Introduction to Spark

- MapReduce enables big data analytics using large, unreliable clusters

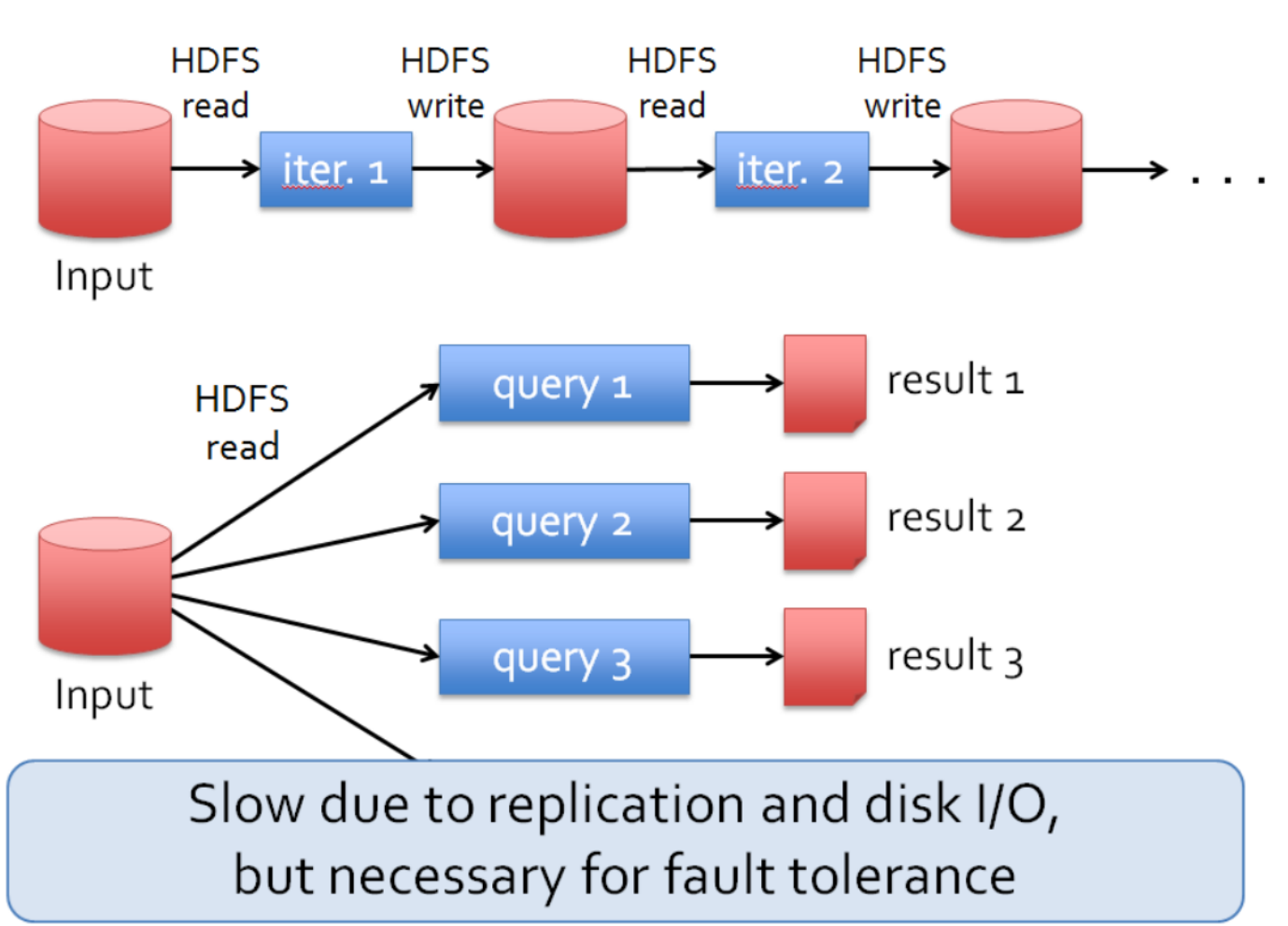
- There are multiple implementation of MapReduce
  - Based on HDFS
  - Based on No-SQL databases
  - Based on cloud

- Can not support Complex/iterative applications efficiently
- Root Cause: Inefficient data sharing

  Only way to share data across jobs is via stable storage
- There is room to further improvement with MapReduce

  - iterative algorithms,

  - interactive ad-hoc queries



Copied from Matei Zaharia, et.al., Resilient Distributed Datasets, A Fault-Tolerant Abstraction for In-Memory Cluster Computing, amplab, UC Berkeley

- In other words, MapReduce lacks efficient primitives for data sharing

- This is where Spark comes into the picture – instead of load the data from disk for every query, why not do In-Memory data sharing?

