Why?

`grep`?

hadoop

Spark

Spark officially sets a new record in large-scale sorting

November 5, 2014 | by Reynold Xin

A month ago, we shared with you our entry to the 2014 Gray Sort competition, a 3rd-party benchmark measuring how fast a system can sort 100 TB of data (1 trillion records). Today, we are happy to announce that our entry has been reviewed by the benchmark committee and we have officially won the Daytona GraySort contest!

In case you missed our earlier blog post, using Spark on 206 EC2 machines, we sorted 100 TB of data on disk in 23 minutes. In comparison, the previous world record set by Hadoop MapReduce used 2100 machines and took 72 minutes. This means that Spark sorted the same data **3X faster** using **10X fewer machines**. All the sorting took place on disk (HDFS), without using Spark's in-memory cache. This entry tied with a UCSD research team building high performance systems and we jointly set a new world record.

	Hadoop MR Record	Spark Record	Spark 1 PB
Data Size	102.5 TB	100 TB	1000 TB
Elapsed Time	72 mins	23 mins	234 mins
# Nodes	2100	206	190
# Cores	50400 physical	6592 virtualized	6080 virtualized
Cluster disk throughput	3150 GB/s (est.)	618 GB/s	570 GB/s
Sort Benchmark Daytona Rules	Yes	Yes	No
Network	dedicated data center, 10Gbps	virtualized (EC2) 10Gbps network	virtualized (EC2) 10Gbps network
Sort rate	**1.42 TB/min**	**4.27 TB/min**	**4.27 TB/min**
Sort rate/node	**0.67 GB/min**	**20.7 GB/min**	**22.5 GB/min**

http://databricks.com/blog/2014/11/05/spark-officially-sets-a-new-record-in-large-scale-sorting.html

Big Data

Big Code

Big Data

```
public class WordCount {
  public static class Map extends 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, Context context) throws IOException,
InterruptedException {
         String line = value.toString();
         StringTokenizer tokenizer = new StringTokenizer(line);
         while (tokenizer.hasMoreTokens()) {
             word.set(tokenizer.nextToken());
             context.write(word, one);
         }
     }
  }

  public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {
     public void reduce(Text key, Iterable<IntWritable> values, Context context)
       throws IOException, InterruptedException {
         int sum = 0;
         for (IntWritable val : values) {
             sum += val.get();
         }
         context.write(key, new IntWritable(sum));
     }
  }

  public static void main(String[] args) throws Exception {
     Configuration conf = new Configuration();

     Job job = new Job(conf, "wordcount");

     job.setOutputKeyClass(Text.class);
     job.setOutputValueClass(IntWritable.class);

     job.setMapperClass(Map.class);
     job.setReducerClass(Reduce.class);

     job.setInputFormatClass(TextInputFormat.class);
     job.setOutputFormatClass(TextOutputFormat.class);

     FileInputFormat.addInputPath(job, new Path(args[0]));
     FileOutputFormat.setOutputPath(job, new Path(args[1]));

     job.waitForCompletion(true);
  }
}
```

Big Code

Big Data

```
public class WordCount {
  public static class Map extends 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, Context context) throws IOException,
InterruptedException {
        String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);
        while (tokenizer.hasMoreTokens()) {
            word.set(tokenizer.nextToken());
            context.write(word, one);
        }
     }
  }

  public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {
     public void reduce(Text key, Iterable<IntWritable> values, Context context)
       throws IOException, InterruptedException {
        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }
        context.write(key, new IntWritable(sum));
     }
  }

  public static void main(String[] args) throws Exception {
     Configuration conf = new Configuration();

     Job job = new Job(conf, "wordcount");

     job.setOutputKeyClass(Text.class);
     job.setOutputValueClass(IntWritable.class);

     job.setMapperClass(Map.class);
     job.setReducerClass(Reduce.class);

     job.setInputFormatClass(TextInputFormat.class);
     job.setOutputFormatClass(TextOutputFormat.class);

     FileInputFormat.addInputPath(job, new Path(args[0]));
     FileOutputFormat.setOutputPath(job, new Path(args[1]));

     job.waitForCompletion(true);
  }
}
```

Tiny Code

```
object WordCount{
  def main(def main(args: Array[String])){
    val sparkConf = new SparkConf()
                        .setAppName("wordcount")
    val sc = new SparkContext(sparkConf)
    sc.textFile(args(0))
      .flatMap(_.split(" "))
      .countByValue
      .saveAsTextFile(args(1))
  }
}
```

Why Spark?

