Enterprise Data Storage and Analysis on

Tim Barr

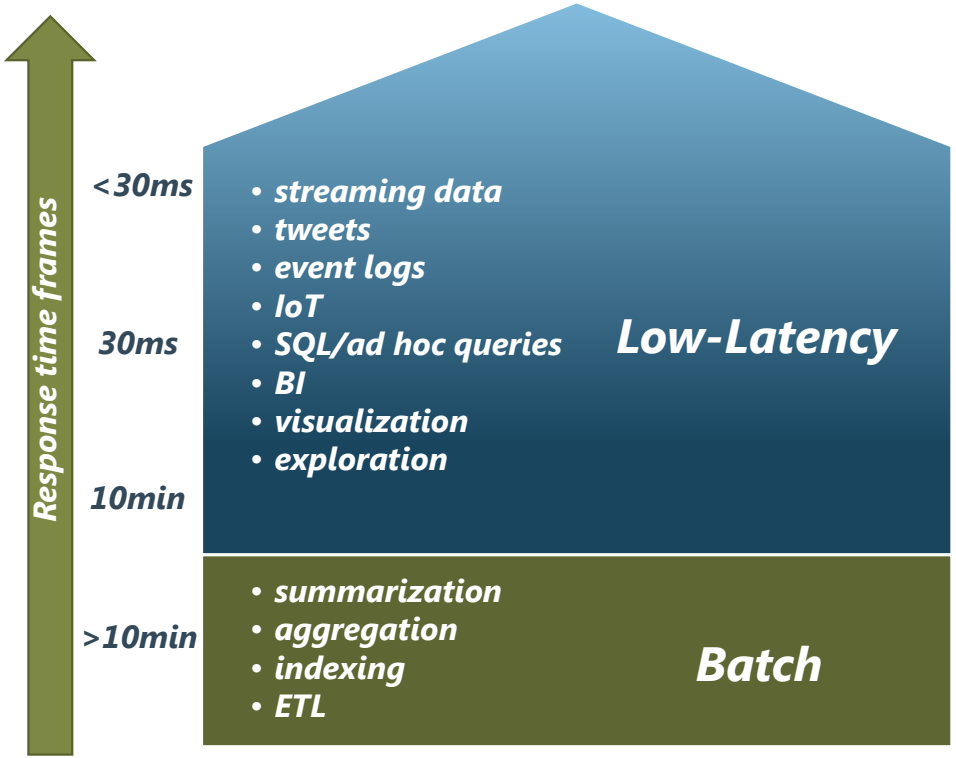
January 15, 2015

Agenda

- Challenges in Big Data Analytics
- Why many Hadoop deployments under deliver
- What is Apache Spark
- Spark Core, SQL, Streaming, MLlib, and GraphX
- Graphs for CyberAnalytics
- Hybrid Spark Architecture
- Why you should love Scala
- Q&A

Challenges in Big Data Analytics

Emergence of Latency-Sensitive Analytics



Higher performance and more innovative use of memory-storage hierarchies and interconnects required here

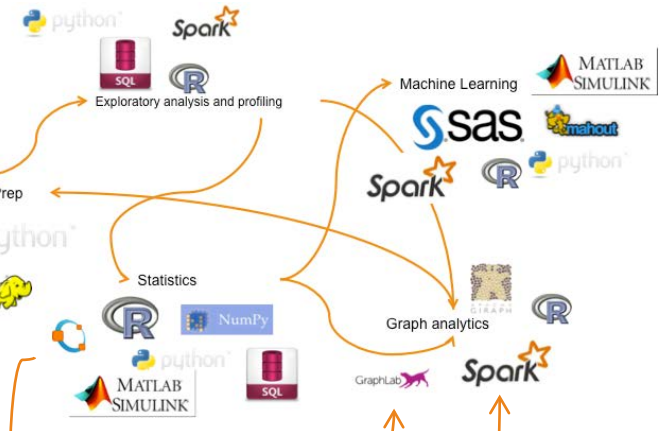
Hadoop tomorrow

Hadoop today

Performance optimizations

Focus on Analytic Productivity

Time to Value Is the Key Performance Metric



Stand up big data clusters

- Sizing
- Provisioning
- Configuration
- Tuning
- Workload management
- Move into production

