

An introduction to Spark

By Todd Lipcon

With thanks to Ted Malaska, Jairam Ranganathan and Sandy Ryza



About the Presenter...



Todd Lipcon

Software Engineer

San Francisco Bay Area | Information Technology and Services

2009-
Current:

Software Engineer at Cloudera

Hadoop Commiter and PMC Member

HBase Committer and PMC Member

Member of the Apache Software Foundation

Past: Software Engineer at Amie Street, Inc.

Co-founder and technical lead at MyArtPlot, LLC

Software Engineering Intern at Google

Agenda

- Introduction to Crunch
- Introduction to Spark
- Spark use cases
- Changing mindsets
- Spark in CDH
- Short-term roadmap

MR is great... but can we do better?

Problems with MR:

- **Very low-level:** requires a lot of code to do simple things
- **Very constrained:** everything must be described as “map” and “reduce”. Powerful but sometimes difficult to think in these terms.

MR is great... but can we do better?

Two approaches to improve on MapReduce:

1. **Special purpose systems** to solve one problem domain well. Ex: Giraph / Graphlab (graph processing), Storm (stream processing)
2. **Generalize the capabilities of MapReduce** to provide a richer foundation to solve problems.
Ex: MPI, Hama/Pregel (BSP), Dryad (arbitrary DAGs)

Both are viable strategies depending on problem!

Apache Crunch

A brief introduction

What is Crunch?

“The *Apache Crunch Java library* provides a framework for writing, testing, and running MapReduce pipelines. Its goal is to make **pipelines** that are composed of many user-defined functions **simple to write, easy to test, and efficient to run.**”

Compared to other Hadoop tools

(e.g Hive/Pig)

- **Developer focused**
 - Java API, easy to integrate with other Java libraries
 - Less boiler plate, more “fluent” API than MR
- **Minimal abstractions**
 - Thin layer on top of MapReduce – equally fast
 - Access to MapReduce APIs when you really need them
- **Flexible data model**
 - Able to use POJOs, protocol buffers, Avro records, tuples, etc.

Crunch concepts: Sources and Targets

- **Source:** a repository to load input data *from*
 - JDBC
 - Files on Hadoop (text/CSV, Avro, protobuf, binary, custom data formats, etc)
 - HBase
 - Yields a **PCollection<T>**
- **Target:** a place to write job output
 - Stores a **PCollection** back onto HDFS or some other storage

Crunch concepts: PCollection<T>

- Represents a “parallel collection” of T objects
 - **.parallelDo(DoFn<T, U> function)**
 - like a Hadoop Mapper
 - Emit 0 to n records for each input record
 - Output: PCollection<U>
 - **.filter(FilterFn<T> filter)**
 - FilterFn must return *true* or *false* for each input record
 - Output: PCollection<T>
 - **.union(PCollection<T>)**
 - Appends two collections together

Crunch concepts: PTable<K, V>

- Sub-interface of **PCollection<Pair<K, V>>**
 - Supports all of the PCollection methods.
 - **.groupByKey()**
 - Output: PGroupedTable<K, V>
 - Like a PCollection<K, Iterable<V>>
 - This is just like the “shuffle” phase of MapReduce!
- **JoinStrategy.join(table1, table2, joinType)**
 - Simple method for SQL-style joins between two tables on their matching keys
 - Supports inner join, outer join, cartesian join, etc.
 - API allows user to provide hints on the best implementation

Crunch concepts: PGroupedTable<K, V>

- The same as a PCollection<Pair<K, V>>
 - Still allows PCollection functions like **parallelDo(...)**
- Methods for aggregating or reducing:
 - **combineValues(Aggregator)**
 - Standard aggregators include summing values, averaging, sampling, top-k, etc.

