

Cloud Computing using MapReduce, Hadoop, Spark

Andy Konwinski
andyk@cs.berkeley.edu



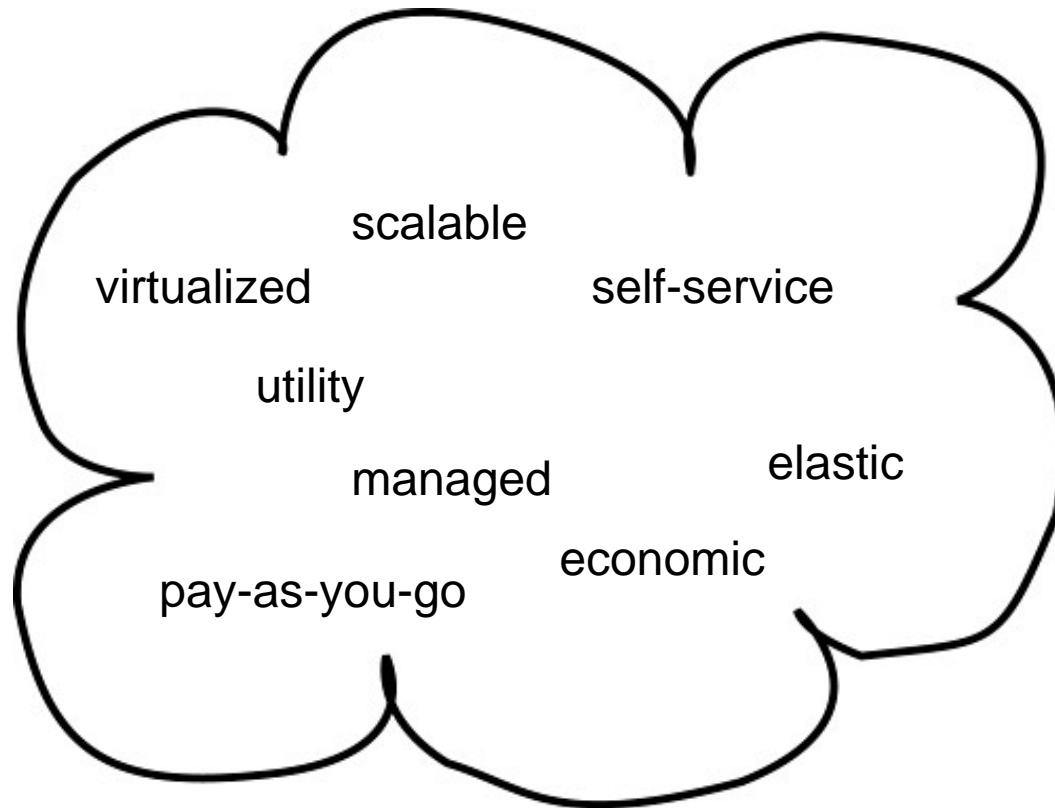
Why this talk?

- From parallel to distributed
 - “Big Data” too big to fit on one computer
- SPMD might not be best for your ...
 - **Application** (compute bound vs. data bound)
 - **Environment** (public clouds)

Outline

- Cloud Overview
- MapReduce
- MapReduce Examples
- Introduction to Hadoop
- Beyond MapReduce
- Summary

What is Cloud Computing?



What is Cloud Computing?

- **Cloud:** large Internet services running on 10,000s of machines (Amazon, Google, Microsoft, etc.)
- **Cloud computing:** services that let external customers rent cycles and storage
 - *Amazon EC2*: virtual machines at 8.5¢/hour, billed hourly
 - *Amazon S3*: storage at 15¢/GB/month
 - *Google AppEngine*: free up to a certain quota

Core Cloud Concepts

- Virtualization
- Self-service (personal credit card) & pay-as-you-go
- Economic incentives
 - Provider: Sell unused resources
 - Customer: no upfront capital costs building data centers, buying servers, etc

Core Cloud Concepts

- Infinite scale ...

From: Andy Konwinski <andyk@cs.berkeley.edu>

To: [REDACTED]

Cc: [REDACTED]

Sent: Wed May 05 12:31:24 2010

Subject: Re: Question on recent AWS usage

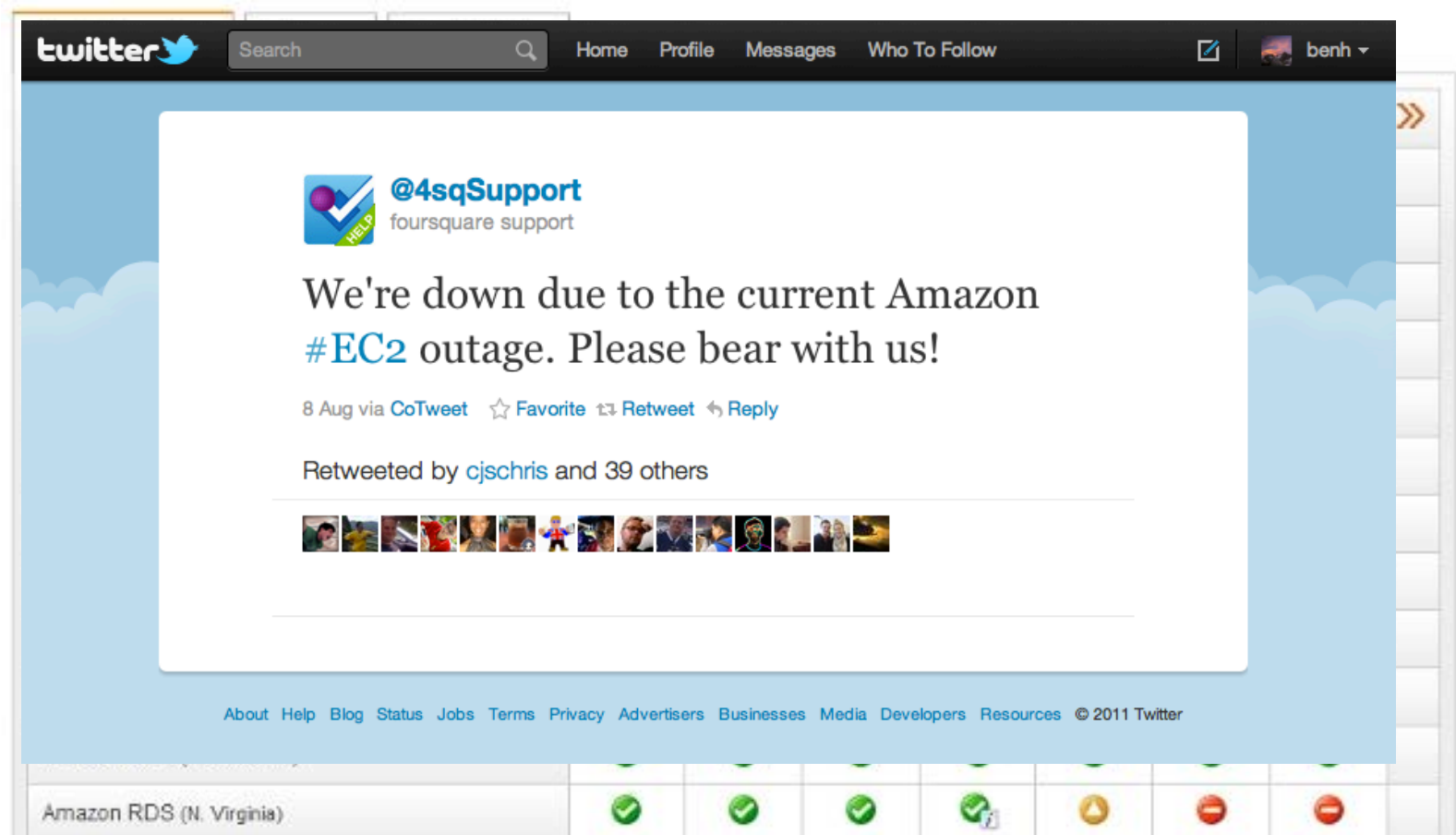
Hi [REDACTED]

Hope things are well with you. I'm not sure if anybody from the RAD Lab has been in touch with you about this, but a big paper deadline is coming up and several projects in the RAD Lab are using EC2 extensively for research experiments and we are hitting our limit. The deadline is Friday and I'm wondering if we can get the limit increased temporarily until Friday. I think our limit may currently be 500 instances, could we get it increased to a 1000 or 2000?

Andy
CS Graduate Student
UC Berkeley

Core Cloud Concepts

- Always available ...



Moving Target

Infrastructure as a Service (virtual machines)

➔ Platforms/Software as a Service

Why?

- Managing lots of machines is still hard
- Programming with failures is still hard

Solution: higher-level frameworks, abstractions

Cloud Environment Challenges

- Cheap nodes fail, especially when you have many
 - Mean time between failures for 1 node = 3 years
 - MTBF for 1000 nodes = 1 day
 - **Solution:** Restrict programming model so you can efficiently “build-in” fault-tolerance (art)
- Commodity network = low bandwidth
 - **Solution:** Push computation to the data

HPC/MPI in the Cloud

- EC2 provides virtual machines, so you can run MPI
- Fault-tolerance:
 - Not standard in most MPI distributions (to the best of my knowledge)
 - Recent restart/checkpointing techniques*, but need the checkpoints to be replicated as well
- Communication?

* <https://ftg.lbl.gov/projects/CheckpointRestart>

HPC/MPI in the Cloud

- LBLN 138pg report on cloud HPC*
- New HPC specific EC2 instance sizes
 - 10 Gbps Ethernet, GPUs

