Big Data Analysis with Apache Spark

Matei Zaharia

Outline

The big data problem

MapReduce

Apache Spark

How people are using it

The Big Data Problem

Data is growing faster than computing power

Growing data sources
- » Mostly machine generated

Cheap storage

Stalling CPU speeds

Examples

Facebook's daily logs: 60 TB

1000 genomes project: 200 TB

Google web index: 10+ PB

Cost of 1 TB of disk: $25

Time to read 1 TB from disk: 6 hours (50 MB/s)

The Big Data Problem

Single machine can no longer process or even store all the data!

Only solution is to **distribute** over large clusters


Google Datacenter
How do we program this thing?

Traditional Network Programming

Message-passing between nodes

Really hard to do at scale:

- How to split problem across nodes?
- How to deal with failures?
- Even worse: stragglers (node is not failed, but slow)

Data-Parallel Models

Restrict the programming interface so that the system can do more automatically

"Here's an operation, run it on all of the data"

- » I don't care *where* it runs (you schedule that)
- » In fact, feel free to run it *twice* on different nodes

Biggest example: MapReduce

MapReduce

First widely popular programming model for data-intensive apps on clusters

Published by Google in 2004

» Processes 20 PB of data / day

Popularized by open-source Hadoop project

MapReduce Programming Model

Data type: key-value records

Map function:

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

Reduce function:

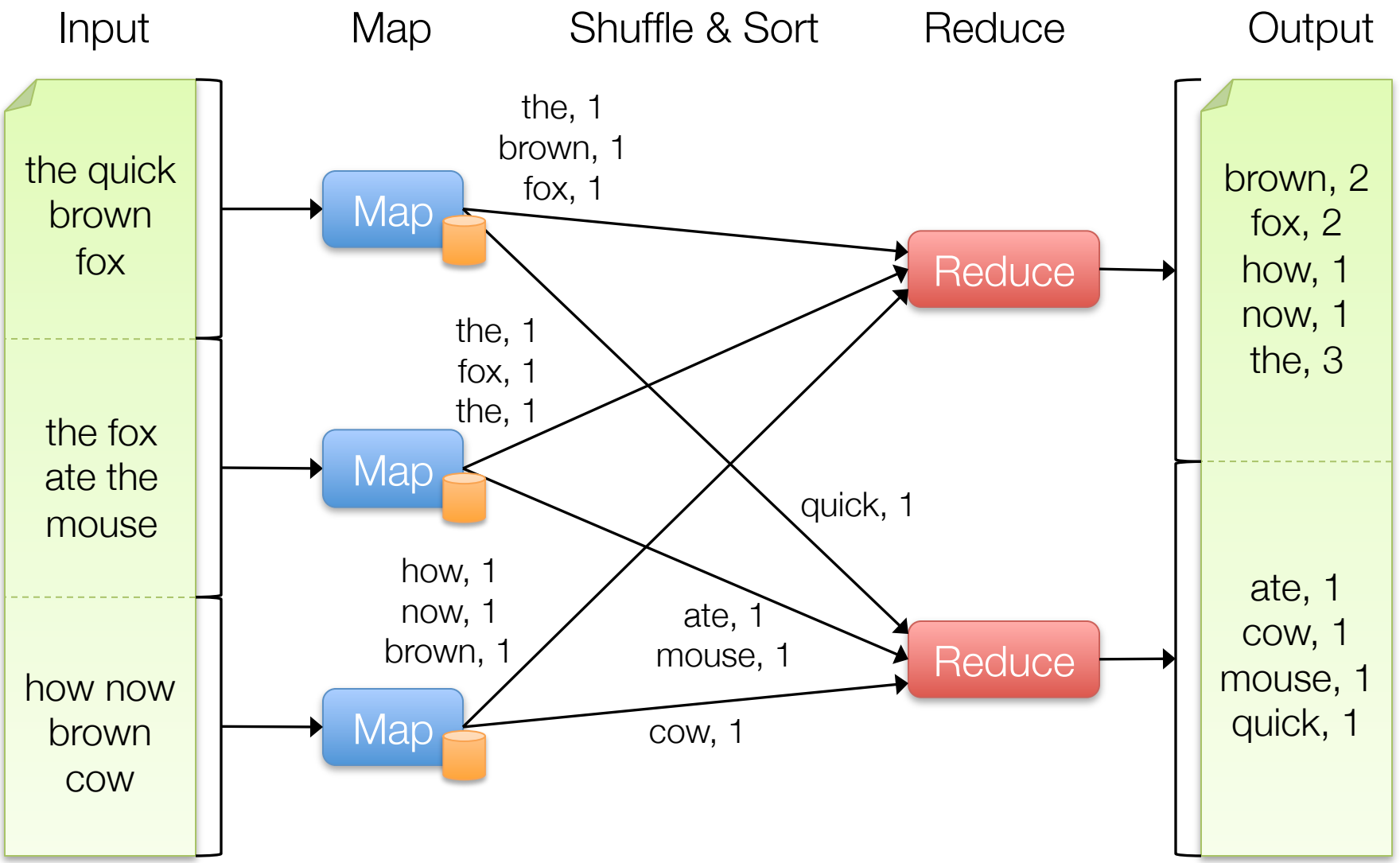
$$(K_{inter}, \text{list}(V_{inter})) \rightarrow \text{list}(K_{out}, V_{out})$$

Example: Word Count

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


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

Word Count Execution



MapReduce Execution

Automatically split work into many small *tasks*

Send tasks to nodes based on data locality

Automatically recover from failures

Summary

Data-parallel programming models let systems automatically manage much of execution:

- » Assigning work, load balancing, fault recovery

But… the story doesn’t end here!