Crunch WordCount (1)

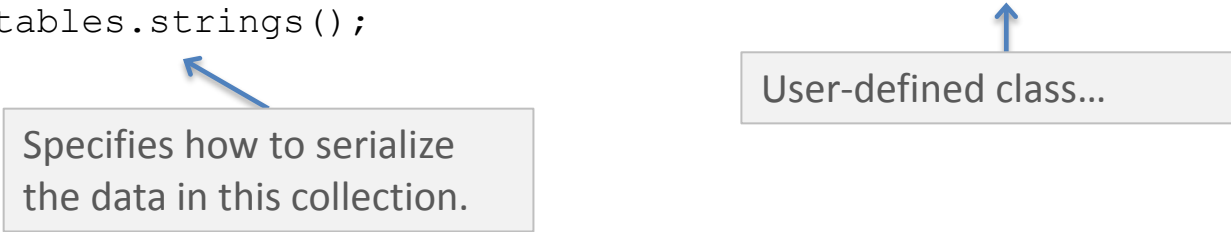
```
1 public class WordCount extends Configured implements Tool {
2     public static void main(String[] args) throws Exception {
3         ToolRunner.run(new Configuration(), new WordCount(), args);
4     }
5
6     void run() {
7         String inputPath = args[0];
8         String outputPath = args[1];
9
10        // Create an object to coordinate pipeline creation and execution
11        Pipeline pipeline = new MRPipeline(WordCount.class, getConf());
        // or... = MemPipeline.getInstance();
```

Standard Hadoop “main”
function which parses
configuration, etc

Specifies whether this
pipeline should execute in
MapReduce or locally in-
memory (for testing)

Crunch WordCount (2)

```
1 // Reference a given text file as a collection of Strings
2 PCollection<String> lines = pipeline.readTextFile(inputPath);
3
4 // Define a function that splits each line in a PCollection of
5 // Strings into a PCollection made up of the individual words
6 // in those lines.
7 PCollection<String> words = lines.parallelDo(new Tokenizer(),
8       Writables.strings());
9
10
11
12
13 PTable<String, Long> counts = words.count();
14 pipeline.writeTextFile(counts, outputPath);
15 }
```



Specifies how to serialize the data in this collection.

User-defined class...

Crunch WordCount (3)

```
1  public static class Tokenizer extends DoFn<String, String> {  
2      @Override  
3      public void process(String line, Emitter<String> emitter) {  
4          for (String word : line.split(" ")) {  
5              emitter.emit(word);  
6          }  
7      }  
8  }  
9  }
```


Crunch advantages

- **DoFns** are like mappers, but many **DoFns** can be chained together in a single mapper, or in the reducer
- Crunch *planner* constructs a pipeline with the minimum number of MR jobs
 - `myTable.parallelDo(x).parallelDo(y).groupByKey().aggregate(Aggregators.sum()).parallelDo(z).writeToFile("/foo")` is only one MapReduce job!
- Makes it easy to write composable, re-usable, testable code
- Can run the same pipeline in MapReduce, or in-memory
 - In the future, can also run on new frameworks like Spark with no code change!

Apache Spark

A brief introduction

What is Spark?

Spark is a general purpose computational framework

Retains the advantages of MapReduce:

- Linear scalability
- Fault-tolerance
- Data Locality based computations



...but offers so much more:

- Leverages distributed memory for better performance
- Improved developer experience
- Full Directed Graph expressions for data parallel computations
- Comes with libraries for machine learning, graph analysis, etc

Spark vs Crunch

- Similar API – higher level functional transformations
- Spark is a service of its own: *not* a wrapper around MapReduce
 - In fact, Crunch can compile down to Spark jobs instead of MR jobs!
- More “batteries included”
 - Machine learning, graph computation, etc

Easy: Get Started Immediately

- Multi-language support
- Interactive Shell

Python

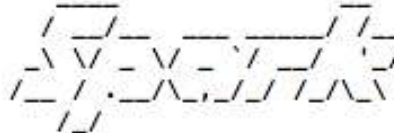
```
lines = sc.textFile(...)
lines.filter(lambda s: "ERROR" in s).count()
```

