

Cloud Computing using MapReduce, Hadoop, Spark

Benjamin Hindman

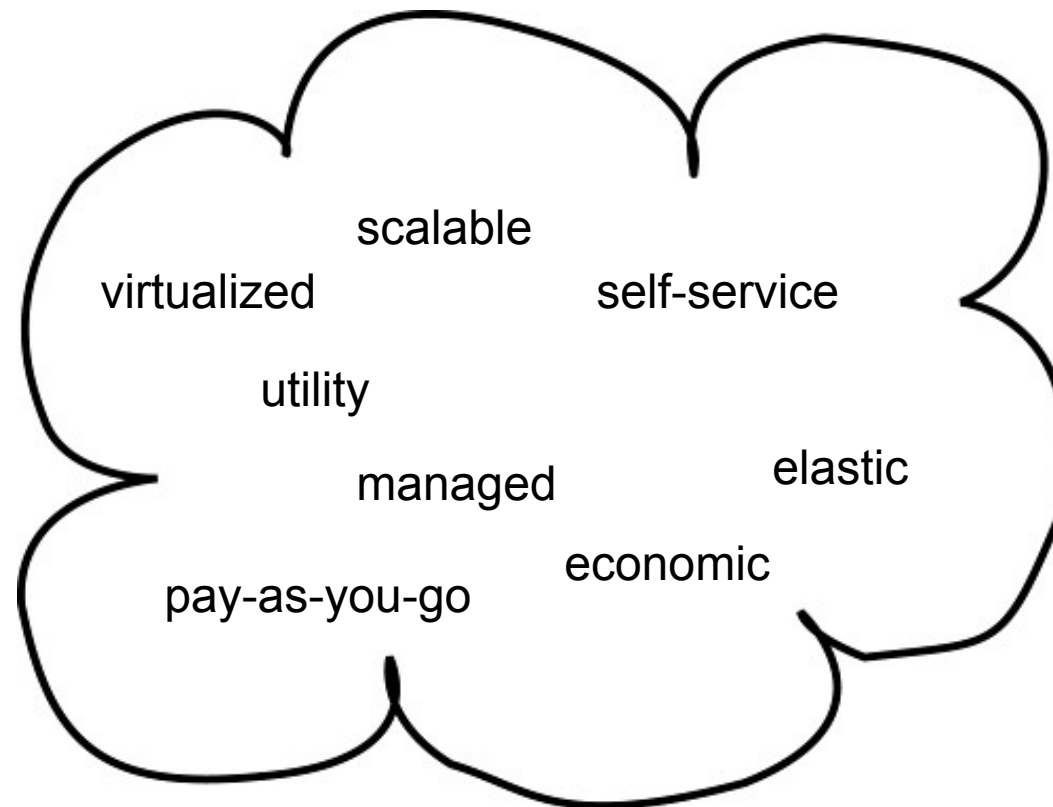
benh@cs.berkeley.edu



Why this talk?

- At some point, you'll have enough data to run your “parallel” algorithms on multiple computers
- SPMD (e.g., MPI, UPC) might not be the best for your **application**, or your **environment**

What is Cloud Computing?



What is Cloud Computing?

- “Cloud” refers to large Internet services running on 10,000s of machines (Amazon, Google, Microsoft, etc)
- “Cloud computing” refers to services by these companies 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
 - Windows Azure: higher-level than EC2, applications use API

What is Cloud Computing?

- Virtualization
 - From co-location, to hosting providers running the web server, the database, etc and having you just FTP your files ... now you do all that yourself again!
- Self-service (use personal credit card) and pay-as-you-go
- Economic incentives
 - Provider: Sell unused resources
 - Customer: no upfront capital costs building data centers, buying servers, etc

“Cloud Computing”

- Infinite scale ...

From: [REDACTED]
To: [REDACTED]
Cc: [REDACTED] Benjamin Hindman <benh@EECS.Berkeley.EDU>; [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?


[REDACTED]
CS Graduate Student
UC Berkeley

“Cloud Computing”

- Always available ...



The image shows a screenshot of a Twitter interface. At the top, the Twitter logo and navigation links (Home, Profile, Messages, Who To Follow) are visible. The main content is a tweet from @4sqSupport (foursquare support) with the text: "We're down due to the current Amazon #EC2 outage. Please bear with us!". The tweet is dated "8 Aug via CoTweet" and includes interaction options like Favorite, Retweet, and Reply. Below the tweet, it says "Retweeted by cjschris and 39 others" and shows a row of profile pictures. At the bottom of the screenshot, there is a footer with links (About, Help, Blog, Status, Jobs, Terms, Privacy, Advertisers, Businesses, Media, Developers, Resources) and a copyright notice "© 2011 Twitter".




twitter Home Profile Messages Who To Follow  behn ▾

 **@4sqSupport**
foursquare support

We're down due to the current Amazon
#EC2 outage. Please bear with us!

