Spark and Shark: High-speed In-memory Analytics over Hadoop Data

May 14, 2013 @ Oracle

Reynold Xin, AMPLab, UC Berkeley

The Big Data Problem

Data is growing faster than computation speeds

Accelerating data sources

» Web, mobile, scientific, …

Cheap storage

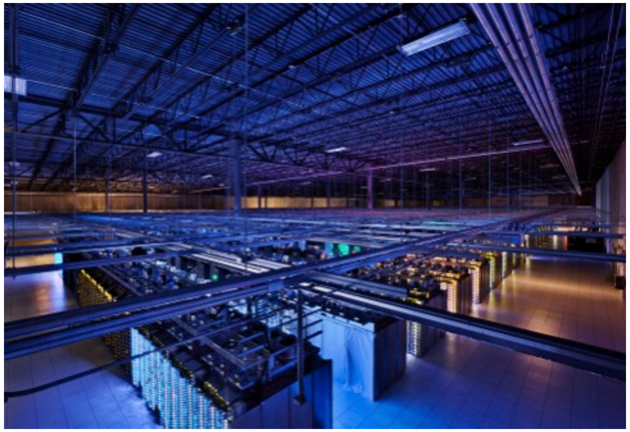
Stalling clock rates

Result

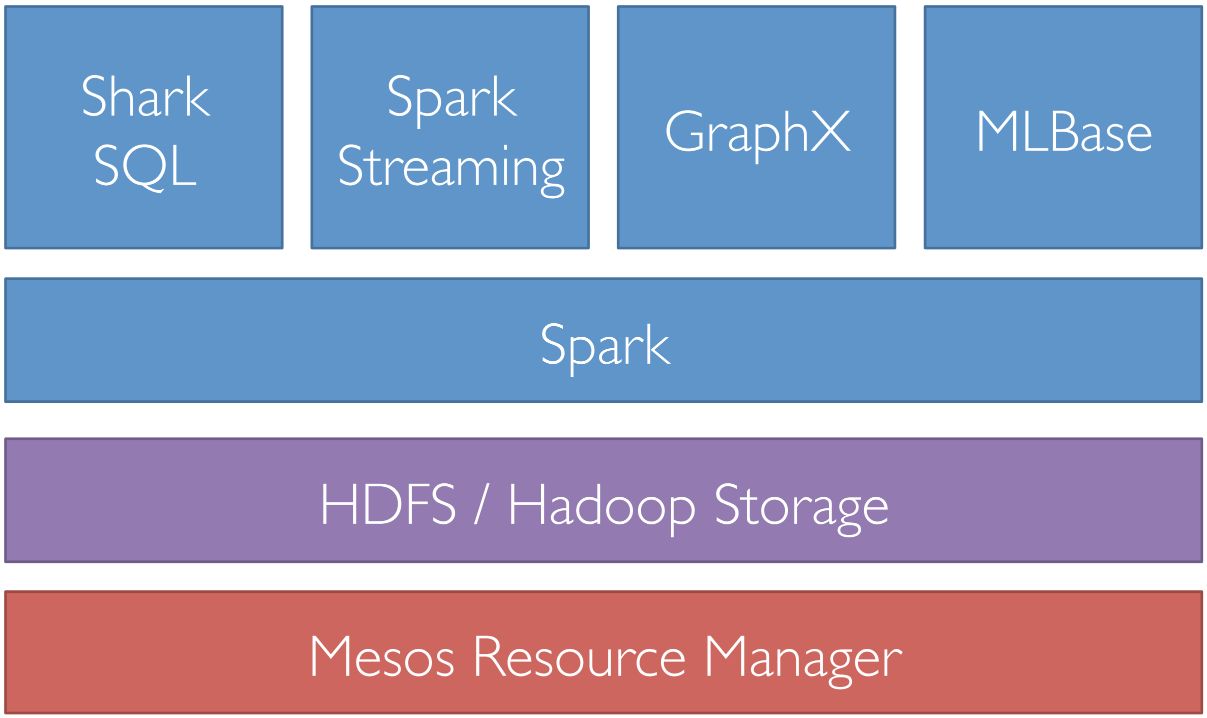
Processing has to scale out over large clusters

Users are adopting a new class of systems

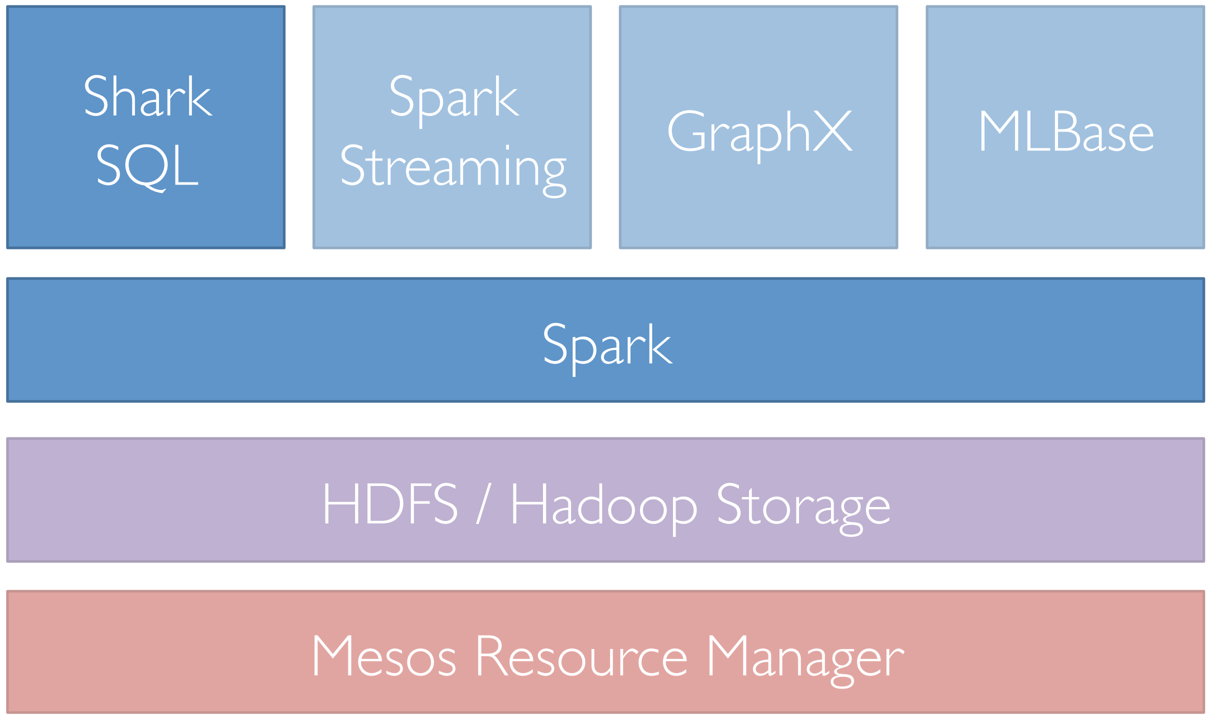
- » Hadoop MapReduce now used at banks, retailers, …
- » $1B market by 2016



Berkeley Data Analytics Stack



Today's Talk


Shark SQL
Spark Streaming
GraphX
MLBase
Spark
HDFS / Hadoop Storage
Mesos Resource Manager

Spark

Separate, fast, MapReduce-like engine

» In-memory storage for fast iterative computations

» General execution graphs

» Up to 100X faster than Hadoop MapReduce

Compatible with Hadoop storage APIs

» Read/write to any Hadoop-supported systems, including HDFS, Hbase, SequenceFiles, etc

Shark

An analytics engine built on top of Spark

» Support both SQL and complex analytics

» Up to 100X faster than Apache Hive

Compatible with Hive data, metastore, queries

» HiveQL

» UDF / UDAF

» SerDes

» Scripts

Community

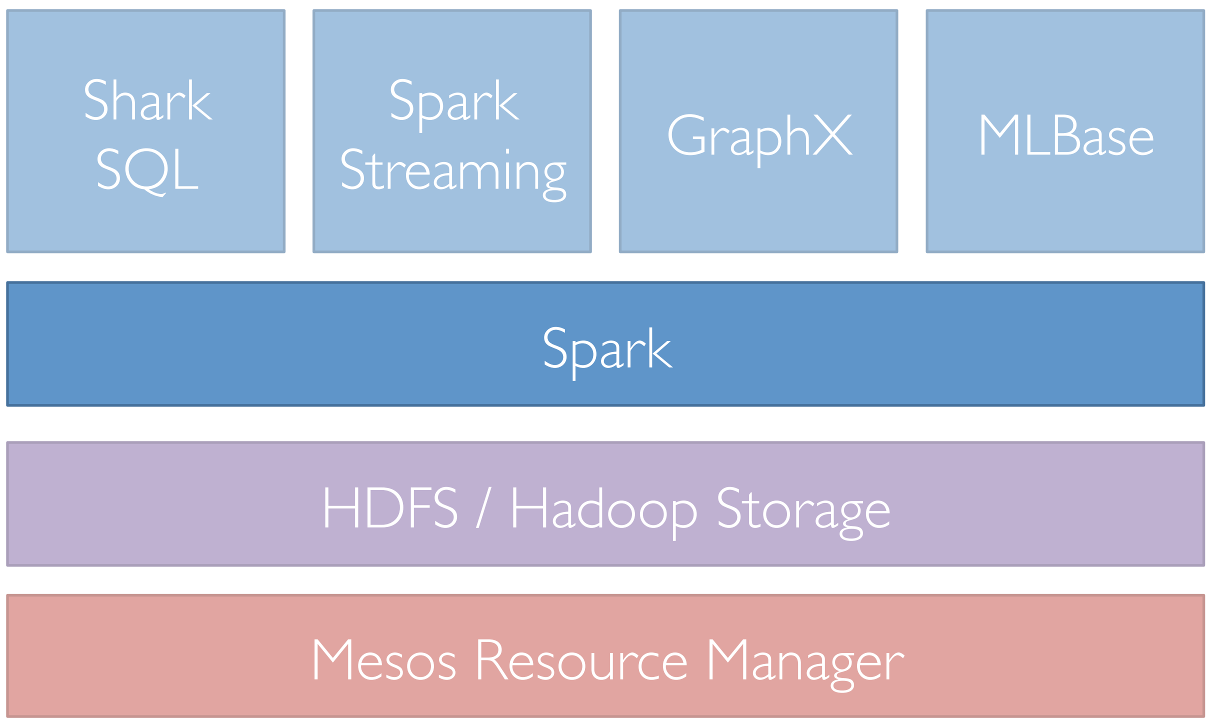
3000 people attended online training

800 meetup members

14 companies contributing

spark-project.org

Today's Talk


Shark SQL
Spark Streaming
GraphX
MLBase
Spark
HDFS / Hadoop Storage
Mesos Resource Manager