- Readability
- Expressiveness
- Fast
- Testability
- Interactive
- Fault Tolerant
- Unify Big Data

Course Overview

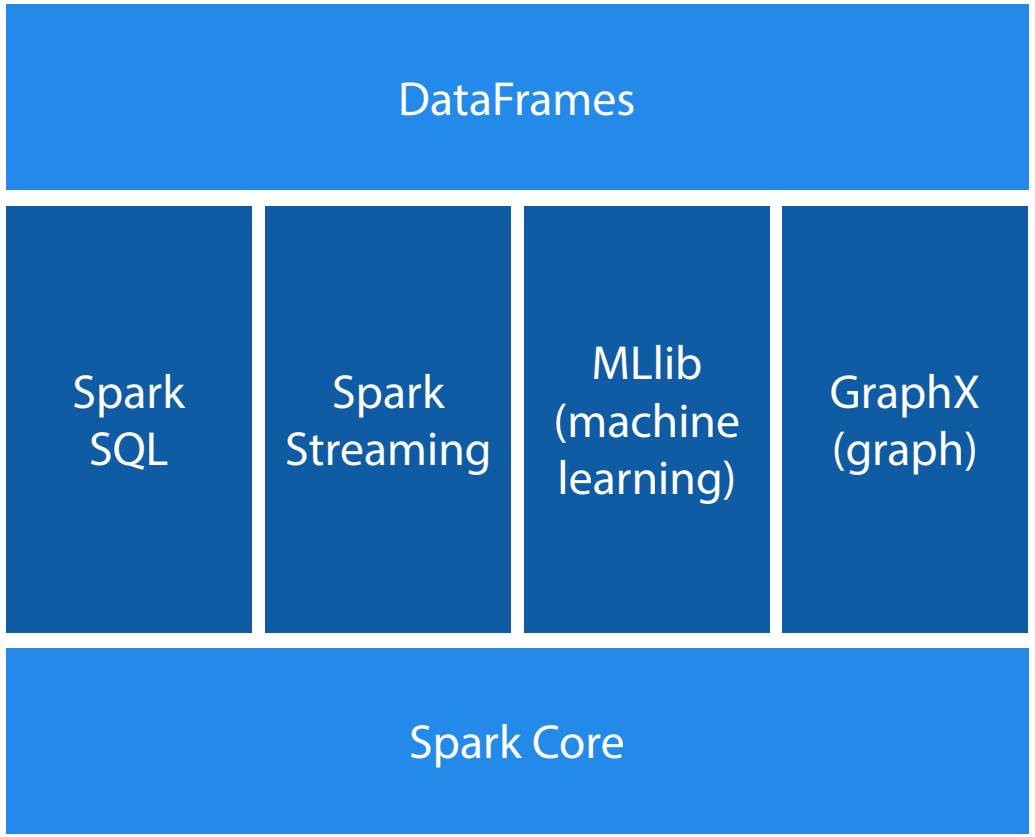
- Basics of Spark
- Core API
- Cluster Managers
- Spark Maintenance
- Libraries
  - SQL
  - Streaming
  - MLlib/GraphX
- Troubleshooting / Optimization
- Future of Spark

