Why Spark?

JIANNAN WANG

Background

UC Berkeley's Research Centers

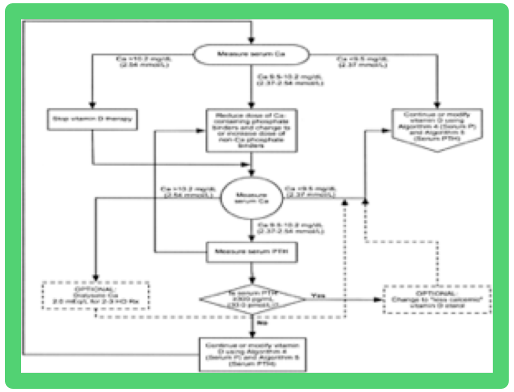
Years	Title	Profs: Director, Co-PIs	Students
1977–1981	X-Tree: Tree Multiprocessor	Despain, Patterson, Sequin	12
1980–1984	RISC: Reduced Instructions	Patterson, Ousterhout, Sequin	17
1983–1986	SOAR: Smalltalk On A RISC	Patterson, Ousterhout	22
1985–1989	SPUR: Symbolic Processing Using RISCs	Patterson, Fateman, Hilfinger, Hodges, Katz, Ousterhout	21
1988–1992	RAID: Redundant Array of Inexpensive Disks	Katz, Ousterhout, Patterson, Stonebraker	16
1993–1998	NOW: Network of Workstations	Culler, Anderson, Brewer, Patterson	25
1997–2002	IRAM: Intelligent RAM	Patterson, Kubiatowicz, Wawrzynek, Yelick	12
2001–2005	ROC: Recovery Oriented Computing Systems	Patterson, Fox	11
2005–2011	RAD Lab: Reliable Adaptive Distributed Computing Lab	Patterson, Fox, Jordan, Joseph, Katz, Shenker, Stoica	30
2007–2013	Par Lab: Parallel Computing Lab	Patterson, Asanovic, Demmel, Fox, Keutzer, Kubiatowicz, Sen, Yelick	40
2011–2017	AMP Lab: Algorithms, Machines, and People	Franklin, Jordan, Joseph, Katz, Patterson, Recht, Shenker, Stoica	40
2013–2018	ASPIRE Lab	Asanovic, Alon, Bachrach, Demmel, Fox, Keutzer, Nikolic, Patterson, Sen, Wawrzynek	40

Requirements

- A common vision
- About 5 years
- At least three faculty
- A dozen students

AMPLab's Vision

Make sense of BIG DATA by tightly integrating algorithms, machines, and people

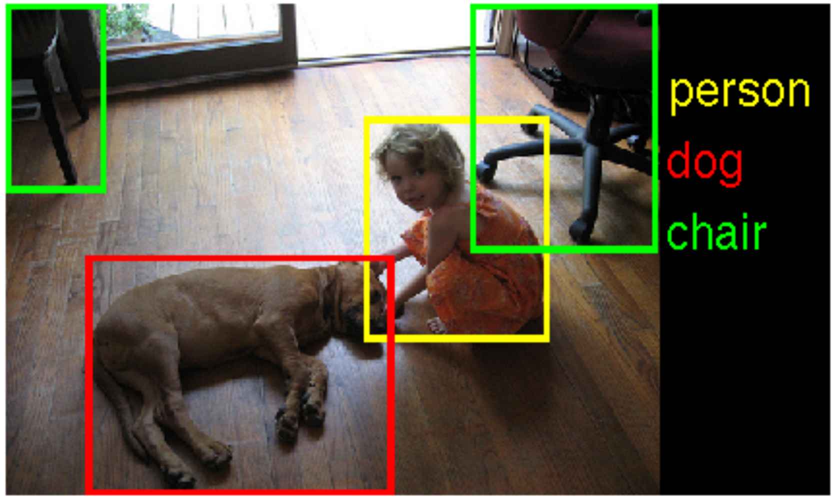


+

+

Example: Extract Value From Image Data

What are in the image?