Scala

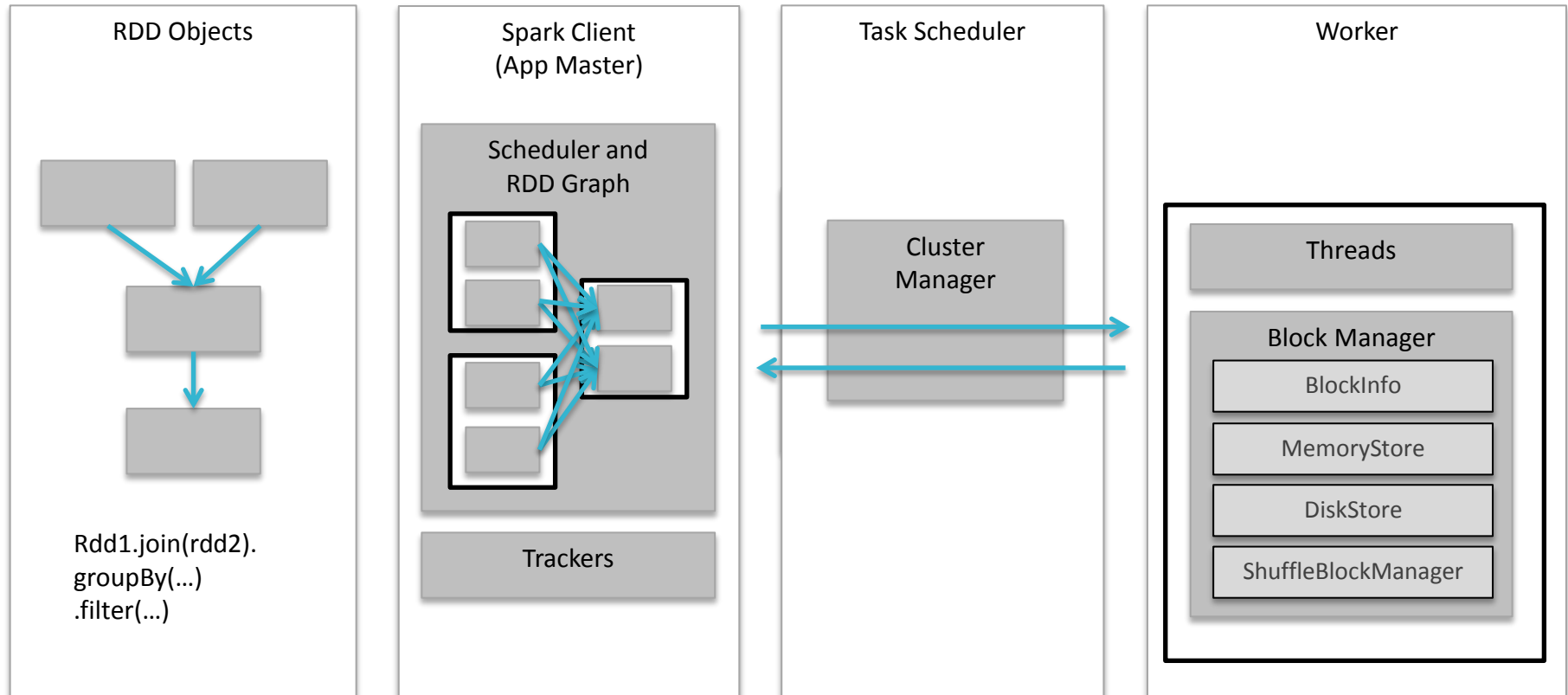
```
val lines = sc.textFile(...)
lines.filter(s => s.contains("ERROR")).count()
```

Java

```
JavaRDD<String> lines = sc.textFile(...);
lines.filter(new Function<String, Boolean>() {
    Boolean call(String s) {
        return s.contains("error");
    }
}).count();
```

```
root@ip-172-31-11-254:~  
[root@ip-172-31-11-254 ~]# /opt/cloudera/parcels/S...  
...  
Welcome to  
 version 0.8.0  
Using Python version 2.6.6 (r266:84292, Sep 11 2012)  
Spark context available as sc.  
...  
>>> file = sc.textFile("hdfs://ip-172-31-11-254.us-east-2.amazonaws.com/hdfs/ec2-data/pageviews/2007/2007-12/pagecounts-200712.txt")  
...  
>>> file.count()  
...  
856769  
>>> file.filter(lambda line: "Holiday" in line).count()  
...  
101
```

