Lessons from Running Large Scale Spark Workloads

Reynold Xin, Matei Zaharia

Feb 19, 2015 @ Strata

databricks™

About Databricks

Founded by the creators of Spark in 2013

Largest organization contributing to Spark

End-to-end hosted service, Databricks Cloud

A slide from 2013 …

Spark

Fast and expressive cluster computing system interoperable with Apache Hadoop

Improves efficiency through:

» In-memory computing primitives
» General computation graphs

→ Up to 100× faster (2-10× on disk)

Improves usability through:

» Rich APIs in Scala, Java, Python
» Interactive shell

→ Often 5× less code

Does Spark scale?

Does Spark scale? Yes!

On-Disk Sort Record:

Time to sort 100TB

2013 Record: Hadoop

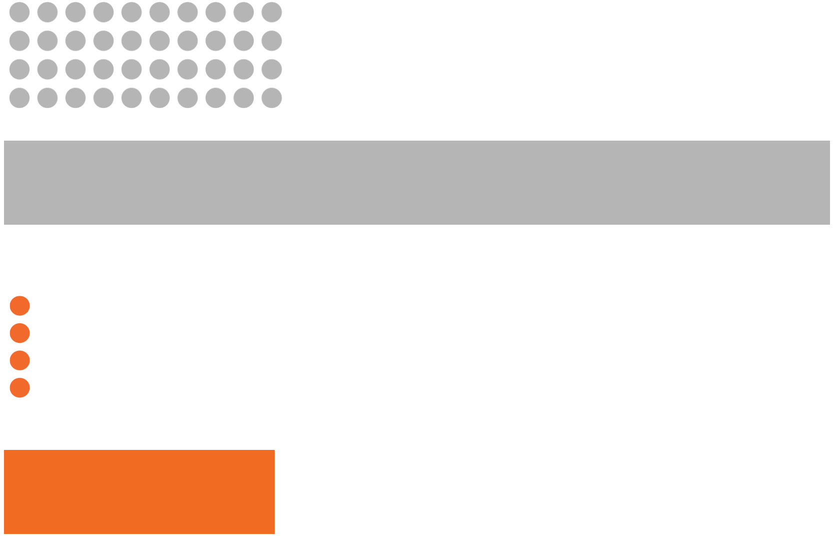
2100 machines

72 minutes

2014 Record: Spark

207 machines

23 minutes



Also sorted 1PB in 4 hours

Source: Daytona GraySort benchmark, sortbenchmark.org

Agenda

Spark “hall of fame”

Architectural improvements for scalability

Q&A

Spark "Hall of Fame"

★

LARGEST CLUSTER

Tencent

(8000+ nodes)

★

LARGEST SINGLE-DAY INTAKE

Tencent

(1PB+ /day)

★

LONGEST-RUNNING JOB

Alibaba

(1 week on 1PB+ data)

★

LARGEST SHUFFLE

Databricks PB Sort

(1PB)

★

MOST INTERESTING APP

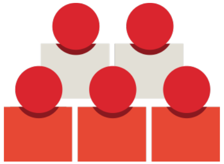
Jeremy Freeman

Mapping the Brain at Scale

(with lasers!)

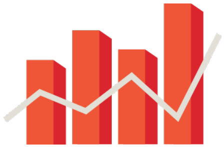
Largest Cluster & Daily Intake

800 million+
active users



8000+
nodes

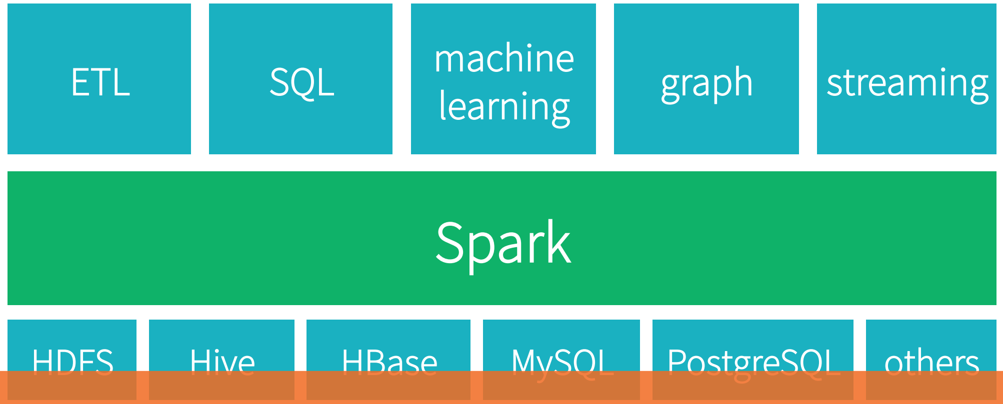
150 PB+
1 PB+/day



Tencent 腾讯 数据平台部

Spark at Tencent

Ads CTR prediction
Similarity metrics on billions of nodes
Spark SQL for BI and ETL
. . .



See Lianhui's session this afternoon

Images courtesy of Lianhui Wang

Longest Running Spark Job

Alibaba Taobao uses Spark to perform image feature extraction for product recommendation.