- Here is how Spark addresses this issue: Resilient Distributed Datasets (RDDs)

- RDD allows applications to keep working sets in memory for reuse.

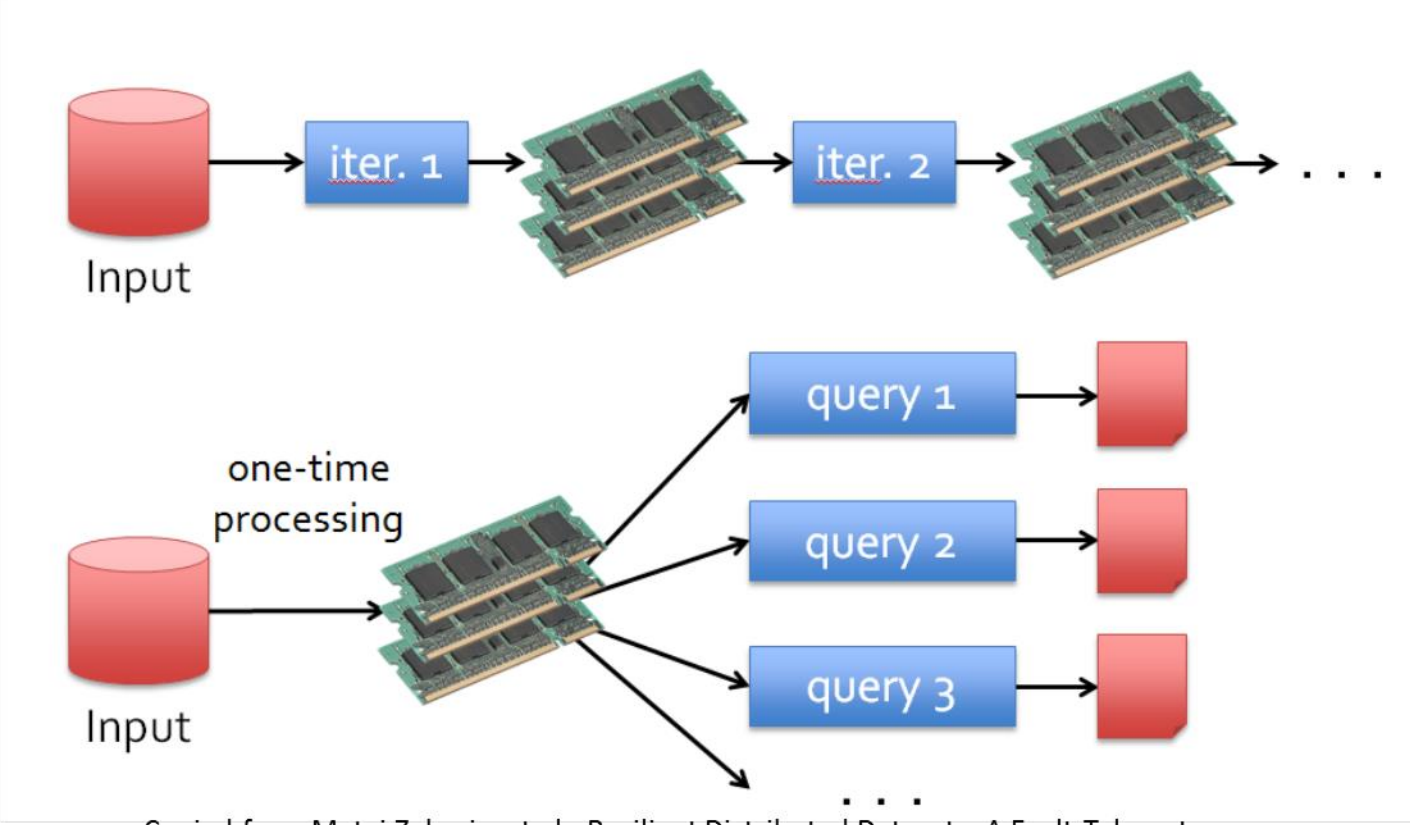
RDD Overview

- A new programming model (RDD)
  - parallel/distributed computing
  - in-memory sharing
  - fault-tolerance
- An implementation of RDD:

Spark's Philosophy

- Generalize MapReduce
- Richer Programming Model
  - ✓Fewer Systems to Master
- Better Memory Management
  - ✓Less data movement leads to better performance for complex analytics

Something like this……



Copied from Matei Zaharia, et.al., Resilient Distributed Datasets, A Fault-Tolerant Abstraction for In-Memory Cluster Computing, amplab, UC Berkeley

But memory is not reliable, how about FT?