Amazon's cloud is the world's 42nd fastest supercomputer

Amazon used its EC2 service to build one of the world's fastest HPC clusters

by Jon Brodtkin - Nov 15 2011, 7:35am PST

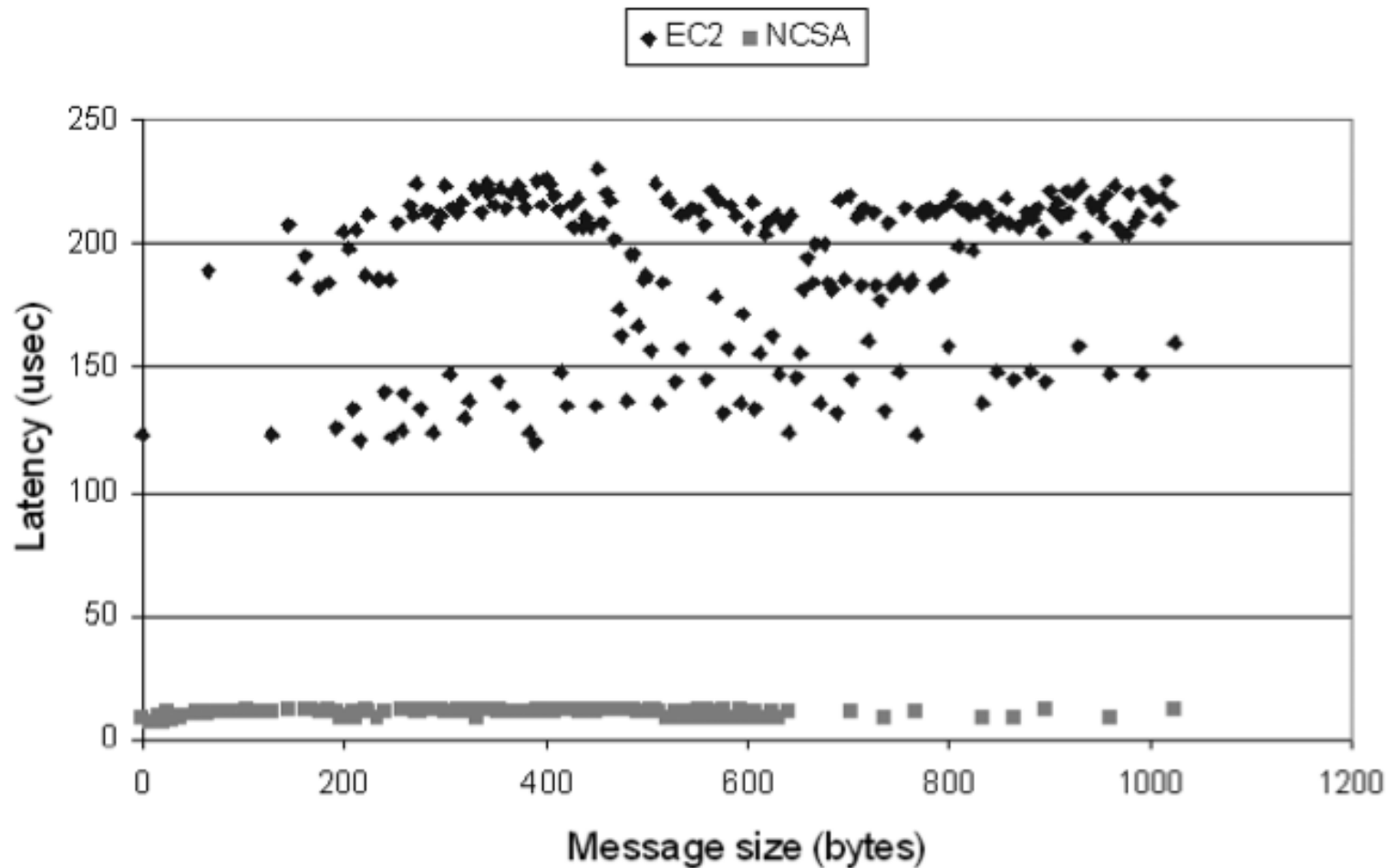
SUPERCOMPUTING

30

The list of the world's 500 fastest supercomputers came out yesterday with a top 10 that was unchanged from the previous ranking issued in June. But further down the list, a familiar name is making a charge: Amazon, with its Elastic Compute Cloud service, built a **17,024-core, 240-teraflop cluster** that now ranks as the 42nd fastest supercomputer in the world.

* tinyurl.com/magellan-report

Latency on EC2 vs Infiniband



Outline

- Cloud Overview
- MapReduce
- MapReduce Examples
- Introduction to Hadoop
- Beyond MapReduce
- Summary

What is MapReduce?

- Data-parallel programming model for clusters of commodity machines
- Pioneered by Google
 - Processes 20 PB of data per day
- Popularized by Apache Hadoop project
 - Used by Yahoo!, Facebook, Amazon, ...

What has MapReduce been used for?

- At Google:
 - Index building for Google Search
 - Article clustering for Google News
 - Statistical machine translation
- At Yahoo!:
 - Index building for Yahoo! Search
 - Spam detection for Yahoo! Mail
- At Facebook:
 - Ad optimization
 - Spam detection

What has MapReduce been used for?

- In research:
 - Analyzing Wikipedia conflicts (PARC)
 - Natural language processing (CMU)
 - Bioinformatics (Maryland)
 - Particle physics (Nebraska)
 - Ocean climate simulation (Washington)
 - <Your application here>

MapReduce Goals

- **Cloud Environment:**
 - Commodity nodes (cheap, but unreliable)
 - Commodity network (low bandwidth)
 - Automatic fault-tolerance (fewer admins)
- **Scalability** to large data volumes:
 - Scan 100 TB on 1 node @ 50 MB/s = 24 days
 - Scan on 1000-node cluster = 35 minutes

MapReduce Programming Model

$$\text{list}\langle T_{\text{in}} \rangle \rightarrow \text{list}\langle T_{\text{out}} \rangle$$

- Data type: key-value *records*

$$\text{list}\langle (K_{\text{in}}, V_{\text{in}}) \rangle \rightarrow \text{list}\langle (K_{\text{out}}, V_{\text{out}}) \rangle$$

MapReduce Programming Model

Map function:

$$(K_{in}, V_{in}) \rightarrow \text{list}\langle(K_{inter}, V_{inter})\rangle$$

Reduce function:

$$(K_{inter}, \text{list}\langle V_{inter} \rangle) \rightarrow \text{list}\langle(K_{out}, V_{out})\rangle$$

Example: Word Count

```
def map(line_num, line):  
    foreach word in line.split():  
        output(word, 1)
```

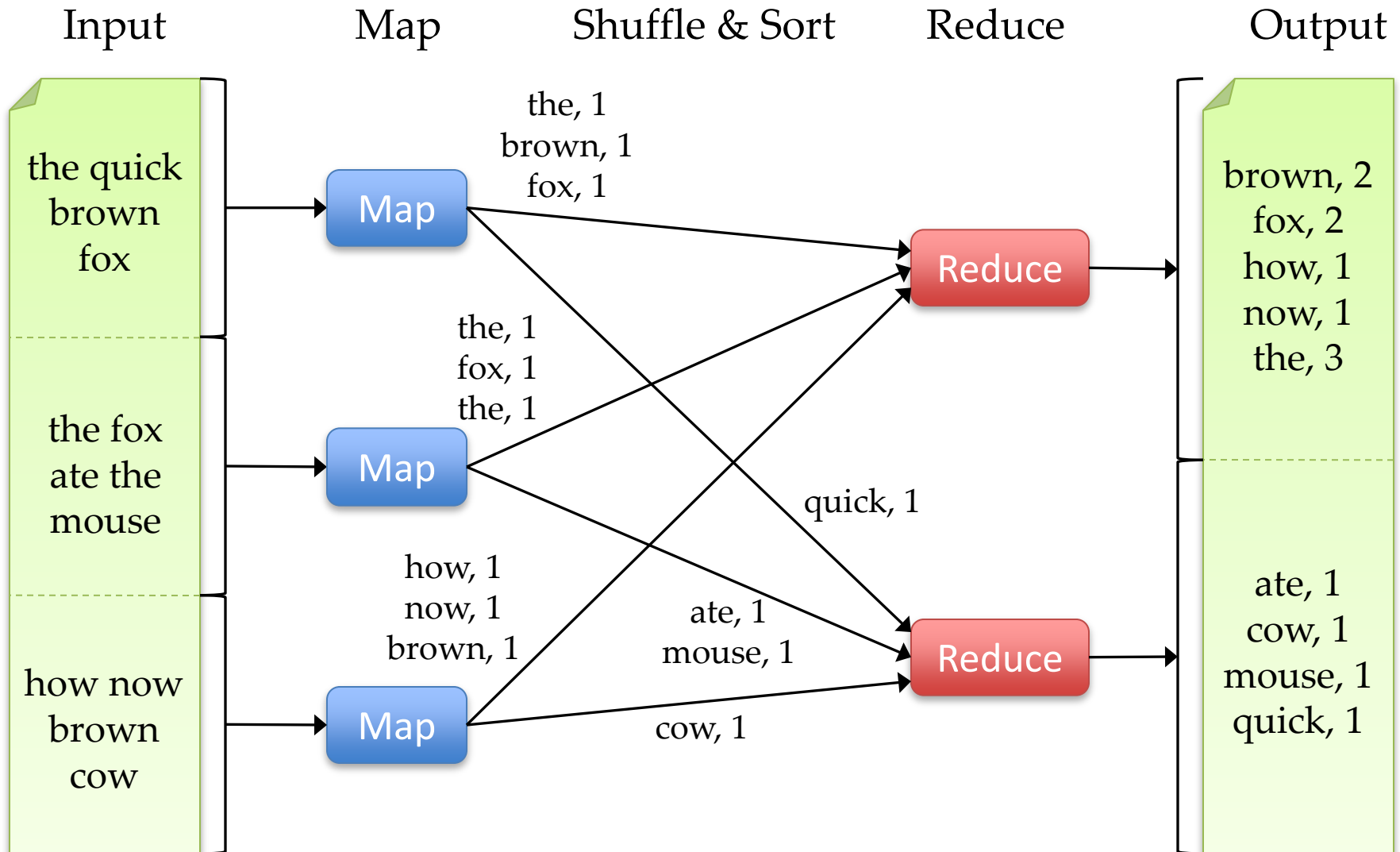
```
def reduce(word, counts):  
    output(word, sum(counts))
```

Example: Word Count

```
def map(line_num, line):  
    foreach word in line.split():  
        output(word, 1)
```

```
def reduce(word, counts):  
    output(word, counts.size())
```

Example: Word Count

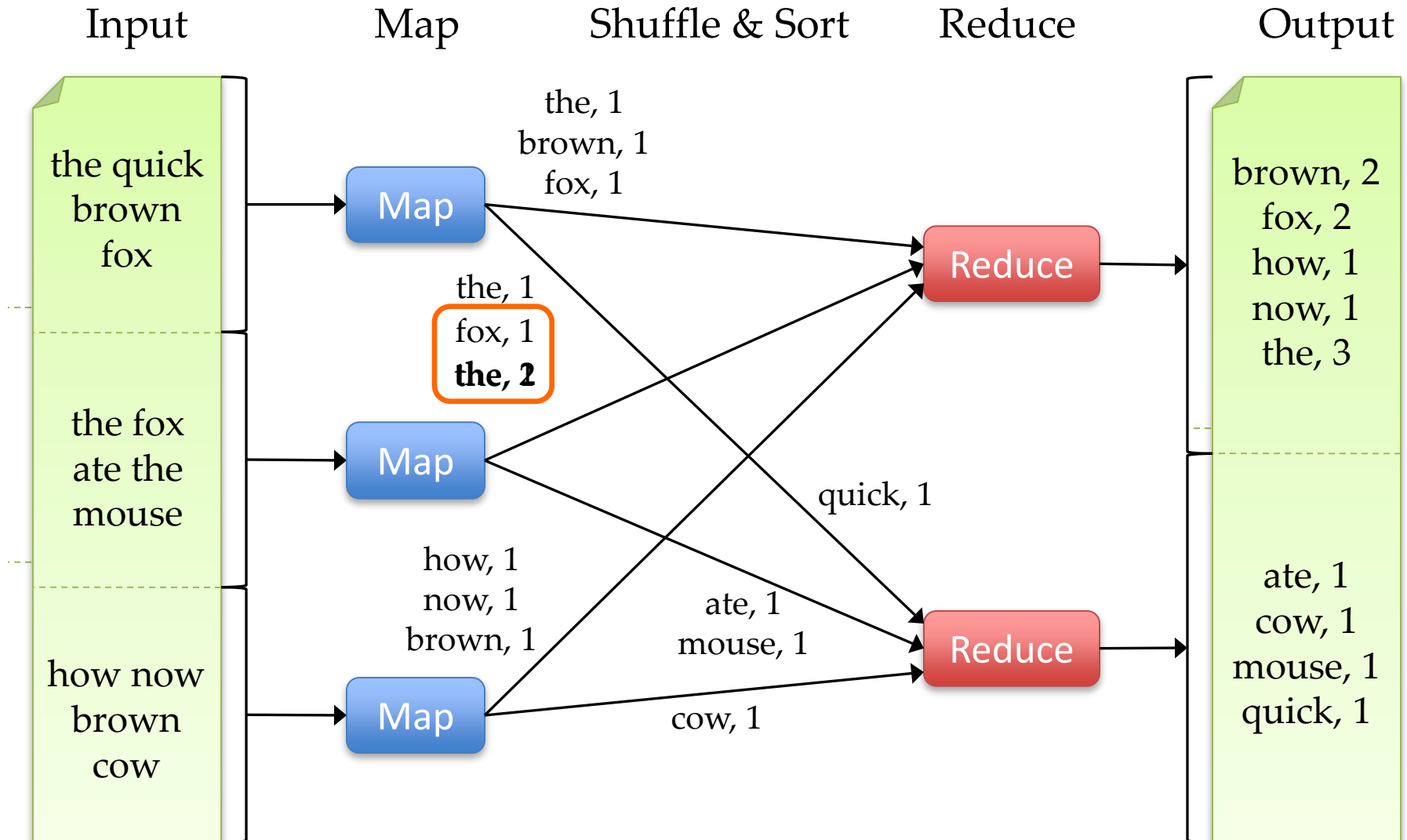


Optimization: Combiner

- Local “reduce” function for repeated keys produced by same map
- For associative ops. like sum, count, max
- Decreases amount of intermediate data
- Example:

```
def combine(key, values):  
    output(key, sum(values))
```

