Applied Data Analysis (CS401)

Lecture 13
Scaling to massive data
13 Dec 2023

Robert West

Announcements

- Homework H2
  - Feedback to be released later this week
  - Reminder (see Ed): Please participate in ML4Ed study by Fri 14:00
- Project milestone P3 due next week (Fri 22 Dec)
- Friday's lab session:
  - Last lab session! ⟶ Last quiz (on lecture 12)
  - Project office hour (same sign-up protocol as last week)
  - Exercises on Spark (useful for your future projects, your job, your love life)
- **Course eval is available on IS-Academia!**
  - Note: different from the eval from a few weeks ago!

Feedback

Give us feedback on this lecture here:
https://go.epfl.ch/ada2023-lec13-feedback

- What did you (not) like about this lecture?
- What was (not) well explained?
- On what would you like more (fewer) details?
- Where is Waldo? 
- …

So far in this class...

- We made one big assumption:
  - All data fits on a single machine
  - Even more, all data fits into memory on a single machine (Pandas)
- Realistic assumption for **prototyping,** but frequently not for production code

The big-data problem

Data is growing faster than computation speed

+ Growing data sources
  (e.g, Web, mobile, sensors, …)

+ Cheap hard-disk storage

– Stalling CPU speeds

– RAM bottlenecks

Examples

Facebook's daily logs: 60 TB

1000 Genomes project: 200 TB

Google Web index: 100+ PB

These numbers (anno domini 2016) are outdated (too small)!

Cost of 1 TB of disk: $50

Time to read 1 TB from disk: 3 hours (100 MB/s)

The big-data problem

A single machine can no longer store, let alone process, all the data

The only solution is to **distribute** over a large cluster of machines

But how much data should you get?

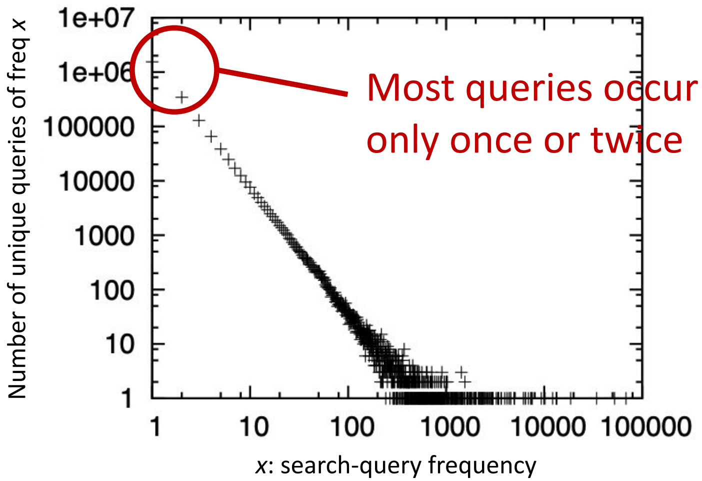
Of course, “it depends”, but for many applications the answer is:

As much as you can get

Big data about people (text, Web, social media) tends to follow heavy-tailed distributions

(e.g., power laws)

Example: Web search



59% of all Web search queries are unique

17% of all queries are made only twice

8% are made three times

Hardware for big data

Budget (a.k.a. commodity) hardware

Not "gold-plated" (a.k.a. custom)

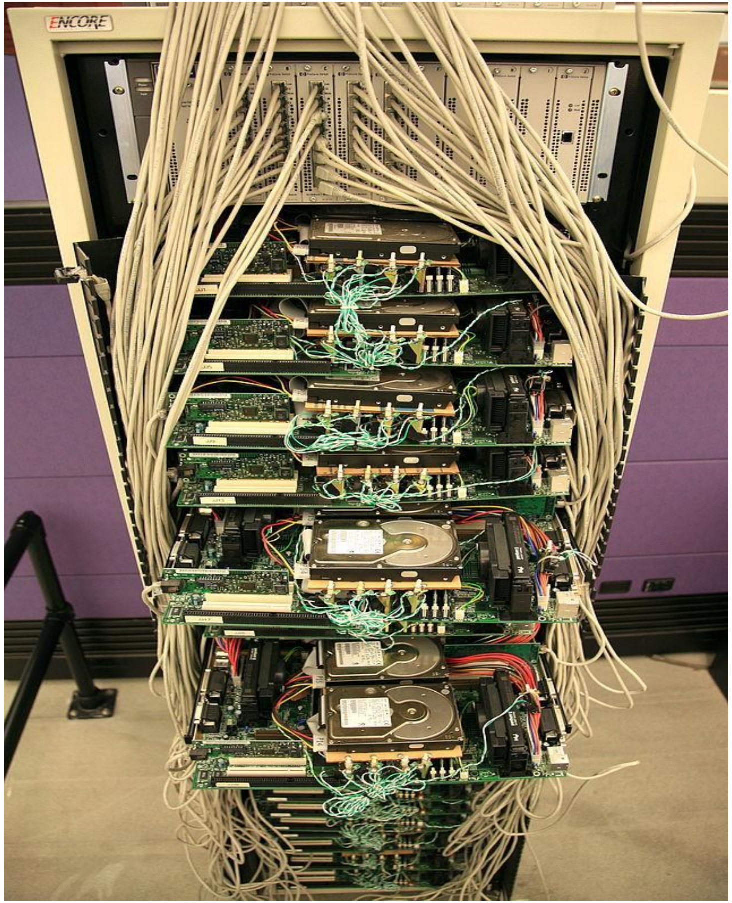
Many low-end servers

Easy to add capacity

Cheaper per CPU and per disk

Increased complexity in software:

- Fault tolerance
- Virtualization (e.g., distributed file systems)



Google Corkboard server: Steve Jurvetson/Flickr

Problems with cheap hardware

Failures, e.g. (Google numbers)