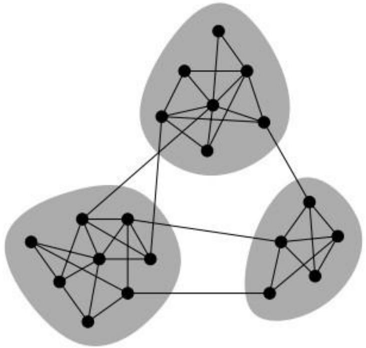
1 week runtime on petabytes of images!

Largest e-commerce site (800m products, 48k sold/min)

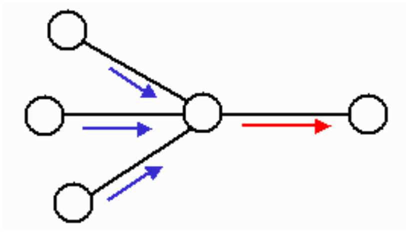
ETL, machine learning, graph computation, streaming

Alibaba Taobao: Extensive Use of GraphX

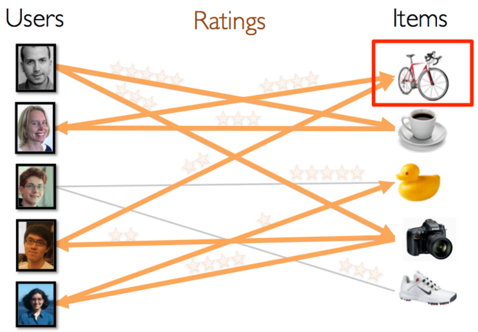
clustering
(community detection)



belief propagation
(influence & credibility)



collaborative filtering
(recommendation)