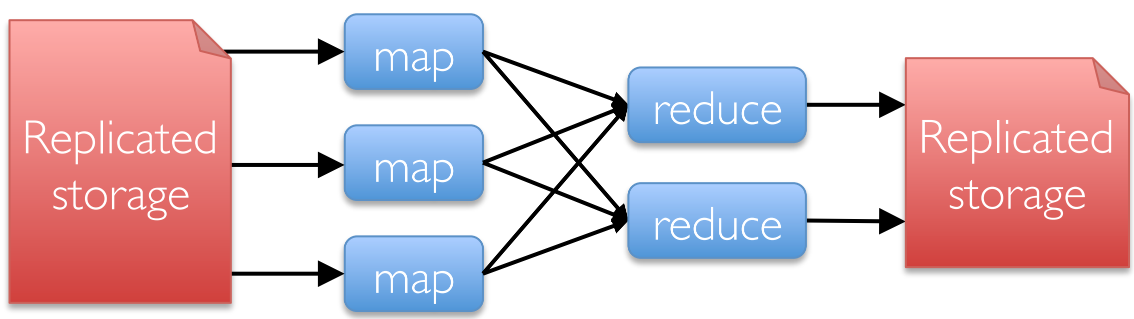
Background

Two things make programming clusters hard:

» **Failures:** amplified at scale (1000 nodes ➔ 1 fault/day)

» **Stragglers:** slow nodes (e.g. failing hardware)

MapReduce brought the ability to handle these automatically



Spark Motivation

MapReduce simplified batch analytics, but users quickly needed more:

» More **complex**, multi-pass applications (e.g. machine learning, graph algorithms)

» More **interactive** ad-hoc queries

» More **real-time** stream processing

One Reaction

Specialized models for some of these apps

- » Google Pregel for graph processing
- » Iterative MapReduce
- » Storm for streaming

Problem:

- » Don't cover all use cases
- » How to *compose* in a single application?

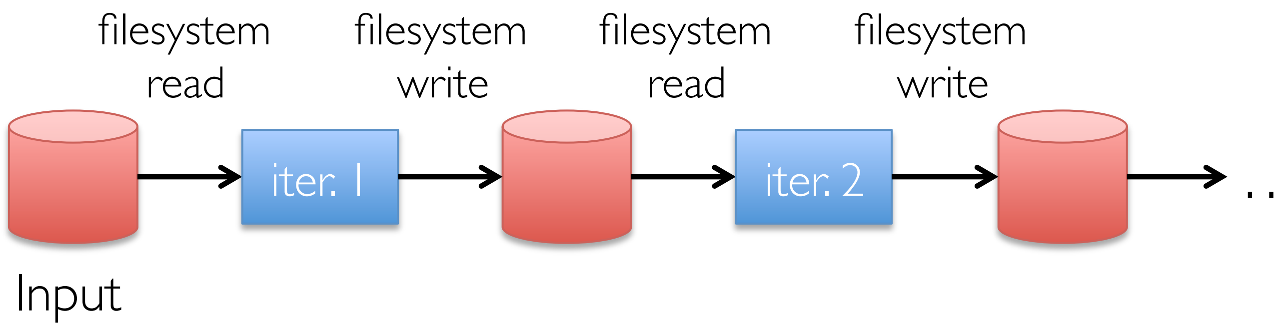
Observation

Complex, streaming and interactive apps all need one thing that MapReduce lacks:

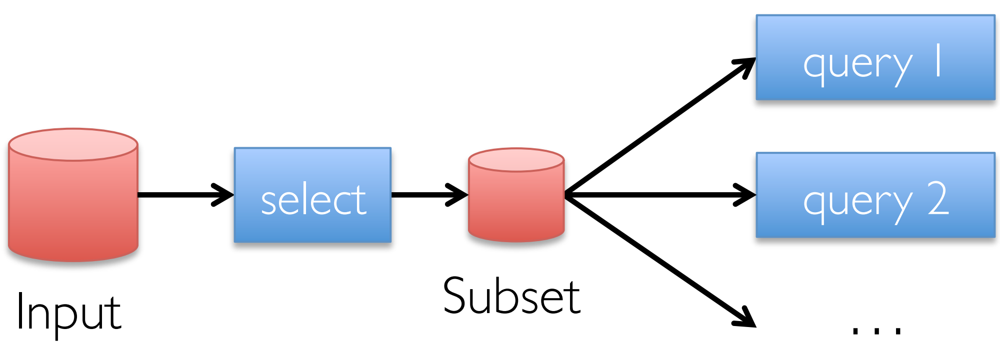
Efficient primitives for **data sharing**

Examples

Iterative:



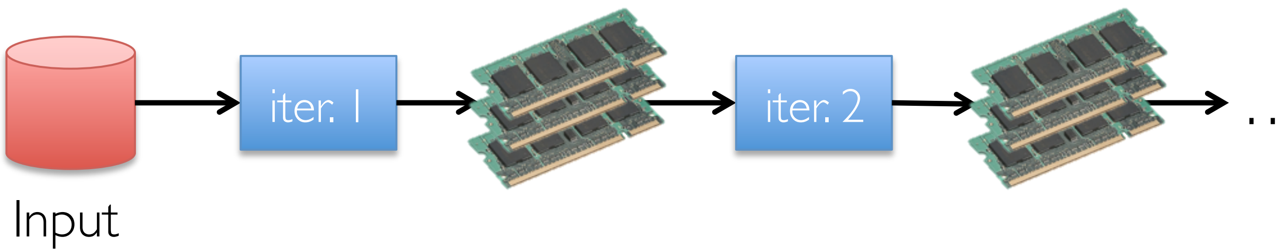
Interactive:



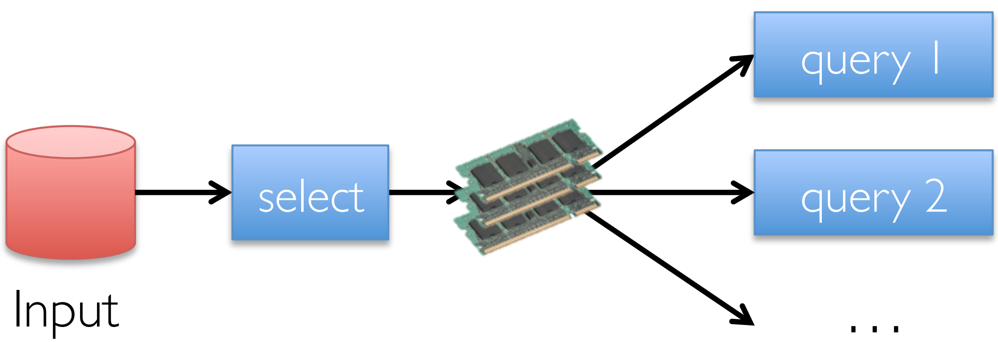
Slow due to replication and disk I/O, but necessary for fault tolerance

Goal: Sharing at Memory Speed

Iterative:



Interactive:



10-100x faster than network/disk, but
how to make fault-tolerant?

Existing Storage Systems

Based on a general "shared memory" model

» Fine-grained updates to mutable state

» E.g. databases, key-value stores, RAMCloud

Requires replicating data across the network for fault tolerance

» 10-100× slower than memory write!

Can we provide fault tolerance without replication?