- 1–5% hard drives/year
- 0.2% DIMMs (dual in-line memory modules)/year

Commodity network (1–10 Gb/s) speeds vs. RAM

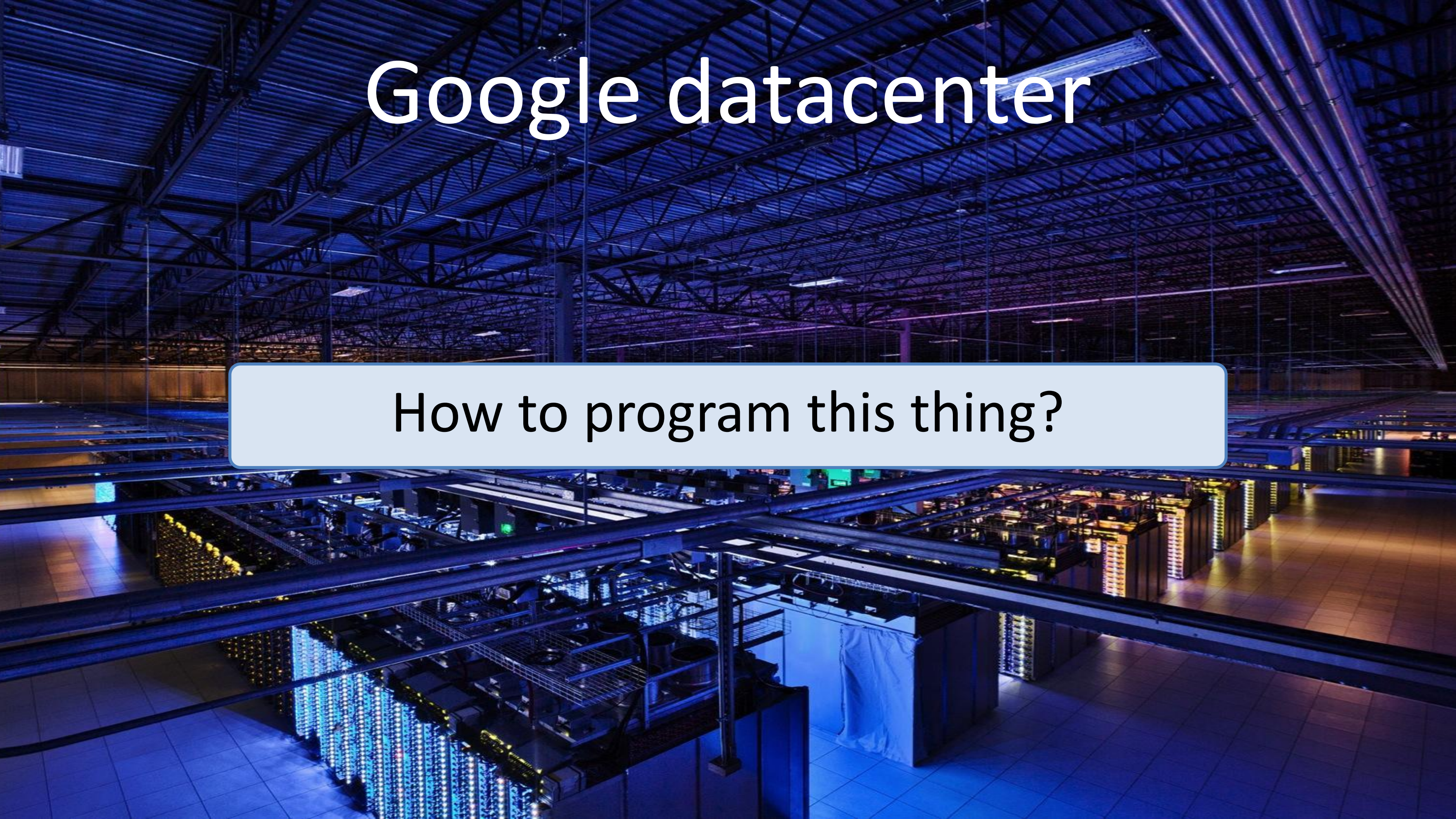
- Much more latency (100–100,000x)
- Lower throughput (100–1,000x)

Uneven performance

- Inconsistent hardware (e.g., old + new)
- Variable network latency
- External loads

These numbers are constantly changing thanks to new technology!

Google datacenter



How to program this thing?

What’s hard about cluster computing?

1. **How to split work across machines?**
2. **How to deal with failures?**

How do you count the number of occurrences of each word in a document?

"I am Sam
I am Sam
Sam I am
Do you like
Green eggs and
ham?"

I: 3
am: 3
Sam: 3
do: 1
you: 1
like: 1
…

A hashtable (a.k.a. dict)!

“I am Sam
I am Sam
Sam I am
Do you like
Green eggs and
ham?”

{}

A hashtable!

"I am Sam
I am Sam
Sam I am
Do you like
Green eggs and
ham?"

{I: 1}

A hashtable!

“I $\boxed{\text{am}}$ Sam
I am Sam
Sam I am
Do you like
Green eggs and
ham?”

{I: 1,
am: 1}

A hashtable!

“I am $\boxed{\text{Sam}}$
I am Sam
Sam I am
Do you like
Green eggs and
ham?”