- Spark's solution: Resilient Distributed Datasets (RDDs)
  - Read only, partitioned collections of objects
  - A distributed *immutable* array
  - Created through parallel transformations on data in stable storage
  - Can be cached for efficient reuse.

Cited from Matei Zahria, Spark fast, interactive, language-integrated cluster computing, amplab, UC Berkeley

- Log the operations that generates the RDD
- Automatically rebuilt on (partial) failure

Spark API

- Spark provides API to support the following languages: Scala, Java, Python, R

- Data Operations
  - Transformations; lazily create RDDs

    wc = dataset.flatMap(tokenize).reduceByKey(add)
  - Actions; execute compilation

    wc.collect()

Abstraction: Dataflow Operators

• **map**	• **reduce**	• **sample**
• **filter**	• count	• **take**
• **groupBy**	• fold	• **first**
• **sort**	• reduceByKey	• **partitionBy**
• **union**	• groupByKey	• **mapWith**
• **join**	• cogroup	• **pipe**
• **leftOuterJoin** • **rightouterjoin**	• **Cross** • **Zip**	• **Save** • **. . .**

Spark Example: Log Mining

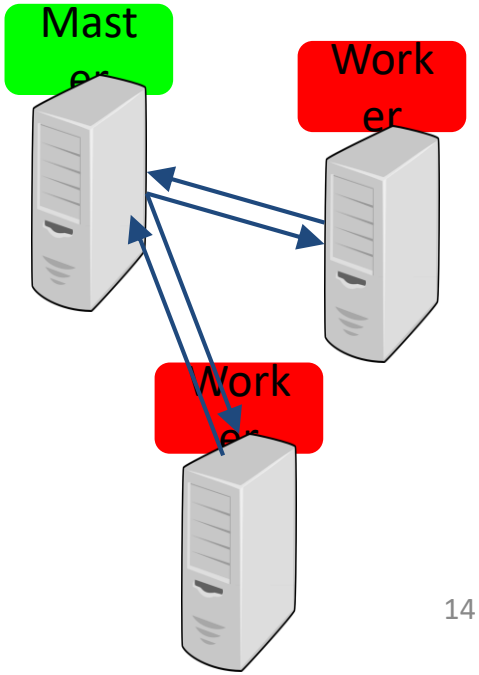
- Load error messages from a log into memory and run interactive queries

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
errors.persist()
errors.filter(_.contains("Foo")).count()
errors.filter(_.contains("Bar")).count()
```

base RDD

transformation

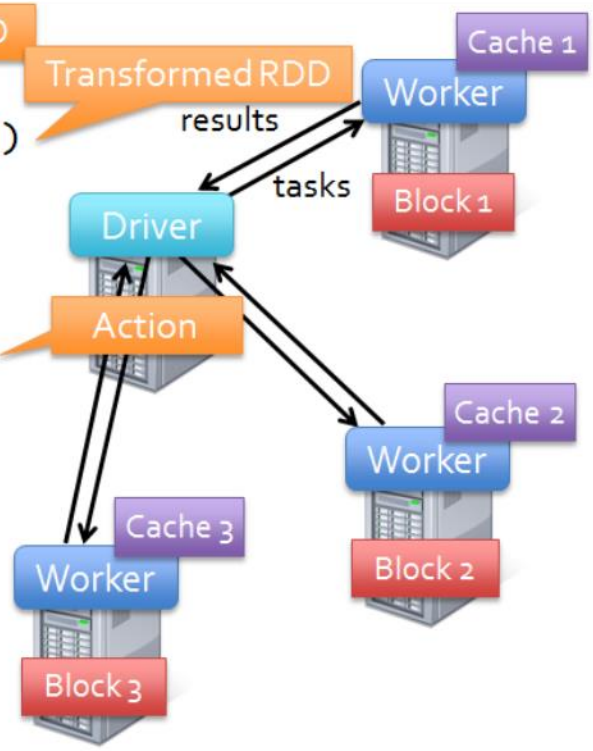
action !



Result: full-text search on 1TB data in 5-7sec vs. 170sec with on-disk data!

Simple Example

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
messages = errors.map(_.split('\t')(2))
cachedMsgs = messages.cache()

cachedMsgs.filter(_.contains("foo")).count
cachedMsgs.filter(_.contains("bar")).count
. . .
```