8 Aug via CoTweet ☆ Favorite ↻ Retweet ↩ Reply

Retweeted by cjschris and 39 others

Amazon RDS (N. Virginia)       

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

Challenges in the Cloud Environment

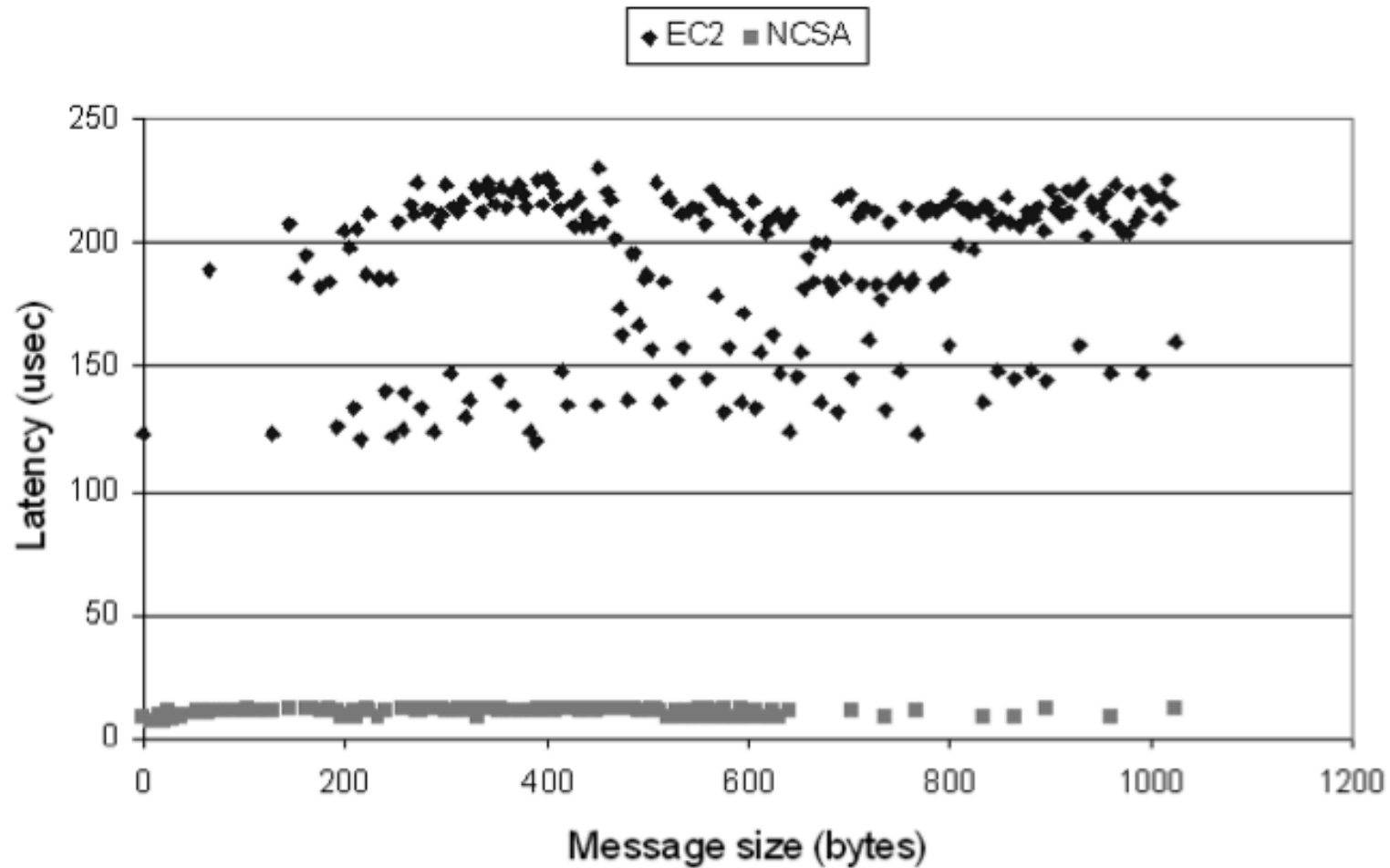
- Cheap nodes fail, especially when you have many
 - Mean time between failures for 1 node = 3 years
 - MTBF for 1000 nodes = 1 day
 - **Solution:** Restrict programming model so you can efficiently “build-in” fault-tolerance (art)
- Commodity network = low bandwidth
 - **Solution:** Push computation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

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>

Latency on EC2 vs Infiniband



Source: Edward Walker. Benchmarking Amazon EC2 for High Performance Computing. *login*, vol. 33, no. 5, 2008.

MPI in the Cloud

- Cloud data centers often use 1 Gbps Ethernet, which is much slower than supercomputer networks
- Studies show poor performance for communication intensive codes, but OK for less intensive ones
- New HPC specific EC2 “sizes” that may help: 10 Gbps Ethernet, and optionally $2 \times$ Nvidia Tesla GPUs