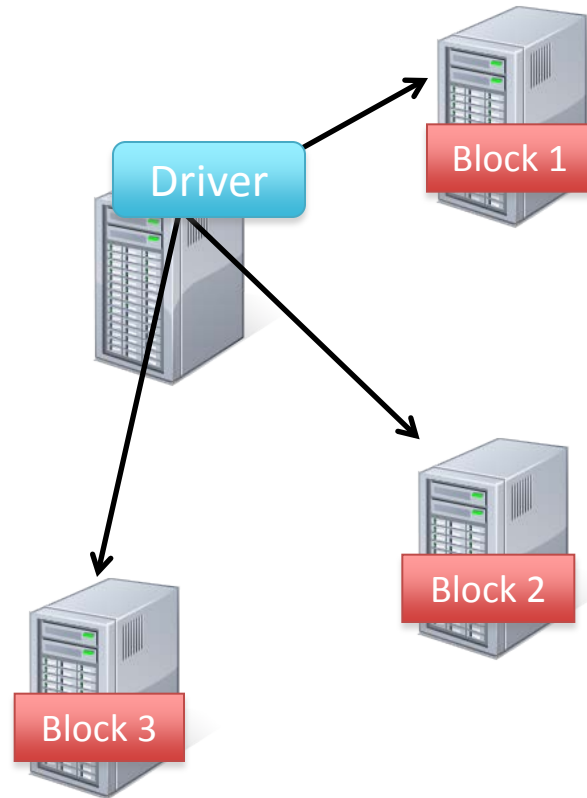

Example: Word Count + Combiner



MapReduce Execution Details

- Data stored on compute nodes
- Mappers preferentially scheduled on same node or same rack as their input block
 - Minimize network use to improve performance
- Mappers save outputs to local disk before serving to reducers
 - Efficient recovery when a reducer crashes
 - Allows more flexible mapping to reducers

MapReduce Execution Details



Fault Tolerance in MapReduce

1. If a task crashes:

- Retry on another node
 - OK for a map because it had no dependencies
 - OK for reduce because map outputs are on disk
- If the same task repeatedly fails, fail the job or ignore that input block

➤ Note: For the fault tolerance to work, *user tasks must be idempotent and side-effect-free*

Fault Tolerance in MapReduce

2. If a node crashes:

- Relaunch its current tasks on other nodes
- Relaunch any maps the node previously ran
 - Necessary because their output files were lost along with the crashed node

Fault Tolerance in MapReduce

3. If a task is going slowly (straggler):
 - Launch second copy of task on another node
 - Take the output of whichever copy finishes first, and kill the other one
- Critical for performance in large clusters (many possible causes of stragglers)

Takeaways

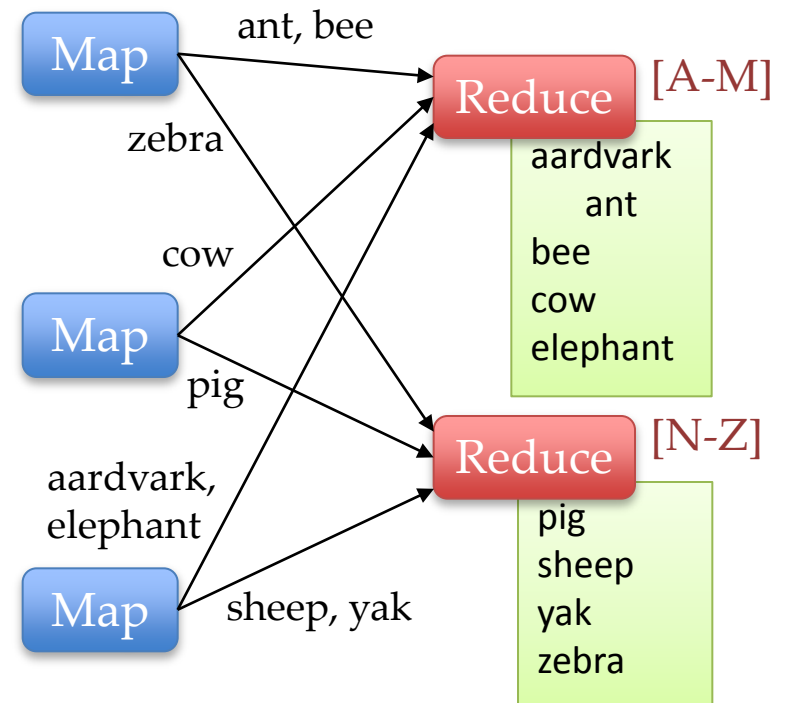
- By providing a restricted programming model, MapReduce can control job execution in useful ways:
 - Parallelization into tasks
 - Placement of computation near data
 - Load balancing
 - Recovery from failures & stragglers

Outline

- Cloud Overview
- MapReduce
- MapReduce Examples
- Introduction to Hadoop
- Beyond MapReduce
- Summary

1. Sort

- **Input:** (key, value) records
- **Output:** same records, sorted by key
- **Map:** identity function
- **Reduce:** identify function
- **Trick:** Pick partitioning function p such that $k_1 < k_2 \Rightarrow p(k_1) < p(k_2)$



2. Search

- **Input:** (filename, line) records
- **Output:** lines matching a given pattern
- **Map:**

```
if (line matches pattern):  
    output(filename, line)
```
- **Reduce:** identity function
 - Alternative: no reducer (map-only job)

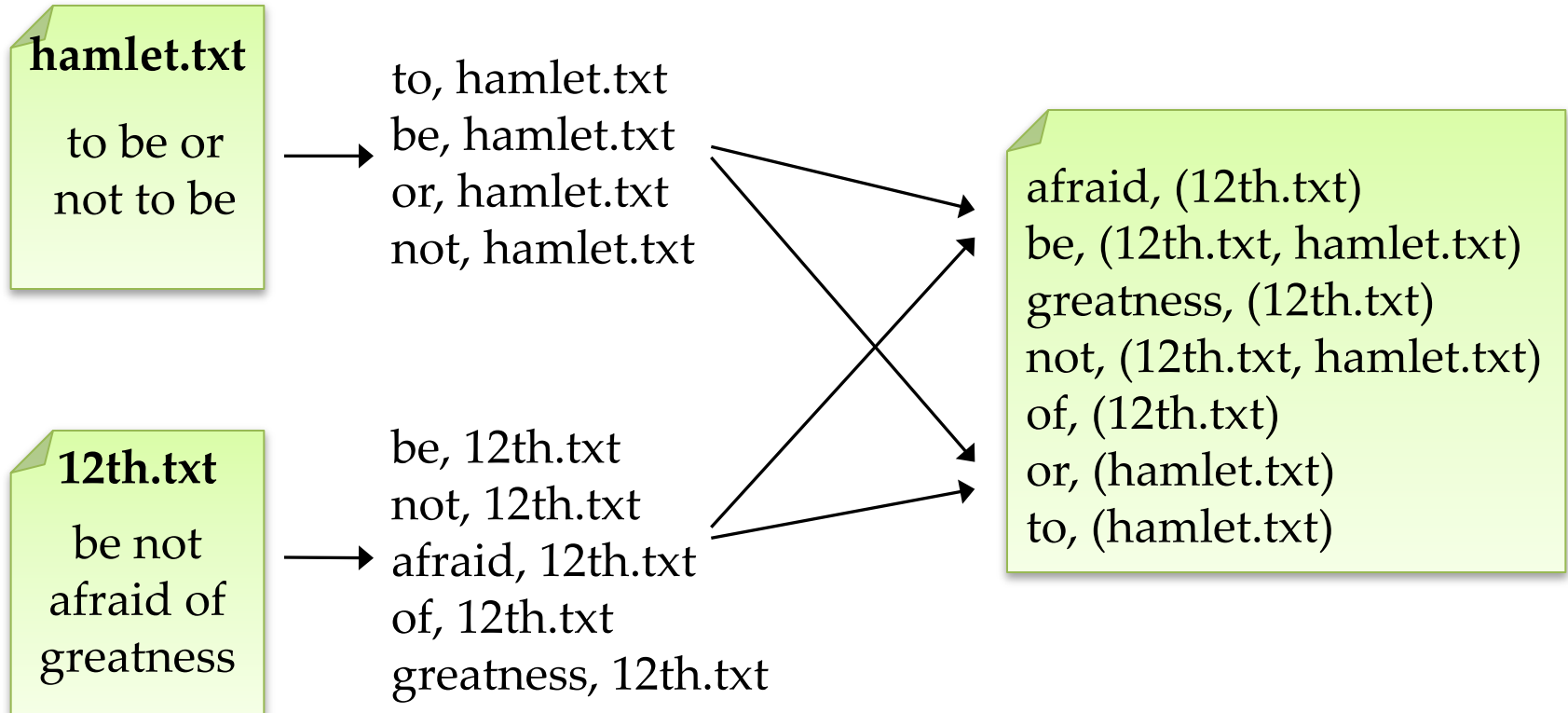
3. Inverted Index

- **Input:** (filename, text) records
- **Output:** list of files containing each word
- **Map:**

```
foreach word in text.split():  
    output(word, filename)
```
- **Combine:** remove duplicates
- **Reduce:**

```
def reduce(word, filenames):  
    output(word, sort(filenames))
```

Inverted Index Example



4. Most Popular Words

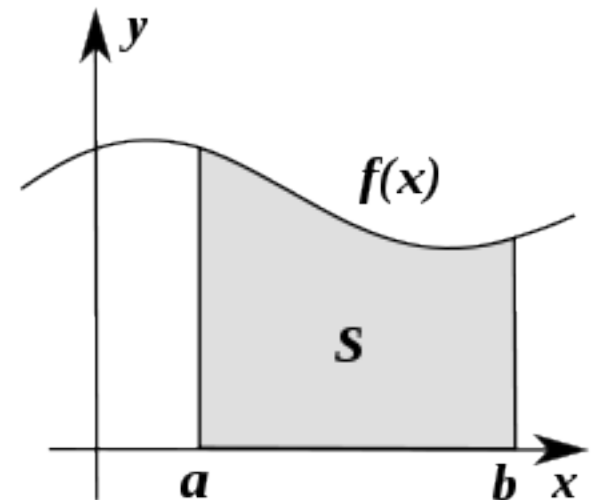
- **Input:** (filename, text) records
- **Output:** the 100 words occurring in most files
- Two-stage solution:
 - **Job 1:**
 - Create inverted index, giving (word, list(file)) records
 - **Job 2:**
 - Map each (word, list(file)) to (count, word)
 - Sort these records by count as in sort job
- Optimizations:
 - Map to (word, 1) instead of (word, file) in Job 1

5. Numerical Integration

- **Input:** (start, end) records for sub-ranges to integrate*
- **Output:** integral of $f(x)$ over entire range
- **Map:**

```
def map(start, end):  
    sum = 0  
    for(x = start; x < end; x += step):  
        sum += f(x) * step  
    output("", sum)
```

