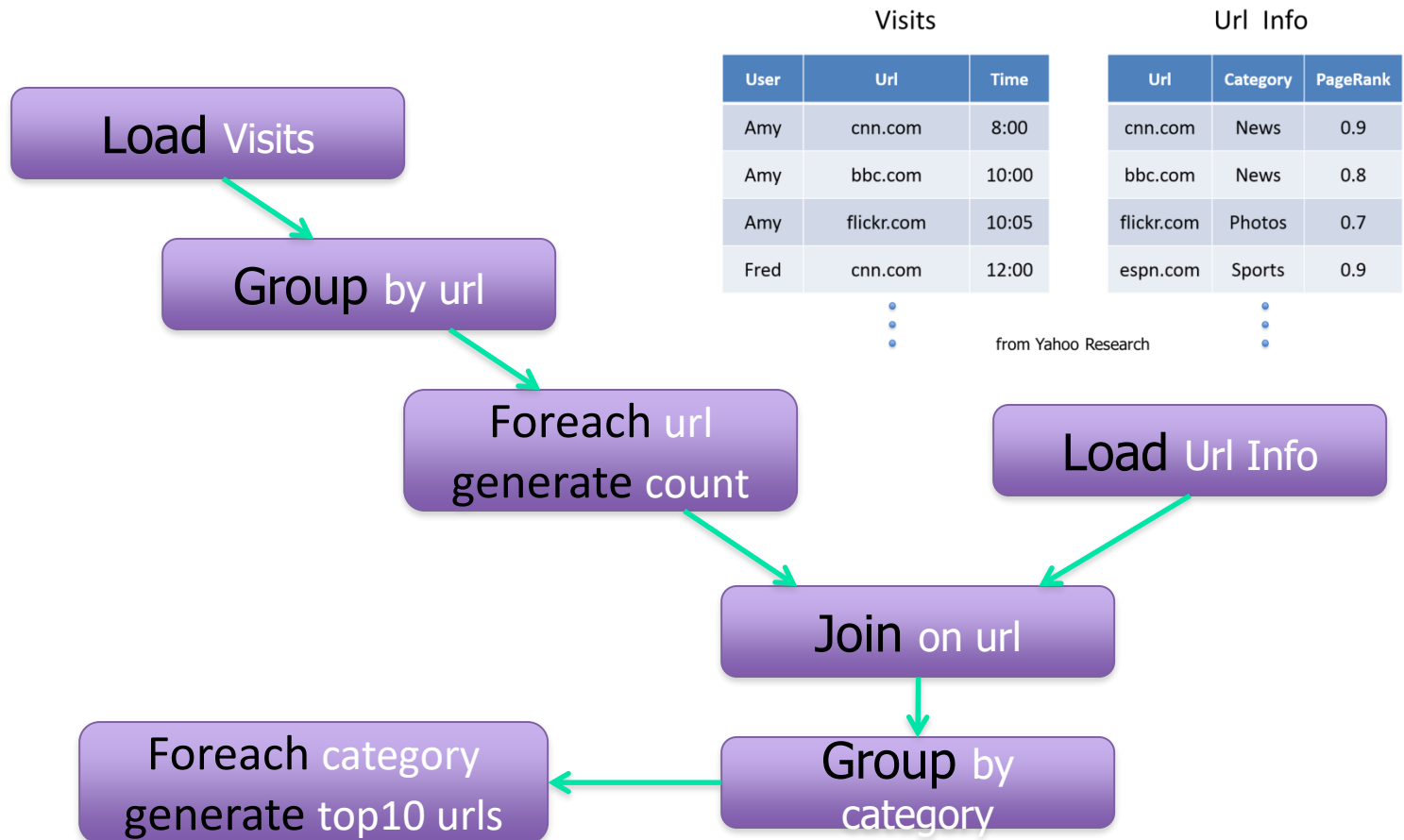


from Yahoo Research

10.2 The Hadoop Ecosystem

10.2.1 Pig

- Find the top 10 most visited pages in each category





10.2 The Hadoop Ecosystem

10.2.1 Pig

```
visits          = load '/data/visits' as (user, url, time);
gVisits         = group visits by url;
visitCounts     = foreach gVisits generate url, count(visits);

urlInfo         = load '/data/urlInfo' as (url, category, pRank);
visitCounts     = join visitCounts by url, urlInfo by url;
// select * from visitCounts v,urlInfo u where v.url = u.url
gCategories     = group visitCounts by category;
topUrls         = foreach gCategories generate top(visitCounts,10);

store topUrls into '/data/topUrls';
```