{I: 1,
am: 1,
Sam: 1}

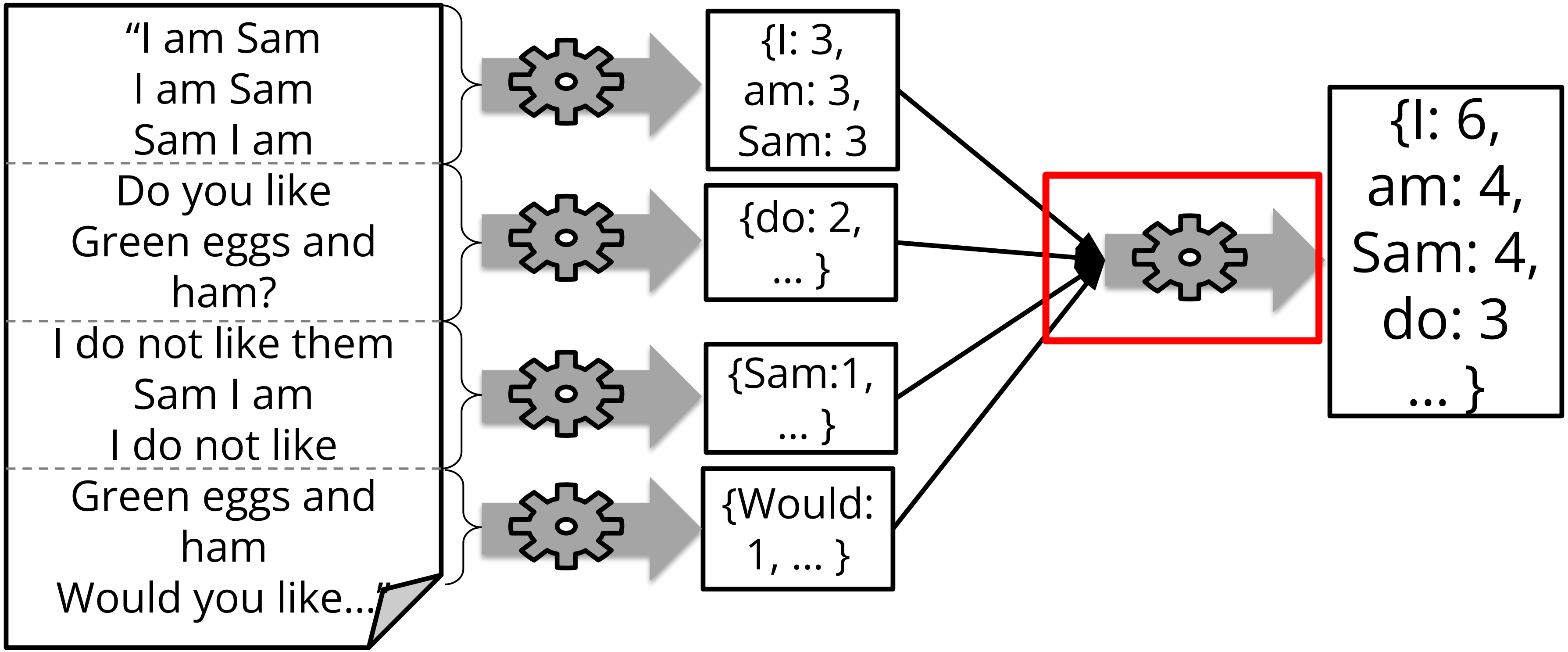
A hashtable!

“I am Sam
$\boxed{\text{I}}$ am Sam
Sam I am
Do you like
Green eggs and
ham?”

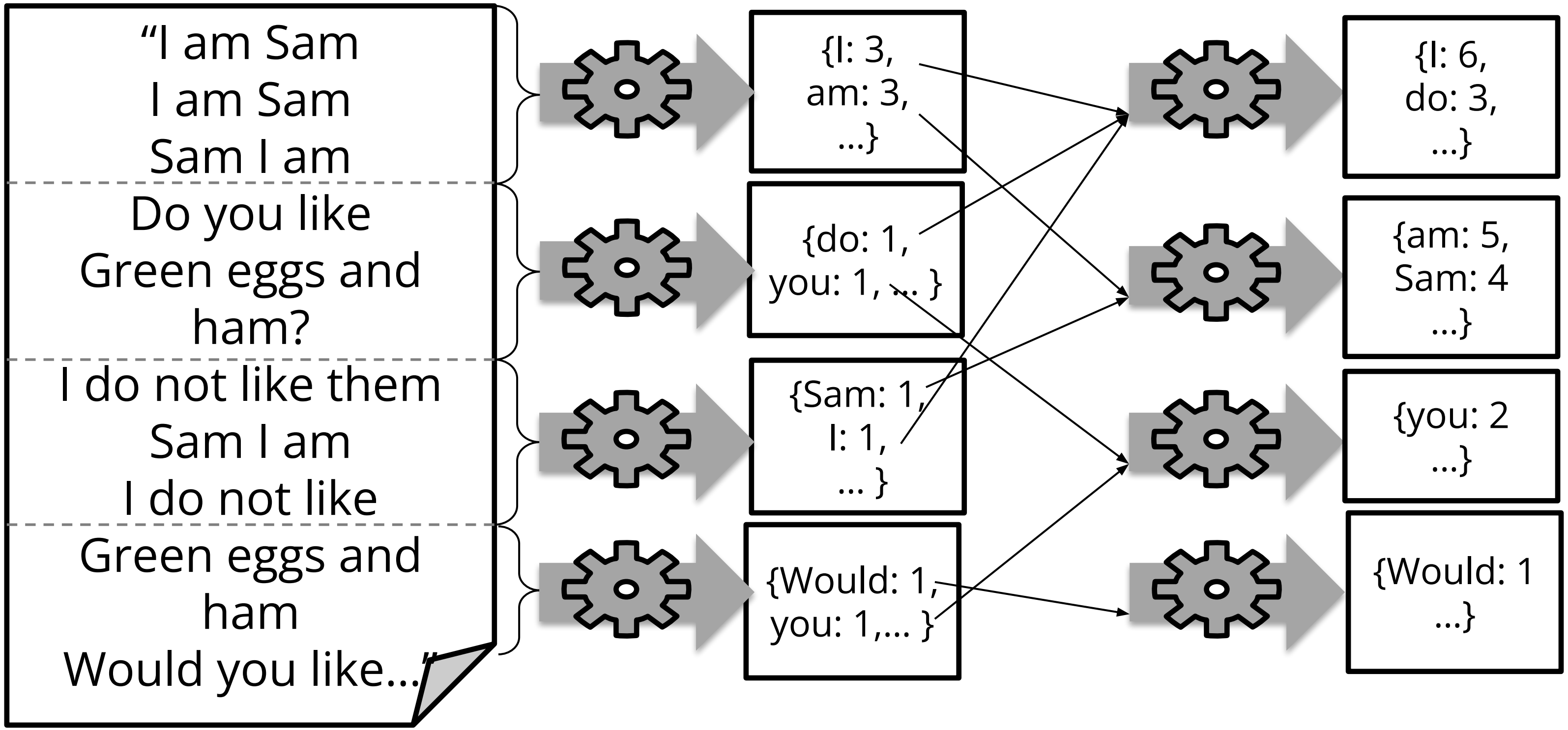
{I: 2,
am: 1,
Sam: 1}

What if the document is really big?