Copied from Matei Zahria, et.al., Spark fast, interactive, language-integrated cluster computing, amplab, UC Berkeley

Simple Yet Powerful

WordCount Implementation: Hadoop vs. Spark

```java
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);
      }
    }
  }

  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);
    }
  }

  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

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

Pregel: iterative graph processing, 200 LoC using Spark

HaLoop: iterative MapReduce, 200 LoC using Spark

WordCount

- val counts = sc.textFile(“hdfs://…”).flatMap(line=>line.split(“”)).map(word=>(word, 1L)).reduceByKey(_+_)

What is Iterative Algorithm?

data = input data

w = <target vector> -(Shared data structure)

At each iteration,

Do something to item in data:

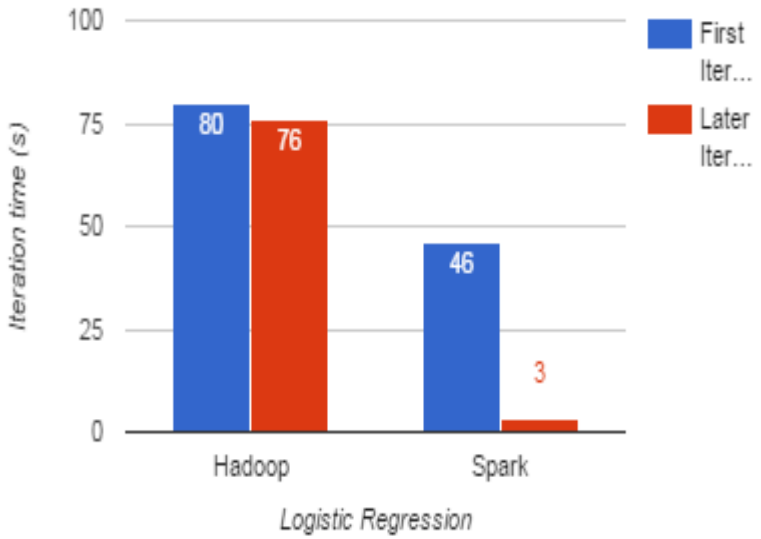
Update (w) -(Update shared data structure)

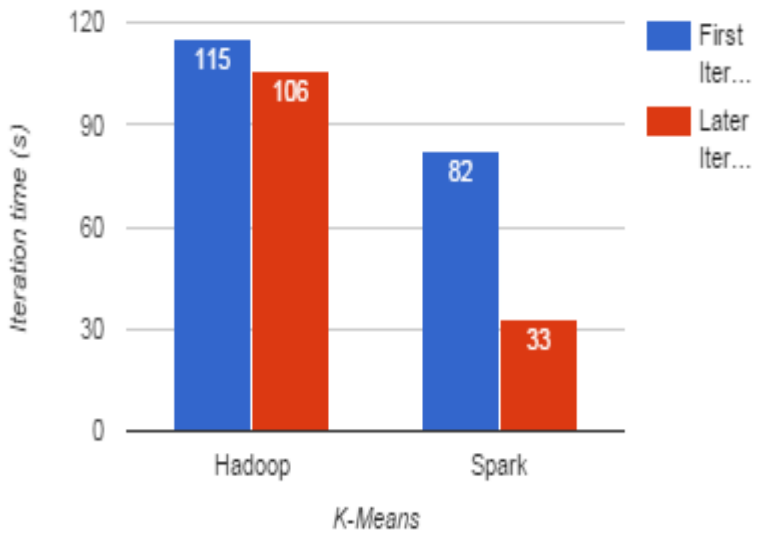
```
val data = spark.textFile(...).map(readPoint).cache()

var w = Vector.random(D)

for (i <- 1 to ITERATIONS) {
  val gradient = data.map(p =>
    (1 / (1 + exp(-p.y*(w dot p.x))) - 1) * p.y * p.x
  ).reduce(_ + _)
  w -= gradient
}