Section Overview

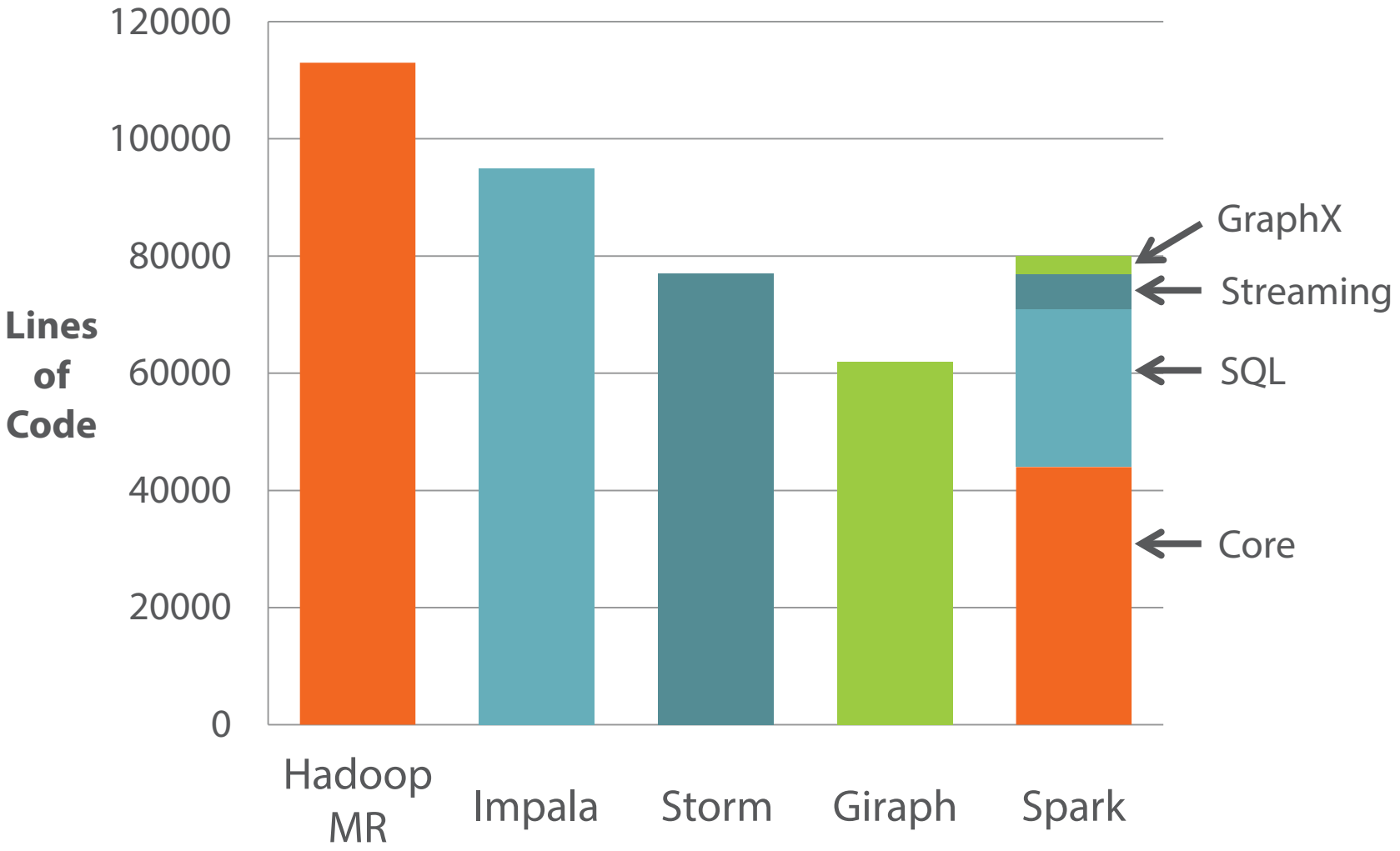
- Basics of Spark
  - Hadoop
  - History of Spark
  - Installation
  - Big Data's Hello World
  - Course Prep