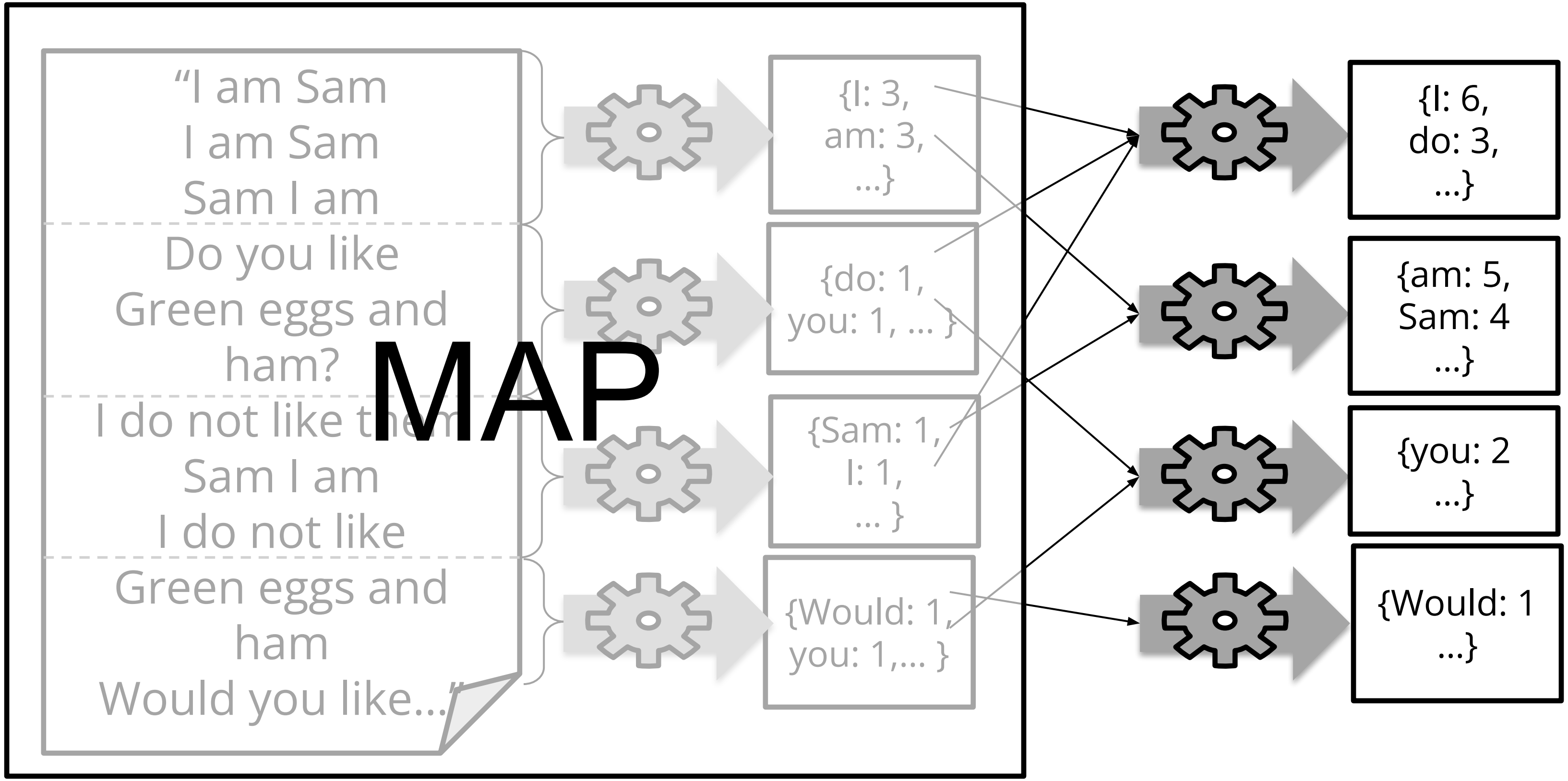
What if the document is really big?



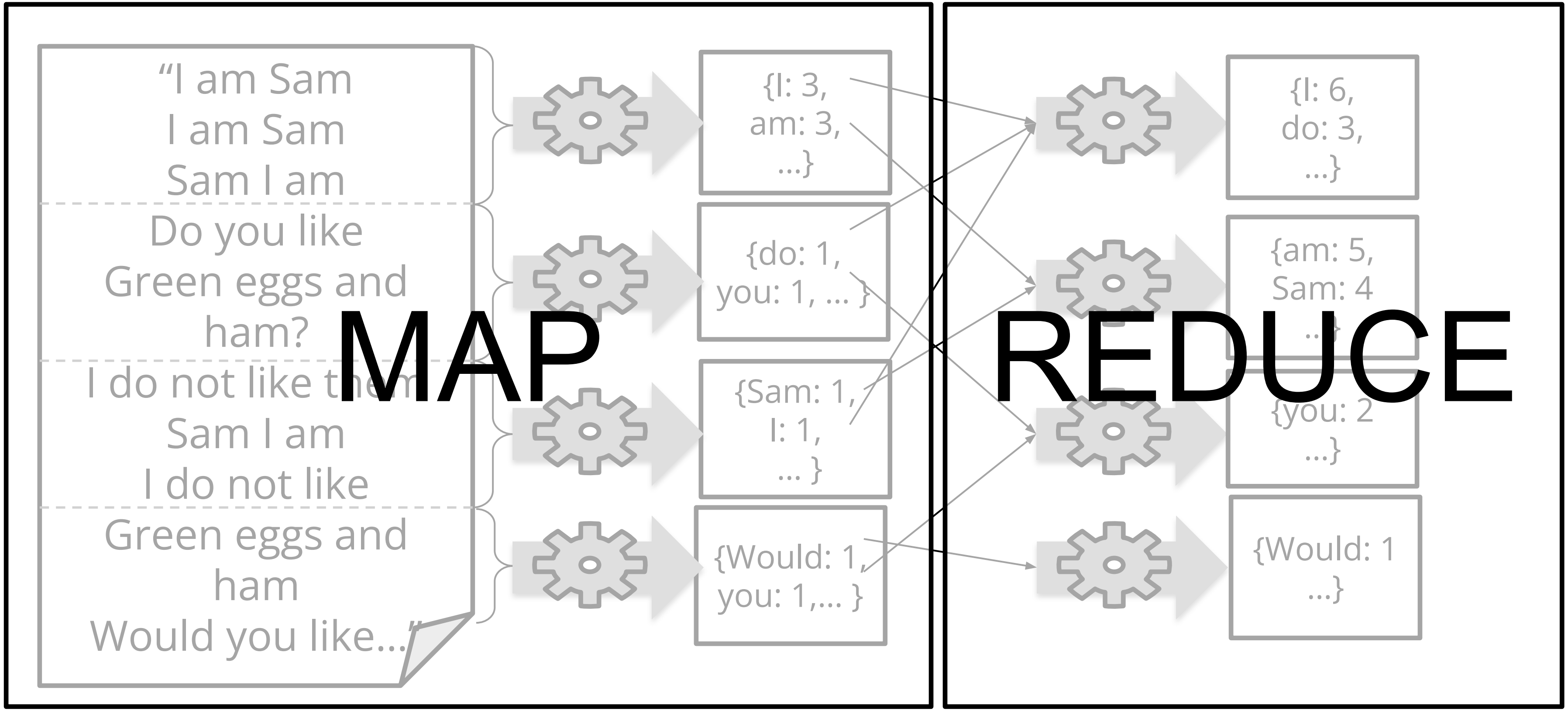
“Divide and Conquer”