Move data

- Copy, load, replication
- Multiple data sources
- Fighting data gravity

Data prep

- Cleansing
- Merge data
- Apply schema

Analyze

- Multiple frameworks
- Analytics pipeline

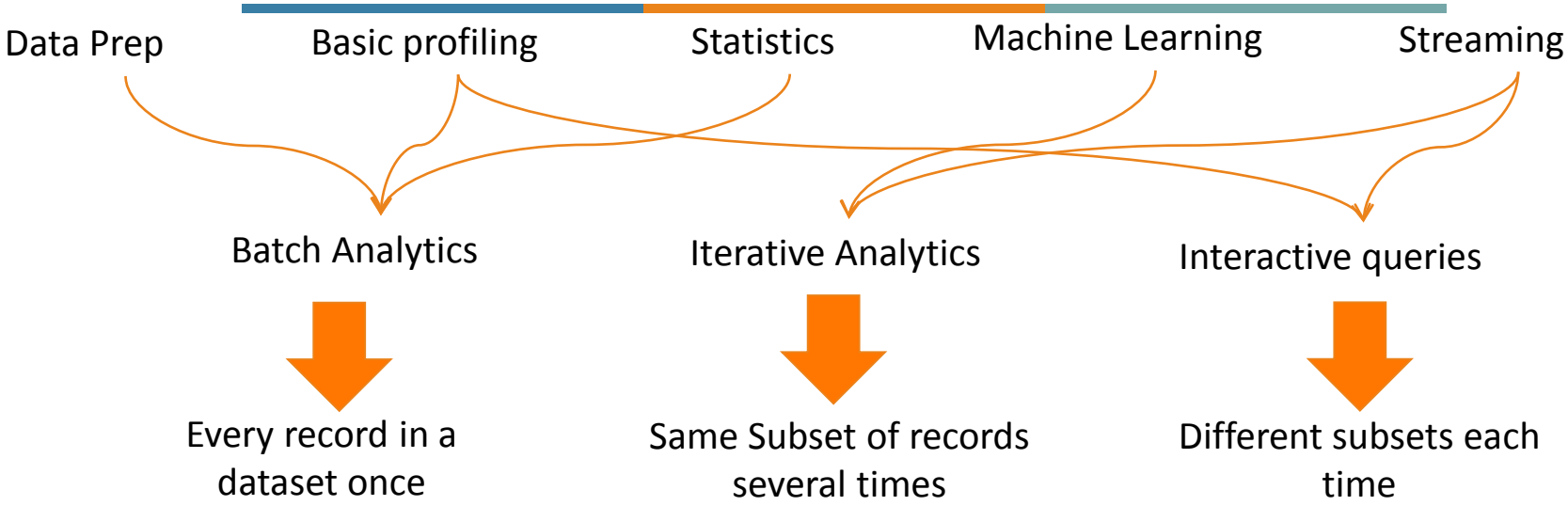
Apply results

- Scoring
- Reports
- Apply to next stage

Map Shuffle Reduce

Job run time is a fraction of the total Time to Value

Integrated Advanced Analytics



- In the real-world, advanced analytics needs multiple, integrated toolsets
- These toolsets require very different computing demands

6

Why many Hadoop Deployments Under Deliver

- Data scientists are critical, but in short supply
- Shortage of big data tools
- Complexity of the MapReduce programming environment
- Cost of Analytic value currently too high
- MapReduce performance does not allow the analyst to follow his/her nose
- Spark is often installed on existing under powered Hadoop clusters leading to undesirable performance

Hadoop: Great Promise but with Challenges

Forbes Article: How to Avoid a Hadoop Hangover

"Hadoop is hard to set up, use, and maintain. In and of itself, grid computing is difficult, and Hadoop doesn't make it any easier. Hadoop is still maturing from a developer's standpoint, let alone from the standpoint of a business user. Because only savvy Silicon Valley engineers can derive value Hadoop, it's not going to make inroads into larger organizations without a lot of handholding and professional services."