- **Reduce:**
 def reduce(key, values):
 output(key, sum(values))



*Can implement using custom InputFormat

Outline

- Cloud Overview
- MapReduce
- MapReduce Examples
- Introduction to Hadoop
- Beyond MapReduce
- Summary

Hadoop Components

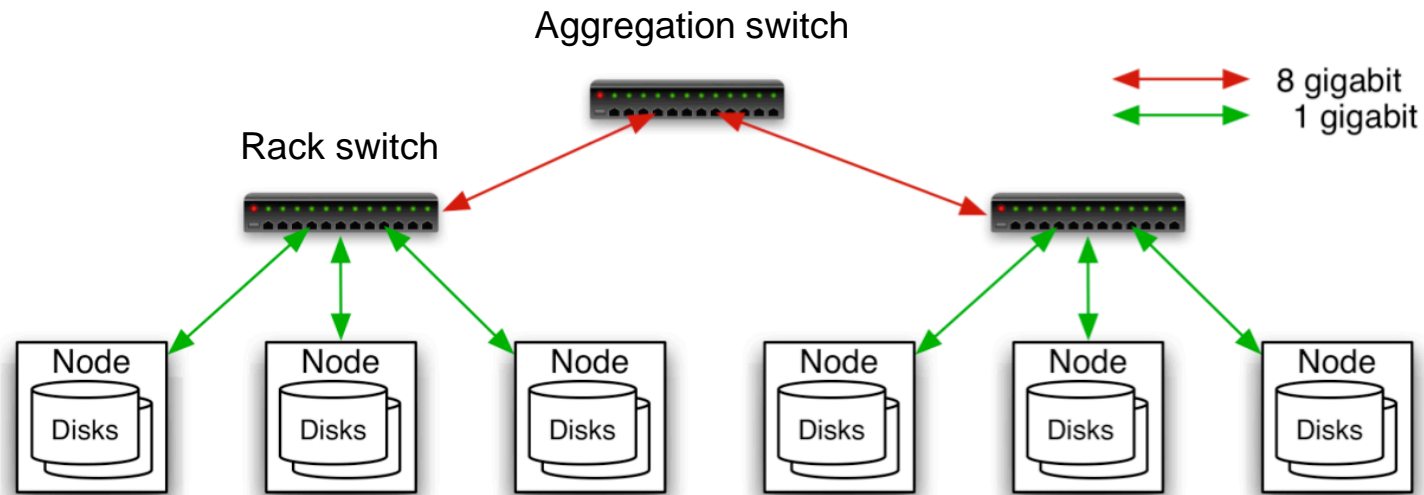
- MapReduce
 - Runs jobs submitted by users
 - Manages work distribution & fault-tolerance
- Distributed File System (HDFS)
 - Runs on same machines!
 - Replicates data 3x for fault-tolerance



Typical Hadoop Cluster



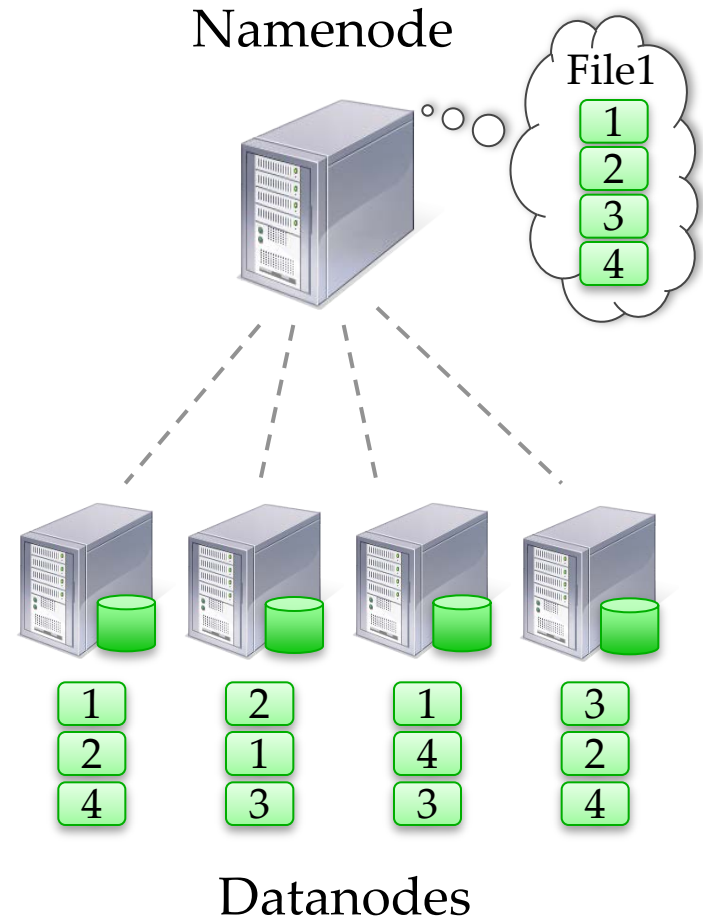
Typical Hadoop cluster



- 40 nodes/rack, 1000-4000 nodes in cluster
- 1 Gbps bandwidth in rack, 8 Gbps out of rack
- Node specs at Facebook:
8-16 cores, 32 GB RAM, 8×1.5 TB disks, no RAID

Distributed File System

- Files split into 128MB blocks
- Blocks replicated across several datanodes (often 3)
- Namenode stores metadata (file names, locations, etc)
- Optimized for large files, sequential reads
- Files are append-only



Hadoop

- Download from hadoop.apache.org
- To install locally, unzip and set JAVA_HOME
- Docs: hadoop.apache.org/common/docs/current
- Three ways to write jobs:
 - Java API
 - Hadoop Streaming (for Python, Perl, etc)
 - Pipes API (C++)

Word Count in Java

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

    private final static IntWritable ONE = new IntWritable(1);

    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()) {
            output.collect(new Text(itr.nextToken()), ONE);
        }
    }
}
```

Word Count in Java

```
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));
    }
}
```

Word Count in Java

```
public static void main(String[] args) throws Exception {  
    JobConf conf = new JobConf(WordCount.class);  
    conf.setJobName("wordcount");  
  
    conf.setMapperClass(MapClass.class);  
    conf.setCombinerClass(Reduce.class);  
    conf.setReducerClass(Reduce.class);  
  
    FileInputFormat.setInputPaths(conf, args[0]);  
    FileOutputFormat.setOutputPath(conf, new Path(args[1]));  
  
    conf.setOutputKeyClass(Text.class); // out keys are words (strings)  
    conf.setOutputValueClass(IntWritable.class); // values are counts  
  
    JobClient.runJob(conf);  
}
```

Word Count in Python with Hadoop Streaming

Mapper.py:

```
import sys
for line in sys.stdin:
    for word in line.split():
        print(word.lower() + "\t" + 1)
```

Reducer.py:


```
import sys
counts = {}
for line in sys.stdin:
    word, count = line.split("\t")
    dict[word] = dict.get(word, 0) + int(count)
for word, count in counts:
    print(word.lower() + "\t" + 1)
```


Amazon Elastic MapReduce

- Simplifies configuring, deploying Hadoop
- Web interface, command-line tools for Hadoop jobs on EC2
- Data in Amazon S3
- Monitors job, shuts down machines when finished

Elastic MapReduce UI

Create a New Job Flow

Cancel 



DEFINE JOB FLOW

SPECIFY PARAMETERS

CONFIGURE EC2 INSTANCES

REVIEW

Creating a job flow to process your data using Amazon Elastic MapReduce is simple and quick. Let's begin by giving your job flow a name and selecting its type. If you don't already have an application you'd like to run on Amazon Elastic MapReduce, samples are available to help you get started.

Job Flow Name*:

The name can be anything you like and doesn't need to be unique. It's a good idea to name the job flow something descriptive.

Type*: ☒ Streaming

A Streaming job flow allows you to write single-step mapper and reducer functions in a language other than java.

☐ Custom Jar (advanced)

A custom jar on the other hand gives you more complete control over the function of Hadoop but must be a compiled java program. Amazon Elastic MapReduce supports custom jars developed for Hadoop 0.18.3.

☐ Pig Program

Pig is a SQL-like language built on top of Hadoop. This option allows you to define a job flow that runs a Pig script, or set up a job flow that can be used interactively via SSH to run Pig commands.

☐ Sample Applications

Select a sample application and click Continue. Subsequent forms will be filled with the necessary data to create a sample Job Flow.

Word Count (Streaming)



Word count is a Python application that counts occurrences of each word in provided documents. [Learn more and view license](#)

Continue



* Required field

Elastic MapReduce UI

Create a New Job Flow

Cancel 

✓
DEFINE JOB FLOW

✓
SPECIFY PARAMETERS

○
CONFIGURE EC2 INSTANCES

REVIEW

Enter the number and type of EC2 instances you'd like to run your job flow on.

Number of Instances*:

The number of EC2 instances to run in your Hadoop cluster.
If you wish to run more than 20 instances, please complete the [limit request form](#).

Type of Instance*:

The type of EC2 instances to run in your Hadoop cluster ([learn more about instance types](#)).

⌵ [Show advanced options](#)

[< Back](#)

Continue 

* Required field

Elastic MapReduce UI



[Contact Us](#) | [Create an AWS Account](#)

[About AWS](#)

[Products](#)

[Solutions](#)

[Resources](#)

[Support](#)

[Your Account](#)

[Home](#) > [Resources](#) > [AWS Management Console](#) [BETA](#) > [Amazon Elastic MapReduce](#)

Welcome, Rad Lab | [Settings](#) | [Sign Out](#)

Amazon EC2

**Amazon Elastic
MapReduce**

Amazon
CloudFront

Your Elastic MapReduce Job Flows

Region: US-East



Create New Job Flow



Terminate



Show/Hide



Refresh



Help

Viewing:

All



1 to 1 of 1 Job Flows



	Name	State	Creation Date	Elapsed Time	Normalized Instance Hours
	My Job Flow	STARTING	2009-08-19 14:50 PDT	0 hours 0 minutes	0