“Divide and Conquer”



“Divide and Conquer”



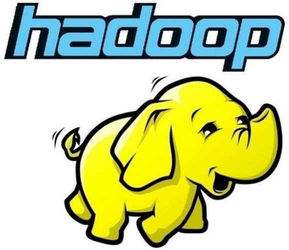
Recall: What's hard about cluster computing?

1. How to divide work across machines?
   - Moving data may be very expensive
   - Must consider network, data locality

2. How to deal with failures?
   - 1 server fails every 3 years ⇒ 10K servers see~10 faults/day
   - Even worse: stragglers (node not failed, but slow)

Solution: MapReduce

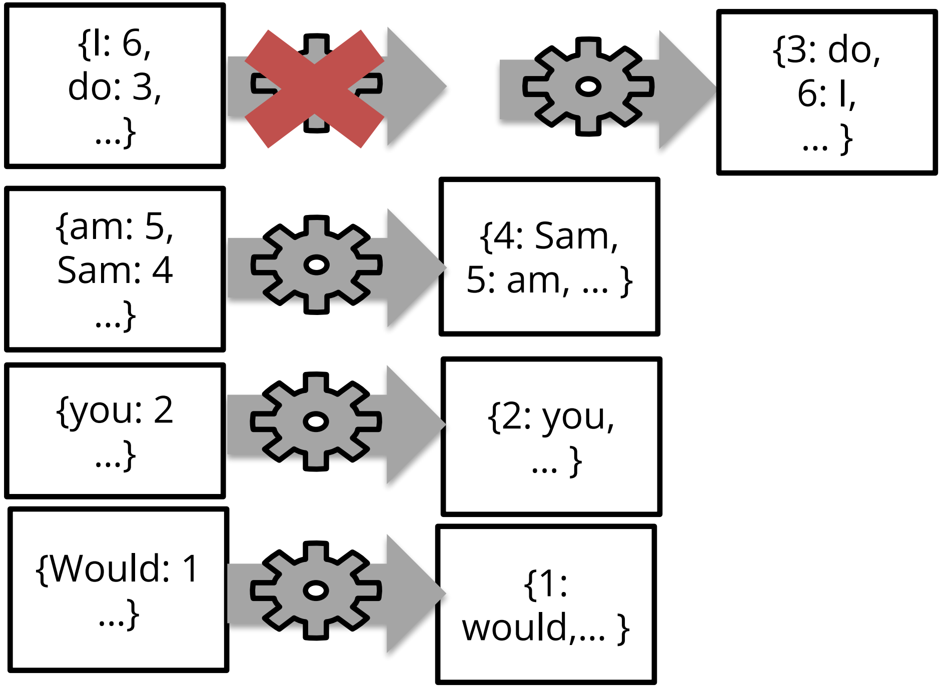
Jeff Dean [facts]



- Smart systems engineers have done all the work for you
  - Task scheduling
  - Virtualization of file system
  - Fault tolerance (incl. data replication)
  - Job monitoring
  - etc.
- “All” you need to do: implement Mapper and Reducer classes