How to solve the problem?

Deep Learning (Algorithms)
GPU Cluster (Machines)
ImageNet (People)

Spark's Initial Idea

Algorithms + Machines

- Run ML Algorithms on Hadoop

Why is it slow?

1. The algorithms are iterative (i.e., multiple scans of data)
2. MapReduce writes/reads data to/from **disk** at each iteration

Solution

- Keep data in **memory**

How About Fault Tolerance?

Resilient Distributed Datasets (RDD)

Main Idea: Logging the transformations (used to build an RDD) rather than the RDD itself

Zaharia et al. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. NSDI 2012: 15-28

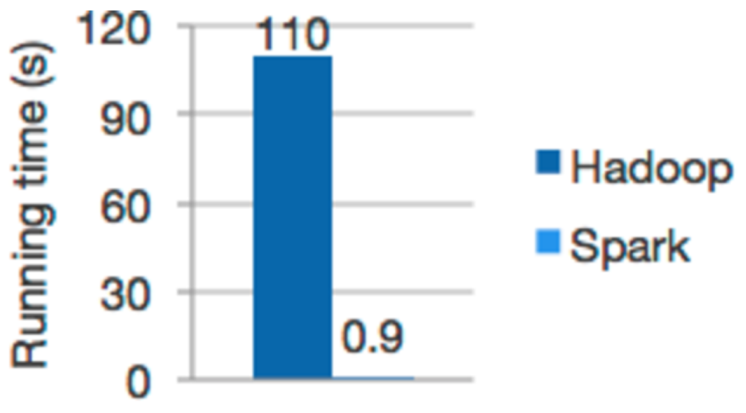
Why Spark?

Fast ✓

Easy to Use ✓

What Makes Spark *Fast*?

In-memory Computation



Logistic regression in Hadoop and Spark

What you save?

- Serialization/Deserialization
- Compression/Decompression
- I/O cost

“CPU (and not I/O) is often the bottleneck”

Ousterhout et al. Making Sense of Performance in Data Analytics Frameworks. NSDI 2015: 293-307

Hardware Trends

	2010	2016	
Storage	50+MB/s (HDD)	500+MB/s (SSD)	10X
Network	1Gbps	10Gbps	10X
CPU	~3GHz	~3GHz	☹

What Makes Spark *Fast*?

Project Tungsten: Bringing Apache Spark Closer to Bare Metal

by Reynold Xin and Josh Rosen
Posted in **ENGINEERING BLOG** | April 28, 2015

1. Memory Management and Binary Processing
2. Cache-aware computation
3. Code generation

Why Spark?

Fast ✓

Easy to Use ✓

What Makes Spark *Easy-to-Use?*

Over 80 High-level Operators

WordCount (Mapreduce)

```java
import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

public class WordCount extends Configured implements Tool {

    public static class TokenizerMapper
    extends Mapper<LongWritable, Text, Text, IntWritable>{

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

        @Override
        public void map(LongWritable key, Text value, Context context
                ) throws IOException, InterruptedException {
            StringTokenizer itr = new StringTokenizer(value.toString());
            while (itr.hasMoreTokens()) {
                word.set(itr.nextToken());
                context.write(word, one);
            }
        }
    }

    public static class IntSumReducer
    extends Reducer<Text, IntWritable, Text, IntWritable> {
        private IntWritable result = new IntWritable();

        @Override
        public void reduce(Text key, Iterable<IntWritable> values,
                Context context
                ) throws IOException, InterruptedException {
            int sum = 0;
            for (IntWritable val : values) {
                sum += val.get();
            }
            result.set(sum);
            context.write(key, result);
        }
    }

    public static void main(String[] args) throws Exception {
        int res = ToolRunner.run(new Configuration(), new WordCount(), args);
        System.exit(res);
    }

    @Override
    public int run(String[] args) throws Exception {
        Configuration conf = this.getConf();
        Job job = Job.getInstance(conf, "word count");
        job.setJarByClass(WordCount.class);

        job.setInputFormatClass(TextInputFormat.class);

        job.setMapperClass(TokenizerMapper.class);
        job.setCombinerClass(IntSumReducer.class);
        job.setReducerClass(IntSumReducer.class);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        job.setOutputFormatClass(TextOutputFormat.class);
        TextInputFormat.addInputPath(job, new Path(args[0]));
        TextOutputFormat.setOutputPath(job, new Path(args[1]));

        return job.waitForCompletion(true) ? 0 : 1;
    }
}
```