1 Job Flow selected

	Id:	j-46JL0YQ7ZPH1	Creation Date:	2009-08-19 14:50 PDT
	Name:	My Job Flow	Start Date:	-
	State:	STARTING	End Date:	-
	Last State Change Reason:	Starting instances		
	Availability Zone:	us-east-1b	Instance Count:	4

Outline

- Cloud Overview
- MapReduce
- MapReduce Examples
- Introduction to Hadoop
- Beyond MapReduce
- Summary

Beyond MapReduce

- Other distributed programming models for distributed computing
 - **Dryad** (Microsoft): general DAG of tasks
 - **Pregel** (Google): bulk synchronous processing
 - **Percolator** (Google): incremental computation
 - **S4** (Yahoo!): streaming computation
 - **Piccolo** (NYU): shared in-memory state
 - **DryadLINQ** (Microsoft): language integration
 - **Spark** (Berkeley): ...

What is Spark?

- *Fast*, MapReduce-like engine
 - In-memory data storage for very fast iterative queries
 - General execution graphs and rich optimizations
 - 40x faster than Hadoop in real apps
- Compatible with Hadoop's storage APIs
 - Can read/write to any Hadoop-supported system, including HDFS, HBase, SequenceFiles, etc

What is Shark?

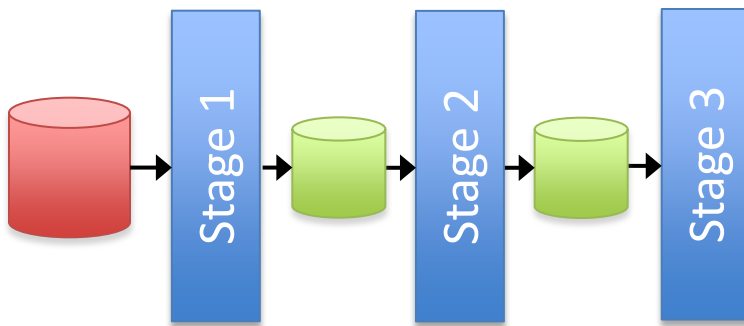
- Port of Apache Hive to run on Spark
- Compatible with existing Hive data, metastores, and queries (HiveQL, UDFs, etc)
- Similar speedups of up to 40x

Why go Beyond MapReduce?

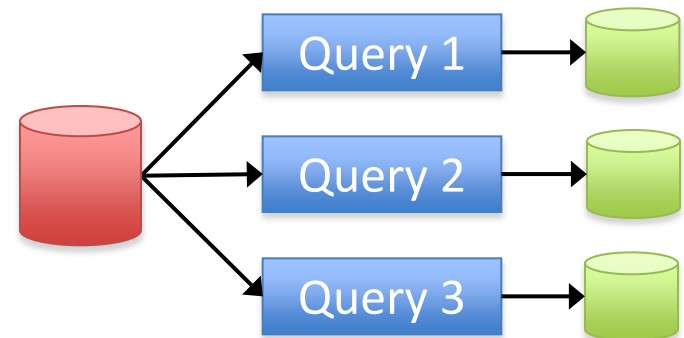
- MapReduce greatly simplified big data analysis
- But as soon as it got popular, users wanted more:
 - More **complex**, multi-stage applications (graph algorithms, machine learning)
 - More **interactive** ad-hoc queries
 - More **real-time** online processing

Why go Beyond MapReduce?

- Complex jobs, streaming, and interactive queries all need one thing that MapReduce lacks:
 - Efficient primitives for **data sharing**



Iterative algorithm



Interactive data mining

Why go Beyond MapReduce?

- Complex jobs, streaming, and interactive queries all need one thing that MapReduce lacks:
 - Efficient primitives for **data sharing**



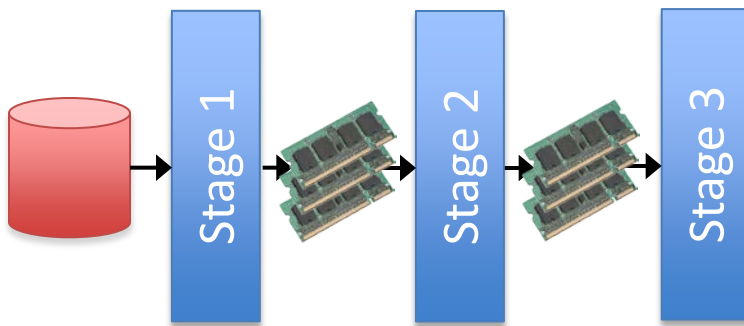
In MapReduce, the only way to share data across jobs is stable storage (e.g. HDFS) -> **slow!**

Iterative algorithm

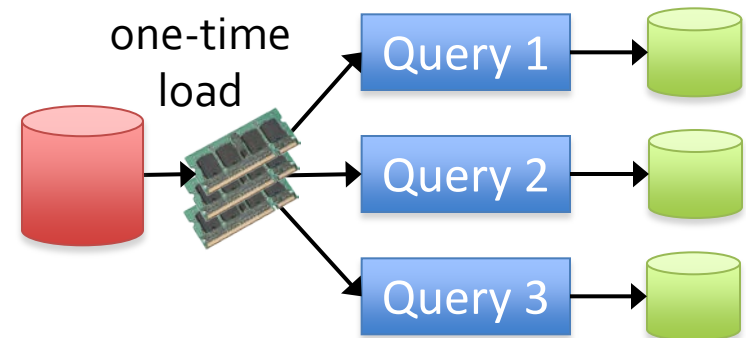
Interactive data mining

How Spark Solves This

- Let applications share data **in memory** through “resilient distributed datasets” (RDDs)
- Support **general graphs** of operators in a query



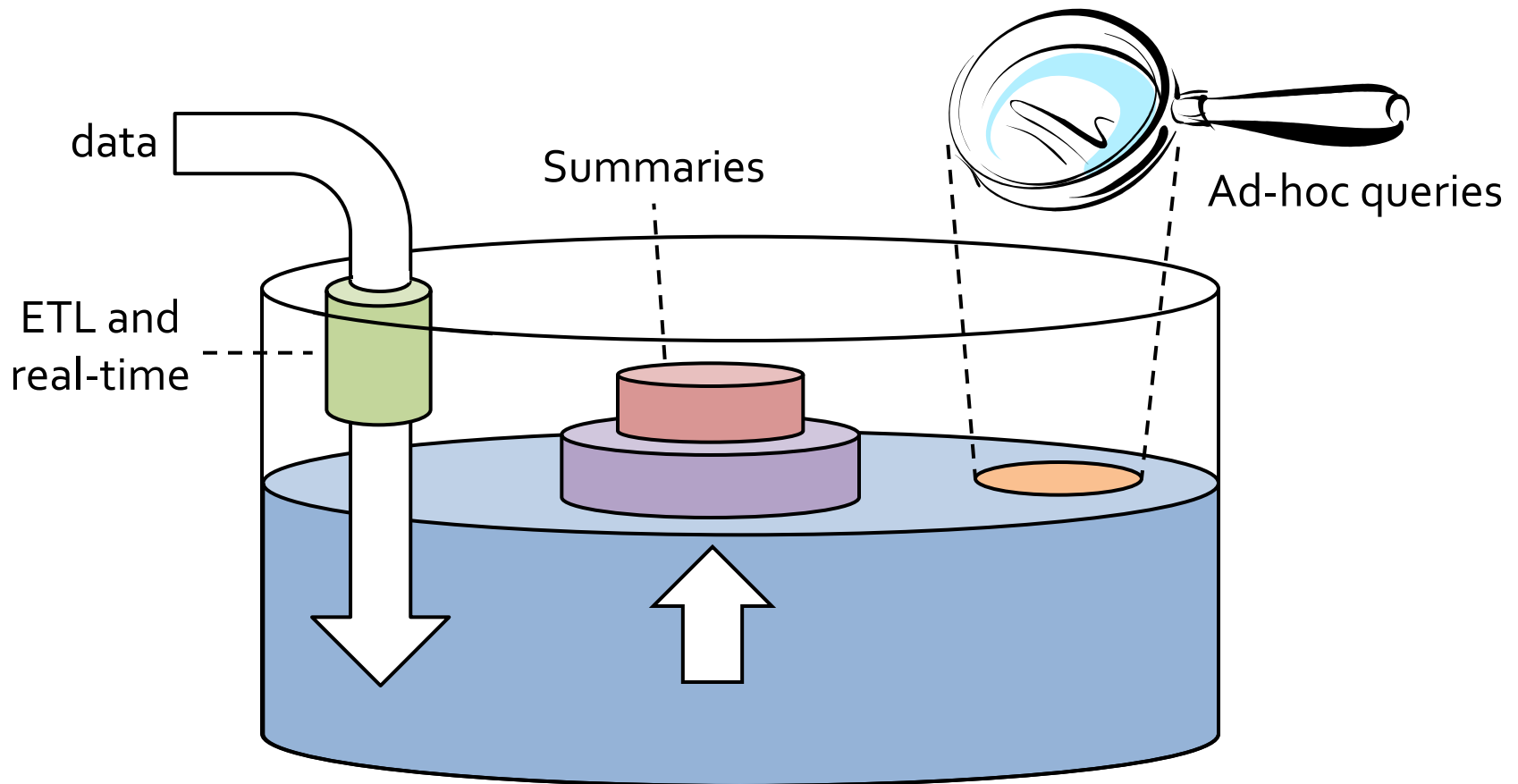
Iterative algorithm



Interactive data mining

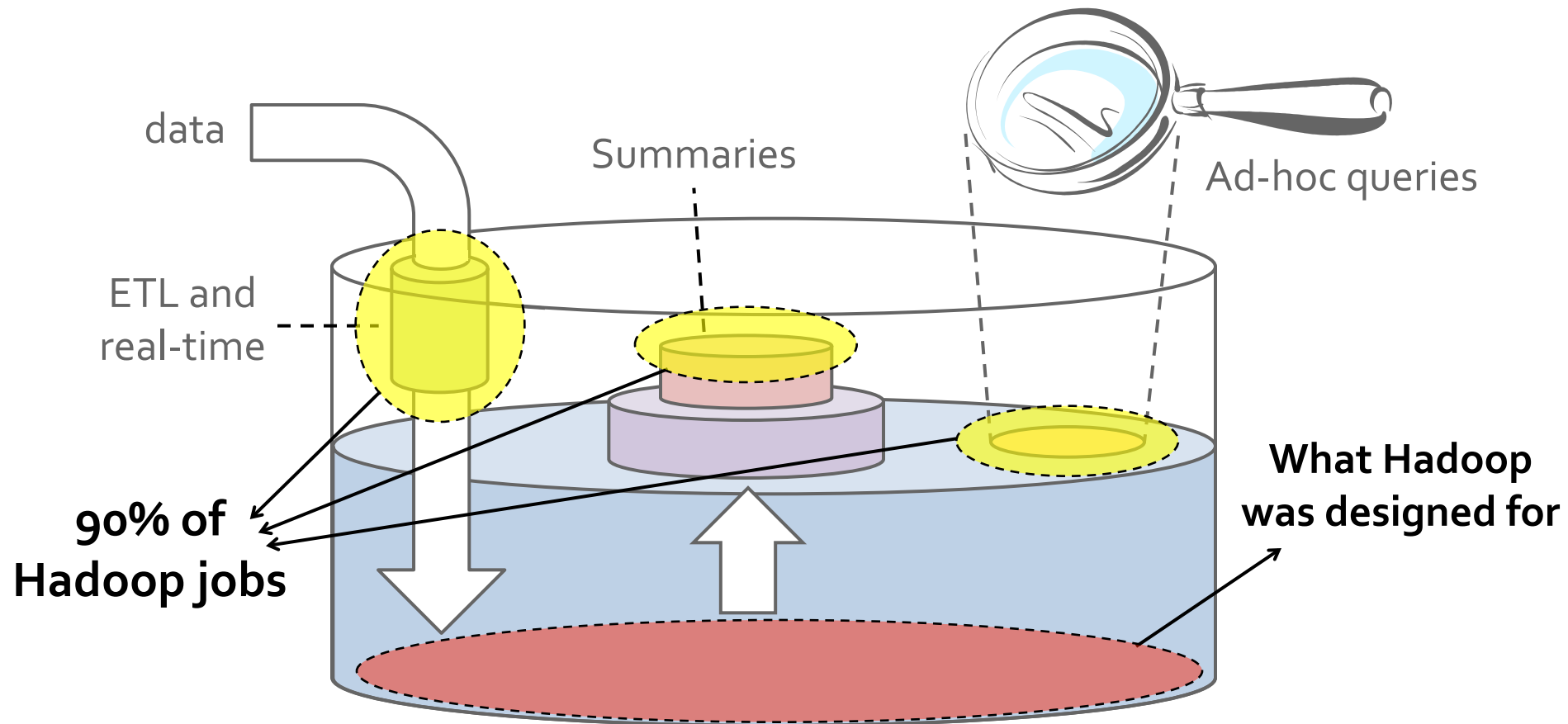
Why Sharing is Fundamental

- “Funnels” view of data lifecycle:



Why Sharing is Fundamental

- “Funnels” view of data lifecycle:



Spark Programming Interface

- Clean language-integrated API in Scala
- Usable *interactively* from Scala interpreter
- Java and SQL also in the works

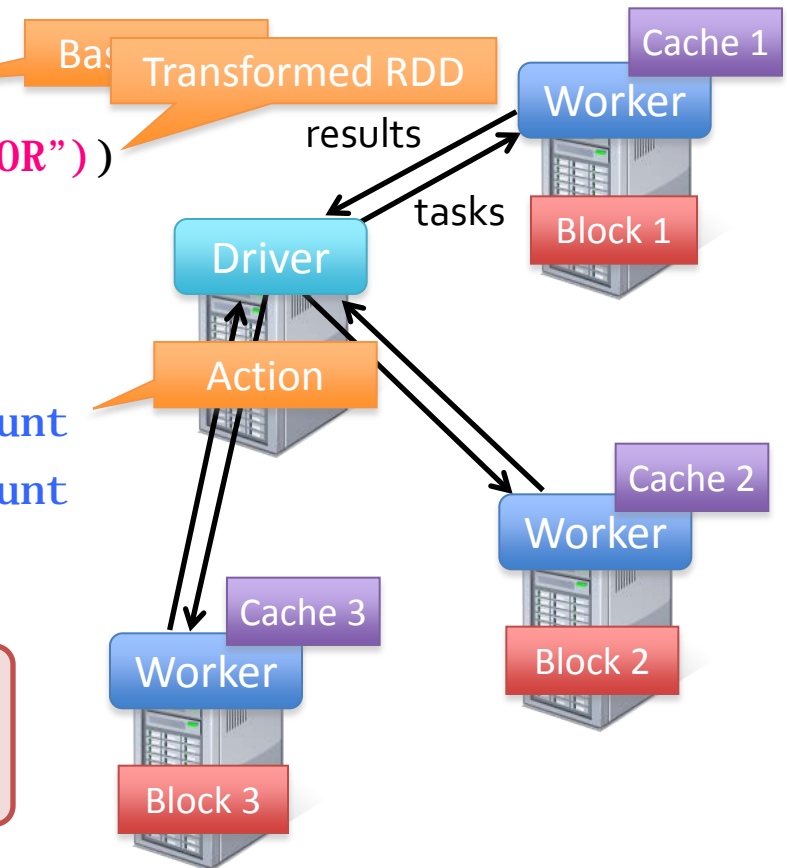
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('\t')(2))
cachedMsgs = messages.cache()

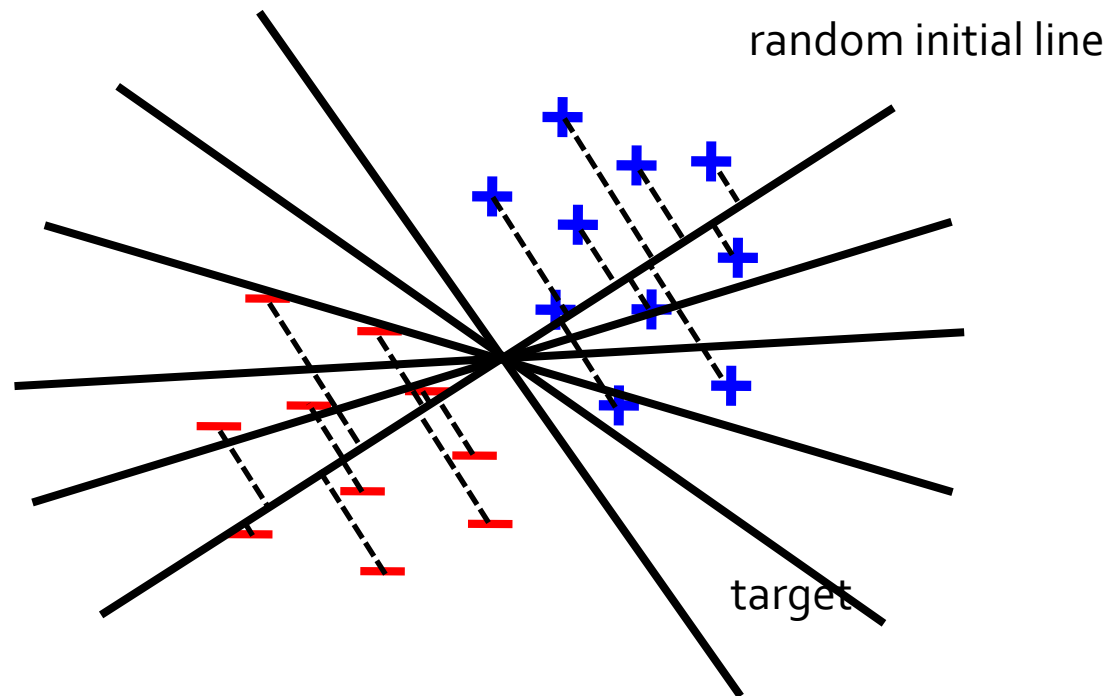
cachedMsgs.filter(_.contains("foo")).count
cachedMsgs.filter(_.contains("bar")).count
. . .
```