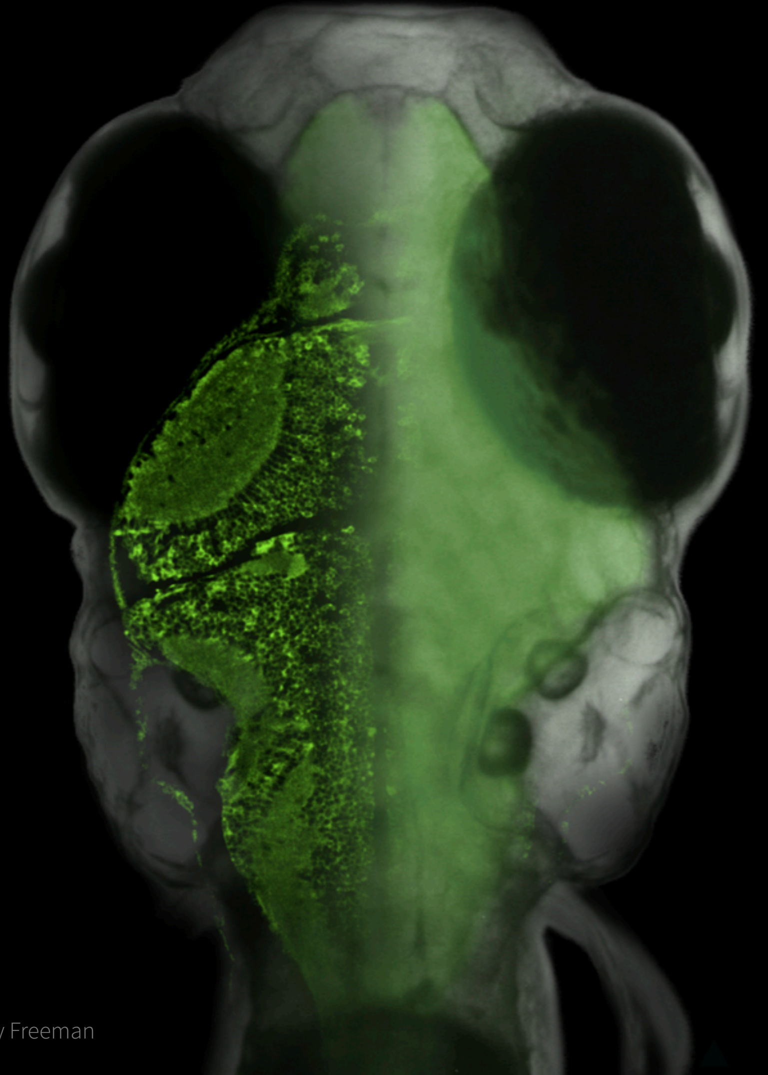


See upcoming talk at [Spark Summit] on streaming graphs

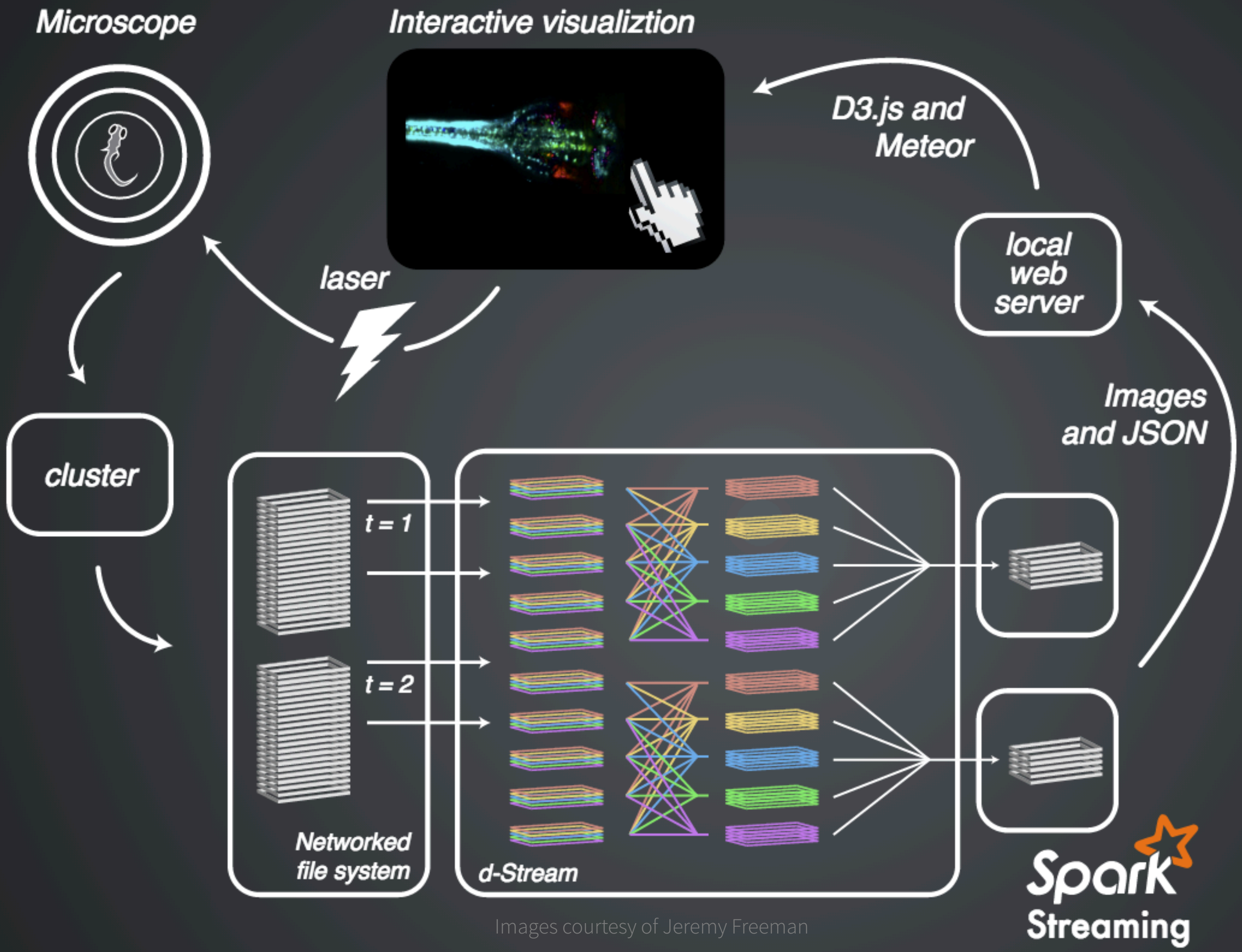
databricks

Images courtesy of Andy Huang, Joseph Gonzales

Mapping the brain at scale

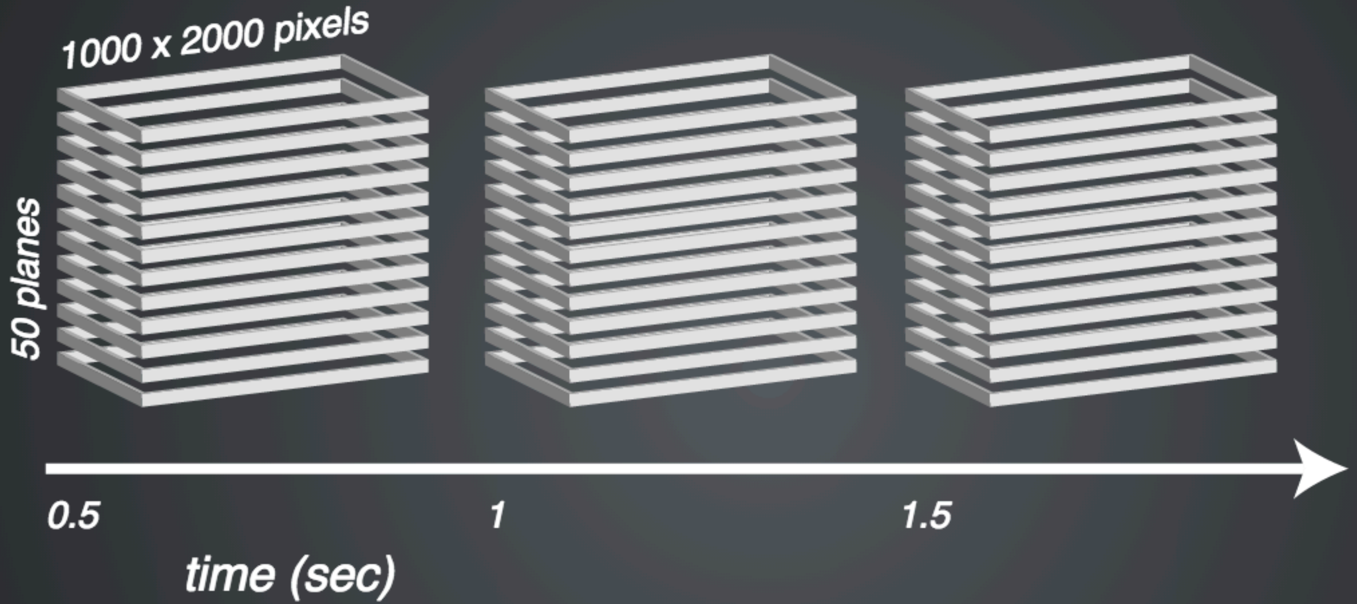