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>**

Outline

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

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)
```

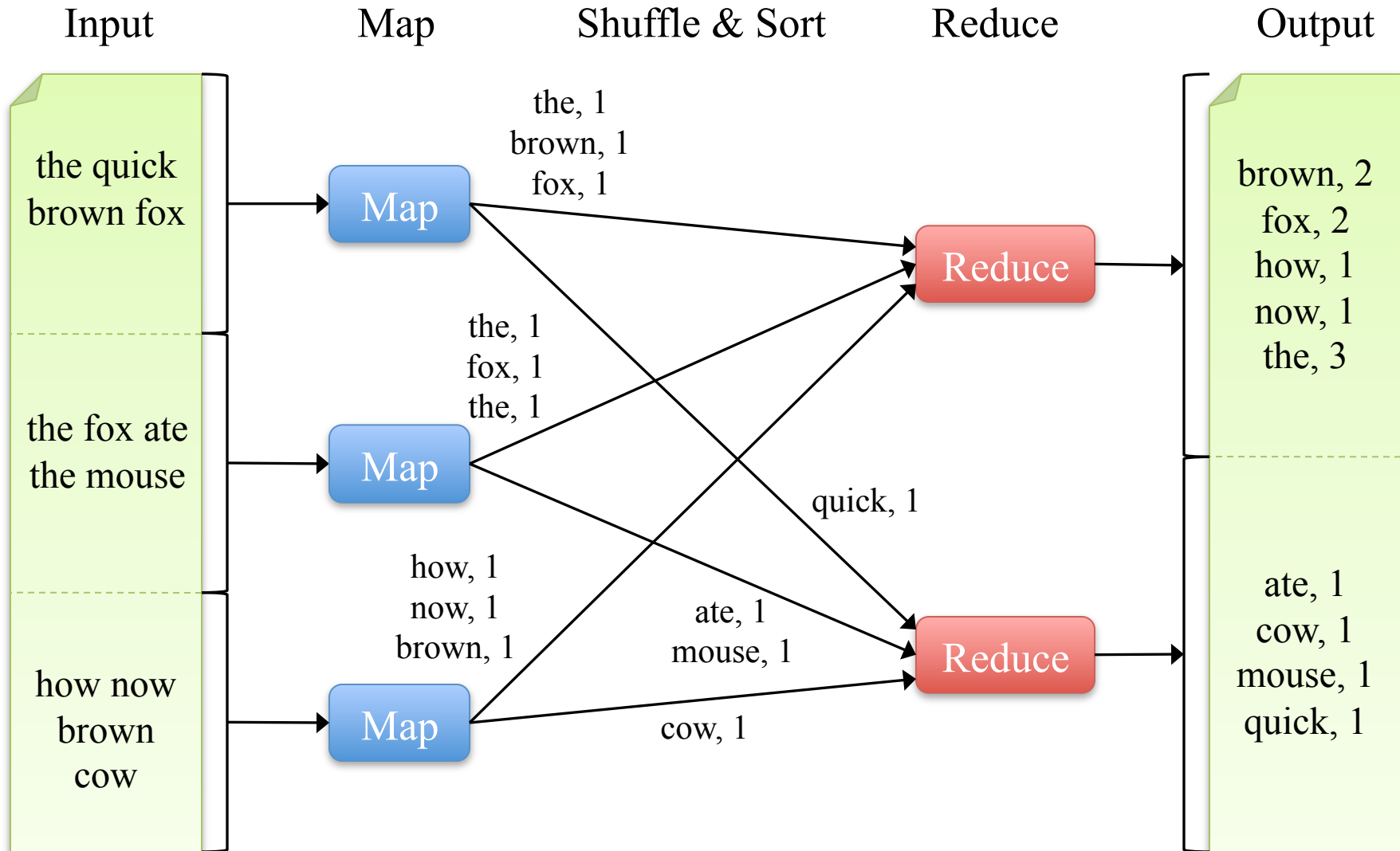
```
def reduce(key, values):  
    output(key, sum(values))
```

Example: Word Count

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

```
def reduce(key, values):  
    output(key, values.size())
```

Example: Word Count

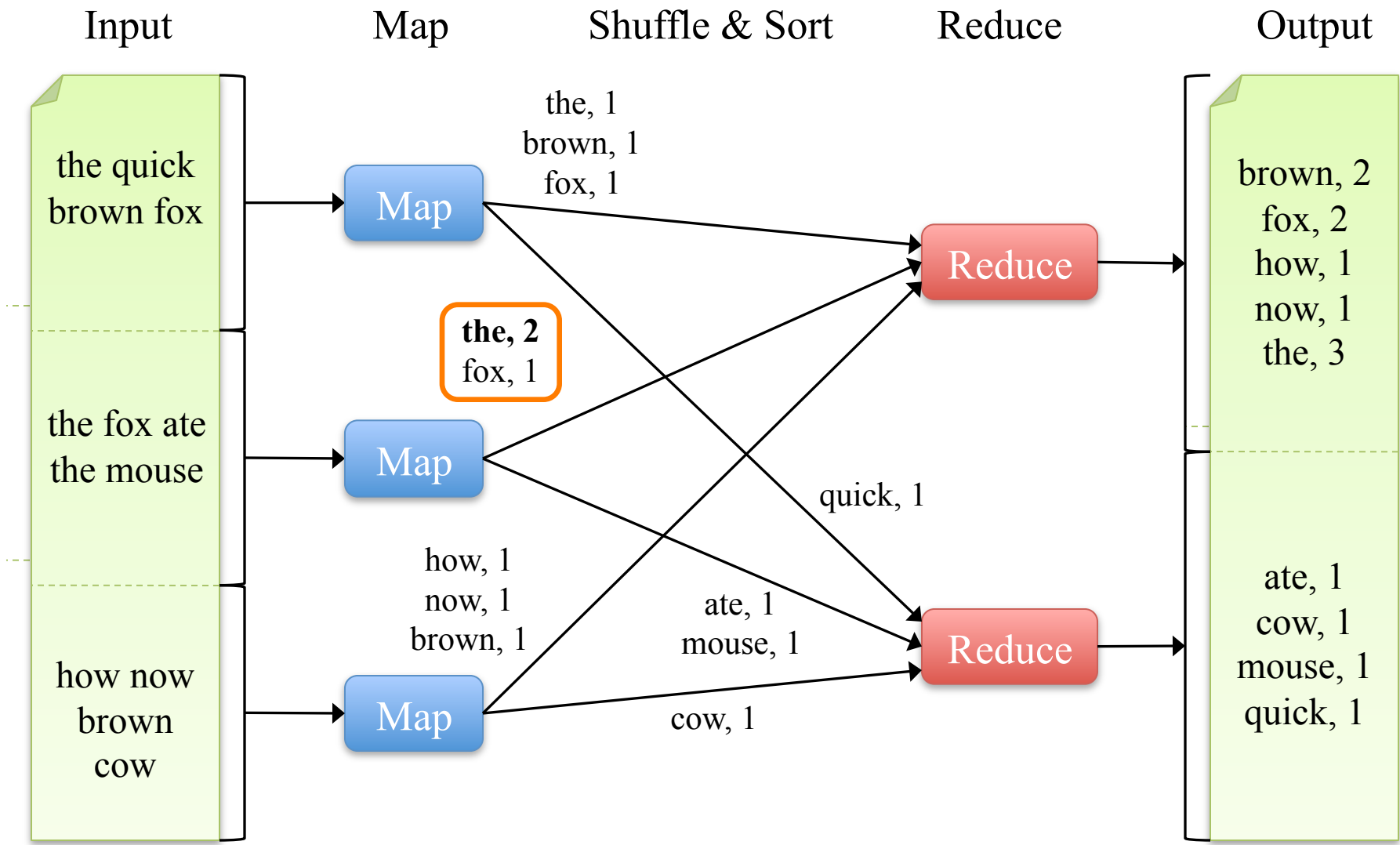


Optimization: Combiner

- 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))
```