Solution: Resilient Distributed Datasets (RDDs)

Restricted form of shared memory

- » Immutable, partitioned sets of records
- » Can only be built through *coarse-grained*, deterministic operations (map, filter, join, …)

Enables fault recovery using *lineage*

- » Log one operation to apply to many elements
- » Recompute any lost partitions on failure

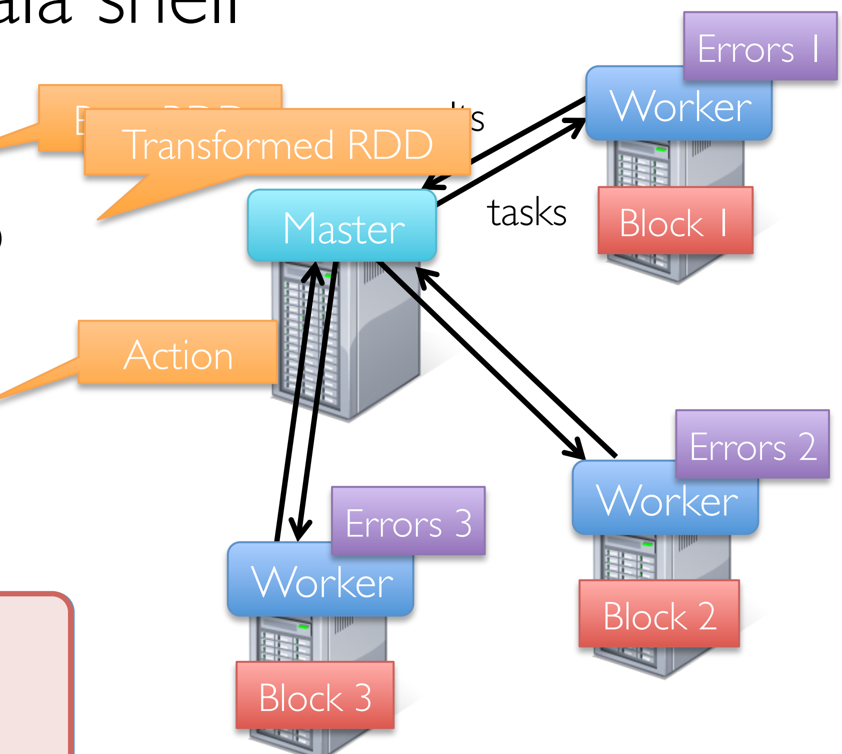
[NSDI 2012]

Example: Log Mining

Exposes RDDs through a functional API in Scala

Usable interactively from Scala shell

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

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



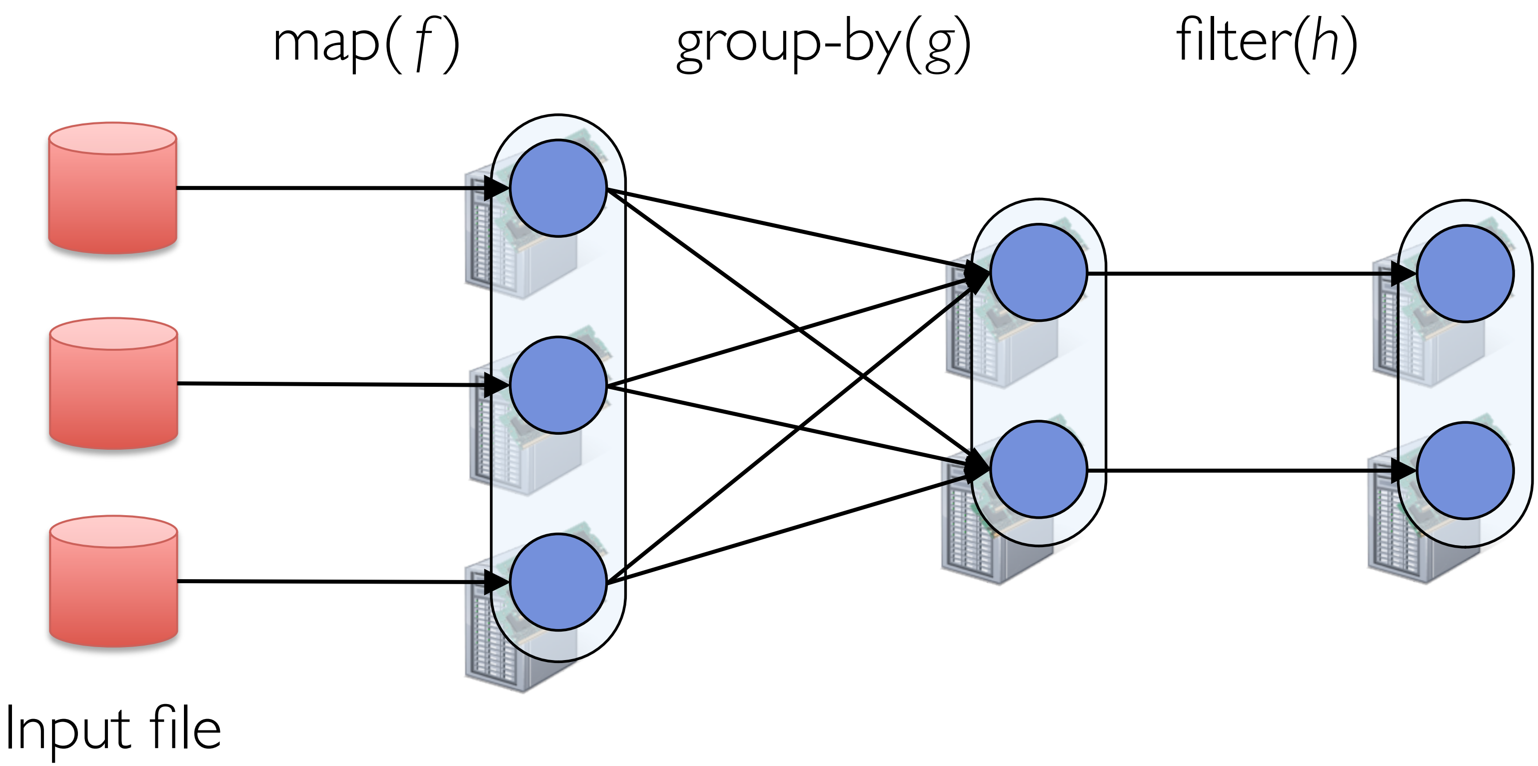
Result: 1 TB data in 5 sec
(vs 170 sec for on-disk data)

```java
public static class WordCountMapClass extends MapReduceBase
  implements 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,
                  OutputCollector<Text, IntWritable> output,
                  Reporter reporter) throws IOException {
    String line = value.toString();
    StringTokenizer itr = new StringTokenizer(line);
    while (itr.hasMoreTokens()) {
      word.set(itr.nextToken());
      output.collect(word, one);
    }
  }
}

public static class WorkdCountReduce extends MapReduceBase
  implements Reducer<Text, IntWritable, Text, IntWritable> {

  public void reduce(Text key, Iterator<IntWritable> values,
                     OutputCollector<Text, IntWritable> output,
                     Reporter reporter) throws IOException {
    int sum = 0;
    while (values.hasNext()) {
      sum += values.next().get();
    }
    output.collect(key, new IntWritable(sum));
  }
}
```

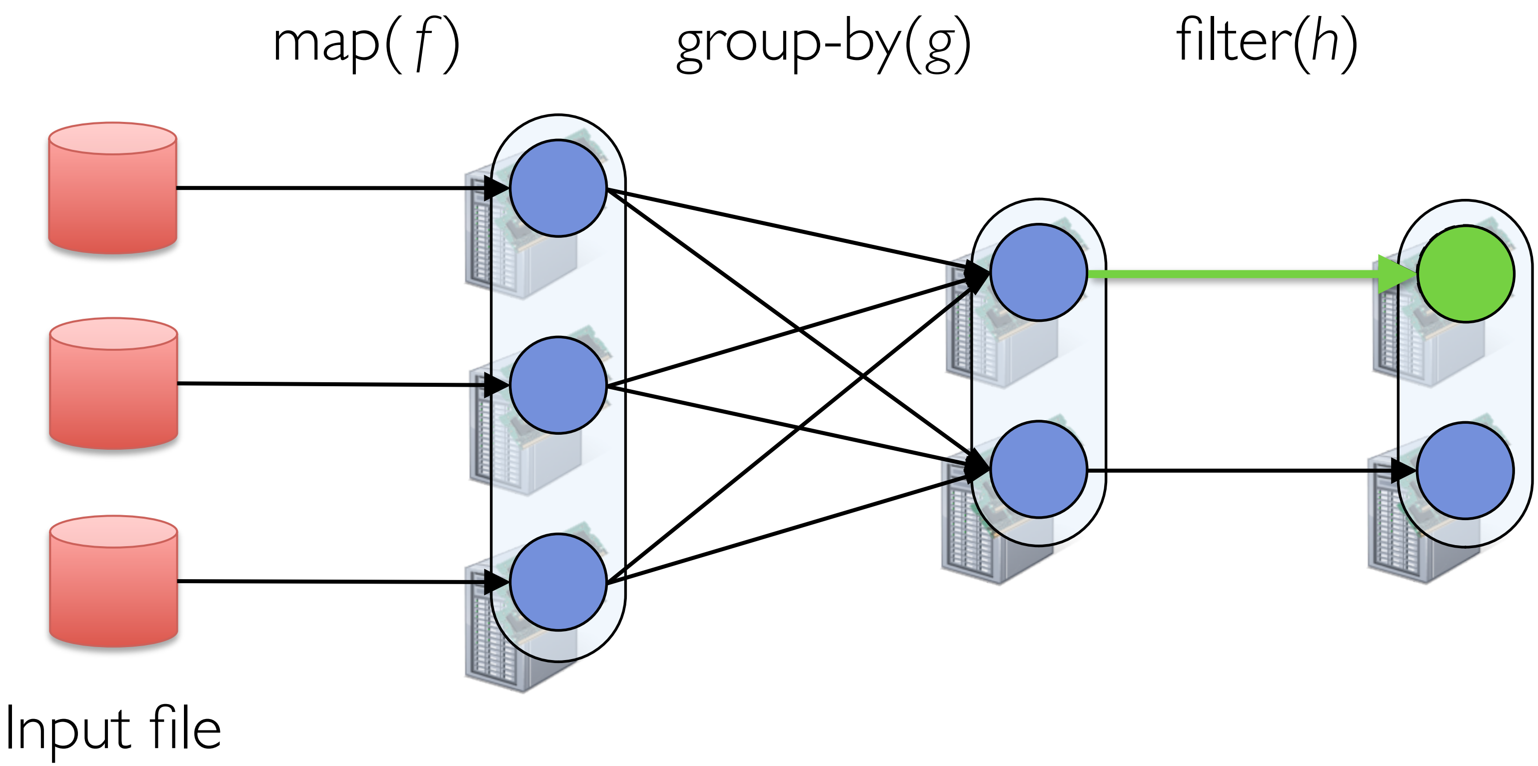
Word Count

```
val docs = sc.textFiles("hdfs://…")

docs.flatMap { doc => doc.split("\s") }
  .map { word => (word, 1) }
  .reduceByKey { case(v1, v2) => v1 + v2 }


docs.flatMap(_.split("\s"))
  .map((_, 1))
  .reduceByKey(_ + _)
```