Mike Driscoll, CEO of Metamarkets

http://www.forbes.com/sites/danwoods/2012/07/27/how-to-avoid-a-hadoop-hangover/

CRAY

Hadoop: Perception versus Reality

Current Perception of Hadoop

- Synonymous with Big Data and openness
- Capable of huge scale with ad-hoc infrastructure

Current Reality of Hadoop

- Many experimenting
- Much expertise in Warehousing – little beyond that
- Data Scientist bottleneck – performance not yet an issue

Current Trajectory of Hadoop

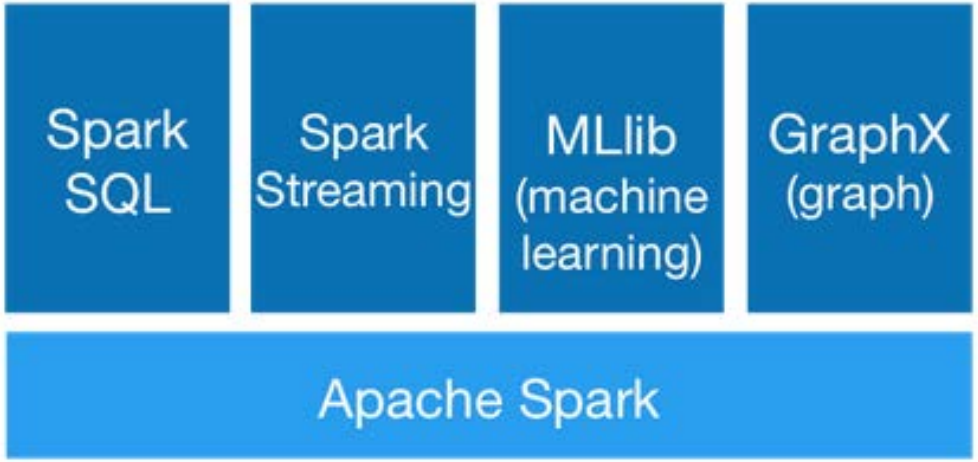
- Industry Momentum – Universities, Govt., Private firms, etc.
- More Users – Beyond Data scientists, Domain Scientists, analysts, etc.
- More Complexity – Multi-layered files, complex algorithms, etc.

Hadoop widely perceived as high potential,
not yet high value, but that's about to change…

CRAY

What is Spark?

- Distributed data analytics engine, generalizing MapReduce
- Core engine, with streaming, SQL, machine learning, and graph processing modules
- Program in Python, Scala, and/or Java



Spark - Resilient Distributed Dataset (RDD)

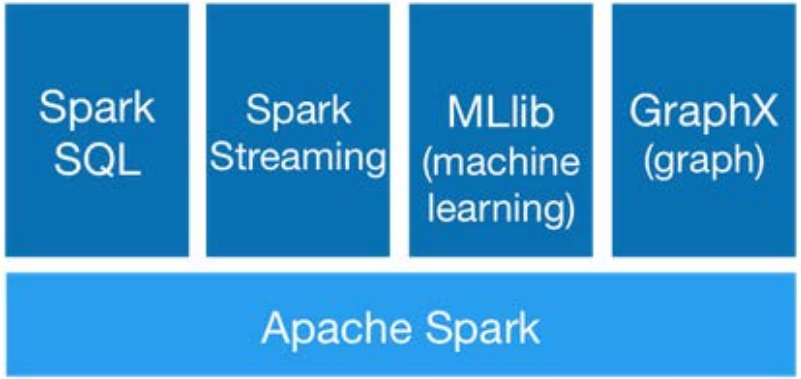
- Distributed collection of objects
- Benefits of RDDs?
  - RDDs exist in-memory
  - Built via parallel transformations (map, filter, …)
  - RDDs are automatically rebuilt on failure

There are two ways to create RDDs:

- Parallelizing an existing collection in your driver program
- Referencing a dataset in an external storage system, such as a shared filesystem, HDFS, HBase, or any data source offering a Hadoop InputFormat.