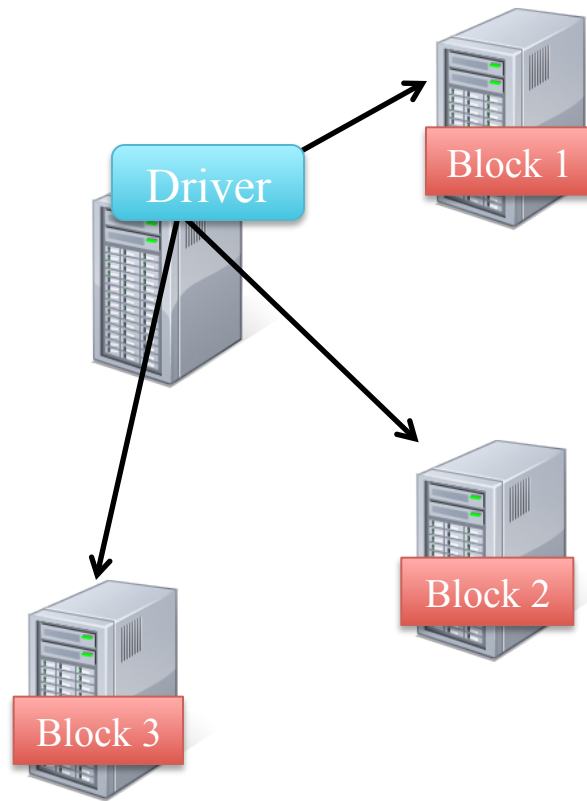
Example: Word Count + Combiner



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

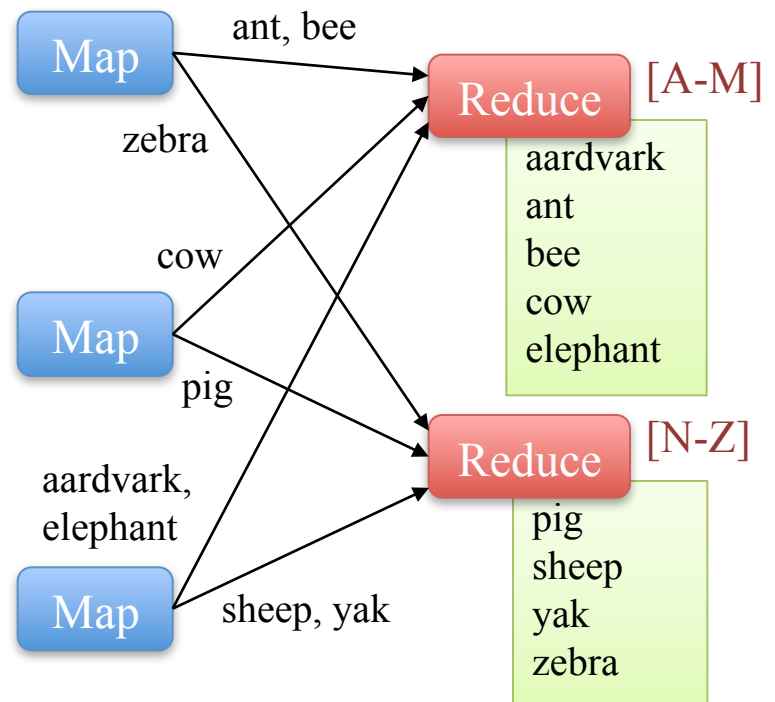
- MapReduce
- MapReduce Examples
- Introduction to Hadoop
- Beyond MapReduce
- Summary

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)