10.2 The Hadoop Ecosystem

10.2.2 Hive

- The Hive language, Hive Query Language (HiveQL), resembles SQL rather than a scripting language
- Example Hive code

```
hive> create table customer (  
    cust_id bigint,  
    first_name string,  
    last_name  string,  
    email_address string)  
row format delimited  
fields terminated by '\t';
```

10.2 The Hadoop Ecosystem

10.2.2 Hive

- Yahoo worked on Pig to facilitate application deployment on Hadoop.
 - Their need mainly was focused on processing unstructured data into structured data.
- Simultaneously, Facebook started working on deploying warehouse solutions on Hadoop that resulted in Hive.
 - The size of data being collected and analyzed in industry for business intelligence (BI) is growing rapidly making traditional warehousing solution prohibitively expensive.

10.2 The Hadoop Ecosystem

10.2.2 Hive

- MapReduce is very low level and requires programmers to write custom programs.
- HIVE supports queries expressed in SQL-like language called HiveQL which are compiled into MR jobs that are executed on Hadoop.
- Hive also allows MR scripts
- It also includes MetaStore that contains schemas and statistics that are useful for data explorations, query optimization, and query compilation.
- At Facebook, Hive warehouse contains tens of thousands of tables; it stores over 700TB and is used for reporting and ad-hoc analyses by 200 Facebook employees.

10.2 The Hadoop Ecosystem

10.2.2 Hive

- Hive structures data into well-understood database concepts such as: tables, rows, cols, partitions
- It supports primitive types: integers, floats, doubles, and strings
- Hive also supports:
 - associative arrays: `map<key-type, value-type>`
 - Lists: `list<element type>`
 - Structs: `struct<file name: file type...>`
- SerDe: serialize and deserialized API is used to move data in and out of tables



10.2 The Hadoop Ecosystem

10.2.2 Hive

- Query Language (HiveQL)
 - Subset of SQL
 - Meta-data queries
 - Limited equality and join predicates



10.2 The Hadoop Ecosystem

10.2.2 Hive

- `CREATE TABLE docs (line STRING);`
- `LOAD DATA INPATH`
`'hdfs://localhost:9000/user/hduser/employee123.txt'`
`OVERWRITE INTO TABLE docs;`
- `CREATE TABLE word_counts AS`
- `SELECT word, count(1) AS count FROM`
- `(SELECT explode(split(line, '\\s')) AS word FROM`
`docs) w`
- `GROUP BY word`
- `ORDER BY word;`



10.2 The Hadoop Ecosystem

10.2.2 Hive

- Hive and Hadoop are extensively used in Facebook for different kinds of operations.
- 700 TB = 2.1Petabyte after replication!

<https://www.facebook.com/notes/facebook-engineering/hive-a-petabyte-scale-data-warehouse-using-hadoop/89508453919/>

https://web.archive.org/web/20110728063630/http://www.sfbayacm.org/wp/wp-content/uploads/2010/01/sig_2010_v21.pdf



10.2 The Hadoop Ecosystem

Pig vs Hive

| Pig | Hive |
|---|---|
| Procedural Data Flow Language | Declarative SQLish Language |
| For Programming | For Reporting |
| Mainly used by Programmers | Mainly used by Data Analysts |
| Operates on the client side of a cluster. | Operates on the server side of a cluster. |
| Does not have a dedicated metadata database. | Makes use of exact variation of dedicated SQL DDL language by defining tables beforehand. |
| Pig is SQL like but varies to a great extent. | Directly leverages SQL and is easy to learn for database experts. |

10.2 The Hadoop Ecosystem

10.2.3 HBase

- HBase provides random read and write access to huge datasets for basic database operations in a key-value fashion.