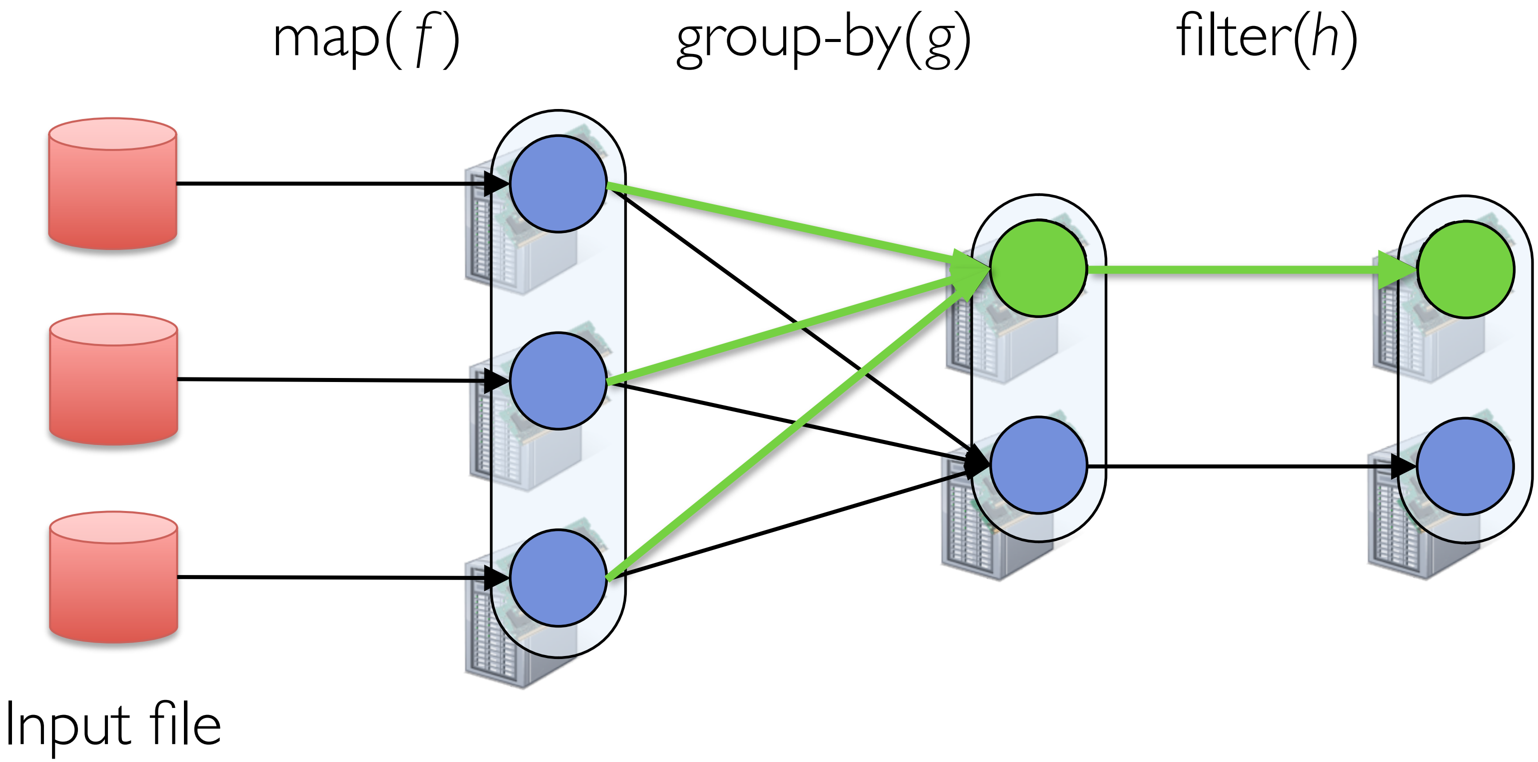
RDD Recovery



RDD Recovery



RDD Recovery



Generality of RDDs

Despite their restrictions, RDDs can express surprisingly many parallel algorithms

» These naturally *apply the same operation to many items*

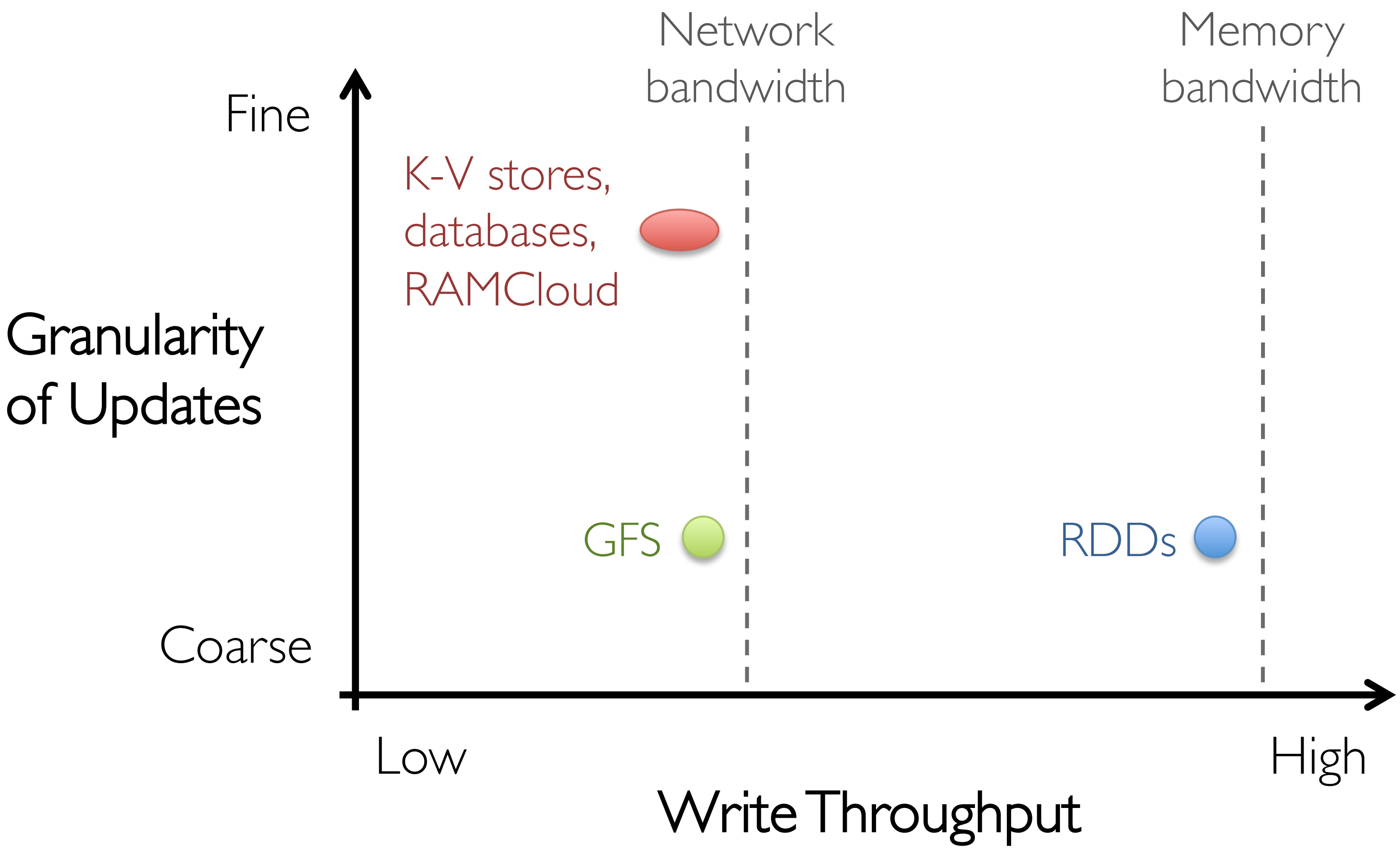
Unify many current programming models

» *Data flow models:* MapReduce, Dryad, SQL, …

» *Specialized models* for iterative apps: Pregel, iterative MapReduce, GraphLab, …

Support new apps that these models don't

Tradeoff Space



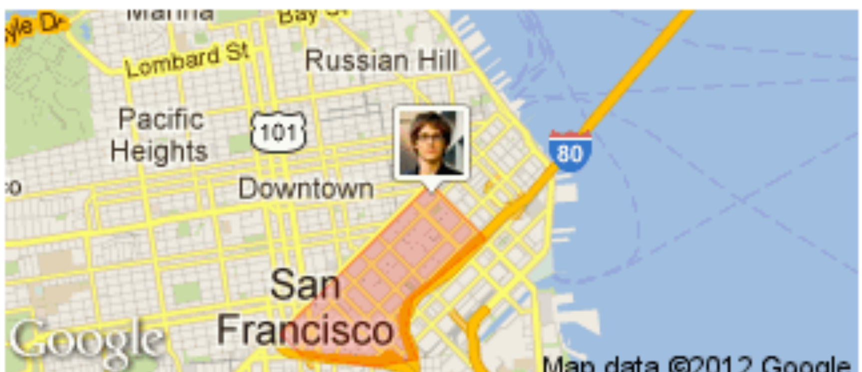
Jure Leskovec
@jure

Following

Median Hadoop job input data size at Microsoft, Yahoo and Facebook is only about 15gb!
research.microsoft.com/pubs/163083/ho...