Spark from a High Level



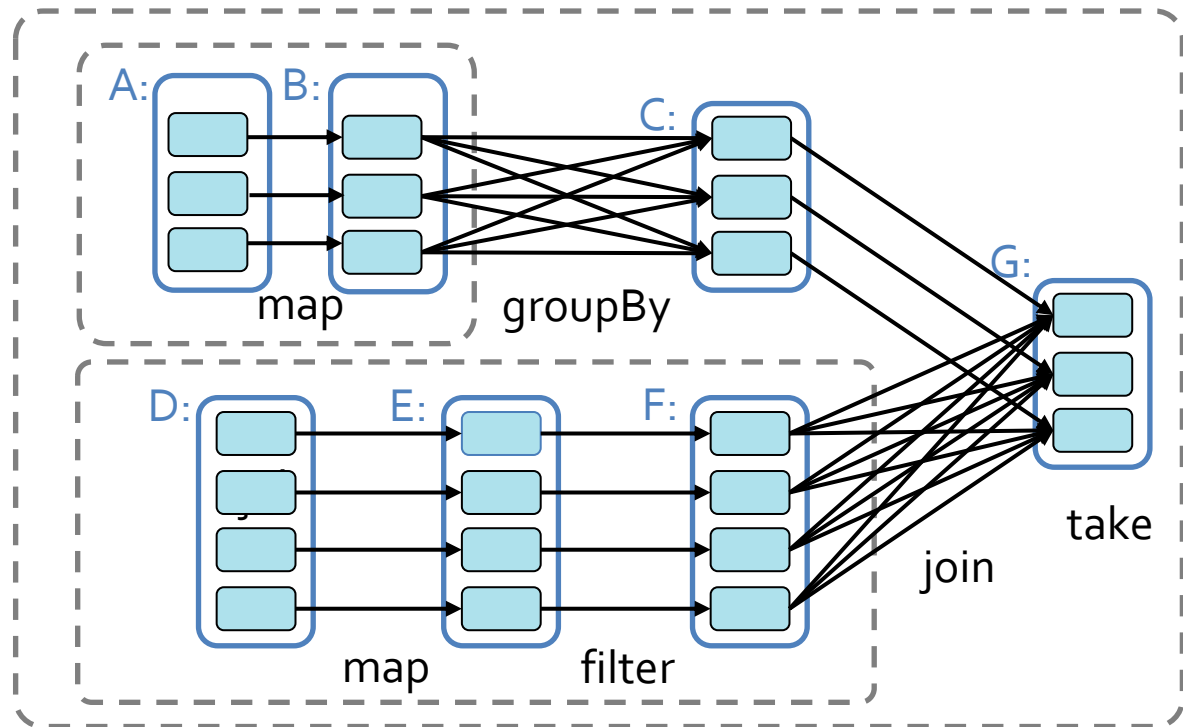
Resilient Distributed Datasets

- **Resilient Distributed Datasets**
- Collections of objects **distributed** across a cluster, stored in RAM or on Disk
 - Like a Crunch *PCollection*
- Built through parallel transformations
- Automatically rebuilt on failure (**resilient**)

Resilient Distributed Datasets

- **Building a Story**

- HadoopRDD
- JdbcRDD
- MappedRDD
- FlatMappedRDD
- FilteredRDD
- OrderedRDD
- PairRDD
- ShuffledRDD
- UnionRDD
-



Easy: Expressive API

- **Transformations**
 - Lazy (like Crunch)
 - Return RDDs
- **Actions**
 - Demand action
 - Return value or store data on HDFS

Easy: Expressive API

- Transformations

- Map
- Filter
- flatMap
- Union
- Reduce
- Sort
- Join
- ...

- Actions

- collect
- Count
- First
- Take(n)
- saveAsTextFile
- foreach(func)
- reduce(func)
- ...