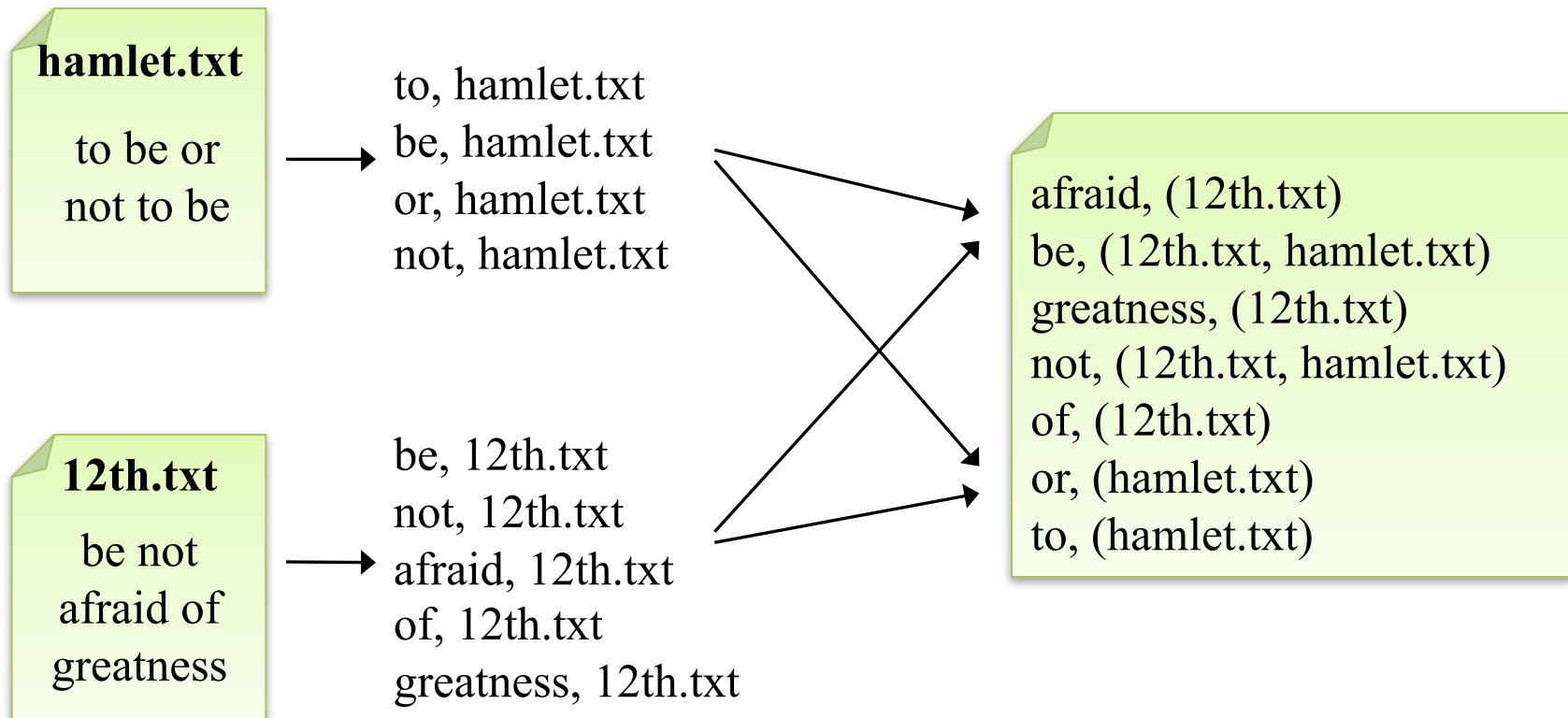
3. Inverted Index

- **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
 - Estimate count distribution by sampling in Job 1.5

5. Numerical Integration

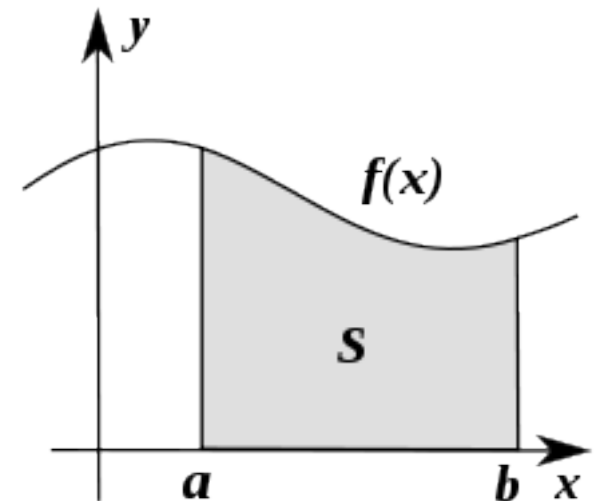
- **Input:** (start, end) records for sub-ranges to integrate*
- **Output:** integral of $f(x)$ over entire range