- Checkout (Step 2 of 4)

Choose a shipping address:

| | | | Last Used |
|--------------------------|---|---|-------------|
| <input type="checkbox"/> | 1600 Pennsylvania Avenue NW Washington DC, 20500 USA | <input type="button" value="Edit"/> <input type="button" value="Delete"/> | 15-Apr-2014 |
| <input type="checkbox"/> | London SW1A 1AA, United Kingdom | <input type="button" value="Edit"/> <input type="button" value="Delete"/> | 15-Mar-2014 |
| <input type="checkbox"/> | आगरा, उत्तर प्रदेश 282001, India | <input type="button" value="Edit"/> <input type="button" value="Delete"/> | 14-Feb-2014 |

10.2 The Hadoop Ecosystem

10.2.3 HBase

- HBase is an **open-source, distributed, column-oriented** database built on top of HDFS based on Google BigTable.
- A distributed NoSQL data store that can scale horizontally to 1,000s of commodity servers and petabytes of indexed storage.
- Designed to operate on top of the Hadoop distributed file system (HDFS) or Kosmos File System (KFS, aka Cloudstore) for scalability, fault tolerance, and high availability.

HBase Data Example

HTable table = ...

Text row = new Text("enclosure1");

Text col1 = new Text("animal:type");

Text col2 = new Text("animal:size");

BatchUpdate update = new BatchUpdate(row); update = new BatchUpdate(row);

update.put(col1, "lion".getBytes("UTF-8")); update.put(col1, "zebra".getBytes("UTF-8"));

update.put(col2, "big".getBytes("UTF-8")); table.commit(update);

table.commit(update);

| Row | Timestamp | Column family: animal: | |
|------------|-----------|---------------------------|-------------|
| | | animal:type | animal:size |
| enclosure1 | t2 | zebra | |
| | t1 | lion | big |
| ... | ... | ... | ... |



HBase Data Example

Retrieve a cell

```
Cell = table.getRow("enclosure1").getColumn("animal:type").getValue();
```

Retrieve a row

```
RowResult = table.getRow( "enclosure1" );
```

Scan through a range of rows

```
Scanner s = table.getScanner( new String[] { "animal:type" } );
```

| Row | Timestamp | Column family: animal: | |
|------------|-----------|---------------------------|-------------|
| | | animal:type | animal:size |
| enclosure1 | t2 | zebra | |
| | t1 | lion | big |
| ... | ... | ... | ... |



10.2 The Hadoop Ecosystem

10.2.3 HBase

HDFS

HDFS is a distributed file system suitable for storing large files.

HDFS does not support fast individual record lookups.

It provides high latency in batch processing; no concept of batch processing.

It provides only sequential access of data.

HBase

HBase is a database built on top of the HDFS.

HBase provides fast lookups for large tables.

It provides low latency access to single rows from billions of records (Random access).

HBase internally uses Hash tables and provides random access, and it stores the data in indexed HDFS files for faster lookups.

https://www.tutorialspoint.com/hbase/hbase_overview.htm

10.2 The Hadoop Ecosystem

10.2.3 HBase

■ Advantages:

- Automatic partitioning
- Scale linearly and automatically with new nodes
- Commodity hardware
- Fault tolerance
- Batch processing
- Random read and write operations