Outline

The big data problem

MapReduce

Apache Spark

How people are using it

Limitations of MapReduce

Programmability: most applications require higher level functions than map / reduce

- » E.g. statistics, matrix multiply, graph search
- » Google ads pipeline had 20 MR steps

Performance: inefficient to combine multiple MapReduce steps into complex programs

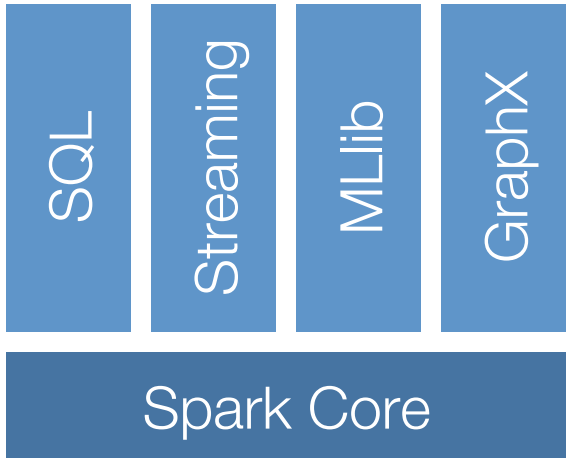
Apache Spark

Programming model that generalizes MapReduce to support more applications

» Adds efficient, in-memory **data sharing**

Large library of built-in functions

APIs in Python, Java, Scala, R



Spark Programmability

WordCount in MapReduce:

```
#include "mapreduce/mapreduce.h"

// User's map function
class SplitWords: public Mapper {
  public:
  virtual void Map(const MapInput& input)
  {
    const string& text = input.value();
    const int n = text.size();
    for (int i = 0; i < n; ) {
      // Skip past leading whitespace
      while (i < n && isspace(text[i]))
        i++;
      // Find word end
      int start = i;
      while (i < n && !isspace(text[i]))
        i++;
      if (start < i)
        Emit(text.substr(
            start,i-start),"1");
    }
  }
};

REGISTER_MAPPER(SplitWords);
```

```
// User's reduce function
class Sum: public Reducer {
  public:
  virtual void Reduce(ReduceInput* input)
  {
    // Iterate over all entries with the
    // same key and add the values
    int64 value = 0;
    while (!input->done()) {
      value += StringToInt(
                 input->value());
      input->NextValue();
    }
    // Emit sum for input->key()
    Emit(IntToString(value));
  }
};

REGISTER_REDUCER(Sum);
```

```
int main(int argc, char** argv) {
  ParseCommandLineFlags(argc, argv);
  MapReduceSpecification spec;
  for (int i = 1; i < argc; i++) {
    MapReduceInput* in= spec.add_input();
    in->set_format("text");
    in->set_filepattern(argv[i]);
    in->set_mapper_class("SplitWords");
  }

  // Specify the output files
  MapReduceOutput* out = spec.output();
  out->set_filebase("/gfs/test/freq");
  out->set_num_tasks(100);
  out->set_format("text");
  out->set_reducer_class("Sum");

  // Do partial sums within map
  out->set_combiner_class("Sum");

  // Tuning parameters
  spec.set_machines(2000);
  spec.set_map_megabytes(100);
  spec.set_reduce_megabytes(100);

  // Now run it
  MapReduceResult result;
  if (!MapReduce(spec, &result)) abort();
  return 0;
}
```