Images courtesy of Jeremy Freeman



Images courtesy of Jeremy Freeman

THESE DATA ARE GETTING BIG, FAST



Ahrens et al., 2013

Images courtesy of Jeremy Freeman

30 min = 1TB

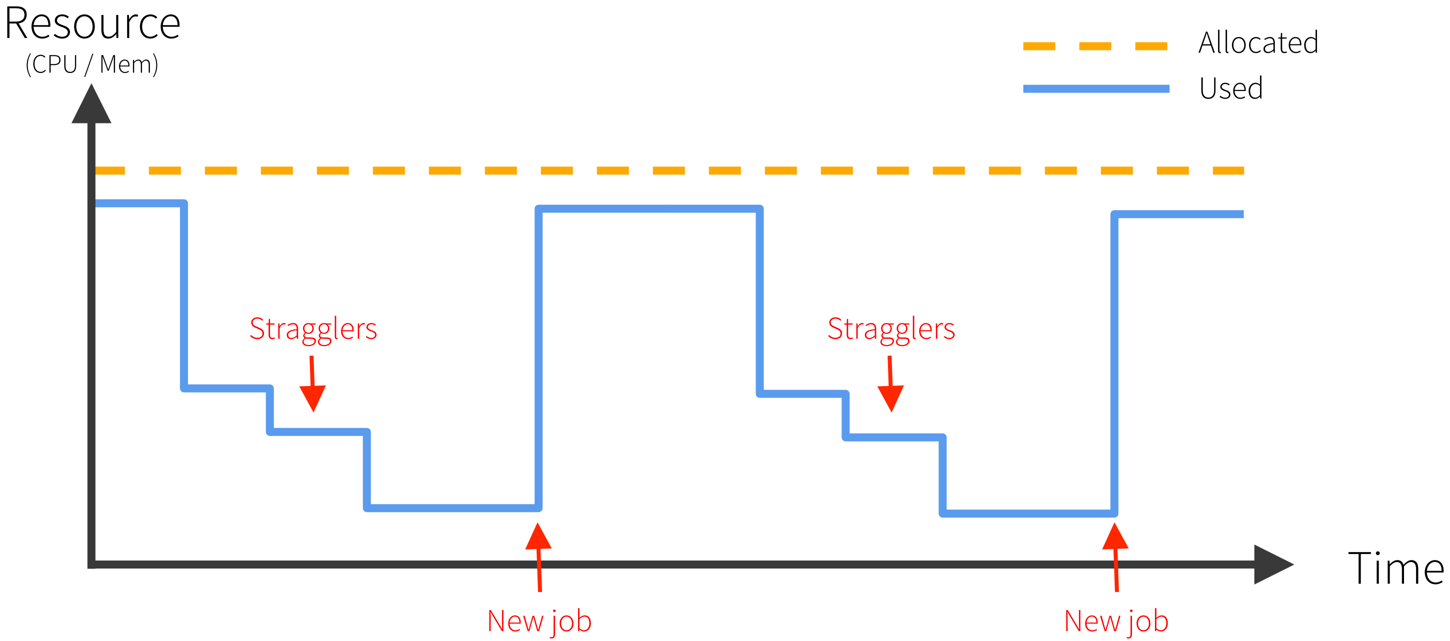
Architectural Improvements

Elastic Scaling: improve resource utilization (1.2)