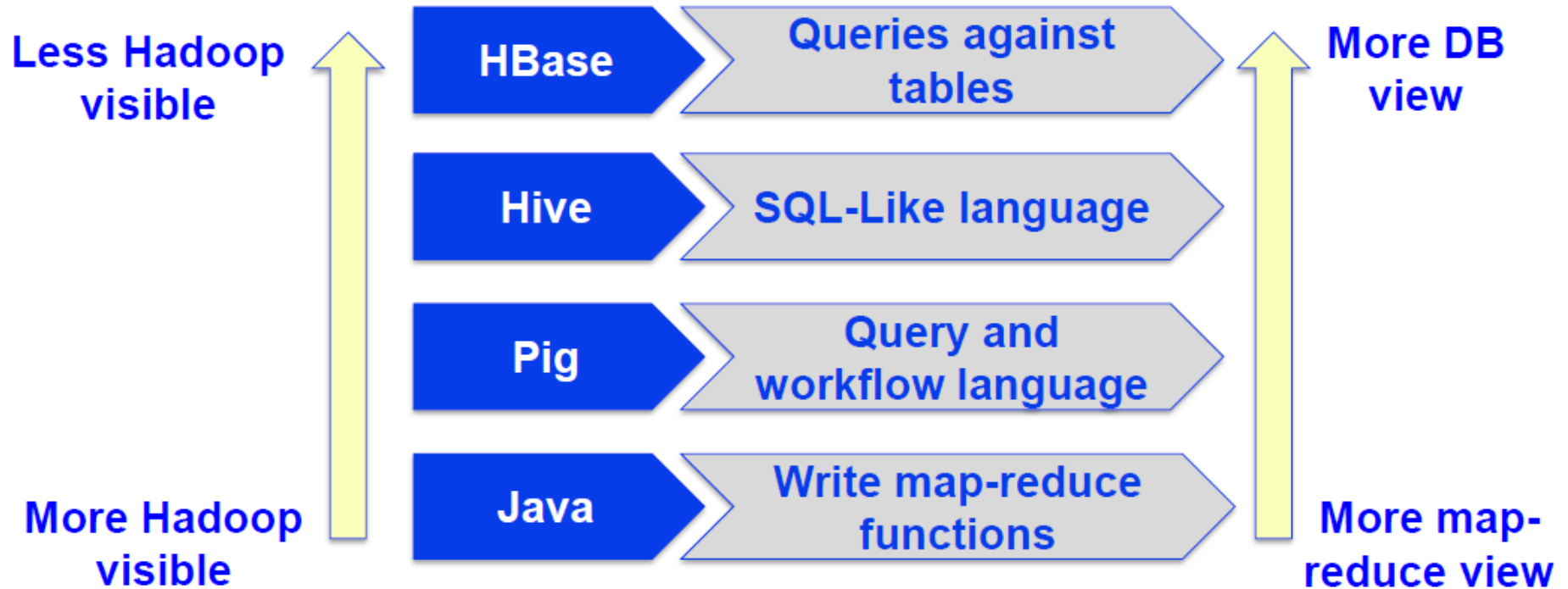
10.2 The Hadoop Ecosystem

10.2.3 HBase

■ Disadvantages:

- We cannot expect completely to use HBase as a replacement for traditional models. Some of the traditional models features cannot be supported by HBase
- HBase cannot perform functions like SQL. It doesn't support SQL structure, so it does not contain any query optimizer
- HBase is CPU and Memory intensive with large sequential input or output access while Map Reduce jobs are primarily input or output bound with fixed memory. HBase integrated with Map-reduce jobs will result in unpredictable latencies

10.2 The Hadoop Ecosystem





References

- Original paper
(<http://labs.google.com/papers/mapreduce.html>)
- On Wikipedia
(<http://en.wikipedia.org/wiki/MapReduce>)
- Hadoop – MapReduce in Java
(<http://lucene.apache.org/hadoop/>)
- <http://code.google.com/edu/parallel/mapreduce-tutorial.html>



References

1. Apache Hadoop Tutorial: <http://hadoop.apache.org>
http://hadoop.apache.org/core/docs/current/mapred_tutorial.html
2. Dean, J. and Ghemawat, S. 2008. **MapReduce: simplified data processing on large clusters.**
Communication of ACM 51, 1 (Jan. 2008), 107-113.
3. Cloudera Videos by Aaron Kimball:
<http://www.cloudera.com/hadoop-training-basic>
4. <http://www.cse.buffalo.edu/faculty/bina/mapreduce.html>



Summary

- 10.1 Analytics for Unstructured Data
 - 10.1.1 Use Cases
 - 10.1.2 MapReduce
 - 10.1.3 Apache Hadoop
- 10.2 The Hadoop Ecosystem
 - 10.2.1 Pig
 - 10.2.2 Hive
 - 10.2.3 HBase
- Summary