The MapReduce Explosion

A Unified Platform for Big Data

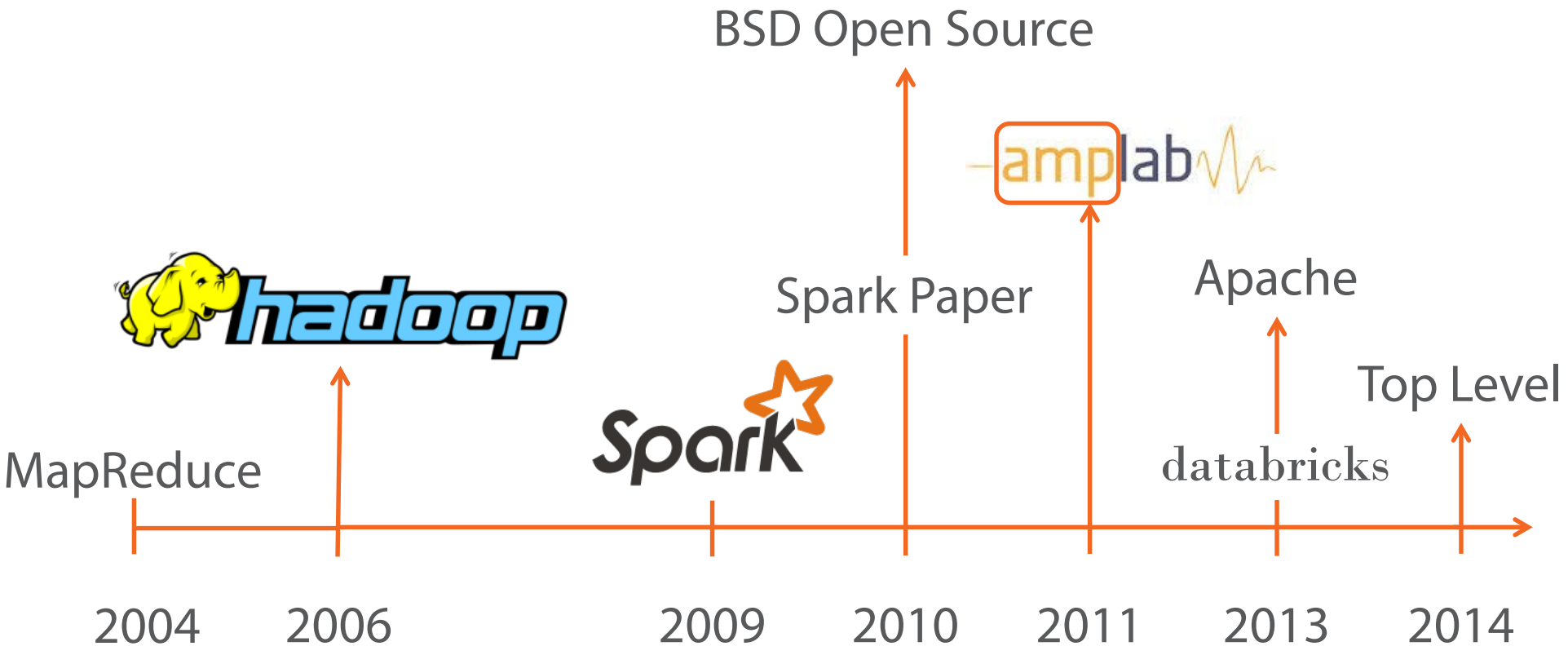


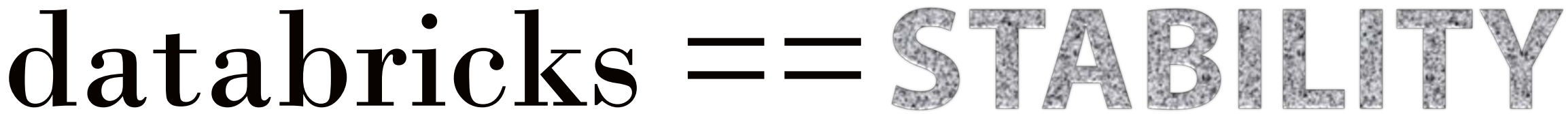
Abstractions FTW



*As of April 2015 using cloc w/o tests or examples

The History of Spark





Stability

Spark SQL

In this release Spark SQL graduates from an alpha project, providing backwards compatibility guarantees for the HiveQL dialect and stable programmatic API's. Spark SQL adds support for writing tables in the data sources API. A new JDBC data source allows importing and exporting from MySQL, Postgres, and other RDBMS systems. A variety of small changes have expanded the coverage of HiveQL in Spark SQL. Spark SQL also adds support schema evolution with the ability to merging compatible schemas in Parquet.

Spark ML/MLlib

In this release Spark MLlib introduces several new algorithms: latent Dirichlet allocation (LDA) for topic modeling, multinomial logistic regression for multiclass classification, Gaussian mixture model (GMM) and power iteration clustering for clustering, FP-growth for frequent pattern mining, and block matrix abstraction for distributed linear algebra. Initial support has been added for model import/export in exchangeable format, which will be expanded in future versions to cover more model types in Java/Python/Scala. The implementations of k-means and ALS receive updates that lead to significant performance gain. PySpark now supports the ML pipeline API added in Spark 1.2, and gradient boosted trees and Gaussian mixture model. Finally, the ML pipeline API has been ported to support the new DataFrames abstraction.

Spark Streaming

Spark 1.3 introduces a new *direct* Kafka API (docs) which enables exactly-once delivery without the use of write ahead logs. It also adds a Python Kafka API along with infrastructure for additional Python API's in future releases. An online version of logistic regression and the ability to read