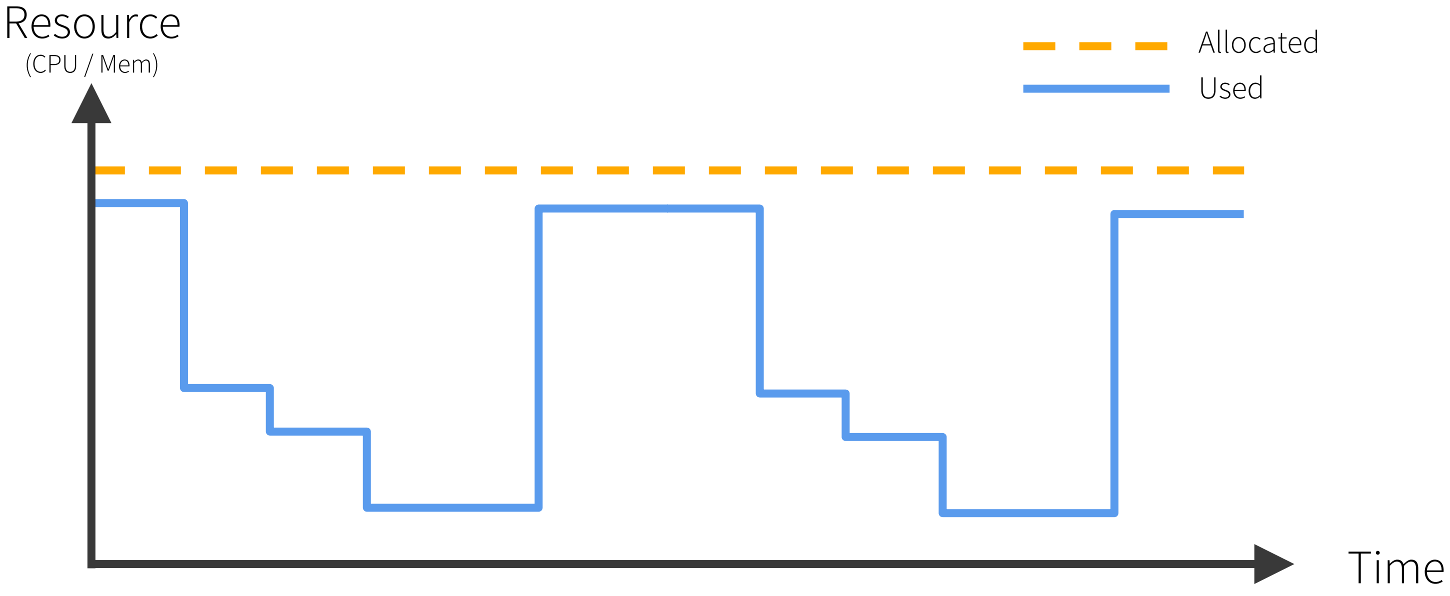
Revamped Shuffle: shuffle at scale (1.2)

DataFrames: easier high performance at scale (1.3)