Spark Programmability

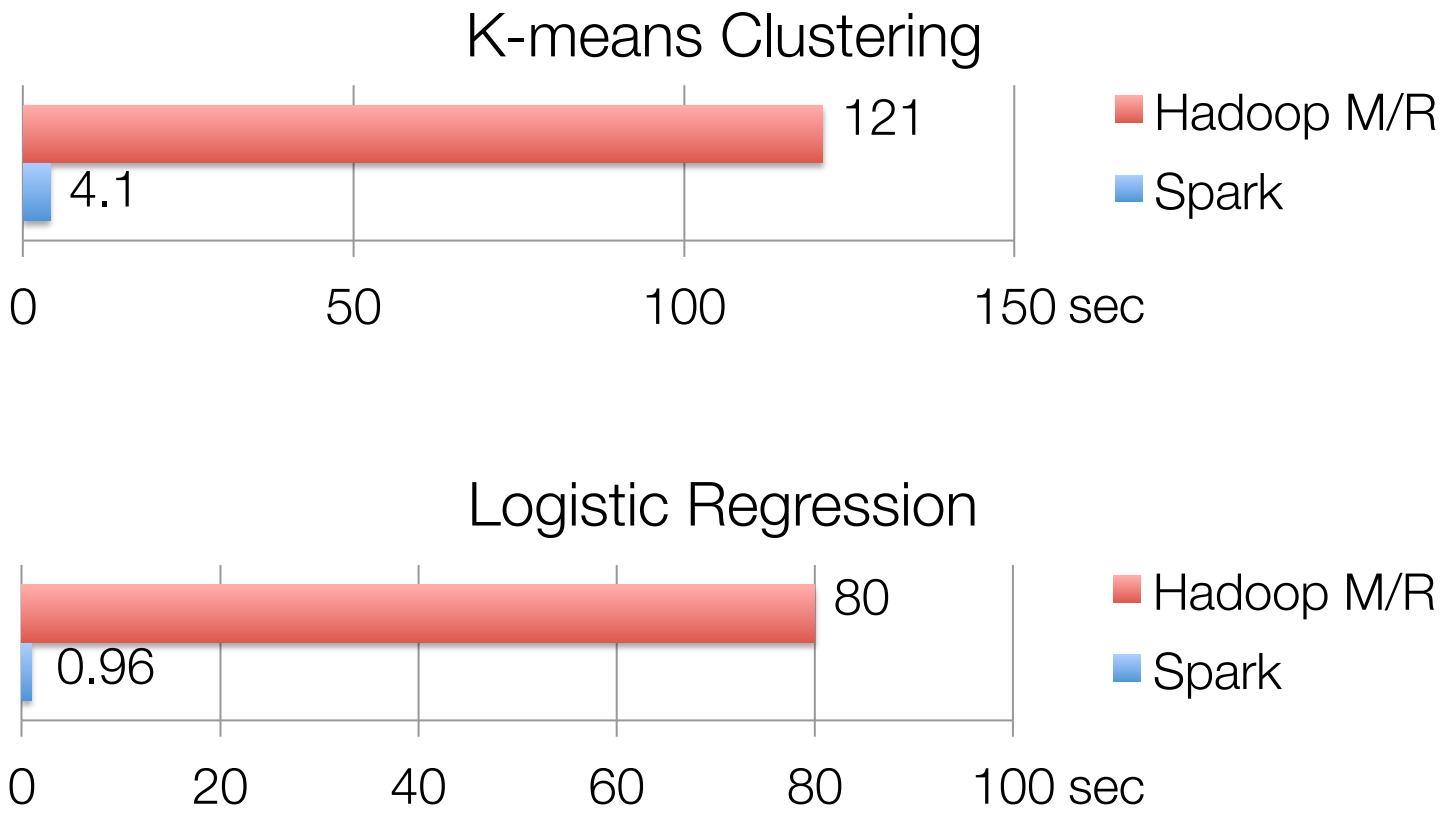
WordCount in Spark:

```
file = spark.textFile("hdfs://...")

counts = file.flatMap(lambda line: line.split(" "))
              .map(lambda word: (word, 1))
              .reduceByKey(lambda a, b: a+b)

counts.save("out.txt")
```

Spark Performance



Programming Model

Write programs in terms of transformations on distributed datasets

Resilient Distributed Datasets (RDDs)

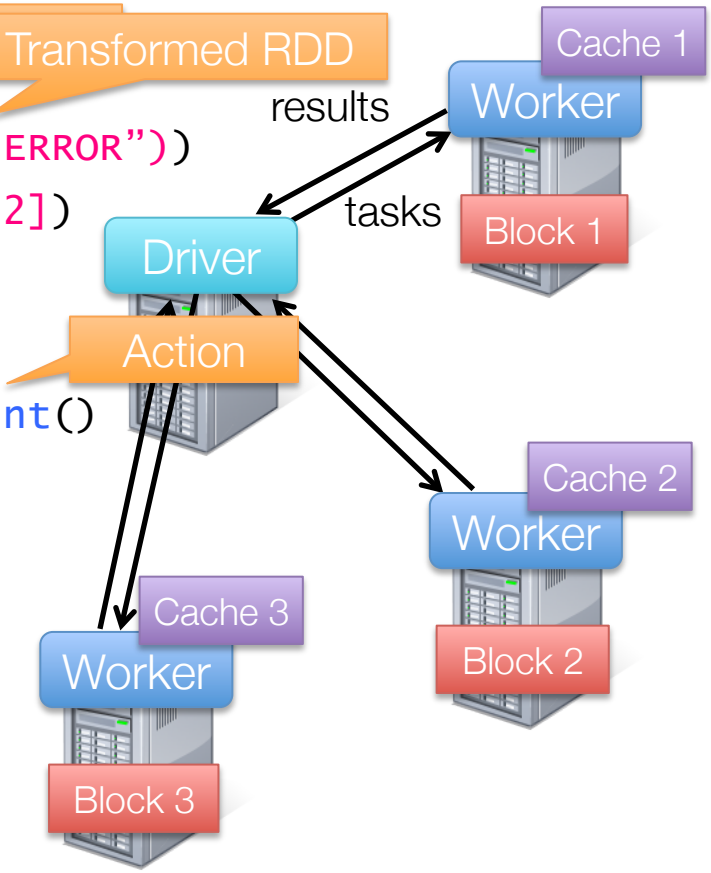
- » Collections of objects that can be stored in memory or disk across a cluster
- » Built via parallel transformations (map, filter, …)
- » Automatically rebuilt on failure

Example: Text Search

Load a large log file into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(lambda s: s.startswith("ERROR"))
messages = errors.map(lambda s: s.split('\t')[2])
messages.cache()