binary records have also been added. For stateful operations, support has been added for loading of an initial state RDD. Finally, the streaming programming guide has been updated to include information about SQL and DataFrame operations within streaming applications, and important clarifications to the fault-tolerance semantics.

GraphX

GraphX adds a handful of utility functions in this release, including conversion into a canonical edge graph.

Upgrading to Spark 1.3

Spark 1.3 is binary compatible with Spark 1.X releases, so no code changes are necessary. This excludes API's marked explicitly as unstable.

As part of stabilizing the Spark SQL API, the `SchemaRDD` class has been renamed to `DataFrame`. Spark SQL's migration guide describes the upgrade process in detail. Spark SQL also now requires that column identifiers which use reserved words (such as "string" or "table") be escaped using backticks.

Known Issues

This release has few known issues which will be addressed in Spark 1.3.1:

- SPARK-6194: A memory leak in PySPark's `collect()`.
- SPARK-6222: An issue with failure recovery in Spark Streaming.
- SPARK-6315: Spark SQL can't read parquet data generated with Spark 1.1.
- SPARK-6247: Errors analyzing certain join types in Spark SQL.

https://spark.apache.org/releases/spark-release-MAJOR-MINOR-REVISION.html

Stability

SparkQA commented 8 minutes ago

Test build #35017 has finished for PR 6841 at commit `276389d`.

- This patch **fails MiMa tests**.
- This patch merges cleanly.
- This patch adds the following public classes *(experimental)*:
  - `class JavaSampleActorReceiver<T> extends UntypedActor`
  - `public class JavaActorWordCount`
  - `class JavaGlobalActorSystem`
  - `trait ActorHelper extends Logging`
  - `class JavaActorHelper(actor: Actor)`
  - `case class Statistics(numberOfMsgs: Int,`
  - `class Supervisor extends Actor`
  - `trait ActorSystemFactory extends Serializable`