Static Resource Allocation

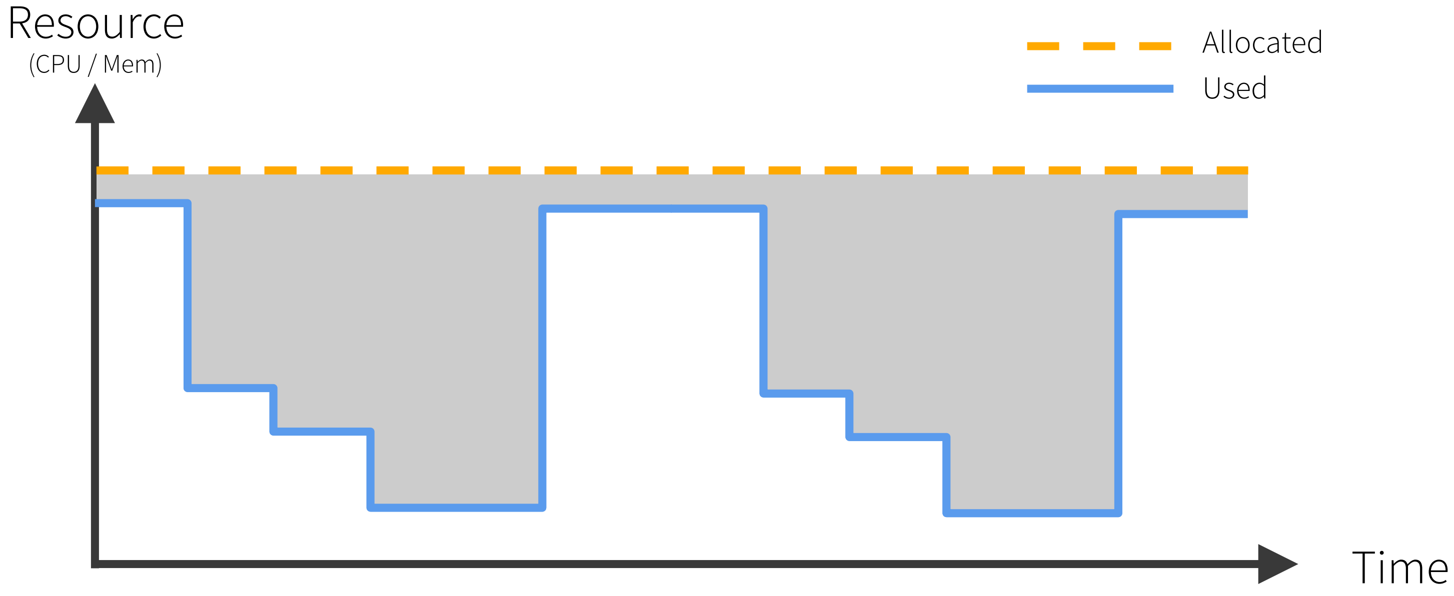


Static Resource Allocation



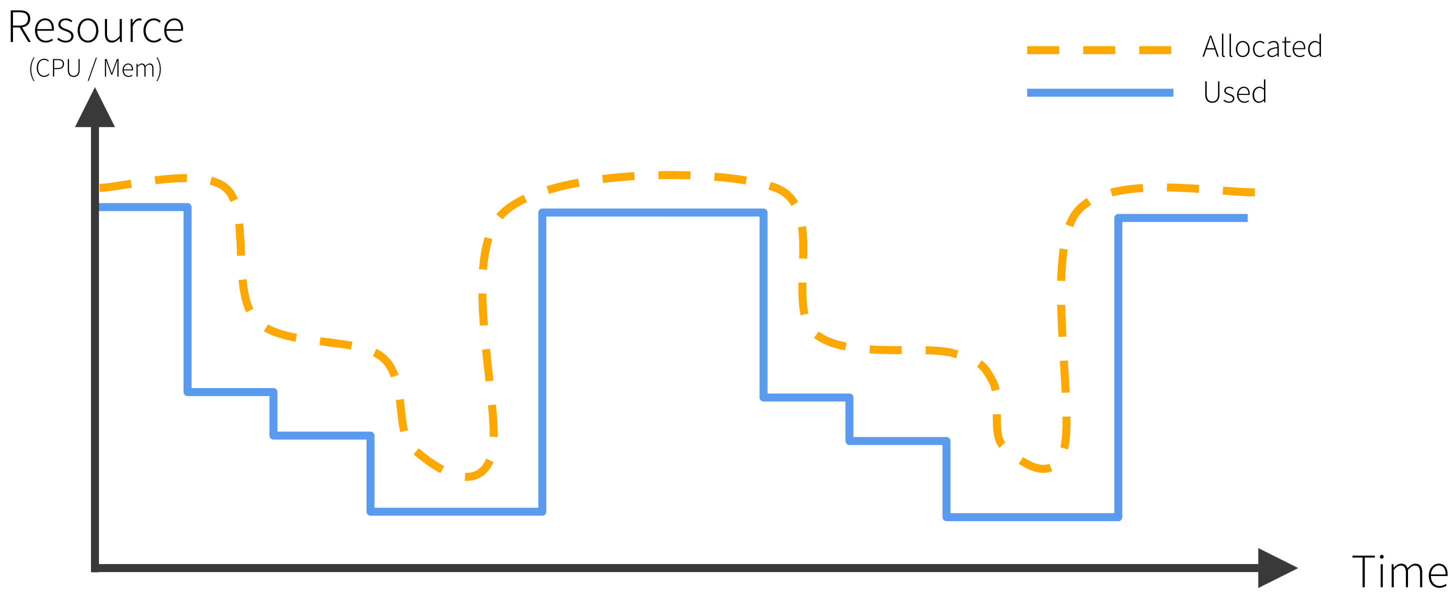
More resources allocated than are used!

Static Resource Allocation



Resources wasted!

Elastic Scaling



Elastic Scaling

spark.dynamicAllocation.enabled	true
spark.dynamicAllocation.minExecutors	3
spark.dynamicAllocation.maxExecutors	20

- Optional -

spark.dynamicAllocation.executorIdleTimeout

spark.dynamicAllocation.schedulerBacklogTimeout

spark.dynamicAllocation.sustainedSchedulerBacklogTimeout

Shuffle at Scale

Sort-based shuffle

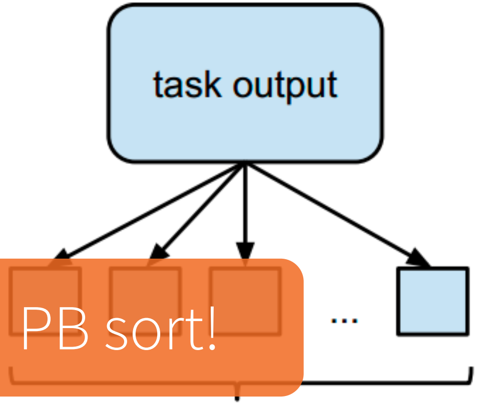
Netty-based transport

Sort-based Shuffle

Old hash-based shuffle: requires R (# reduce tasks) concurrent streams with buffers; limits R.

New sort-based shuffle: sort records by partition first, and then write them; one active stream at a time

Went up to 250,000 reduce tasks in PB sort!