println("Final w: " + w)
```

Copied from Matei Zahria, et.al., Spark fast, interactive, language-integrated cluster computing, amplab, UC Berkeley

Evaluation

10 iterations on 100GB data using 25-100 machines





Spark Ecosystem

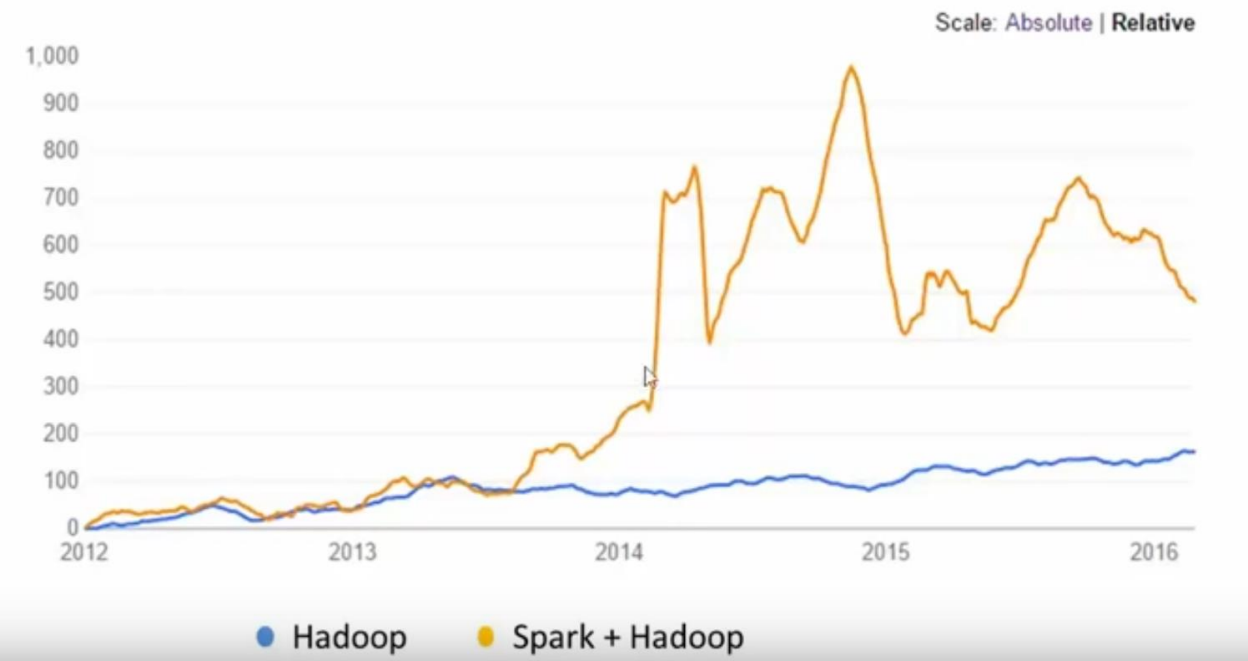
Copied from Matei Zaharia Spark's Role in the Big Data Ecosystem, Databricks

- Spark Core: is the execution engine for the spark platform. It provides distributed in-memory computing capabilities
- Spark SQL: is an engine for Hadoop Hive that enables unmodified Hadoop Hive queries to run up to 100x faster on existing deployments and data
- Spark Streaming: is an engine that enables powerful interactive and analytical applications on streaming data
- MLLib: is a scalable machine learning library
- GraphX is a graph computation engine built on top of Spark

Hadoop Vs Spark

- Spark is a computing framework that can be deployed upon Hadoop.

- You can view Hadoop as an operating system for a distributed computing cluster, while Spark is an application running on the system to provide in-memory analytics functions.


Scale: Absolute | Relative
1,000
900
800
700
600
500
400
300
200
100
0
2012
2013
2014
2015
2016
Hadoop
Spark + Hadoop


SAP
Pivotal
Hortonworks
STRATIO
ORACLE
guavus
cloudera
bluedata
MicroStrategy
Typesafe
elasticsearch.
pentaho
ADATAO
tresata
Atigeo
INUBE
ZOOMDATA
Qlik
Alpine
APERVI
DIYOTTA
talend


tableau
MicroStrategy
Qlik
elasticsearch.
pentaho
talend
tresata
TRIFACTA
SKYTREE
THE MACHINE LEARNING COMPANY
Alpine
atscale
looker
technicolor
virdata
FAiMDATA
ADATAO
DATA INTELLIGENCE FOR ALL
DiYOTTA
ZOOMDATA
platfora
APERVI
INUBE
Atigeo
日志易
rizhiyi.com
ZALONI
Typesafe