Reply · Retweeted · Favorite · Pocket

36 RETWEETS · 23 FAVORITES



from SoMa
San Francisco, CA

4:33 PM - 9 Jul 12 via Twitter for iPhone · Embed this Tweet

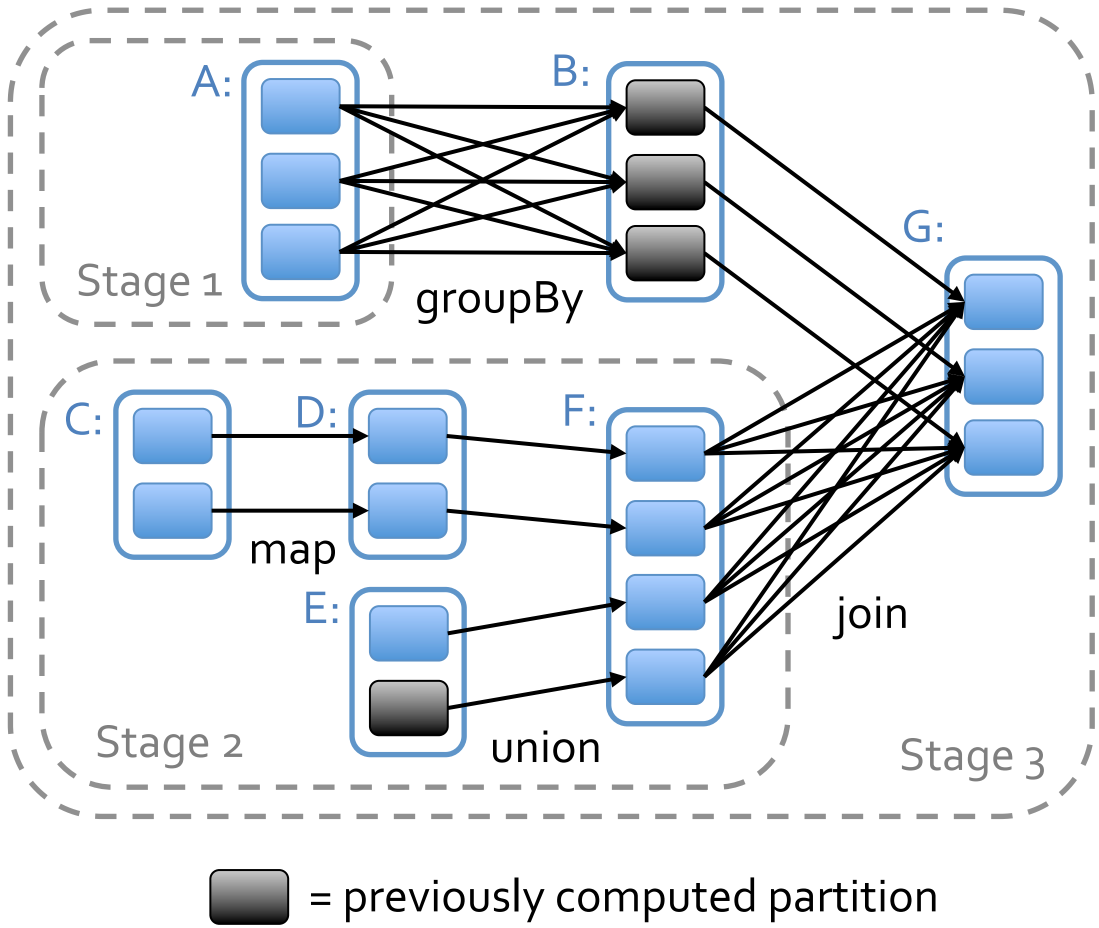
Scheduler

Dryad-like task DAG

Pipelines functions within a stage

Cache-aware data locality & reuse

Partitioning-aware to avoid shuffles



Example: Logistic Regression

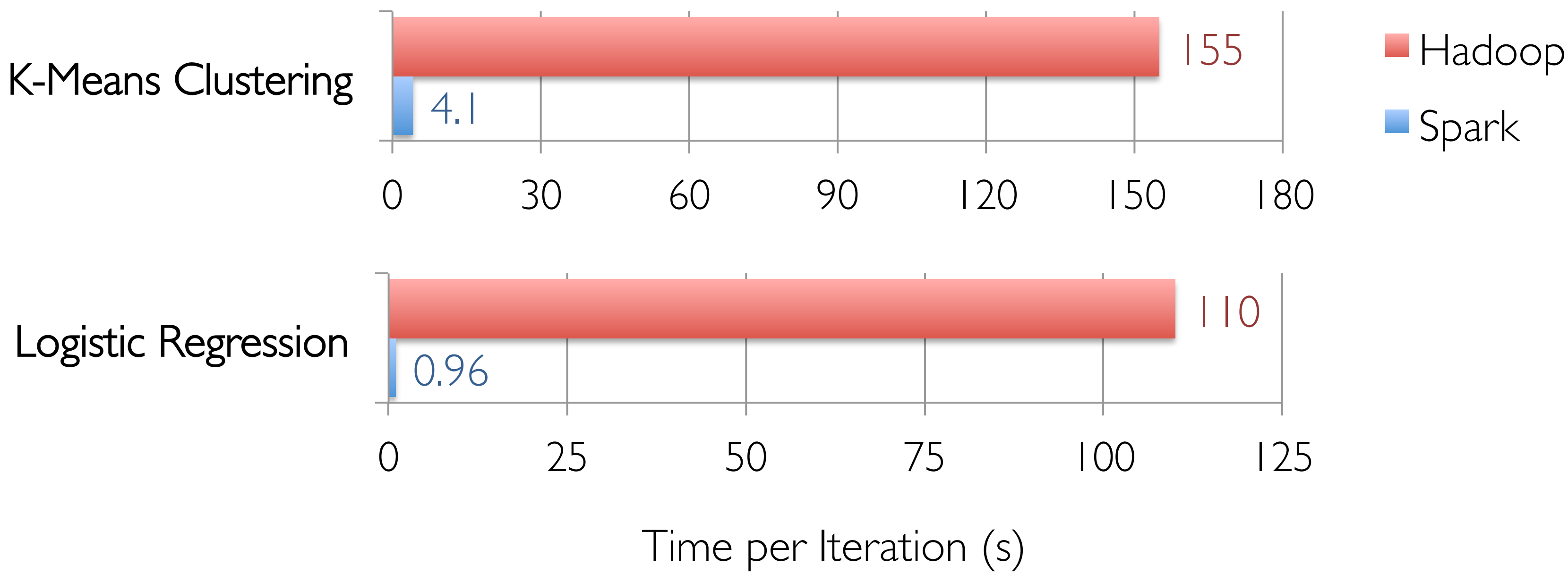
```
val data = spark.textFile(...).map(readPoint).cache()

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)
```

Load data in memory once

Initial parameter vector

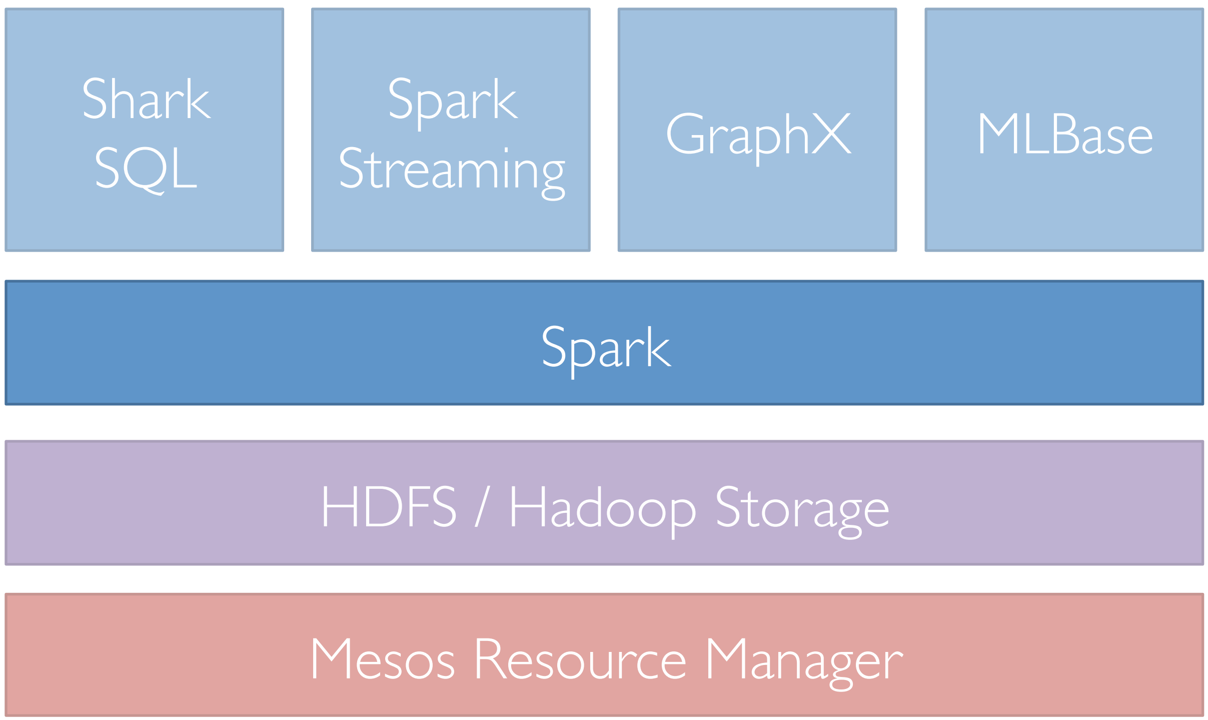
Repeated MapReduce steps to do gradient descent