R concurrent file outputs

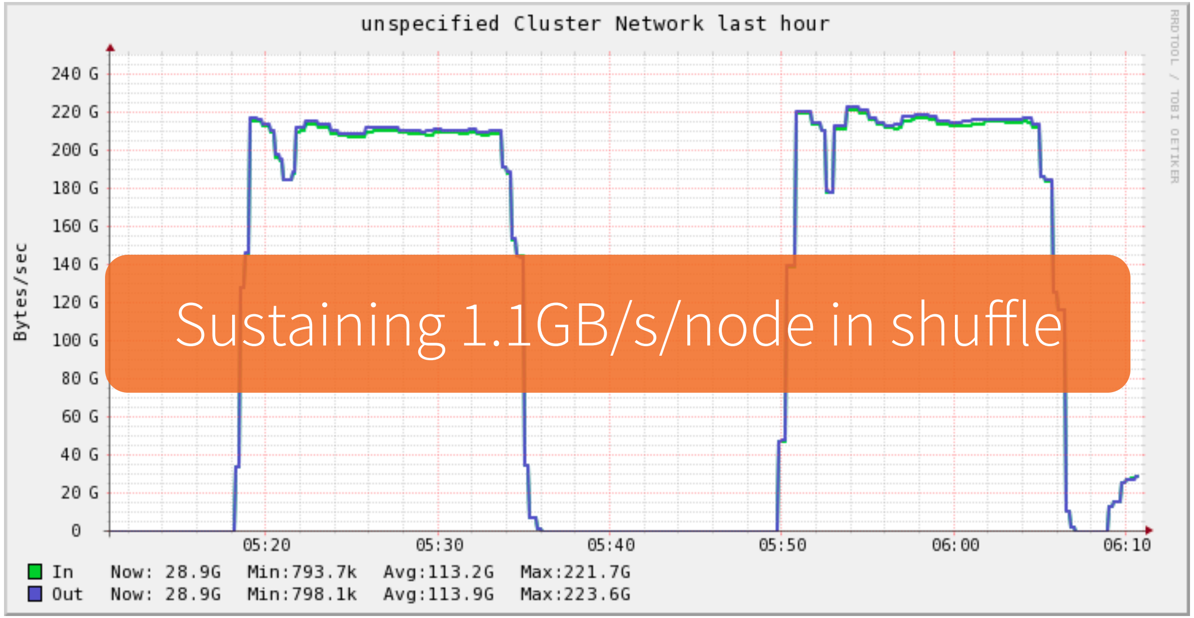
Netty-based Network Transport

Zero-copy network send

Explicitly managed memory buffers for shuffle (lowering GC pressure)

Auto retries on transient network failures

Network Transport



databricks

DataFrames in Spark

Current Spark API is based on Java / Python objects

– Hard for engine to store compactly
– Cannot understand semantics of user functions

DataFrames make it easy to leverage structured data

– Compact, column-oriented storage
– Rich optimization via Spark SQL's Catalyst optimizer

DataFrames in Spark

Distributed collection of data with known schema

Domain-specific functions for common tasks

- Metadata
- Sampling
- Project, filter, aggregation, join, …
- UDFs

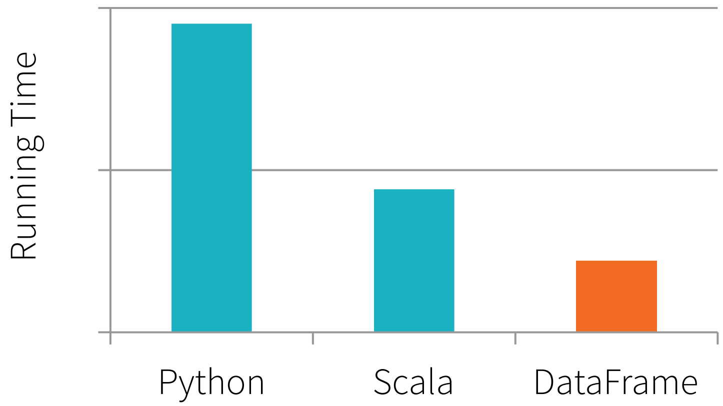
DataFrames

Similar API to data frames in R and Pandas

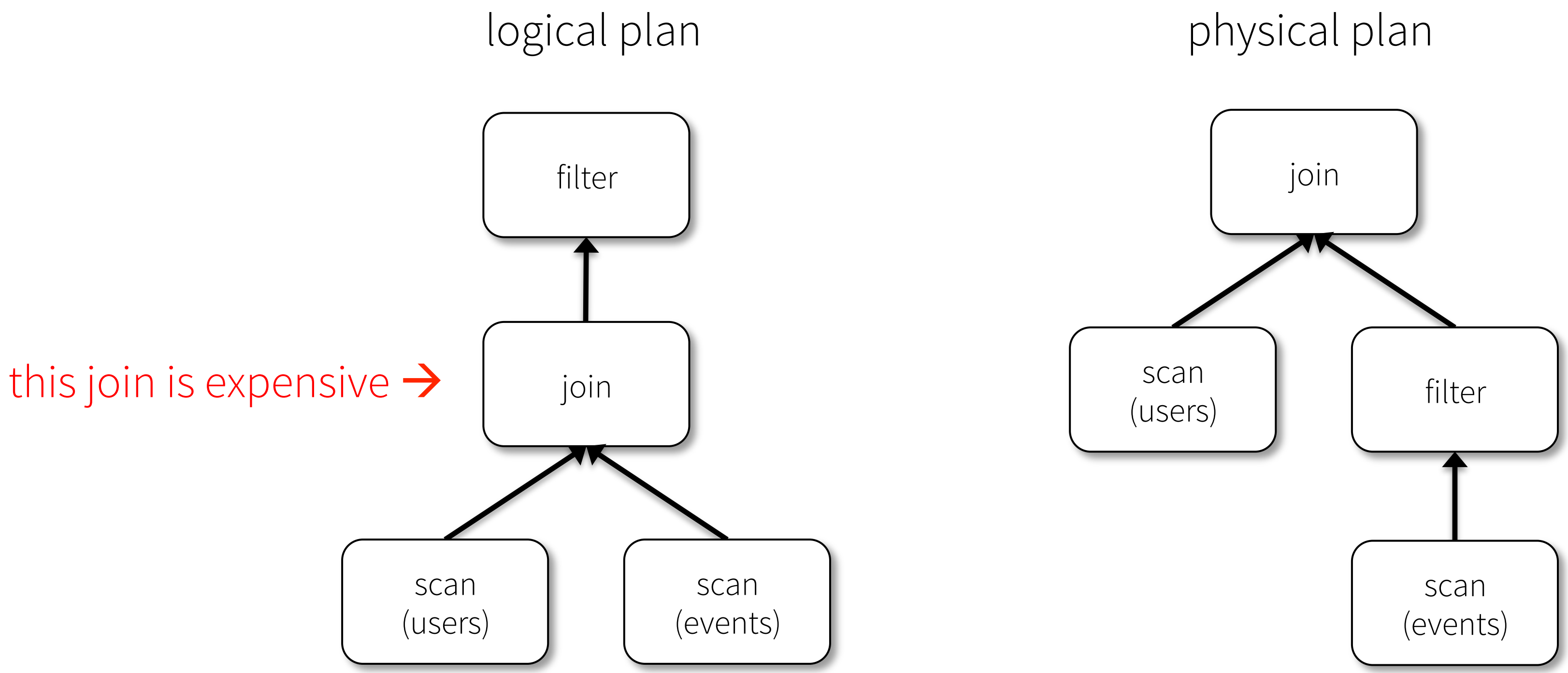
Optimized and compiled to bytecode by Spark SQL