Benefits of a Unified Platform

- No copying or ETL of data between systems
- Combine processing types in one program
- Code reuse
- One system to learn
- One system to maintain

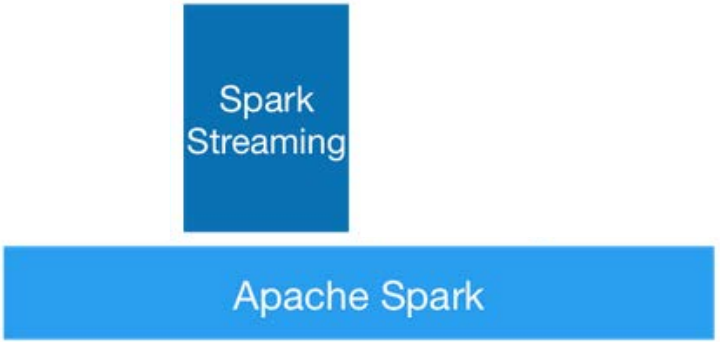


Spark SQL

- Unified data access with with SchemaRDDs
- Tables are a representation of (Schema + Data) = SchemaRDD
- Hive Compatibility
- Standard Connectivity via ODBC and/or JDBC

Spark Streaming

- Spark Streaming expresses streams as a series of RDDs over time
- Combine streaming with batch and interactive queries
- Stateful and Fault Tolerant



Spark Streaming – Inputs/Outputs

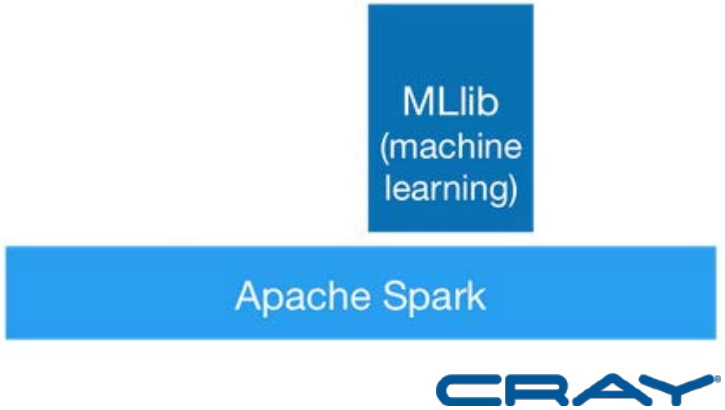
CRAY

Spark Machine Learning

- Iterative computation
- Vectors, Matrices = RDD[Vector]

Current MLlib 1.1 Algorithms

- linear SVM and logistic regression
- classification and regression tree
- k-means clustering
- recommendation via alternating least squares
- singular value decomposition
- linear regression with L1- and L2-regularization
- multinomial naive Bayes
- basic statistics
- feature transformations



Spark GraphX

- Unifies graphs with RDDs of edges and vertices
- View the same data as both graphs and collections
- Custom iterative graph algorithms via Pregel API

Current GraphX Algorithms

- PageRank
- Connected components
- Label propagation
- SVD++
- Strongly connected components
- Triangle count