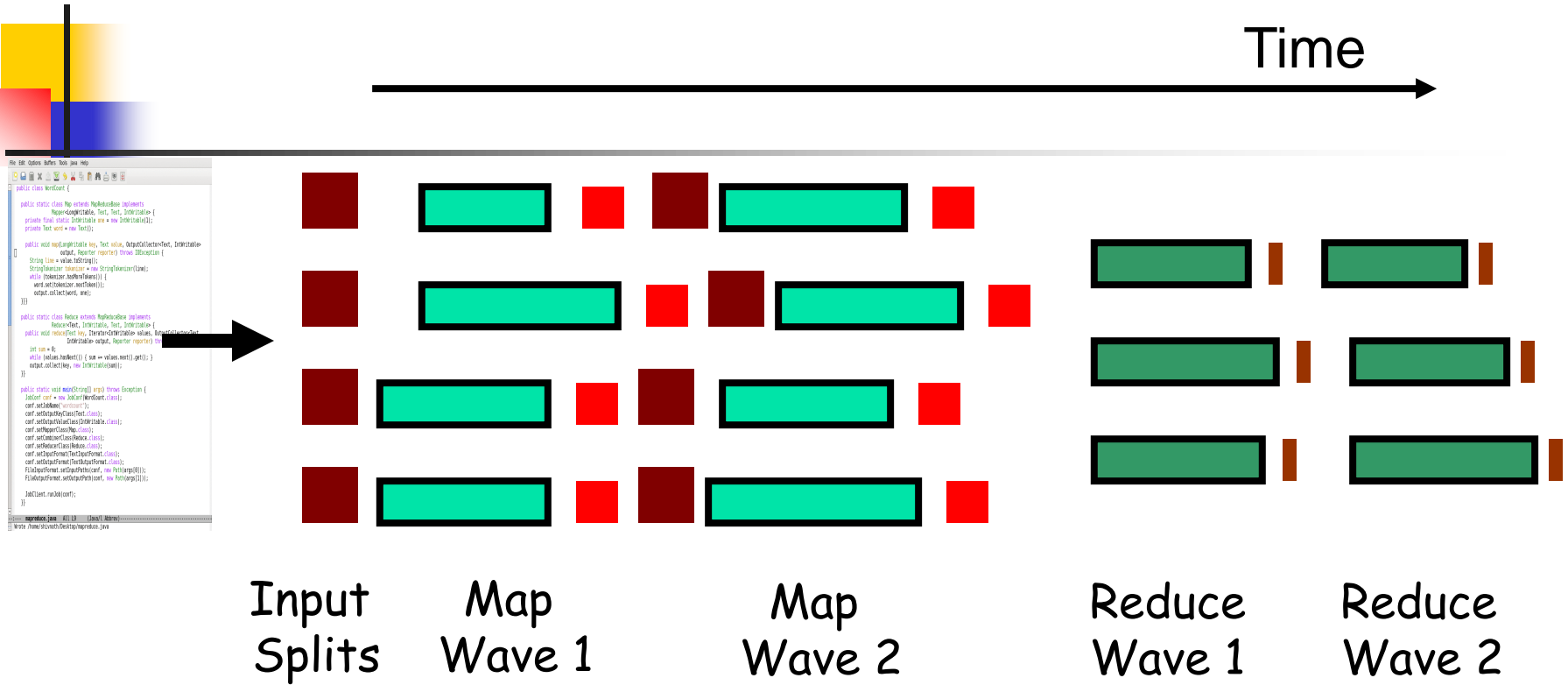
- The Partitioner interface defines the `getPartition()` method
 - ◆ **Input:** a *key*, a *value* and the number of partitions
 - ◆ **Output:** a partition id for the given *key*, *value* pair
- The default Partitioner is the HashPartitioner:

```
int getPartition(K key, V value, int numPartitions) {  
    return key.hashCode() % numPartitions;  
}
```

| Key | key.hashCode() | partitionId (0-2) |
|--------|----------------|-------------------|
| hello | 12847 | 1 |
| world | 23874 | 0 |
| map | 82375 | 1 |
| reduce | 12839 | 2 |

numPartitions = 3

Lifecycle of a MapReduce Job





10.3 NoSQL

- NoSQL = Not only Structured Query Language
- Four major categories of NoSQL tools
 - Key/value stores – contains data (the value) accessed by the key
 - Document stores – good when the value of the key/value pair is a file
 - Column family stores – good for sparse datasets
 - Graph stores – good for items and relationships between them
 - Social networks like Facebook and LinkedIn



10.3 NoSQL

- Examples of NoSQL Data Stores

| Category | Data Store | Website |
|---------------|------------|--|
| Key/Value | Redis | <code>redis.io</code> |
| | Voldemort | <code>www.project-voldemort.com/voldemort</code> |
| Document | CouchDB | <code>couchdb.apache.org</code> |
| | MongoDB | <code>www.mongodb.org</code> |
| Column family | Cassandra | <code>cassandra.apache.org</code> |
| | HBase | <code>hbase.apache.org/</code> |
| Graph | FlockDB | <code>github.com/twitter/flockdb</code> |
| | Neo4j | <code>www.neo4j.org</code> |