Read/write to Hive, JSON, Parquet, ORC, Pandas

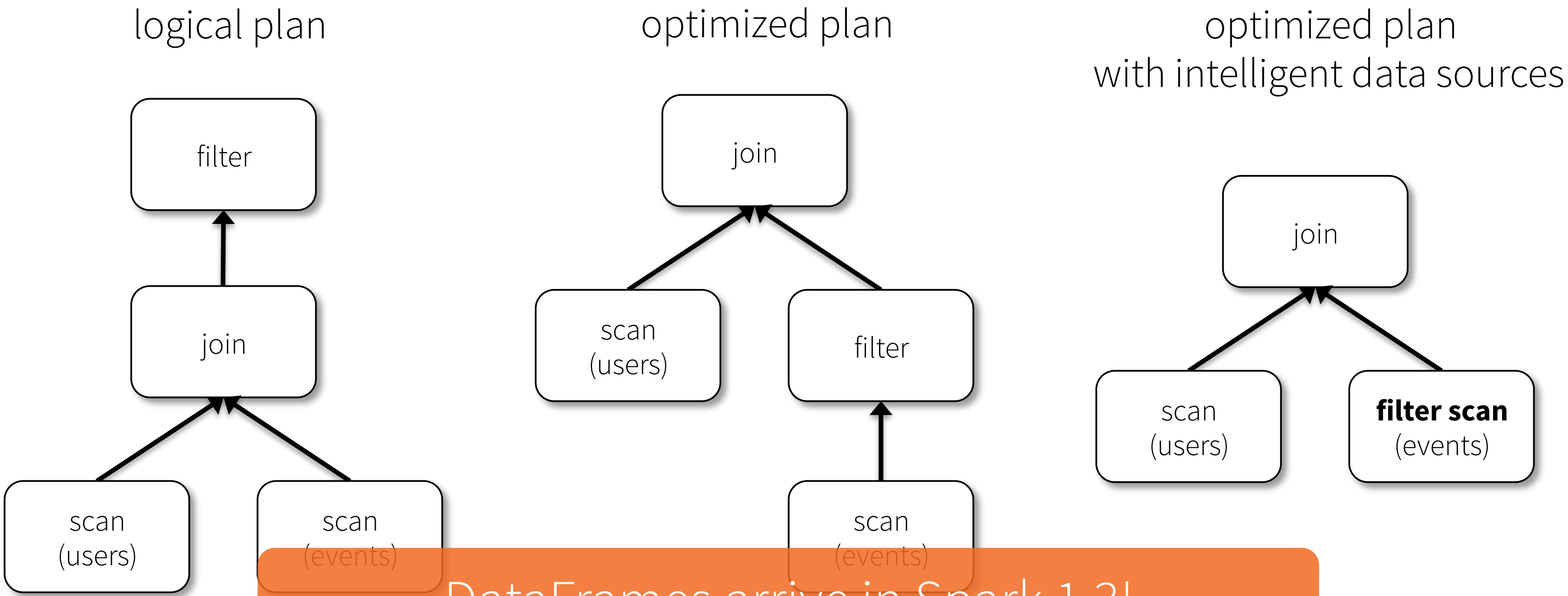
```
df = jsonFile("tweets.json")

df[df.user == "matei"]
  .groupBy("date")
  .sum("retweets")
```



```
joined = users.join(events, users.id == events.uid)
filtered = joined.filter(events.date >= "2015-01-01")
```



```
joined = users.join(events, users.id == events.uid)
filtered = joined.filter(events.date >= "2015-01-01")
```



DataFrames arrive in Spark 1.3!

From MapReduce to Spark

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

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

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

public static class WorkdCountReduce 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));
  }
}
```

```scala
val file = spark.textFile("hdfs://...")
val counts = file.flatMap(line => line.split(" "))
                 .map(word => (word, 1))
                 .reduceByKey(_ + _)
counts.saveAsTextFile("hdfs://...")
```

databricks

```
pdata.map(lambda x: (x.dept, [x.age, 1])) \
  .reduceByKey(lambda x, y: [x[0] + y[0], x[1] + y[1]]) \
  .map(lambda x: [x[0], x[1][0] / x[1][1]]) \
  .collect()
```

```
data.groupBy("dept").avg("age")
```

Thank you! Questions?