- **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))
```

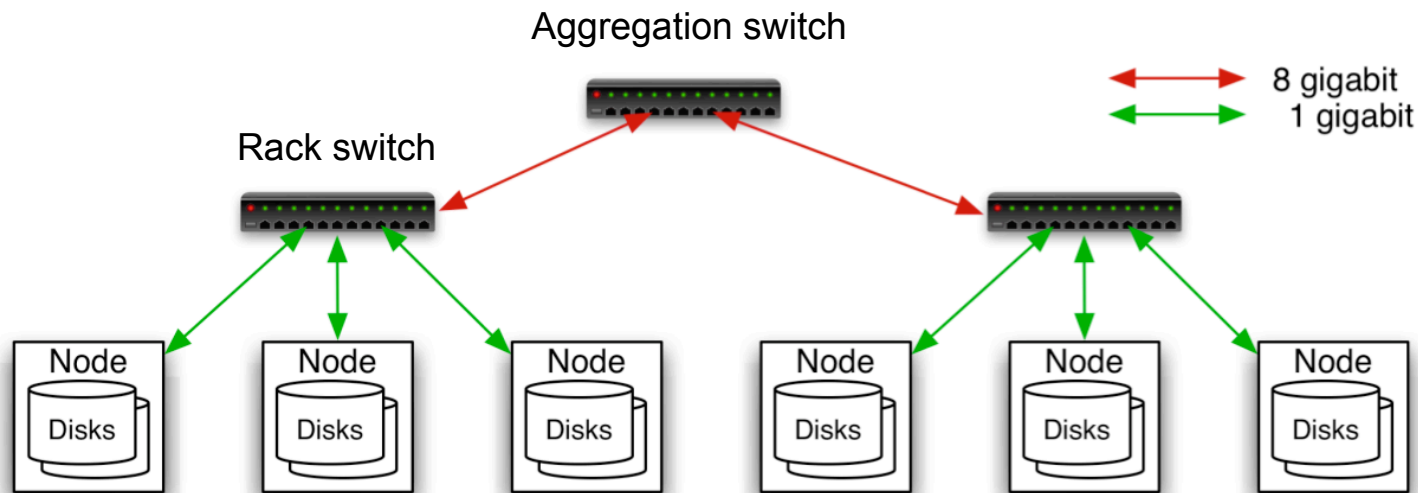


*Can implement using custom InputFormat

Outline

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

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

Typical Hadoop Cluster



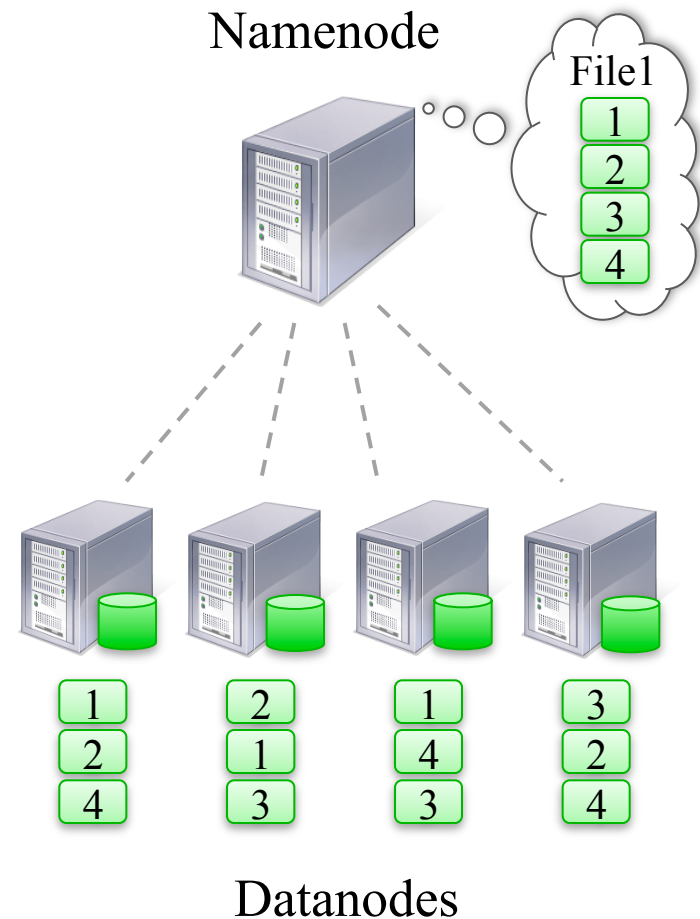
Hadoop Components

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



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

- (When you've had enough with configuring and deploying a Hadoop clusters manually)
- Web interface and command-line tools for running Hadoop jobs on EC2
- Data stored in Amazon S3
- Monitors job and shuts down machines when finished

Elastic MapReduce UI

Create a New Job Flow

Cancel 



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 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](#) * 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

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

Beyond MapReduce

- Many other projects follow MapReduce's example of restricting the programming model for efficient execution in datacenters
 - **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): ...

Spark

- **Motivation:** iterative jobs (common in machine learning, optimization, etc)
- **Problem:** iterative jobs reuse the same *working set* of data over and over, but MapReduce / Dryad / etc require acyclic data flows
- **Solution:** “resilient distributed datasets” that are cached in memory but can be rebuilt on failure

Spark Programming Model

- Resilient distributed datasets (RDDs)
 - Immutable, partitioned collections of objects
 - Created through parallel *transformations* (map, filter, groupBy, join, ...) on data in stable storage
 - Can be *cached* for efficient reuse
- *Actions* on RDDs
 - Count, reduce, collect, save, ...

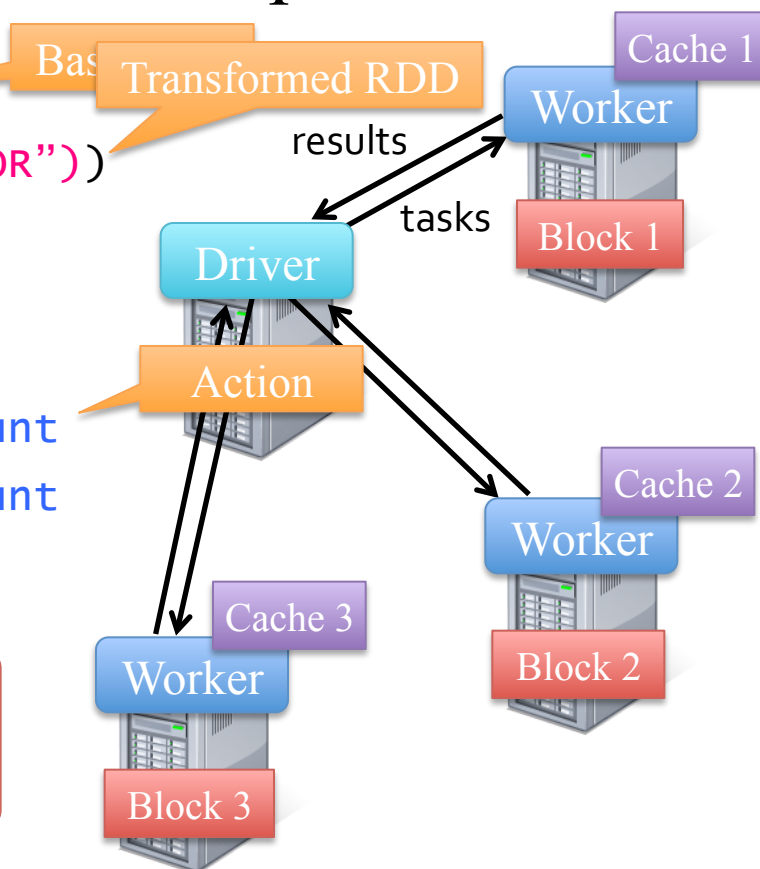
Example: Log Mining

- 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  
...
```