messages.filter(lambda s: "Illumina" in s).count()
messages.filter(lambda s: "Dell" in s).count()
. . .
```

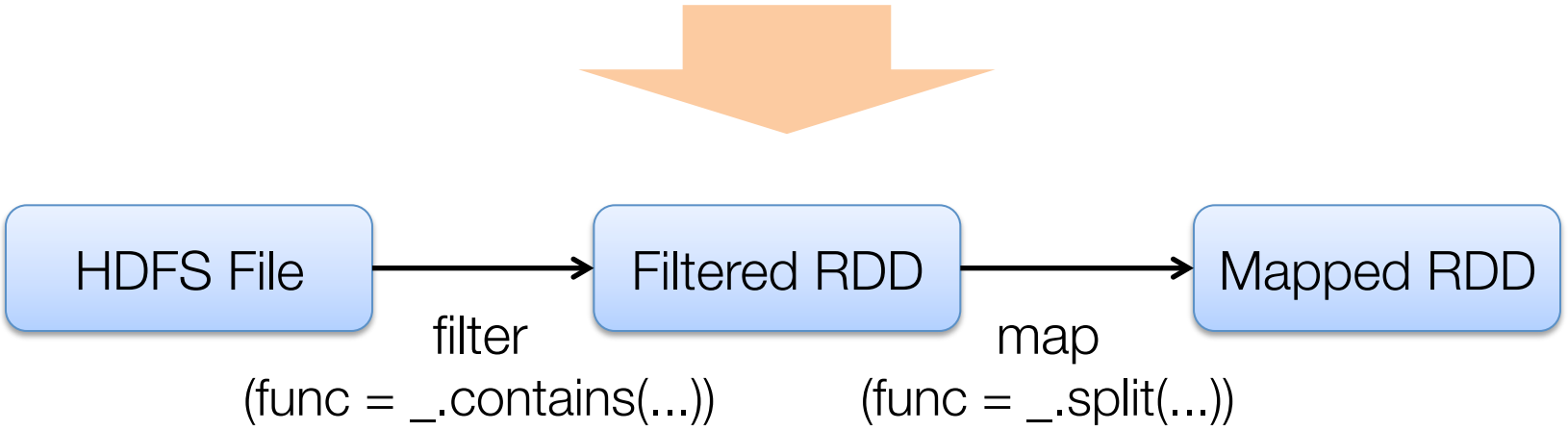


Result: scaled to 1 TB data in 7 sec
(vs 180 sec for on-disk data)

Fault Recovery

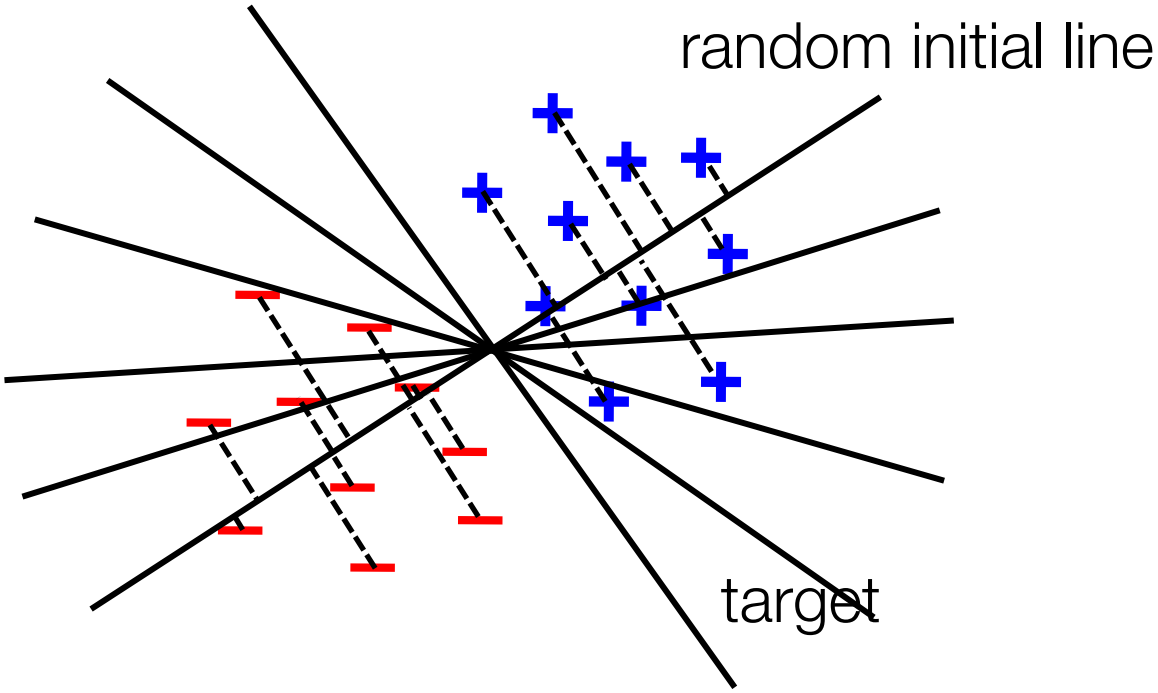
RDDs track *lineage* information that can be used to efficiently reconstruct lost partitions