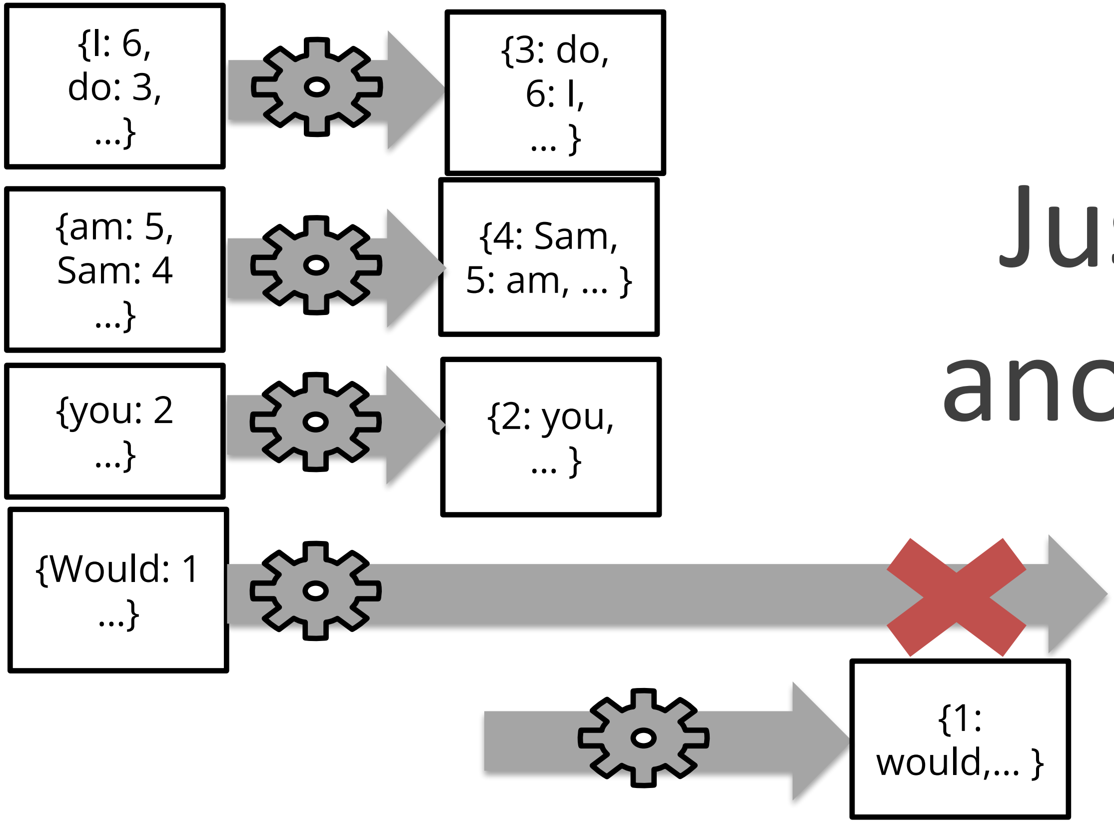
Applied Machine Learning Days ’19 [link]

How to deal with failures?



Just launch another task!

How to deal with slow tasks?



Just launch another task!

Solution: MapReduce

Jeff Dean

- Smart systems engineers have done all the work for you
  - Task scheduling
  - Virtualization of file system
  - Fault tolerance (incl. data replication)
  - Job monitoring
  - etc.
- “All” you need to do: implement Mapper and Reducer classes

Need to break more complex jobs into sequence of MapReduce jobs

Example task

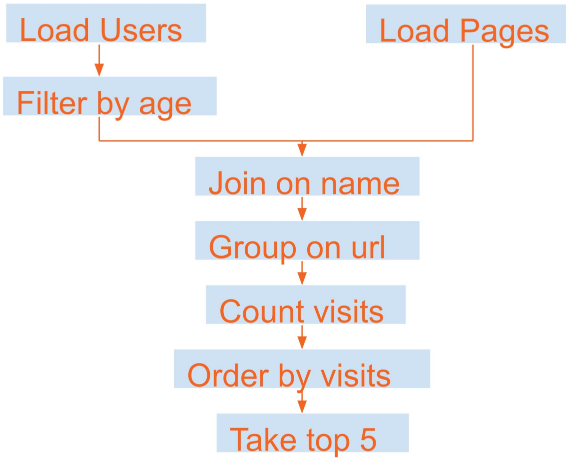
Suppose you have user info in one file, website logs in another, and you need to find the top 5 pages most visited by users aged 18–25

Example from http://wiki.apache.org/pig-data/attachments/PigTalksPapers/attachments/ApacheConEurope09.ppt

In MapReduce

```java
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.MapReduceBase;
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.Job;
import org.apache.hadoop.mapred.jobcontrol.JobC ontrol;
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.sub string(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 w e 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
store it
            // accordingly.
            List<String> first = new ArrayList<String>();
            List<String> second = new ArrayList<String>();

            while (iter.hasNext()) {
                Text t = iter.next();
                String value = t.to String();
                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(',', first Comma);
            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 lp = new JobConf(MRExample.class);
        lp.setJobName("Load Pages");
        lp.setInputFormat(TextInputFormat.class);

        lp.setOutputKeyClass(Text.class);
        lp.setOutputValueClass(Text.class);
        lp.setMapperClass(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.setMapperClass(LoadAndFilterUsers.class);
        FileInputFormat.add InputPath(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(IdentityMap per.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.se tOutputPath(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(MRE xample.class);
        group.setJobName("Group URLs");
        group.setInputFormat(KeyValueTextInputFormat.class);
        group.setOutputKeyClass(Text.class);
        group.setOutputValueClass(LongWritable.class);
        group.setOutputFormat(SequenceFi leOutputFormat.class);
        group.setMapperClass(LoadJoined.class);
        group.setCombinerClass(ReduceUrls.class);
        group.setReducerClass(ReduceUrls.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(SequenceFileOutputF ormat.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/top100sitesforusers18to25"));
        top100.setNumReduceTasks(1);
        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();
    }
}
```