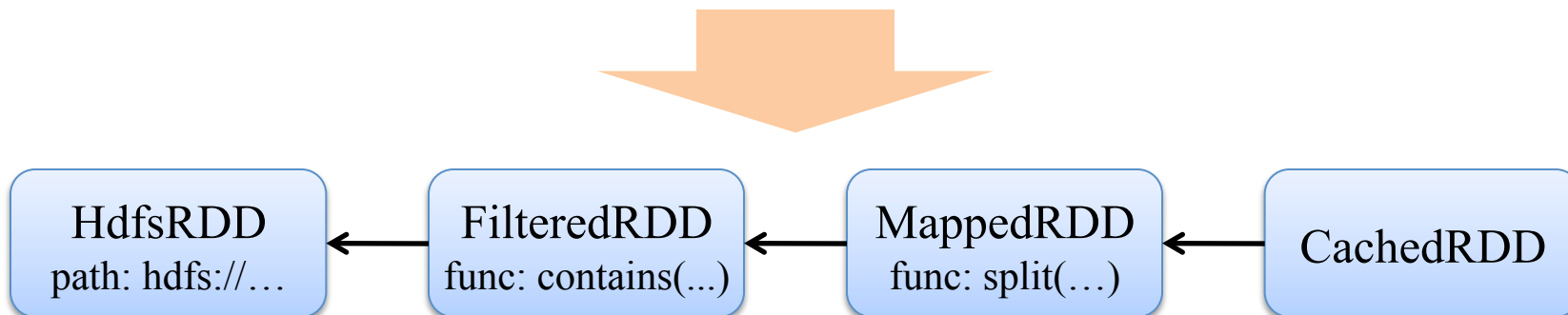
Result: scaled to 1 TB data in 5-7 sec
(vs 170 sec for on-disk data)



Fault Tolerance in Spark

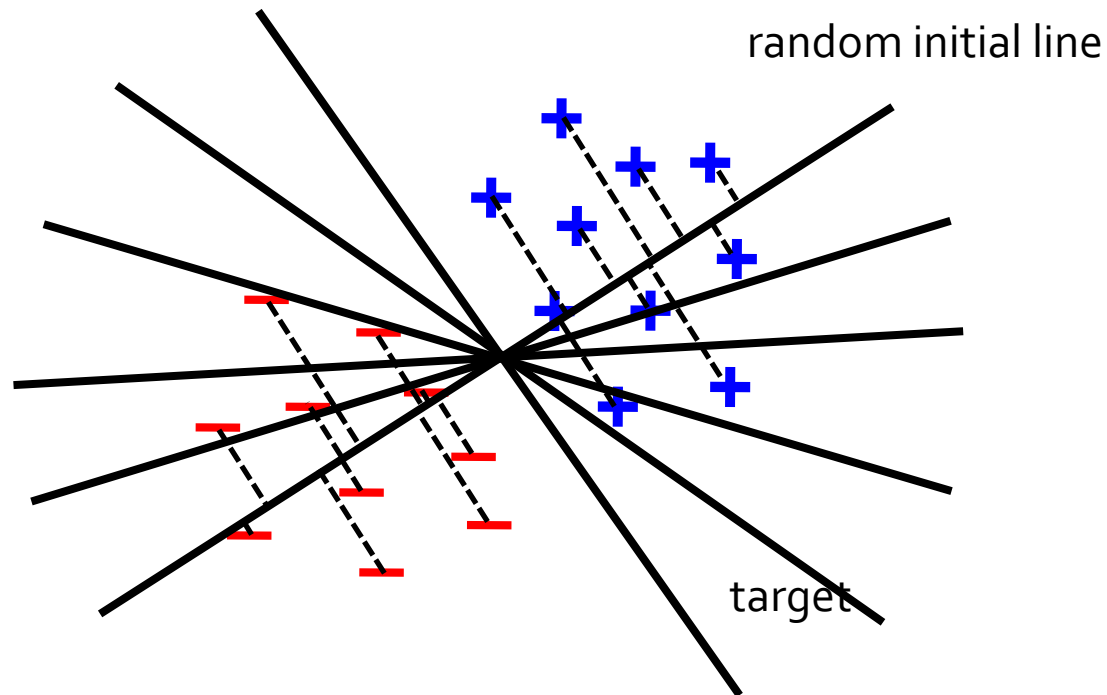
- RDDs maintain *lineage* information that can be used to reconstruct lost partitions
- Example:

```
cachedMsgs = textFile(...).filter(_.contains("error"))  
                          .map(_.split('\t')(2))  
                          .cache()
```



Example: Logistic Regression

- Goal: find best line separating two sets of points



Example: Logistic Regression

```
val data = spark.textFile(...).map(readPoint).cache()
```

```
var w = Vector.random(D)
```

```
for (i <- 1 to ITERATIONS) {
```