Easy: Example – Word Count (Java version)


```
1 // Create a context: specifies cluster configuration, app name, etc.
2 JavaSparkContext sc = new JavaSparkContext(...);
3 JavaRDD<String> lines = sc.textFile("hdfs://...");
4
5 // Create a new RDD with each word separated
6 JavaRDD<String> words = lines.flatMap(
7     new FlatMapFunction<String, String>() {
8         public Iterable<String> call(String s) {
9             return Arrays.asList(s.split(" "));
10         }
11     });
12
13
14
15
```

Easy: Example – Word Count (Java version)

```
1  // Create an RDD with tuples like ("foo", 1)
2  JavaRdd<String> ones = words.map(
3      new PairFunction<String, String, Integer>() {
4          public Tuple2<String, Integer> call(String s) {
5              return new Tuple2(s, 1);
6          }
7      });
8
9  // Run the "reduce" phase to count occurrences
10 JavaPairRDD<String, Integer> counts = ones.reduceByKey(
11     new Function2<Integer, Integer, Integer>() {
12         public Integer call(Integer i1, Integer i2) {
13             return i1 + i2;
14         }
15     }
16 );
17 counts.saveAsTextFile("/out
```

Easy: Example – Word Count (Scala)

```
1 val spark = new SparkContext(master, appName, [sparkHome], [jars])
2 val file = spark.textFile("hdfs://...")
3 val counts = file.flatMap(line => line.split(" "))
4                     .map(word => (word, 1))
5                     .reduceByKey(_ + _)
6 counts.saveAsTextFile("hdfs://...")
```



Scala's type inference and lambda syntax make this very concise!

Easy: Out of the Box Functionality

- **Hadoop Integration**

- Works with Hadoop Data
- Runs With YARN

- **Libraries**

- MLlib
- Spark Streaming
- GraphX (alpha)

- **Roadmap**

- Language support:

- Improved Python support
- SparkR
- Java 8

- Better ML


- Sparse Data Support
- Model Evaluation Framework
- Performance Testing

Example: Logistic Regression

```
data = spark.textFile(...).map(parseDataPoint).cache()
w = numpy.random.rand(D)

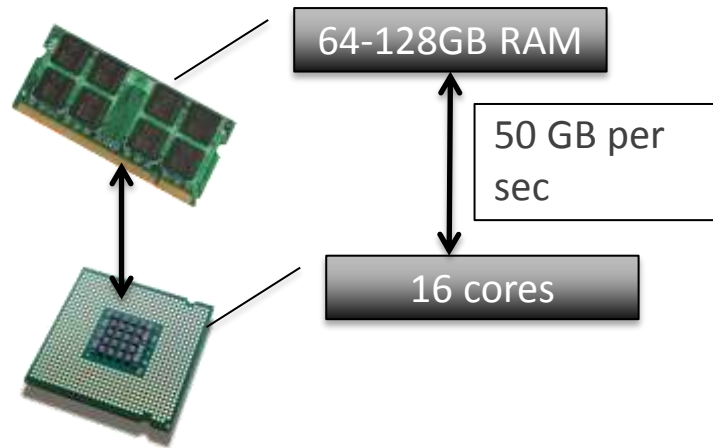
for i in range(iterations):
    gradient = data
        .map(lambda p: (1 / (1 + exp(-p.y * w.dot(p.x))))
            * p.y * p.x)
        .reduce(lambda x, y: x + y)
    w -= gradient

print "Final w: %s" % w
```



Spark will load the **parsed** dataset into memory!

Memory management leads to greater performance



- Hadoop cluster with 100 nodes contains 10+TB of RAM today and will double next year
- 1 GB RAM ~ \$10-\$20