Ex:
```
msgs = textFile.filter(lambda s: s.startsWith("ERROR"))
               .map(lambda s: s.split("\t")[2])
```

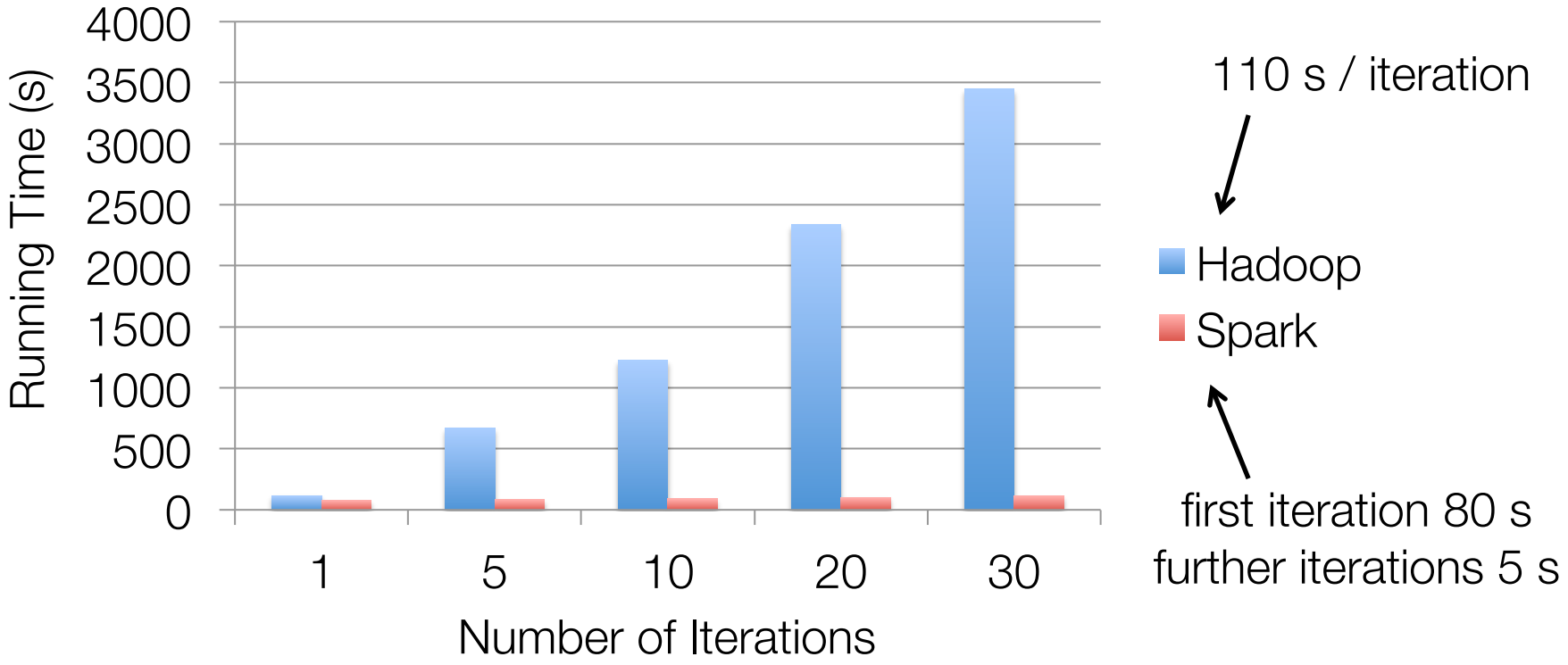


Example: Logistic Regression

Goal: find line separating two sets of points



Logistic Regression Performance



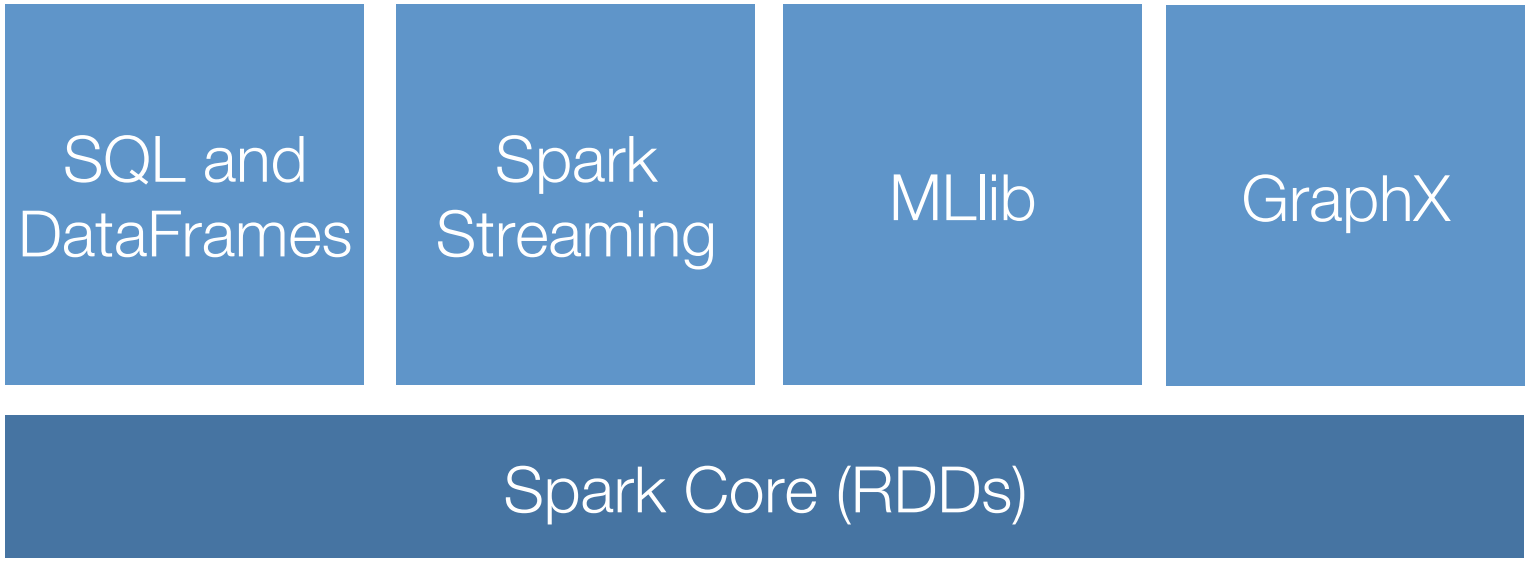
Supported Operators