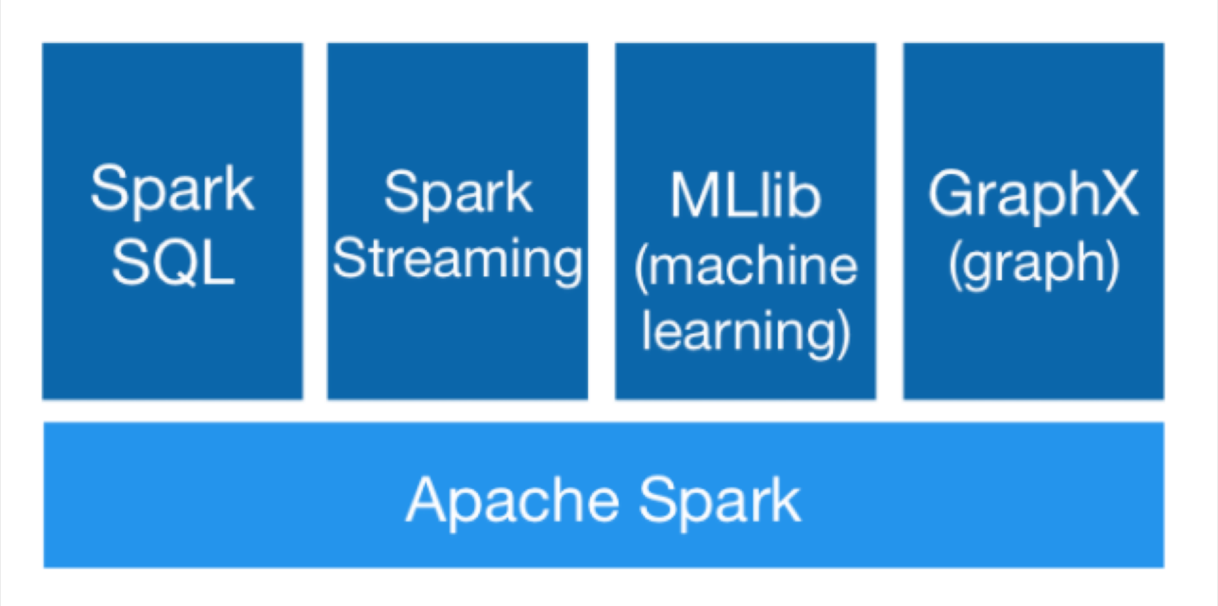
WordCount (Spark)

```python
text_file = spark.textFile("hdfs://...")

text_file.flatMap(lambda line: line.split())
    .map(lambda word: (word, 1))
    .reduceByKey(lambda a, b: a+b)
```

What Makes Spark *Easy-to-Use?*

Unified Engine



Easy to manage, learn, and combine functionality

Analogy

Specialized Devices

Unified Device

What Makes Spark *Easy-to-Use?*

Integrate Broadly

Languages: Scala python R Java

The Big Data world is diversified

Data Sources: cassandra MySQL elastic APACHE HBASE HIVE S3 PostgreSQL JSON

Environments: AWS docker Google Cloud Platform openstack CLOUD SOFTWARE MESOS hadoop YARN Azure

Summary

A brief history of Spark
- UC Berkeley's AMPLab
- Spark's Initial Idea

Spark is fast
- In-memory Computation
- Tungsten Project

Spark is easy-to-use
- High-level Operators
- Unified Engine
- Integrate Broadly