Iterative Algorithms

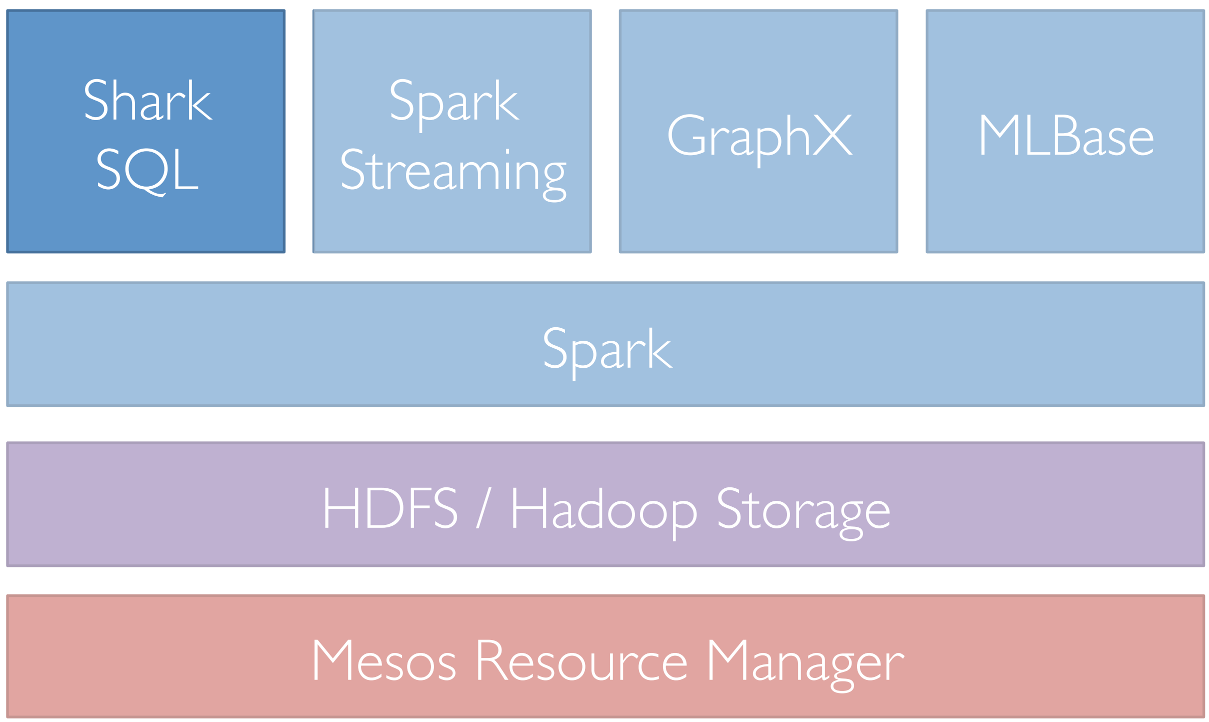


Similar speedups to other in-memory engines (e.g. Piccolo), but offers fine-grained fault tolerance

Spark


Shark SQL
Spark Streaming
GraphX
MLBase
Spark
HDFS / Hadoop Storage
Mesos Resource Manager

Shark


Shark SQL
Spark Streaming
GraphX
MLBase
Spark
HDFS / Hadoop Storage
Mesos Resource Manager

MPP Databases

Oracle, Vertica, HANA, Teradata, Dremel…

Pros

» Very mature and highly optimized engine.

» Fast!

Cons

» Generally not fault-tolerant; challenging for long running queries as clusters scale up

» Lack rich analytics (machine learning)

MapReduce

Hadoop, Hive, Google Tenzing, Turn Cheetah…

Pros

» Deterministic, idempotent tasks enable fine-grained fault-tolerance

» Beyond SQL (machine learning)

Cons

» High-latency, dismissed for interactive workloads

MapReduce: A major step backwards

By David DeWitt on January 17, 2008 4:20 PM | Permalink | Comments (44) | TrackBacks (1)
[Note: Although the system attributes this post to a single author, it was written by David J. DeWitt and Michael Stonebraker]

On January 8, a Database Column reader asked for our views on new distributed database research efforts, and we'll begin here v
to discuss it, since the recent trade press has been filled with news of the revolution of so-called "cloud computing." This paradigr
processors working in parallel to solve a computing problem. In effect, this suggests constructing a data center by lining up a large
much smaller number of high-end servers.

For example, IBM and Google have announced plans to make a 1,000 processor cluster available to a few select universities to te
software tool called MapReduce [1]. Berkeley has gone so far as to plan on teaching their freshman how to program using the Ma

As both educators and researchers, we are amazed at the hype that the MapReduce proponents have spread about how it represen
data-intensive a[...]ood idea for writing certa[...]tations, but to the

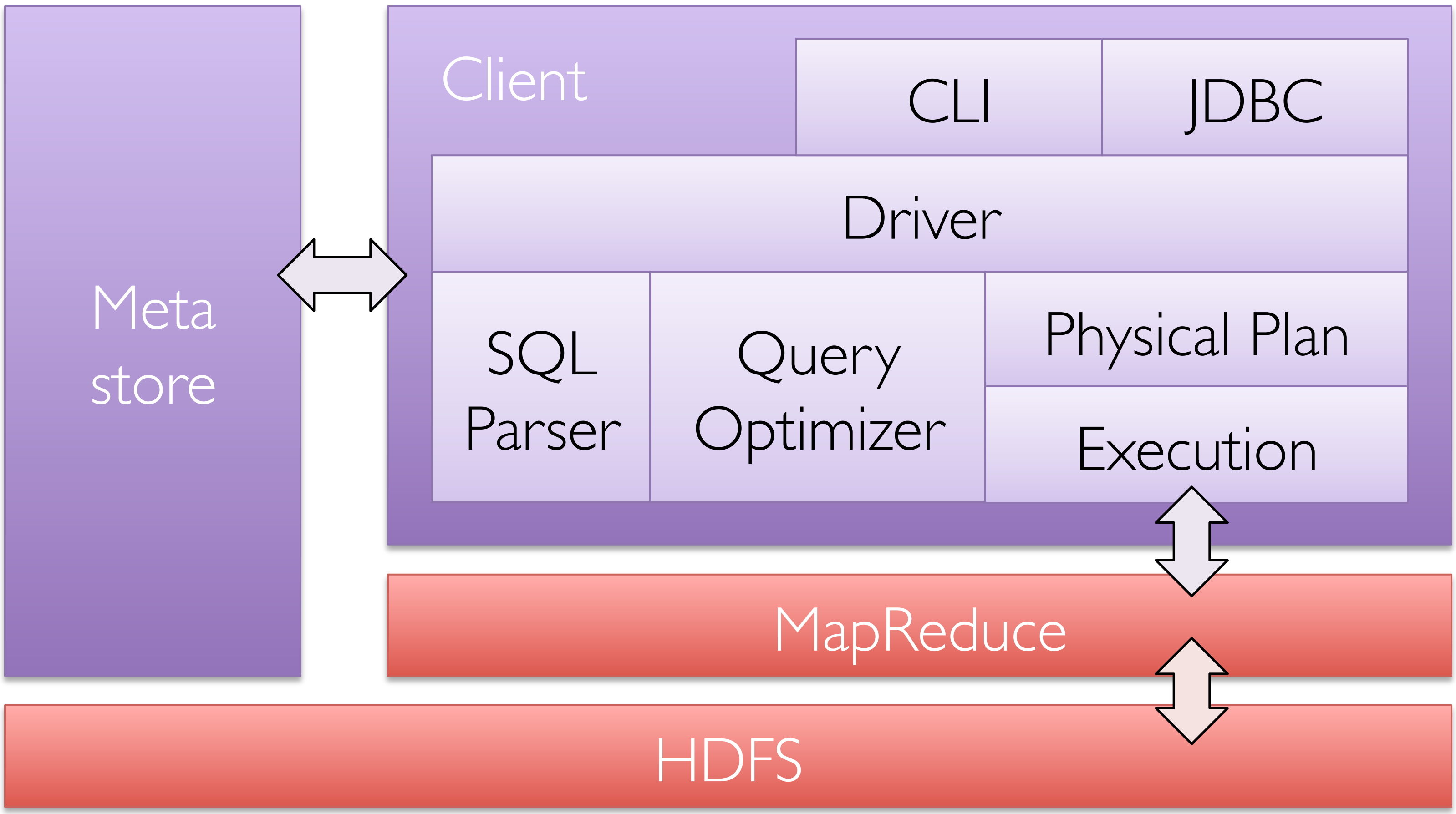
1. A giant st[...]aradigm for large-scale d
2. A sub-op[...] brute force instead of ind
3. Not nove[...]lementation of well know[...]years ago
4. Missing m[...]y included in current DBI