Enter: Spark



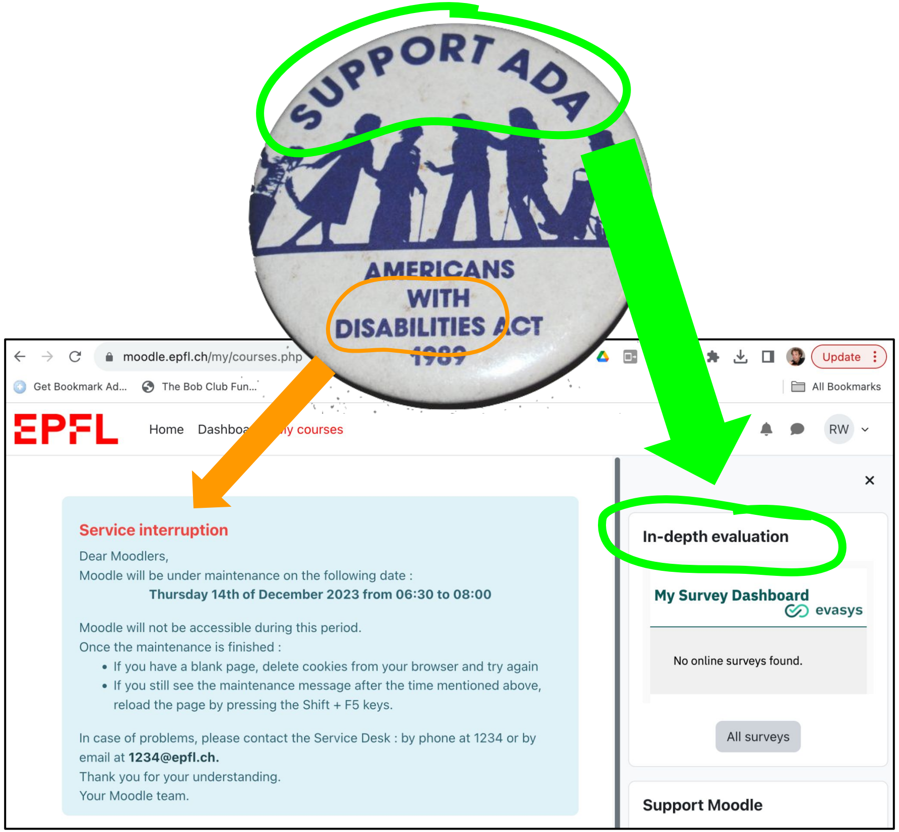
- A high-level API for programming MapReduce-like jobs

```
sc = SparkContext()
print “I am a regular Python program, using the pyspark lib”
users = sc.textFile(‘users.tsv’)  # user <TAB> age
          .map(lambda s: tuple(s.split(‘\t’)))
          .filter(lambda (user, age): age>=18 and age<=25)
pages = sc.textFile(‘pageviews.tsv’)  # user <TAB> url
          .map(lambda s: tuple(s.split(‘\t’)))
counts = users.join(pages)
              .map(lambda (user, (age, url)): (url, 1))
              .reduceByKey(add)
              .takeOrdered(5)
```

Spark

- Implemented in Scala (go EPFL!)
- Additional APIs in
  - Python
  - Java
  - R

Commercial break



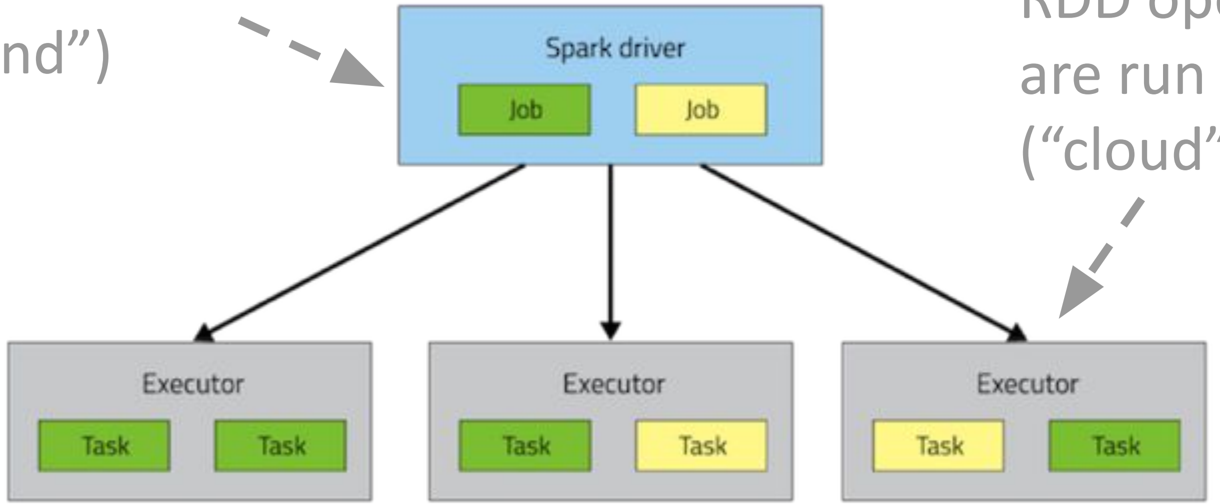
RDD: resilient distributed dataset

- To programmer: looks like one single list (each element represents a “row” of a dataset)
- Under the hood: oh boy…
    - RDDs “live in the cloud”: split over several machines, replicated, etc.
    - Can be processed in parallel
    - Can be transformed to a single, real list (if small…)
    - Typically read from the distributed file system (HDFS)
    - Can be written to the distributed file system

Spark architecture

Your Python script runs in the driver ("ground")

RDD operations are run in executors ("cloud")



RDD operations

- **“Transformations”**
  - Input: RDD; output: another RDD
  - Everything remains “in the cloud”
  - Example: for every entry in the input RDD, count chars
    - RDD:[‘I’, ‘am’, ‘you’] → RDD:[1, 2, 3]
- **“Actions”**
  - Input: RDD; output: a value that is returned to the driver
  - Result is transferred “from cloud to ground”
  - Examples: take a sample of entries from RDD and print it on the driver’s shell; or store results to file (local or distributed)

Lazy execution [unrelated]

- **Transformations** (i.e., RDD→RDD operations) are not executed until it's really necessary (a.k.a. "lazy execution")
- Execution of transformations triggered by **actions**
- Why?
  - If you never look at the data, there's no point in manipulating it…
  - Smarter query processing possible:
    E.g., rdd2 = rdd1.map(f1)
    rdd3 = rdd2.filter(f2)
    Can be done in one go — no need to materialize rdd2

"I have good news and bad news"

RDD transformations [full list]

- **map**(*func*): Return a new distributed dataset formed by passing each element of the source through a function *func*
  - {1,2,3}.map(lambda x: x*2) ⟶ {2,4,6}
- **filter**(*func*): Return a new dataset formed by selecting those elements of the source on which *func* returns true
  - {1,2,3}.filter(lambda x: x <= 2) ⟶ {1,2}
- **flatMap**(*func*): Similar to map, but each input item can be mapped to 0 or more output items (so *func* should return a list rather than a single item)
  - {1,2,3}.flatMap(lambda x: [x,x*10]) ⟶ {1,10,2,20,3,30}

RDD transformations [full list]

- **sample**(*withReplacement?*, *fraction*, *seed*): Sample a fraction *fraction* of the data, with or without replacement, using a given random number generator *seed*
- **union**(*otherDataset*): Return a new dataset that contains the union of the elements in the source dataset and the argument.
- **intersection**(*otherDataset*): …
- **distinct**(): Return a new dataset that contains the distinct elements of the source dataset.