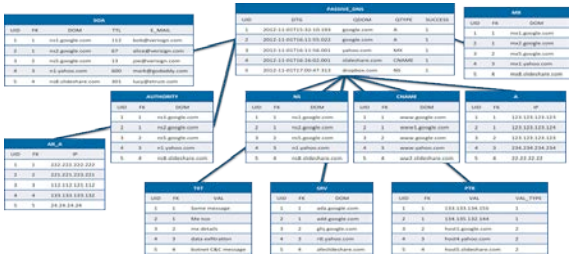
CRAY

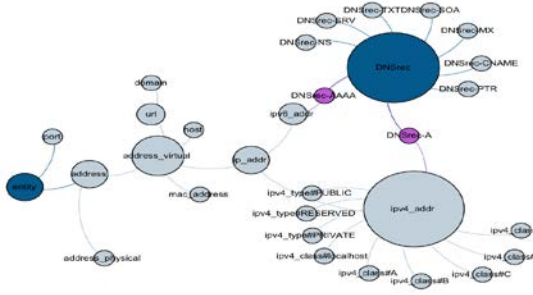
Applying Graphs to CyberAnalytics

Use the graph as a pre-merged perspective of all the available data sets

Graphs enable discovery

- It's called a network! – represent that information in the more natural and appropriate format
- Graphs are optimized to show the relationships present in metadata
- "fail fast, fail cheap" – choose a graph engine that supports rapid hypothesis testing
  - Returning answers before the analyst forgets why he asked them, this enables the investigative discovery flow
- Using this framework, analysts can more easily and more quickly find unusual things – this matters significantly when there is the constant threat of new *unusual* things
- When all focus is no longer on dealing with the known, there is bandwidth for discovery
- When all data can be analyzed in a holistic manner, new patterns and relationships can be seen





Using Graph Analysis to Identify Patterns

Example mature cyber-security questions

- Who hacked us? What did they touch in our network? Where else did they go?
- What unknown botnets are we hosting?
- What are the vulnerabilities in our network configuration?
- Who are the key influencers in the company / on the network?
- What's weird that's happening on the network?

Proven graph algorithms help answer these questions

- Subgraph identification
- Alias identification
- Shortest-path identification
- Common-node identification
- Clustering / community identification
- Graph-based cyber-security discovery environment

Analytic tradecraft and algorithms mature together

- General questions require swiss army knives
- Specific, well-understood questions use exacto knives

CRAY

Spark System Requirements

Storage Systems

It is important to place it as close to this system as possible. If at all possible, run Spark on the same nodes as HDFS. The simplest way is to set up a Spark standalone mode cluster on the same nodes, and configure Spark and Hadoop's memory and CPU usage to avoid interference

Local Disks

While Spark can perform a lot of its computation in memory, it still uses local disks to store data that doesn't fit in RAM, as well as to preserve intermediate output between stages. We recommend having 4-8 disks per node, configured without RAID

https://spark.apache.org/docs/latest/hardware-provisioning.html

Spark System Requirements (continued)

Memory

Spark runs well with anywhere from 8 GB to hundreds of gigabytes of memory per machine. In all cases, we recommend allocating only at most 75% of the memory for Spark; leave the rest for the operating system and buffer cache.

Network

When the data is in memory, a lot of Spark applications are network-bound. Using a 10 Gigabit or higher network is the best way to make these applications faster. This is especially true for "distributed reduce" applications such as group-bys, reduce-bys, and SQL joins.

CPU Cores

Spark scales well to tens of CPU cores per machine because it performs minimal sharing between threads. You should likely provision at least 8-16 cores per machine.

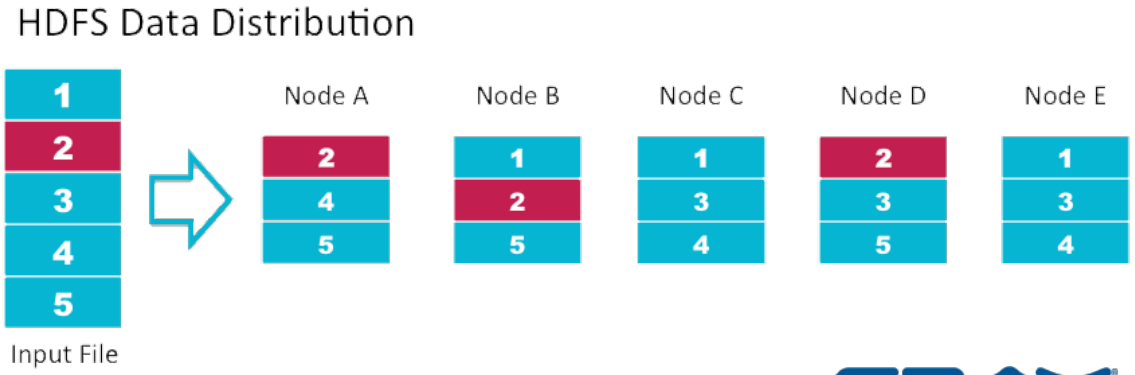
https://spark.apache.org/docs/latest/hardware-provisioning.html