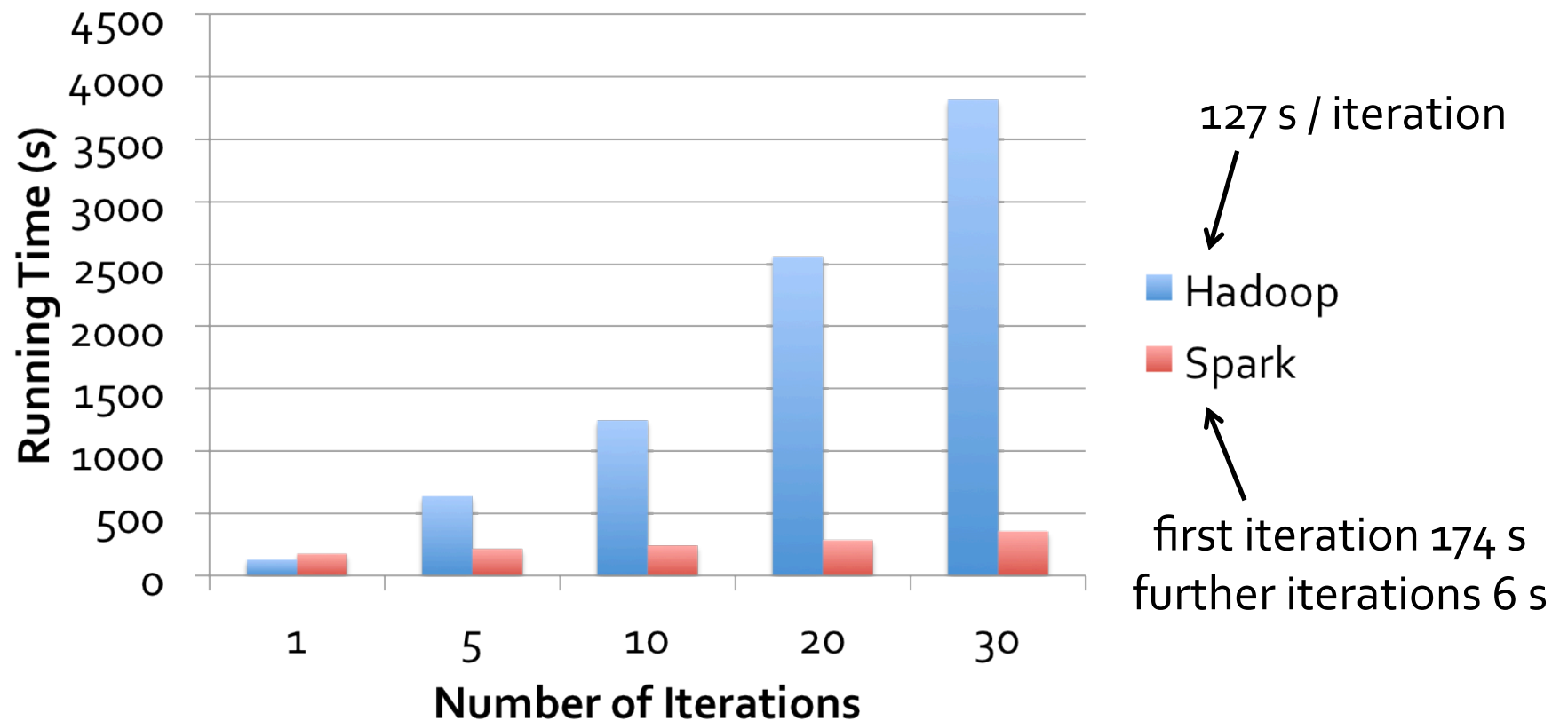
```
  val gradient = data.map(p =>  
    (1 / (1 + exp(-p.y*(w dot p.x))) - 1) * p.y * p.x  
  ).reduce(_ + _)
```

```
  w -= gradient
```

```
}
```

```
println("Final w: " + w)
```

Logistic Regression Performance

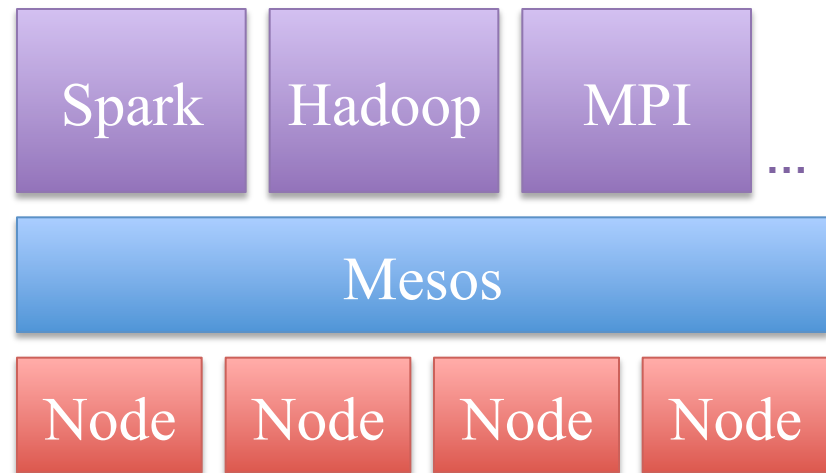


Interactive Spark

- Ability to cache datasets in memory is great for *interactive* data analysis: extract a working set, cache it, query it repeatedly
- Modified Scala interpreter to support interactive use of Spark
- Result: full-text search of Wikipedia in 0.5s after 20-second initial load

Beyond Spark

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



www.mesos-project.org

Outline

- 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/](http://docs.amazonwebservices.com/ElasticMapReduce/latest/GettingStartedGuide/)
- Spark: <http://spark-project.org>
- Mesos: <http://mesos-project.org>

Thanks!