Shark

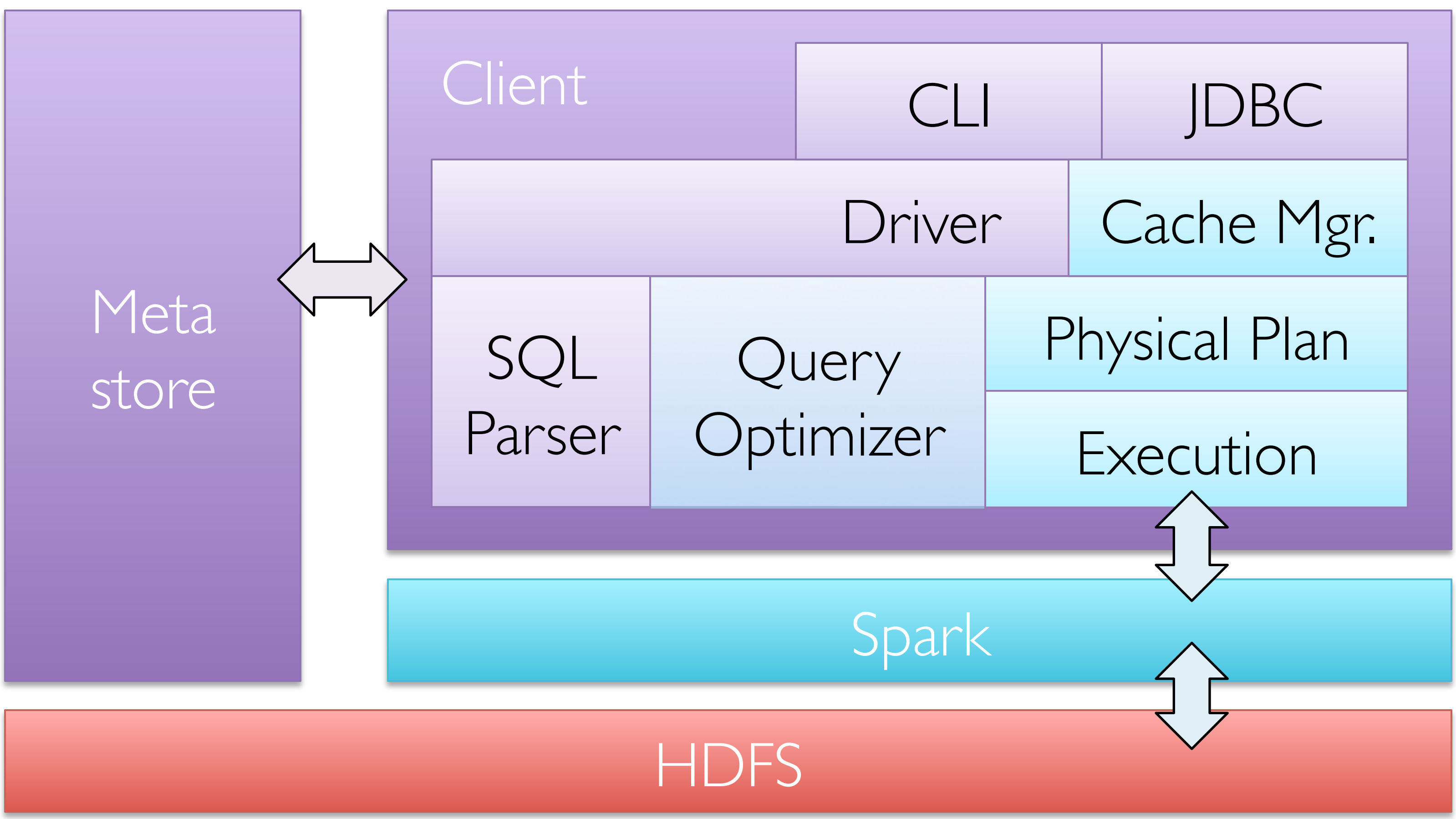
A data analytics system that

- » builds on Spark,
- » scales out and tolerate worker failures,
- » supports low-latency, interactive queries through in-memory computation,
- » supports ***both*** SQL and complex analytics,
- » is compatible with Hive (storage, serdes, UDFs, types, metadata).

Hive Architecture



Shark Architecture



Engine Features

Dynamic Query Optimization

Columnar Memory Store

Machine Learning Integration

Data Co-partitioning & Co-location

Partition Pruning based on Range Statistics

…

How do we optimize:

```
SELECT * FROM table1 a JOIN table2 b ON a.key=b.key
  WHERE my_crazy_udf(b.field1, b.field2) = true;
```

How do we optimize:

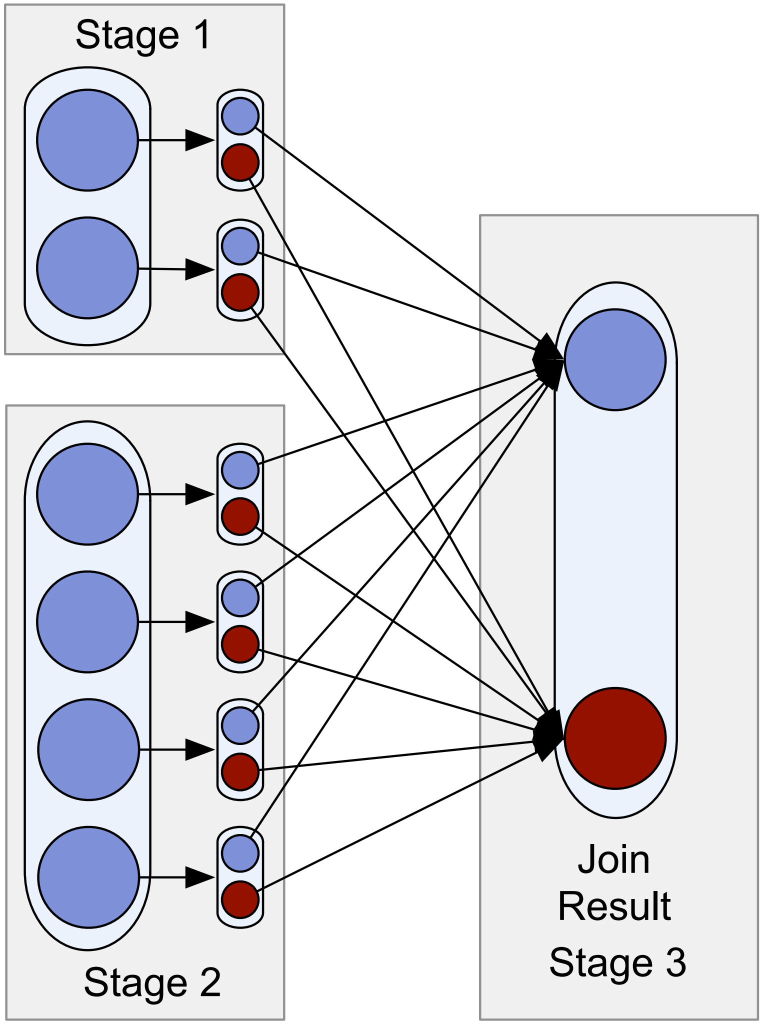
```
SELECT * FROM table1 a JOIN table2 b ON a.key=b.key
  WHERE my_crazy_udf(b.field1, b.field2) = true;
```

Hard to estimate cardinality!

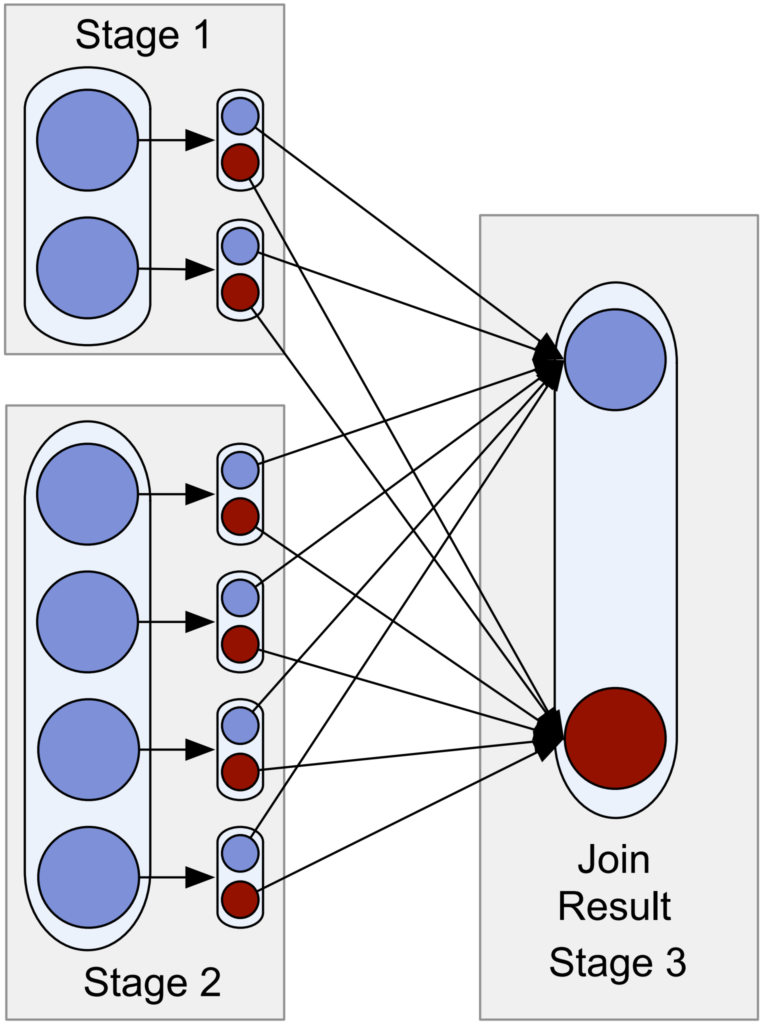
Partial DAG Execution (PDE)

Lack of statistics for fresh data and the prevalent use of UDFs necessitate dynamic approaches to query optimization.