RDD transformations [full list]

- **sample**(*withReplacement?*, *fraction*, *seed*): Sample a fraction *fraction* of the data, with or without replacement, using a given random number generator *seed*

Why ***relative fraction,*** and not ***absolute number?***

POLLING TIME

Scan QR code or go to https://web.speakup.info/room/join/66626

RDD transformations [full list]

- **groupByKey**(): When called on a dataset of (K, V) pairs, returns a dataset of (K, Iterable<V>) pairs.
  - {(1,a), (2,b), (1,c)}.groupByKey() ⟶ {(1,[a,c]), (2,[b])}
- **reduceByKey**(*func*): When called on a dataset of (K, V) pairs, returns a dataset of (K, V) pairs where the values for each key are aggregated using the given reduce function *func*, which must be of type (V, V) => V.
  - {(1, 3.1), (2, 2.1), (1, 1.3)}.reduceByKey(lambda (x,y): x+y) ⟶ {(1, 4.4), (2, 2.1)}

RDD transformations [full list]

- **sortByKey**(): When called on a dataset of (K, V) pairs, returns a dataset of (K, V) pairs sorted by keys
- **join**(*otherDataset*): When called on datasets of type (K, V) and (K, W), returns a dataset of (K, (V, W)) pairs with all pairs of elements for each key
  - {(1,a), (2,b)}.join({(1,A), (1,X)}) → {(1, (a,A)), (1, (a,X))}
- Analogous: **leftOuterJoin**, **rightOuterJoin**, **fullOuterJoin**
- (There are several other RDD transformations, and some of the above have additional arguments; cf. tutorial)

RDD actions [full list]

- **collect**(): Return all the elements of the dataset as an array at the driver program. This is usually useful after a filter or other operation that returns a sufficiently small subset of the data.
- **count**(): Return the number of elements in the dataset.
- **take**(*n*): Return an array with the “first” *n* elements of the dataset.
- **saveAsTextFile**(*path*): Write the elements of the dataset as a text file in a given directory in the local filesystem or HDFS.
- (There are several other RDD actions; cf. tutorial)

Broadcast variables

- my_set = set(range(1e80))
  rdd2 = rdd1.filter(lambda x: x in my_set)
  ^ This is a bad idea: my_set needs to be shipped with every task (one task per data partition, so if rdd1 is spread over *N* partitions, the above will require copying the same object *N* times)
- Better:
  my_set = sc.broadcast(set(range(1e80)))
  rdd2 = rdd1.filter(lambda x: x in my_set.value)
  ^ This way, my_set is copied to each executor only once and persists across all tasks (one per partition) on the same executor
- Broadcast variables are **read-only**

Accumulators

- def f(x): return x*2
  rdd2 = rdd1.map(f)
  ^ How can we easily know how many rows there are in rdd1 (without running a costly reduce operation)?
- Side effects via accumulators!
  counter = sc.accumulator(0)
  def f(x): counter.add(1); return x*2
  rdd2 = rdd1.map(f)
- Accumulators are **write-only** (“add-only”) for executors
- Only driver can read the value: counter.value

RDD persistence

```
rdd2 = rdd1.map(f1)
list1 = rdd2.filter(f2).collect()
list2 = rdd2.filter(f3).collect()
```

rdd1.map(f1) transformation is executed twice

```
rdd2 = rdd1.map(f1)
rdd2.persist()
list1 = rdd2.filter(f2).collect()
list2 = rdd2.filter(f3).collect()
```

Result of rdd1.map(f1) transformation is cached and reused (can choose between memory and disk for caching)

Spark DataFrames

- Bridging the gap between your experience with Pandas and the need for distributed computing
  - RDD = list of rows
  - DataFrame = table with rows and typed columns
- Important to understand what RDDs are and what they offer, but today most of the tasks can be accomplished with DataFrames **(higher level of abstraction ⇒ less code)**
- https://www.databricks.com/spark/getting-started-with-apache-spark/dataframes

Spark SQL [link]

```
sc = SparkContext()

sqlContext = HiveContext(sc)

df = sqlContext.sql("SELECT * from table1 GROUP BY id")
```

Spark's Machine Learning Toolkit

MLlib: Algorithms [more details]

Classification

- Logistic regression, decision trees, random forests

Regression

- Linear (with L1 or L2 regularization)

Unsupervised:

- Alternating least squares
- K-means
- SVD
- Topic modeling (LDA)

Optimizers

- Optimization primitives (SGD, L-BGFS)

Example:
Logistic regression with MLLib

```
from pyspark.mllib.classification \
   import LogisticRegressionWithSGD

trainData = sc.textFile("...").map(...)
testData = sc.textFile("...").map(...)
model = LogisticRegressionWithSGD.train(trainData)
predictions = model.predict(testData)
```

Remarks

- This lecture is not enough to teach you Spark!
- To use it in practice, you'll need to delve into further online material
- Also: Friday's lab session
- You can't learn it without some frustration :(
- Important skill: assess whether you'd benefit from Spark
  – E.g., >1TB: yes, you'll need Spark
  – 20GB: it depends…

Feedback

Give us feedback on this lecture here:
https://go.epfl.ch/ada2023-lec13-feedback

- What did you (not) like about this lecture?
- What was (not) well explained?
- On what would you like more (fewer) details?
- Where is Waldo?
- …

Cluster etiquette

- Develop and debug locally
  - Install Spark locally on your personal computer
  - Use a small subset of the data
- When ready, launch your script on the cluster using spark-submit
- **Never (never!) use the Spark shell (a.k.a. pyspark) -- it's hereby officially forbidden**
- Useful trench report from a dlab member: "What I learned from processing big data with Spark"