```
map
filter
groupBy
union
join
leftOuterJoin
rightOuterJoin
```

```
reduce
count
fold
reduceByKey
groupByKey
cogroup
flatMap
```

```
take
first
partitionBy
pipe
distinct
save
...
```

Built-in Libraries



Largest integrated standard library for big data

Combining Libraries

```
# Load data using SQL
ctx.jsonFile(“tweets.json”).registerTempTable(“tweets”)
points = ctx.sql(“select latitude, longitude from tweets”)


# Train a machine learning model
model = KMeans.train(points, 10)


# Apply it to a stream
sc.twitterStream(...)
  .map(lambda t: (model.predict(t.location), 1))
  .reduceByWindow(“5s”, lambda a, b: a+b)
```

Summary

Libraries + function-based interface let users write parallel programs similar to sequential code

Can use Spark interactively in Python, R, etc

Outline

The big data problem

MapReduce

Apache Spark

How people are using it

Spark Community

1000+ deployments, clusters up to 8000 nodes

Applications

Large-scale machine learning

Analysis of neuroscience data

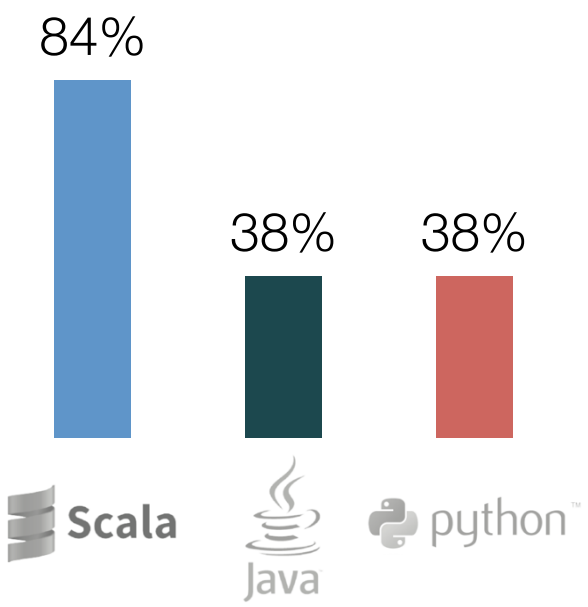
Network security

SQL and data clustering

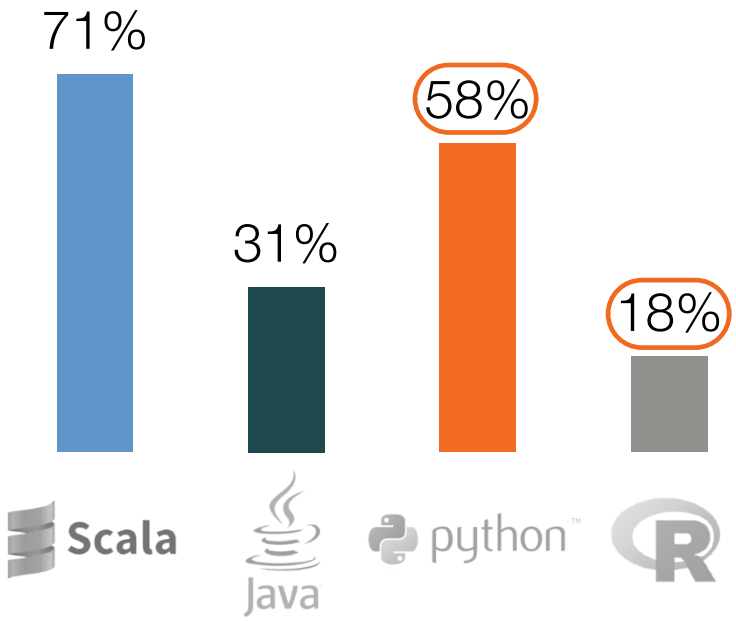
Trends & recommendations

Programming Languages

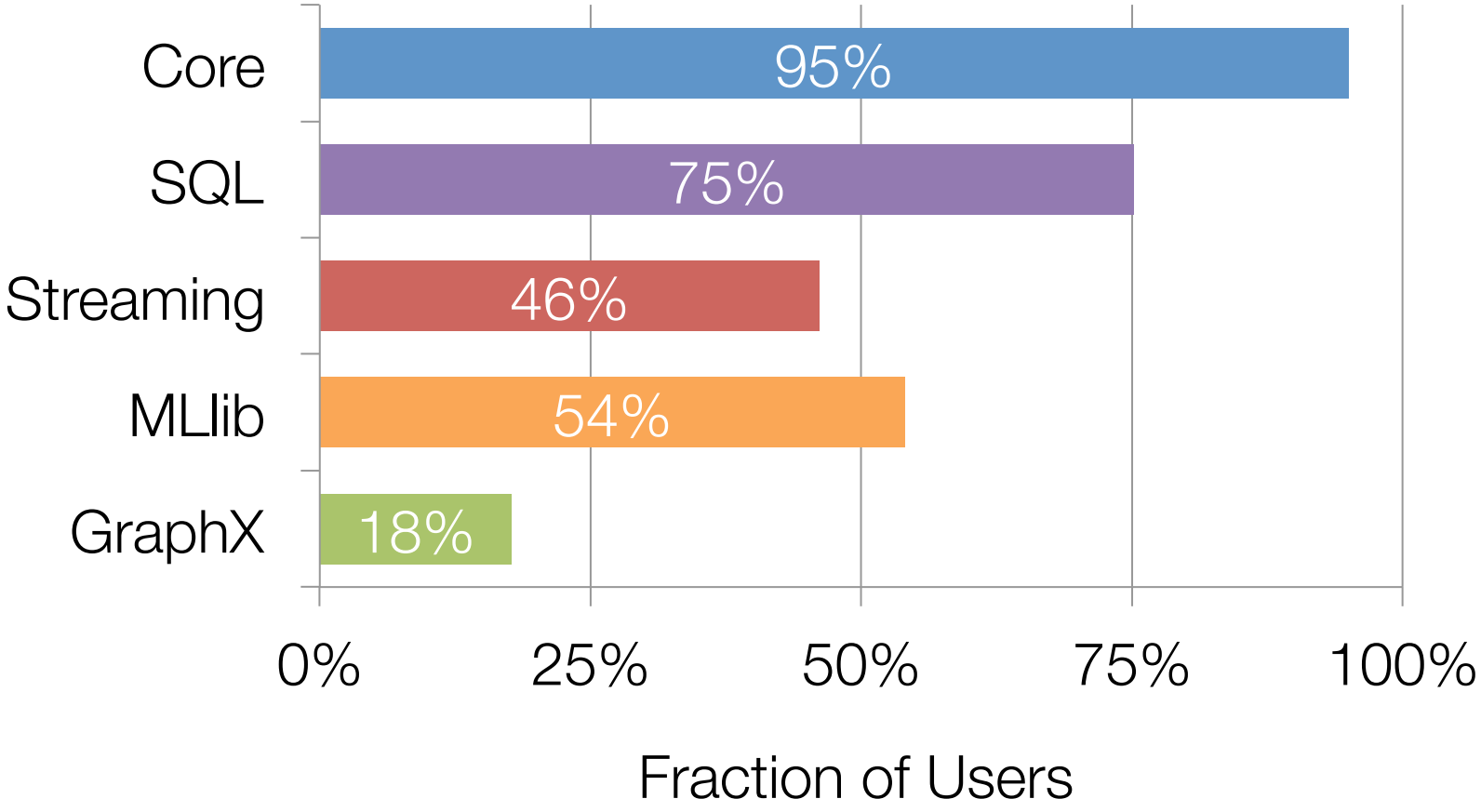
2014 Languages Used



2015 Languages Used

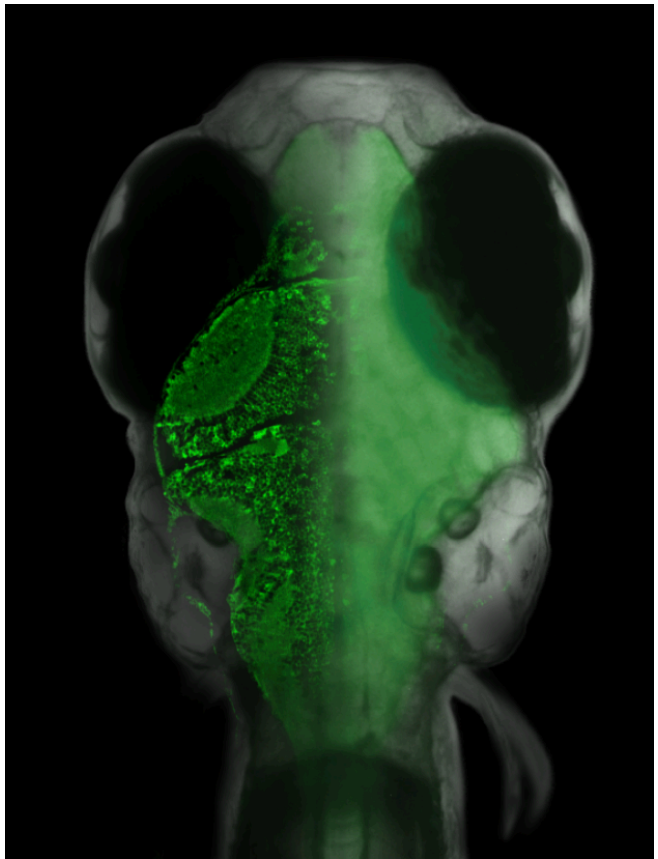


Libraries Used



Example: Neuroscience

HHMI Janelia Farm analyzes data from full-brain imaging of neural activity



Larval zebrafish

+

Light-sheet imaging

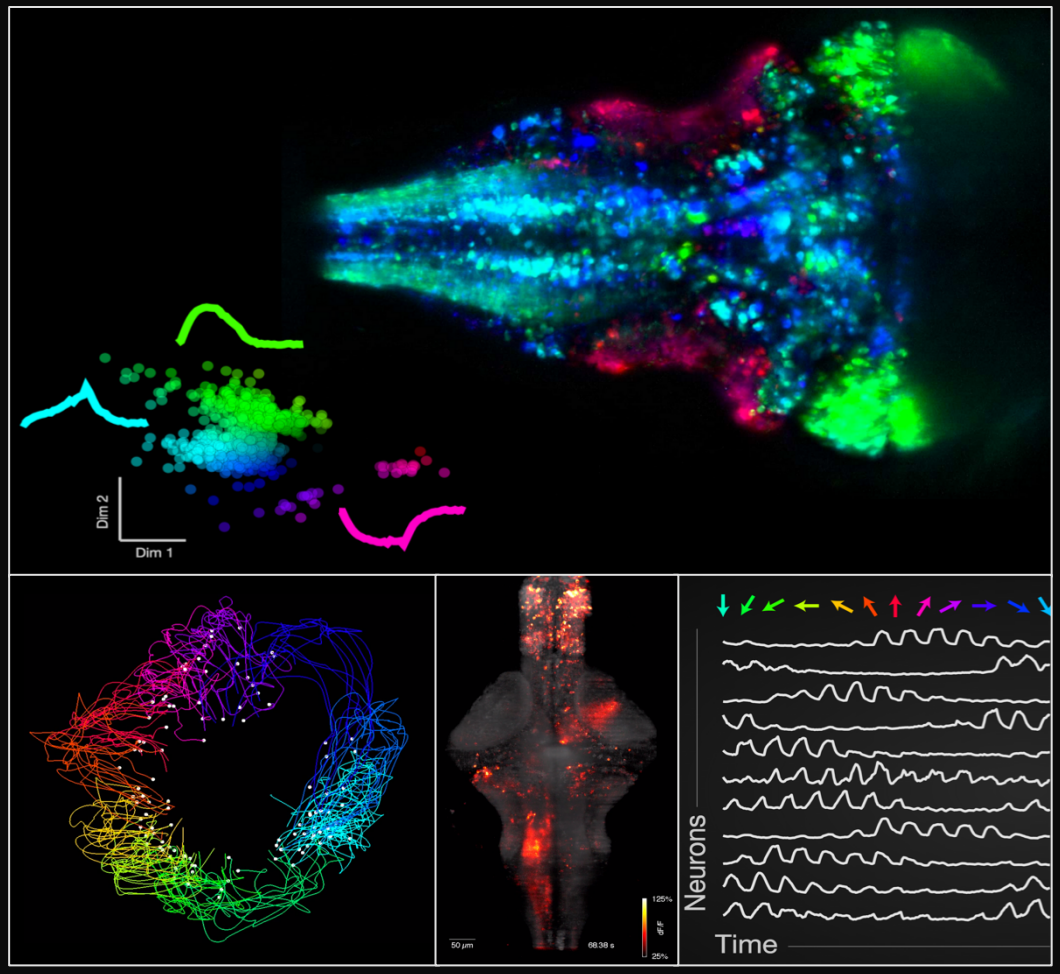
=

2 TB / hour of data

Images from Jeremy Freeman

Data Analysis

Streaming code does clustering, dimens. reduction on 80-node cluster



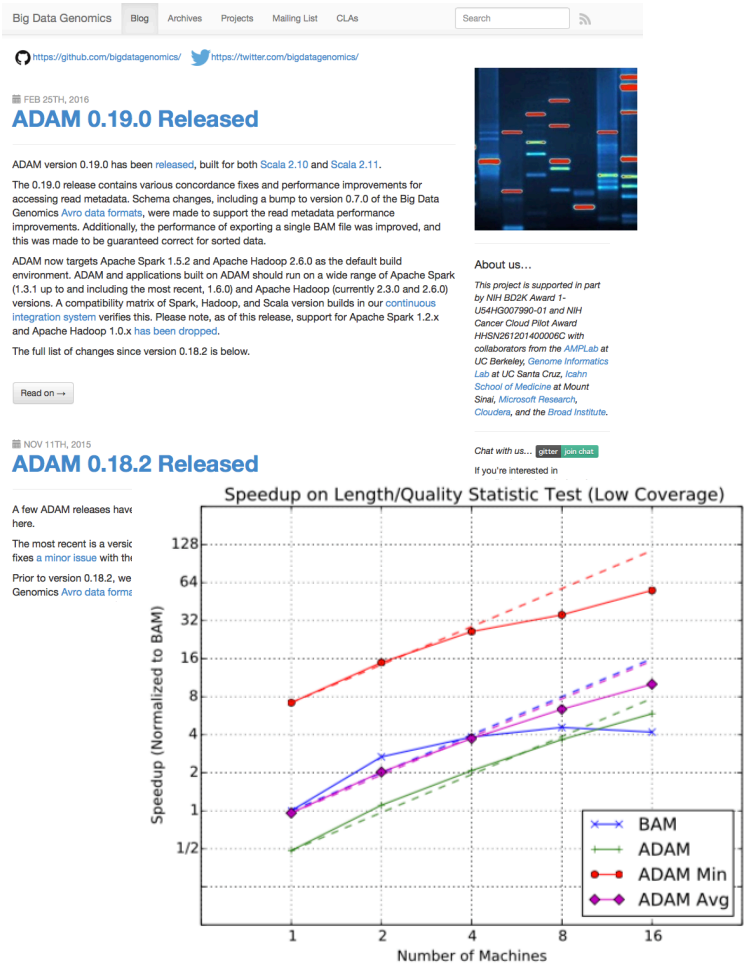
Images from Jeremy Freeman

Example: Berkeley ADAM

Stores and processes reads with standard big data tools and formats

25% smaller than BAM(!), linear scale-up

bdgenomics.org



GATK4/Spark

https://github.com/broadinstitute/gatk

GATK4 is a Spark-native application for genetic sequencing analyses. Currently in alpha testing, with good scalability to hundreds of cores, e.g.:

1. MarkDuplicates – took hours to run and was single-core only, now GATK4 runs in 3 minutes (on 30 GB exome)

2. Depth of coverage – took days to run on 200GB whole genome, now GATK4 runs in 4 minutes

3. Whole genome metrics (e.g., insert-size distribution) runs in 2-3 minutes on 300GB whole genome

Open-source, modular, scalable platform for statistical genetics in development by the Neale lab at Broad

Combine genetic and phenotypic data to uncover the biology of disease

Built using Scala and Spark, currently in alpha

Wall time of QC pipeline down from weeks to minutes

Conclusion

Apache Spark offers a fast, high-level interface to work with big data based on data-parallel model

Large set of existing libraries

Easy to try on just your laptop!
spark.apache.org

To Learn More

Free MOOCs on edX

edx.org

Use case videos at Spark Summit

spark-summit.org