Benefits of HDFS

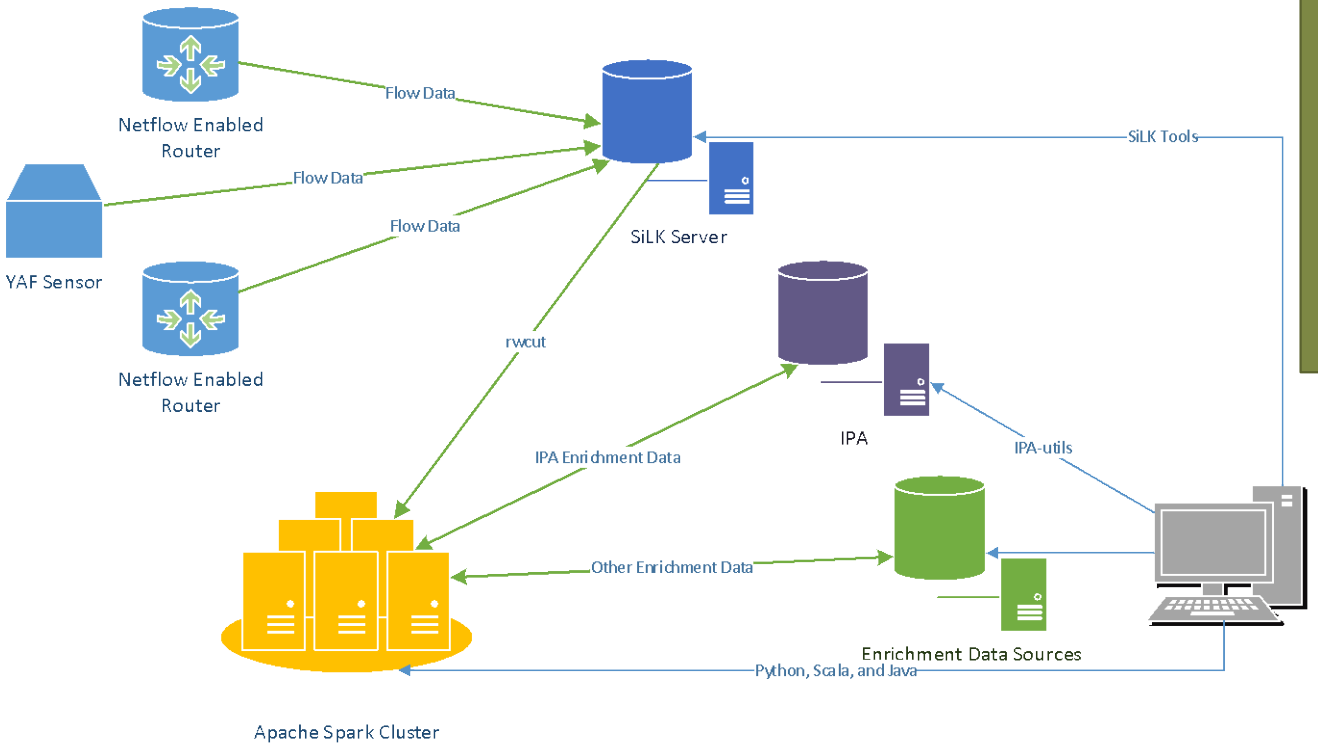
Scale-Out Architecture: Add servers to increase capacity
High Availability: Serve mission-critical workflows and applications
Fault Tolerance: Automatically and seamlessly recover from failures
Flexible Access: Multiple and open frameworks for serialization and file system mounts
Load Balancing: Place data intelligently for maximum efficiency and utilization
Configurable Replication: Multiple copies of each file provide data protection and computational performance

HDFS Sequence Files

A Sequence file is a data structure for binary key-value pairs. it can be used as a common format to transfer data between MapReduce jobs. Another important advantage of a sequence file is that it can be used as an archive to pack smaller files.