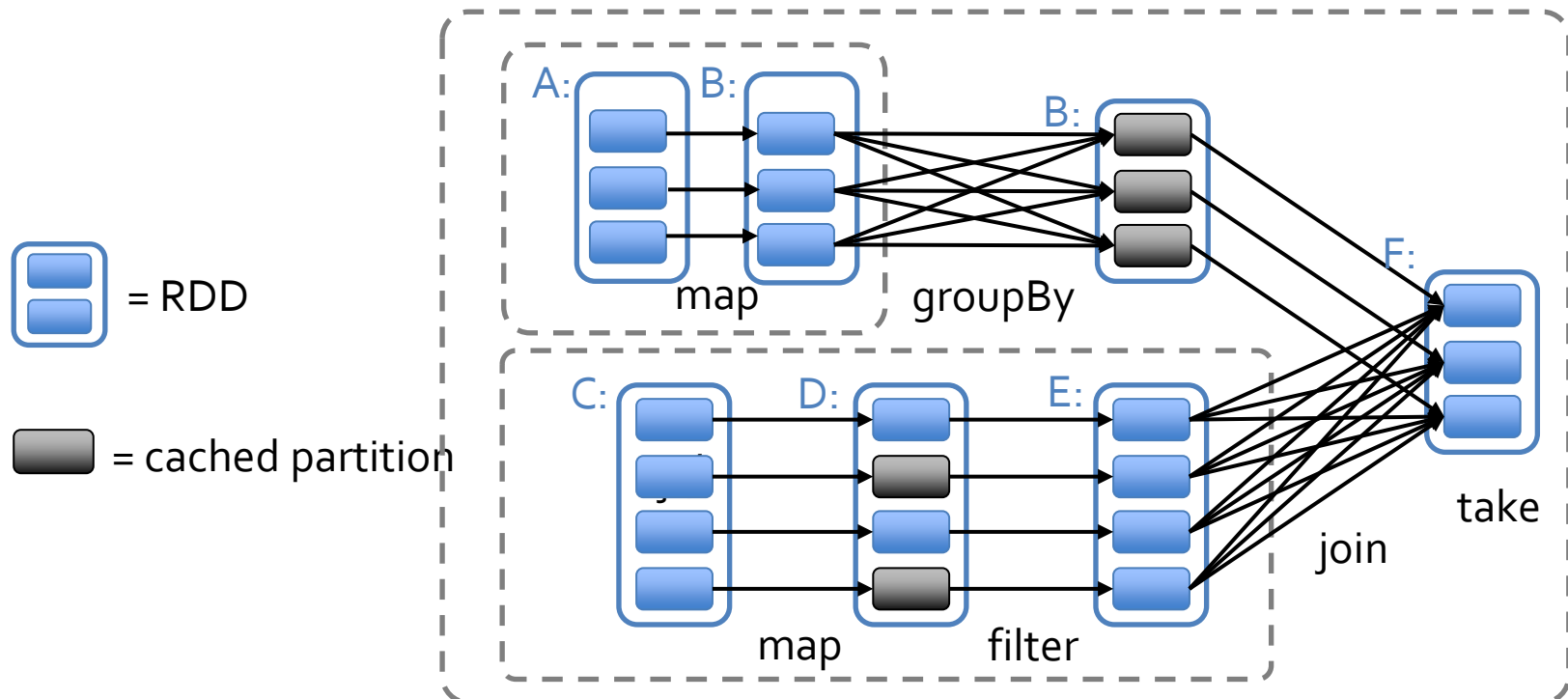
Trends:

½ price every 18 months
2x bandwidth every 3 years

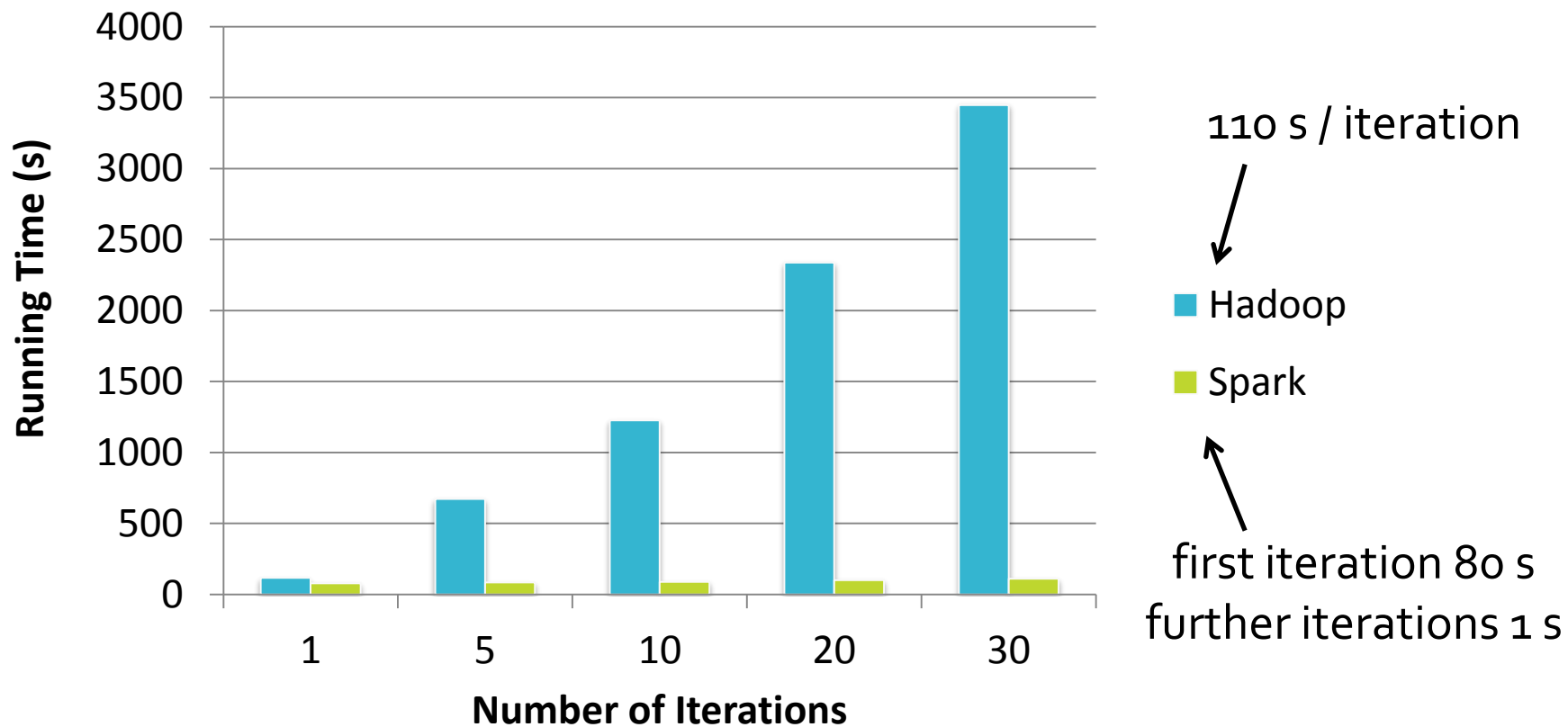
**Memory can be enabler for high performance
big data applications**

Fast: Using RAM, Operator Graphs

- **In-memory Caching**
 - Data Partitions read from RAM instead of disk
- **Operator Graphs**
 - Scheduling Optimizations
 - Fault Tolerance



Logistic Regression Performance (data fits in memory)



Spark MLlib

- “Batteries included” library for Machine Learning
 - Binary Classification (Support Vector Machines)
 - Linear and logistic regression
 - Collaborative filtering (recommendation engines)
 - Stochastic gradient descent
 - Clustering (K-means)

MLLib example (Scala)