Result: scaled to 1 TB data in 5-7 sec
(vs 170 sec for on-disk data)

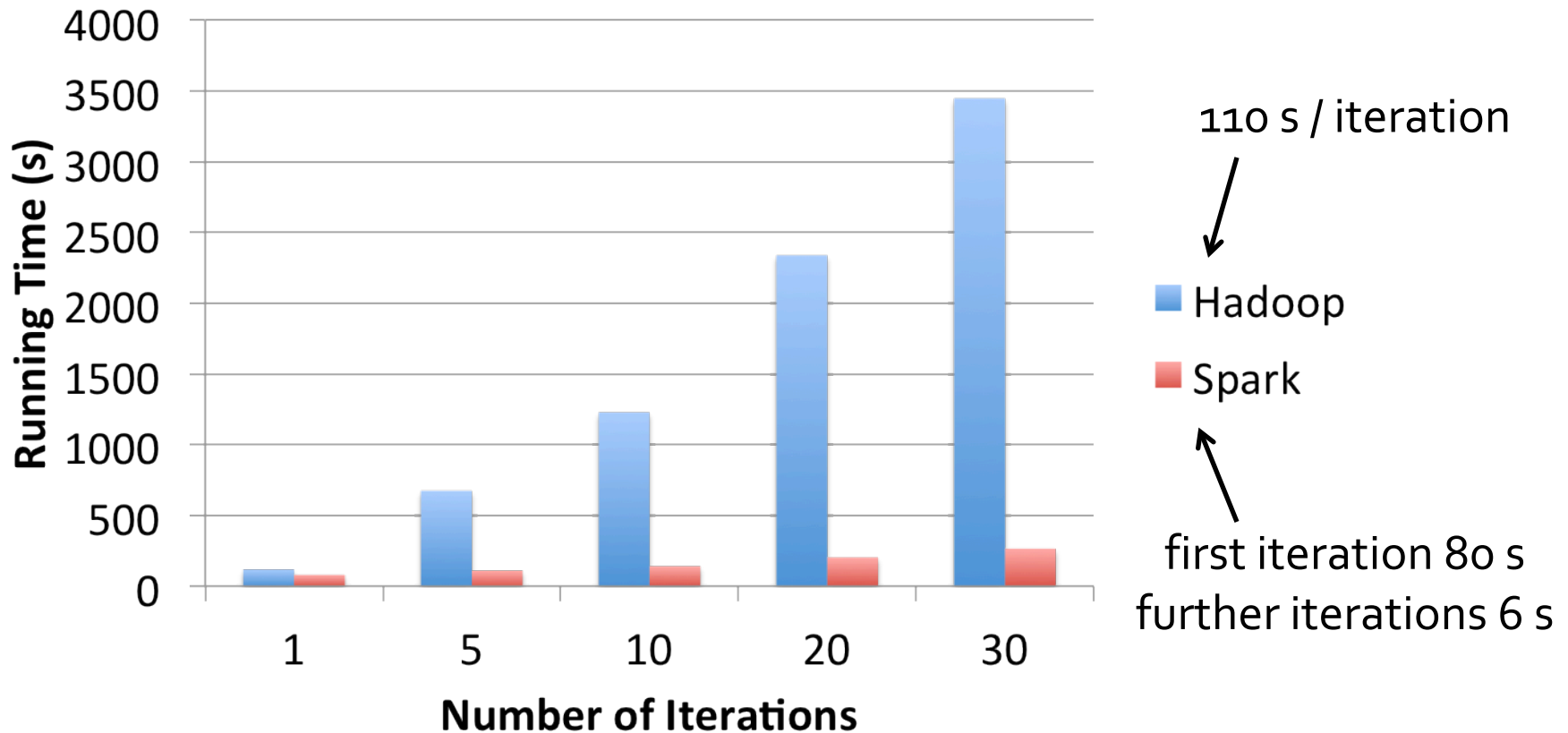


Example: Logistic Regression

Goal: find best line separating two sets of points

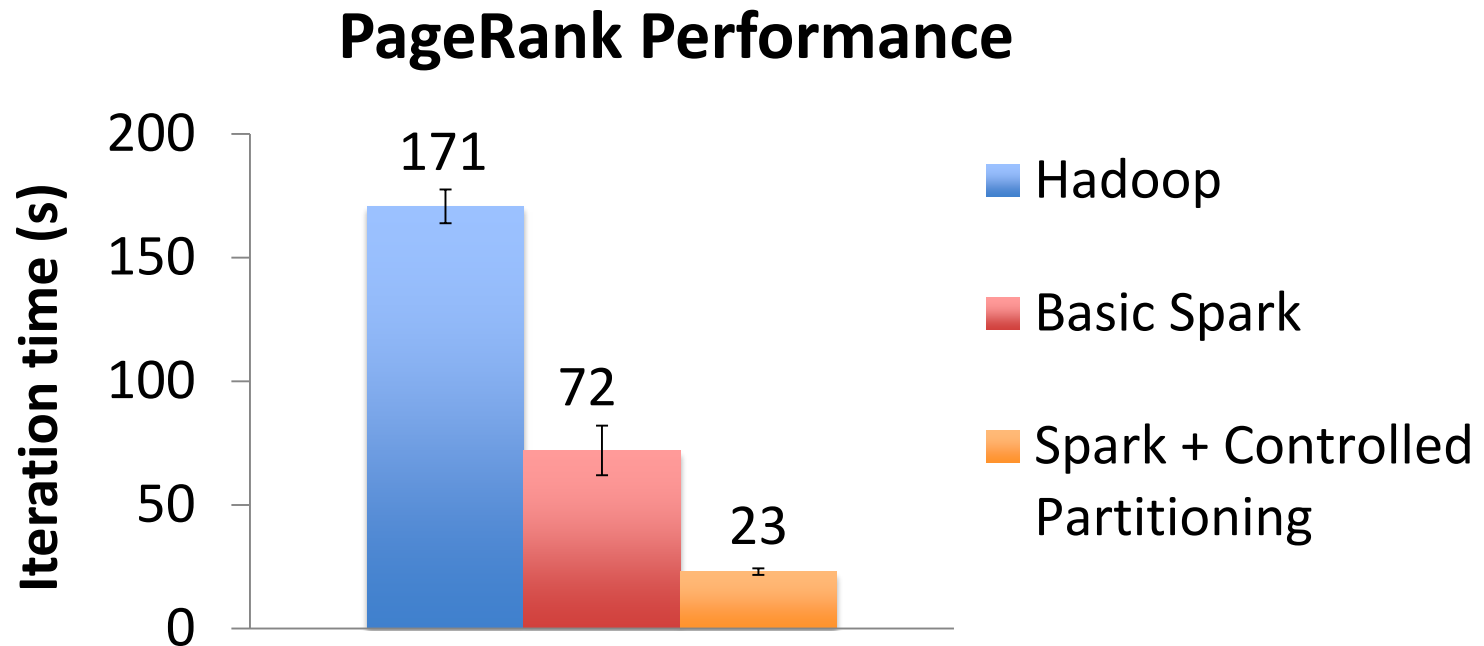


Logistic Regression Performance



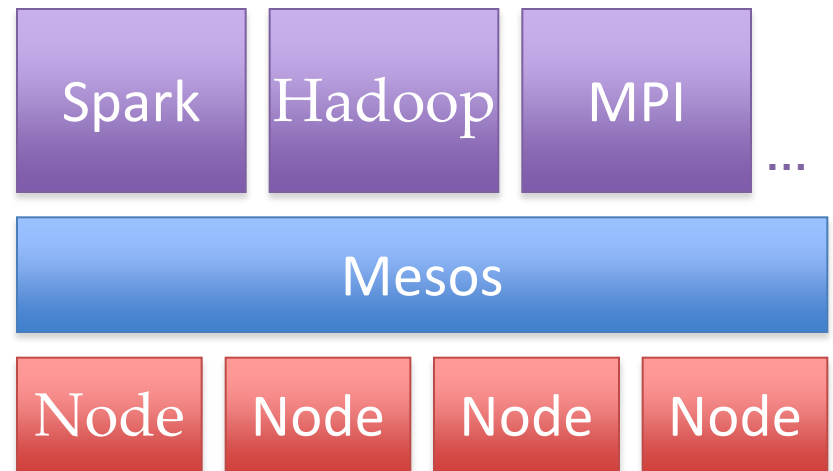
Other Engine Features

- Controllable data partitioning to minimize communication



Beyond Spark

- Write your own framework using Mesos, letting it efficiently share resources and data with Spark, Hadoop & others



www.mesos-project.org

Outline

- Cloud Overview
- MapReduce
- MapReduce Examples
- Introduction to Hadoop
- Beyond MapReduce
- Summary

Summary

- MapReduce's data-parallel programming model hides complexity of distribution and fault tolerance
- Principal philosophies:
 - *Make it scale*, so you can throw hardware at problems
 - *Make it cheap*, saving hardware, programmer and administration costs (but necessitating fault tolerance)
- MapReduce is not suitable for all problems, new programming models and frameworks still being created

Resources

- Hadoop: <http://hadoop.apache.org/common>
- Video tutorials: www.cloudera.com/hadoop-training
- Amazon Elastic MapReduce: <http://docs.amazonwebservices.com/ElasticMapReduce/latest/GettingStartedGuide/>
- Spark: <http://spark-project.org>
- Mesos: <http://mesos-project.org>

Thanks!

HPC Cloud Projects

- Magellan (DOE, Argonne, LBNL)
 - 720 nodes, 5760 cores, InfiniBand network
 - Goals: explore suitability of cloud model, APIs and hardware to scientific computing, and implications on security and cost
- SGI HPC Cloud (“Cyclone”)
 - Commercial on-demand HPC offering
 - Includes CPU and GPU nodes
 - Includes “software as a service” for select domains
- Probably others as well

Outline

- MapReduce architecture
- Sample applications
- Introduction to Hadoop
- Higher-level query languages: Pig & Hive
- Cloud programming research
- Clouds and HPC

Motivation

- MapReduce is powerful: many algorithms can be expressed as a series of MR jobs
- But it's fairly low-level: must think about keys, values, partitioning, etc.
- Can we capture common “job patterns”?

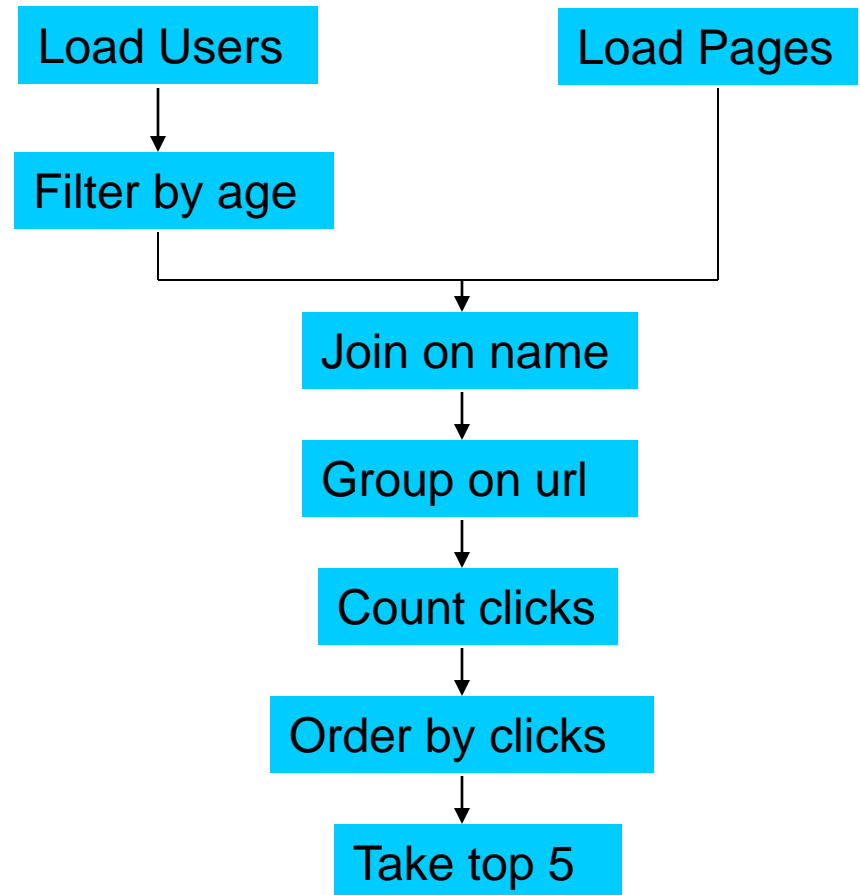
Pig

- Started at Yahoo! Research
- Runs about 50% of Yahoo!'s jobs
- Features:
 - Expresses sequences of MapReduce jobs
 - Data model: nested “bags” of items
 - Provides relational (SQL) operators (JOIN, GROUP BY, etc)
 - Easy to plug in Java functions



An Example Problem

Suppose you have user data in one file, website data in another, and you need to find the top 5 most visited pages by users aged 18-25.



In MapReduce

```

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

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.Writable;
import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.mapred.FileInputFormat;
import org.apache.hadoop.mapred.FileOutputFormat;
import org.apache.hadoop.mapred.JobConf;
import org.apache.hadoop.mapred.KeyValueTextInputFormat;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.SequenceBase;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.RecordReader;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.Reporter;
import org.apache.hadoop.mapred.SequenceFileInputFormat;
import org.apache.hadoop.mapred.SequenceFileOutputFormat;
import org.apache.hadoop.mapred.TextInputFormat;
import org.apache.hadoop.mapred.JobControl;
import org.apache.hadoop.mapred.JobControl.JobCntrl;
import org.apache.hadoop.mapred.lib.IdentityMapper;

public class MRExample {
    public static class LoadPages extends MapReduceBase
        implements Mapper<LongWritable, Text, Text, Text> {

        public void map(LongWritable k, Text val,
            OutputCollector<Text, Text> oc,
            Reporter reporter) throws IOException {
            // Pull the key out
            String line = val.toString();
            int firstComma = line.indexOf(',');
            String key = line.substring(0, firstComma);
            String value = line.substring(firstComma + 1);
            Text outKey = new Text(key);
            // Prepend an index to the value so we know which file
            // it came from.
            Text outVal = new Text("1" + value);
            oc.collect(outKey, outVal);
        }
    }

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

        public void map(LongWritable k, Text val,
            OutputCollector<Text, Text> oc,
            Reporter reporter) throws IOException {
            // Pull the key out
            String line = val.toString();
            int firstComma = line.indexOf(',');
            String value = line.substring(firstComma + 1);
            int age = Integer.parseInt(value);
            if (age < 18 || age > 25) return;
            String key = line.substring(0, firstComma);
            Text outKey = new Text(key);
            // Prepend an index to the value so we know which file
            // it came from.
            Text outVal = new Text("2" + value);
            oc.collect(outKey, outVal);
        }
    }

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

        public void reduce(Text key,
            Iterator<Text> iter,
            OutputCollector<Text, Text> oc,
            Reporter reporter) throws IOException {
            // For each value, figure out which file it's from and
            // accordingly.
            List<String> first = new ArrayList<String>();
            List<String> second = new ArrayList<String>();

            while (iter.hasNext()) {
                Text t = iter.next();
                String value = t.toString();
                if (value.charAt(0) == '1')
                    first.add(value.substring(1));
                else second.add(value.substring(1));
            }
        }
    }
}

```

```

        reporter.setStatus("OK");
    }

    // Do the cross product and collect the values
    for (String s1 : first) {
        for (String s2 : second) {
            String outval = key + "," + s1 + "," + s2;
            oc.collect(null, new Text(outval));
            reporter.setStatus("OK");
        }
    }
}

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

    public void map(
        Text k,
        Text val,
        OutputCollector<Text, LongWritable> oc,
        Reporter reporter) throws IOException {
        // Find the url
        String line = val.toString();
        int firstComma = line.indexOf(',');
        int secondComma = line.indexOf(',', firstComma);
        String key = line.substring(firstComma, secondComma);
        // drop the rest of the record, I don't need it anymore,
        // just pass a 1 for the combiner/reducer to sum instead.
        Text outkey = new Text(key);
        oc.collect(outkey, new LongWritable(1L));
    }
}

public static class ReduceUrls extends MapReduceBase
    implements Reducer<Text, LongWritable, WritableComparable,
Writable> {

    public void reduce(
        Text key,
        Iterator<LongWritable> iter,
        OutputCollector<WritableComparable, Writable> oc,
        Reporter reporter) throws IOException {
        // Add up all the values we see

        long sum = 0;
        while (iter.hasNext()) {
            sum += iter.next().get();
            reporter.setStatus("OK");
        }

        oc.collect(key, new LongWritable(sum));
    }
}

public static class LoadClicks extends MapReduceBase
    implements Mapper<WritableComparable, Writable, LongWritable,
Text> {

    public void map(
        WritableComparable key,
        Writable val,
        OutputCollector<LongWritable, Text> oc,
        Reporter reporter) throws IOException {
        oc.collect((LongWritable)val, (Text)key);
    }
}

public static class LimitClicks extends MapReduceBase
    implements Reducer<LongWritable, Text, LongWritable, Text> {

    int count = 0;

    public void reduce(
        LongWritable key,
        Iterator<Text> iter,
        OutputCollector<LongWritable, Text> oc,
        Reporter reporter) throws IOException {

        // Only output the first 100 records
        while (count < 100 && iter.hasNext()) {
            oc.collect(key, iter.next());
            count++;
        }
    }
}

public static void main(String[] args) throws IOException {
    JobConf jp = new JobConf(Example.class);
    jp.setJobName("Load Pages");
    jp.setInputFormat(TextInputFormat.class);
}