https://github.com/apache/spark/pull/6841

Stability

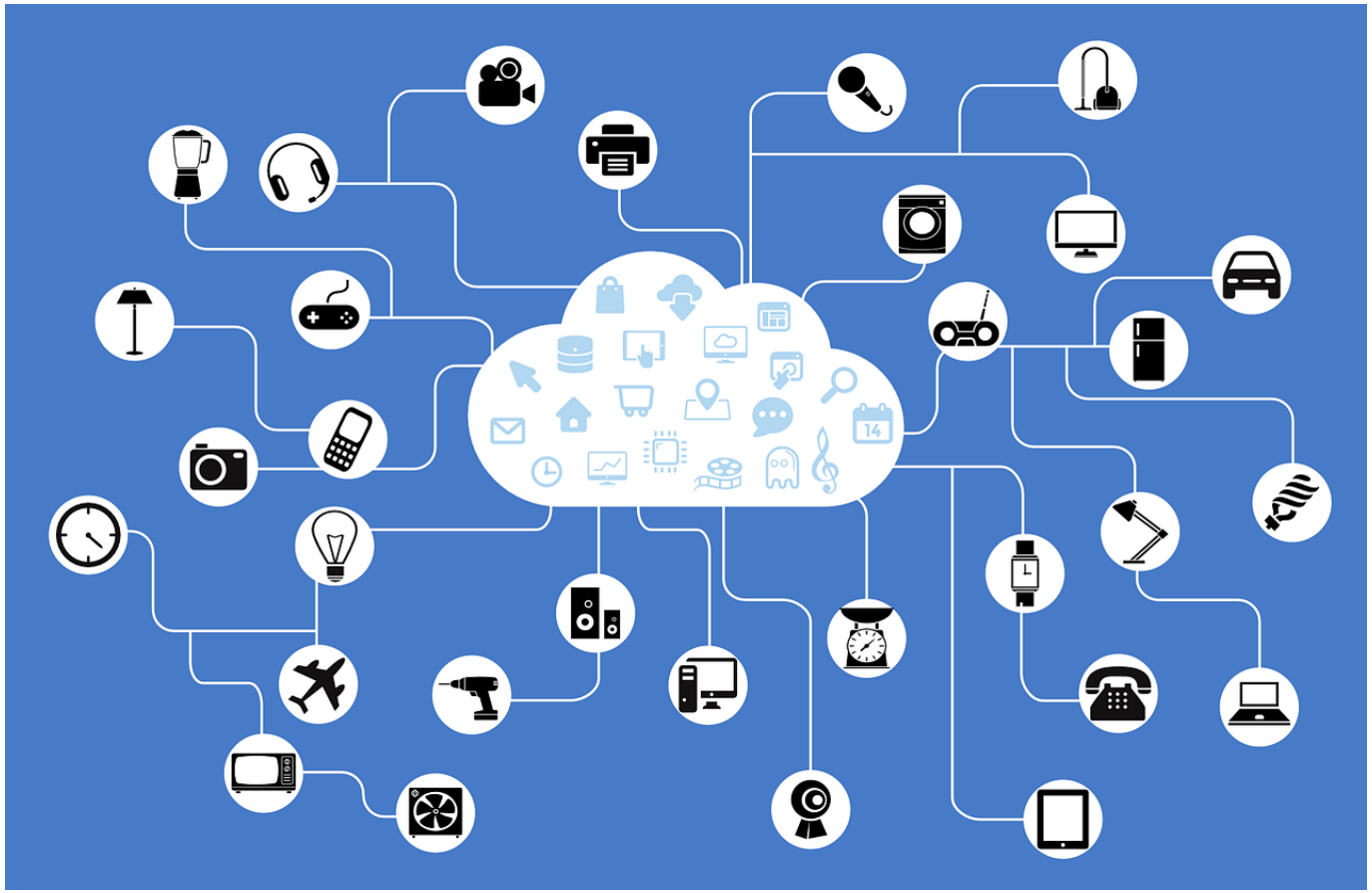
Who Is Using Spark?

Yahoo!

Spark Languages

Spark Languages

Big Data



Big Data

Big Data

Spark Logistics

Summary

- Why
- MapReduce Explosion
- Spark's History
- Installation
- Hello Big Data!
- Additional Resources