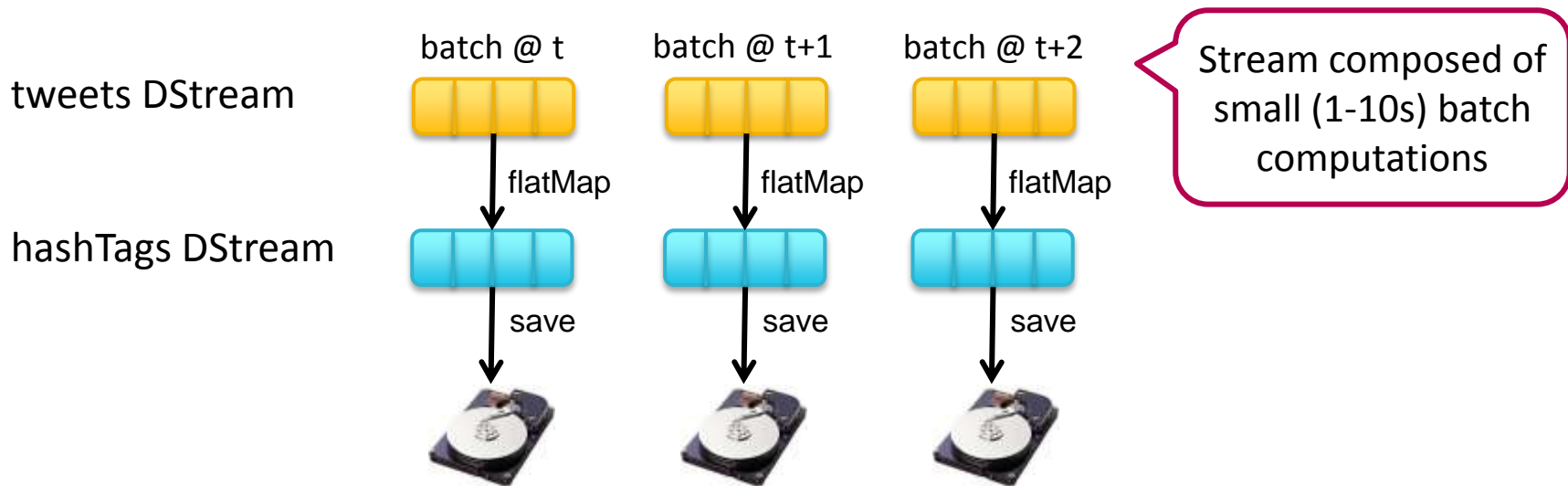
```
1  import org.apache.spark.mllib.clustering.KMeans
2
3  // Load and parse the data
4  val data = sc.textFile("kmeans_data.txt")
5  val parsedData = data.map( _.split(' ').map(_.toDouble))
6
7  // Cluster the data into two classes using KMeans
8  val numIterations = 20
9  val numClusters = 2
10 val clusters = KMeans.train(parsedData, numClusters, numIterations)
11
12 // Evaluate clustering by computing Within Set Sum of Squared Errors
13 val WSSSE = clusters.computeCost(parsedData)
14 println("Within Set Sum of Squared Errors = " + WSSSE)
```

Spark Streaming

- Run *continuous* processing of data using Spark's core API.
- Extends Spark concept of RDD's to *DStreams* (Discretized Streams) which are fault tolerant, transformable streams. Users can re-use existing code for batch/offline processing.
- Adds “rolling window” operations. E.g. compute rolling averages or counts for data over last five minutes.
- Example use cases:
 - “On-the-fly” ETL as data is ingested into Hadoop/HDFS.
 - Detecting anomalous behavior and triggering alerts.
 - Continuous reporting of summary metrics for incoming data.

“Micro-batch” Architecture

```
val tweets = ssc.twitterStream()  
val hashTags = tweets.flatMap (status => getTags(status))  
hashTags.saveAsHadoopFiles("hdfs://...")
```



Sample use cases

User	Use Case	Spark's Value
Conviva	Optimize end user's online video experience by analyzing traffic patterns in real-time, enabling fine-grained traffic shaping control.	<ul style="list-style-type: none">• Rapid development and prototyping• Shared business logic for offline and online computation• Open machine learning algorithms
Yahoo!	Speedup model training pipeline for Ad Targeting, increasing feature extraction phase by over 3x. Content recommendation using Collaborative filtering	<ul style="list-style-type: none">• Reduced latency in broader data pipeline.• Use of iterative ML• Efficient P2P broadcast
Anonymous (Large Tech Company)	"Semi real-time" log aggregation and analysis for monitoring, alerting, etc.	<ul style="list-style-type: none">• Low latency, frequently running "mini" batch jobs processing latest data sets
Technicolor	Real-time analytics for their customers (i.e., telcos); need ability to provide streaming results and ability to query them in real time	<ul style="list-style-type: none">• Easy to develop; just need Spark & SparkStreaming• Arbitrary queries on live data

Building on Spark Today

- **What kind of Apps?**

- ETL
- Machine Learning
- Streaming
- Dashboards

**Growing Number of Successful
Use Cases & 3rd Party
Applications**

Current project status

- Spark promoted to Apache TLP in mid-Feb
- 100+ contributors and 25+ companies contributing
- Includes: Intel, Yahoo, Quantifind, Bizo, etc
- Dozens of production deployments

Questions?

- Thank you for attending!
- I'll be happy to answer any additional questions now...
- <http://spark.apache.org/> has getting started guides, tutorials, etc.