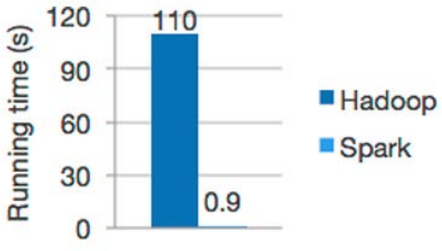
Hybrid Spark Architecture



Apache Spark should…

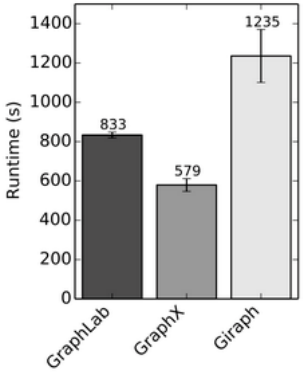
- be complimentary to your existing architecture
- enhance existing system capabilities
- assume some of the analytic workload
- handle archive storage

Spark Performance



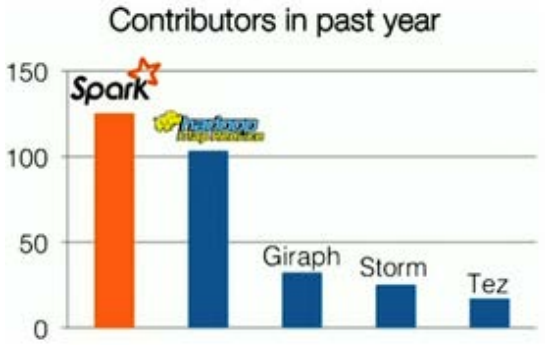
Logistic regression in Hadoop and Spark

In-Memory Performance



End-to-end PageRank performance (20 iterations, 3.7B edges)

Order of Magnitude Graph Performance



Active Open Source Community

Performance – Spark wins Daytona Gray Sort 100TB Benchmark

They used Spark and sorted 100TB of data using 206 EC2 i2.8xlarge machines in 23 minutes. The previous world record was 72 minutes, set by a Hadoop MapReduce cluster of 2100 nodes. 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.

Outperforming large Hadoop MapReduce clusters on sorting not only validates the vision and work done by the Spark community, but also demonstrates that Spark is fulfilling its promise to serve as a faster and more scalable engine for data processing of all sizes.

https://spark.apache.org/news/spark-wins-daytona-gray-sort-100tb-benchmark.html

CRAY

Why you should love Scala

(If you don't already)

Word Count Example – Spark Scala

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

Word Count Example – MapReduce

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

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
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.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.FileSplit;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;
```

Word Count Example – MapReduce (continued)

```
public class WordCount {

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

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

        public void map(Object 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);
            }
```

Word Count Example – MapReduce (continued)

```
}

  public static class IntSumReducer extends
     Reducer<Text, IntWritable, Text, IntWritable> {

     private IntWritable result = new IntWritable();

     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);
```

Word Count Example – MapReduce (continued)

```
}
  }

  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    String[] otherArgs = new GenericOptionsParser(conf, args)
      .getRemainingArgs();

    Job job = new Job(conf, "word count");

    job.setJarByClass(WordCount.class);

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

Word Count Example – MapReduce (continued)

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

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

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

5 Lines of Spark Scala Code vs. 57 Lines of MapReduce Code

Useful Resources

Apache Spark
https://spark.apache.org/

Spark Summit 2014
http://spark-summit.org/2014

Apache Spark Reference Card
http://refcardz.dzone.com/refcardz/apache-spark

Apache Spark Meetups
http://spark.meetup.com/

Thank You!
Questions?