```

```

        lp.setOutputKeyClass(Text.class);
        lp.setOutputValueClass(Text.class);
        lp.setInputMapperClass(LoadPages.class);
        FileInputFormat.addInputPath(lp, new
Path("/user/gates/pages"));
        FileOutputFormat.setOutputPath(lp,
        new Path("/user/gates/tmp/indexed_pages"));
        lp.setNumReduceTasks(0);
        Job loadPages = new Job(lp);

        JobConf lfu = new JobConf(MRExample.class);
        lfu.setJobName("Load and Filter Users");
        lfu.setInputFormat(TextInputFormat.class);
        lfu.setOutputKeyClass(Text.class);
        lfu.setOutputValueClass(Text.class);
        lfu.setInputMapperClass(LoadAndFilterUsers.class);
        FileInputFormat.addInputPath(lfu, new
Path("/user/gates/users"));
        FileOutputFormat.setOutputPath(lfu,
        new Path("/user/gates/tmp/filtered_users"));
        lfu.setNumReduceTasks(0);
        Job loadUsers = new Job(lfu);

        JobConf join = new JobConf(MRExample.class);
        join.setJobName("Join Users and Pages");
        join.setInputFormat(KeyValueTextInputFormat.class);
        join.setOutputKeyClass(Text.class);
        join.setOutputValueClass(Text.class);
        join.setMapperClass(IdentityMapper.class);
        join.setReducerClass(Join.class);
        FileInputFormat.addInputPath(join, new
Path("/user/gates/tmp/indexed_pages"));
        FileInputFormat.addInputPath(join, new
Path("/user/gates/tmp/filtered_users"));
        FileOutputFormat.setOutputPath(join, new
Path("/user/gates/tmp/joined"));
        join.setNumReduceTasks(50);
        Job joinJob = new Job(join);
        joinJob.addDependingJob(loadPages);
        joinJob.addDependingJob(loadUsers);

        JobConf group = new JobConf(MRExample.class);
        group.setJobName("Group URLs");
        group.setInputFormat(KeyValueTextInputFormat.class);
        group.setOutputKeyClass(Text.class);
        group.setOutputValueClass(LongWritable.class);
        group.setOutputFormat(SequenceFileOutputFormat.class);
        group.setMapperClass(LoadJoined.class);
        group.setCombinerClass(ReduceUris.class);
        group.setReducerClass(ReduceUris.class);
        FileInputFormat.addInputPath(group, new
Path("/user/gates/tmp/joined"));
        FileOutputFormat.setOutputPath(group, new
Path("/user/gates/tmp/grouped"));
        group.setNumReduceTasks(50);
        Job groupJob = new Job(group);
        groupJob.addDependingJob(joinJob);

        JobConf top100 = new JobConf(MRExample.class);
        top100.setJobName("Top 100 sites");
        top100.setInputFormat(SequenceFileInputFormat.class);
        top100.setOutputKeyClass(LongWritable.class);
        top100.setOutputValueClass(Text.class);
        top100.setOutputFormat(SequenceFileOutputFormat.class);
        top100.setMapperClass(LoadClicks.class);
        top100.setCombinerClass(LimitClicks.class);
        top100.setReducerClass(LimitClicks.class);
        FileInputFormat.addInputPath(top100, new
Path("/user/gates/tmp/grouped"));
        FileOutputFormat.setOutputPath(top100, new
Path("/user/gates/top100/sitesforusers18to25"));
        top100.setNumReduceTasks(5);
        Job limit = new Job(top100);
        limit.addDependingJob(groupJob);

        JobControl jc = new JobControl("Find top 100 sites for users
18 to 25");
        jc.addJob(loadPages);
        jc.addJob(loadUsers);
        jc.addJob(joinJob);
        jc.addJob(groupJob);
        jc.addJob(limit);
        jc.run();
    }
}

```

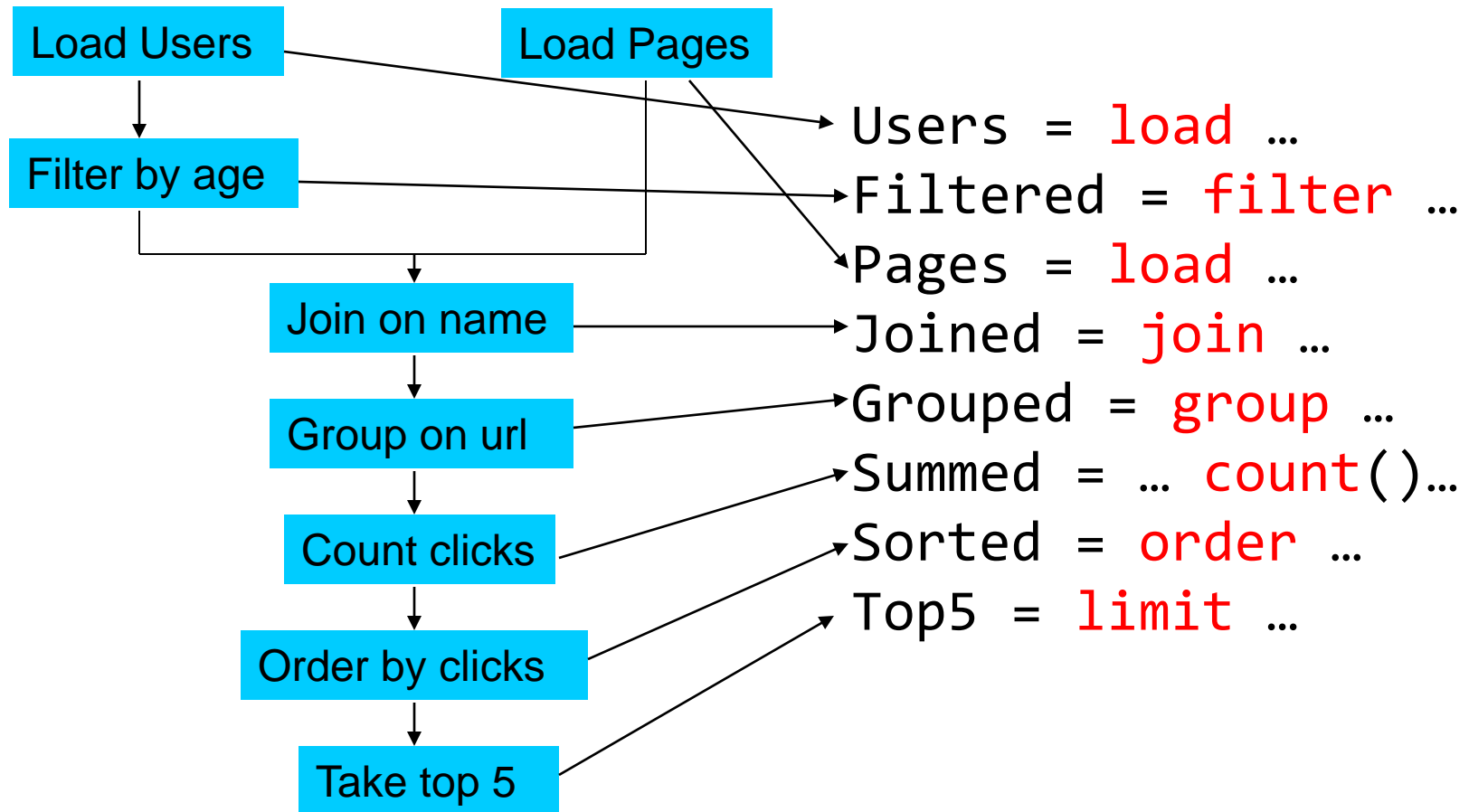
In Pig Latin

```
Users      = load 'users' as (name, age);
Filtered   = filter Users by
              age >= 18 and age <= 25;
Pages      = load 'pages' as (user, url);
Joined     = join Filtered by name, Pages by user;
Grouped    = group Joined by url;
Summed     = foreach Grouped generate group,
              count(Joined) as clicks;
Sorted     = order Summed by clicks desc;
Top5       = limit Sorted 5;

store Top5 into 'top5sites';
```

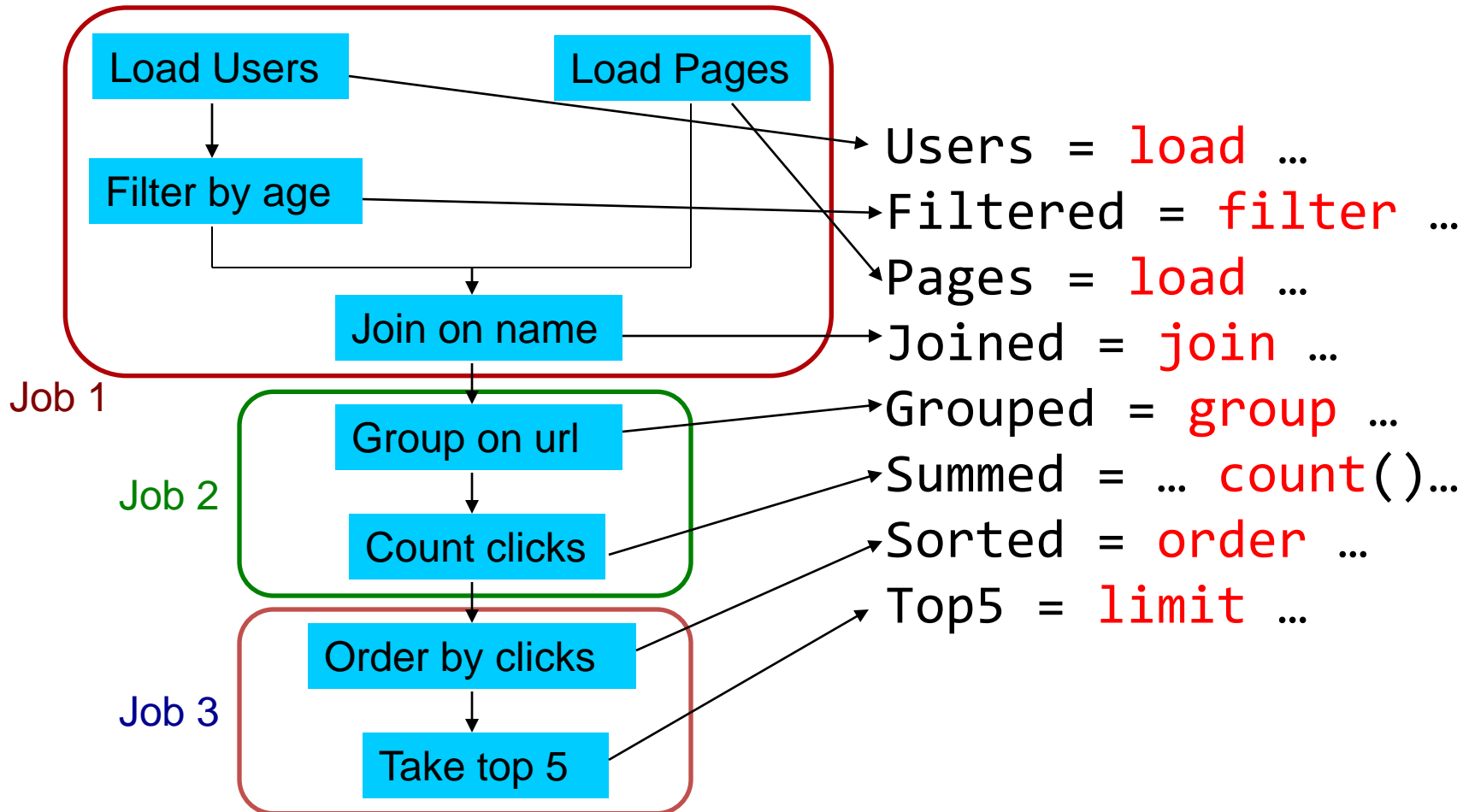
Translation to MapReduce

Notice how naturally the components of the job translate into Pig Latin.



Translation to MapReduce

Notice how naturally the components of the job translate into Pig Latin.



Hive

- Developed at Facebook
- Used for most Facebook jobs
- Relational database built on Hadoop
 - Maintains table schemas
 - SQL-like query language (which can also call Hadoop Streaming scripts)
 - Supports table partitioning, complex data types, sampling, some query optimization