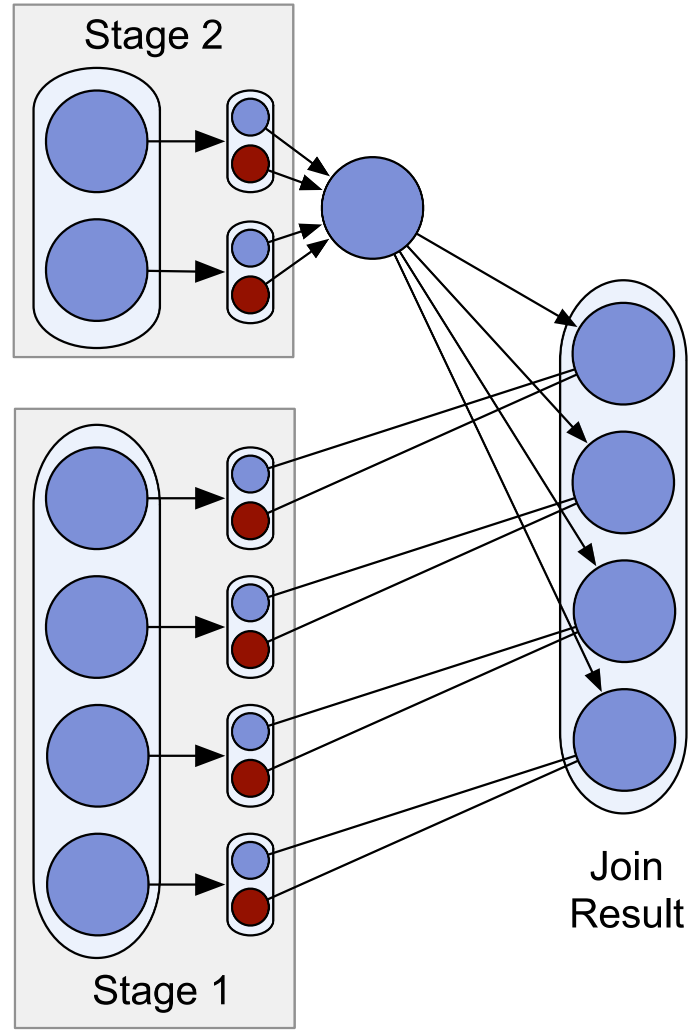
PDE allows *dynamic alternation of query plans* based on statistics collected at run-time.



Shuffle Join



Shuffle Join



Map Join (Broadcast Join)
minimizes network traffic

PDE Statistics

1. Gather customizable statistics at per-partition granularities while materializing map output.
   » partition sizes, record counts (skew detection)
   » "heavy hitters"
   » approximate histograms

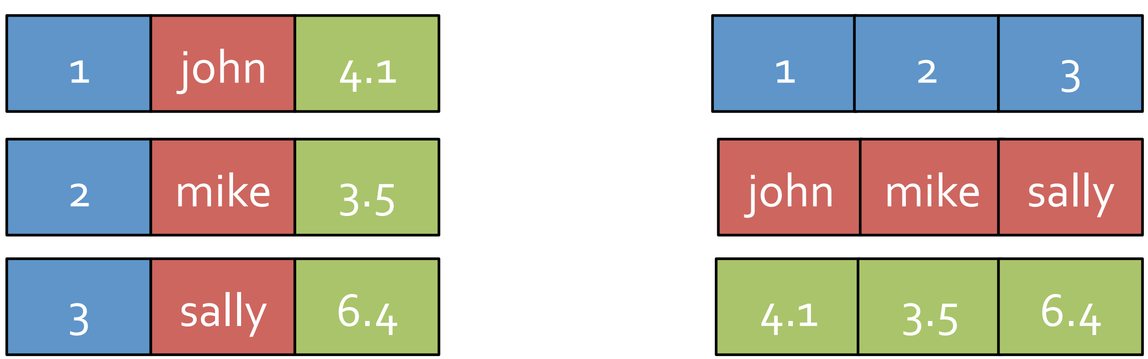
2. Alter query plan based on such statistics
   » map join vs shuffle join
   » symmetric vs non-symmetric hash join
   » skew handling

Columnar Memory Store

Simply caching Hive records as JVM objects is inefficient.

Shark employs column-oriented storage.

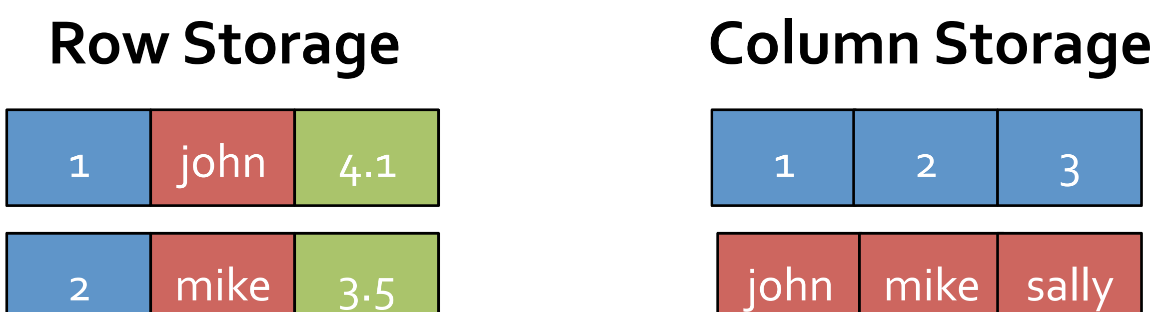
Row Storage **Column Storage**



Columnar Memory Store

Simply caching Hive records as JVM objects is inefficient.

Shark employs column-oriented storage.



Benefit: compact representation, CPU efficient compression, cache locality.

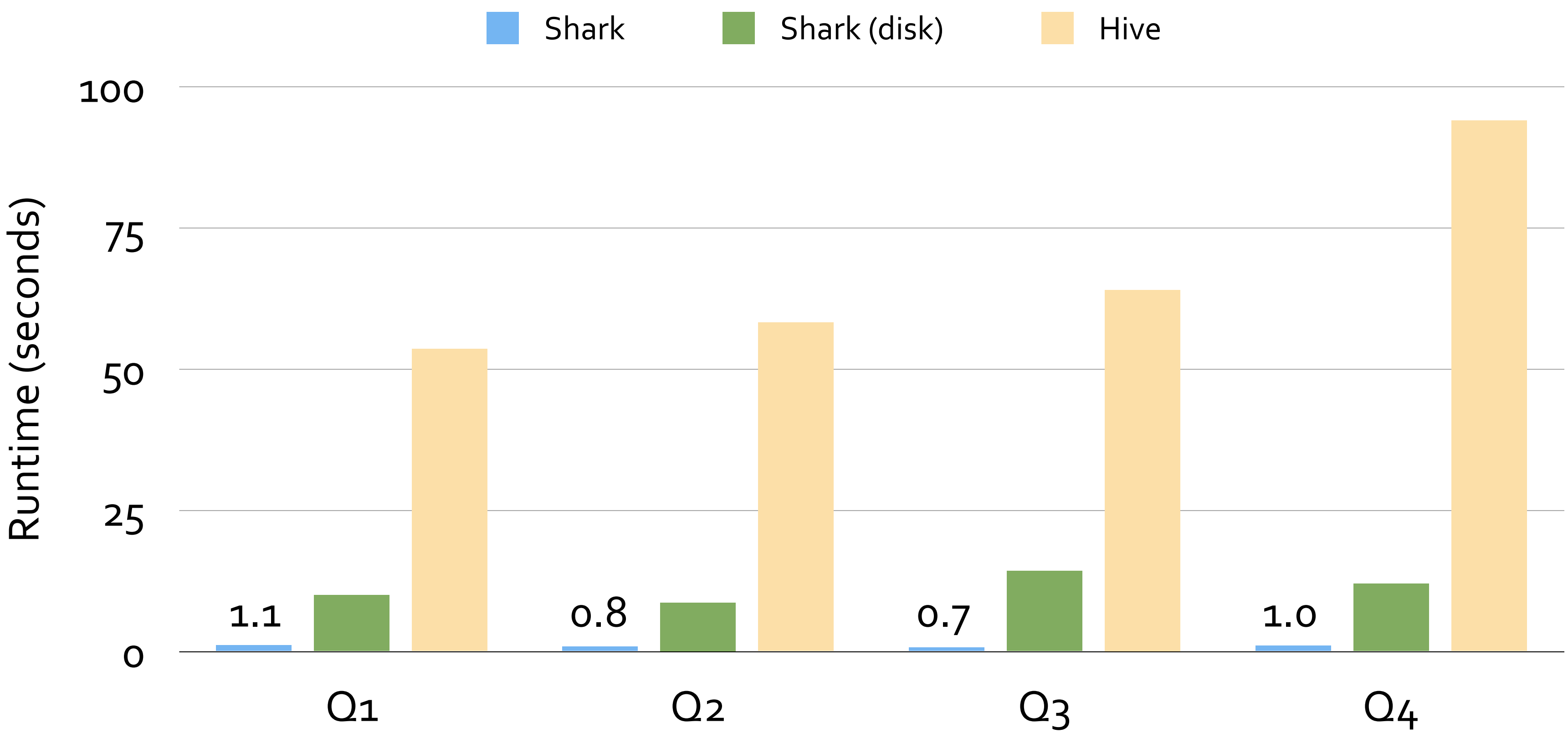
Machine Learning Integration

Unified system for query processing and machine learning

Query processing and ML share the same set of workers and caches

```
def logRegress(points: RDD[Point]): Vector {
  var w = Vector(D, _ => 2 * rand.nextDouble - 1)
  for (i <- 1 to ITERATIONS) {
    val gradient = points.map { p =>
      val denom = 1 + exp(-p.y * (w dot p.x))
      (1 / denom - 1) * p.y * p.x
    }.reduce(_ + _)
    w -= gradient
  }
  w
}

val users = sql2rdd("SELECT * FROM user u
   JOIN comment c ON c.uid=u.uid")

val features = users.mapRows { row =>
  new Vector(extractFeature1(row.getInt("age")),
             extractFeature2(row.getStr("country")),
             ...)}
val trainedVector = logRegress(features.cache())
```

Performance



1.7 TB Real Warehouse Data on 100 EC2 nodes

Why are previous MapReduce-based systems slow?

Why are previous MR-based systems slow?

1. Disk-based intermediate outputs.
2. Inferior data format and layout (no control of data co-partitioning).
3. Execution strategies (lack of optimization based on data statistics).
4. Task scheduling and launch overhead!

Scheduling Overhead!

Hadoop uses heartbeat to communicate scheduling decisions.

» Task launch delay 5 - 10 seconds.

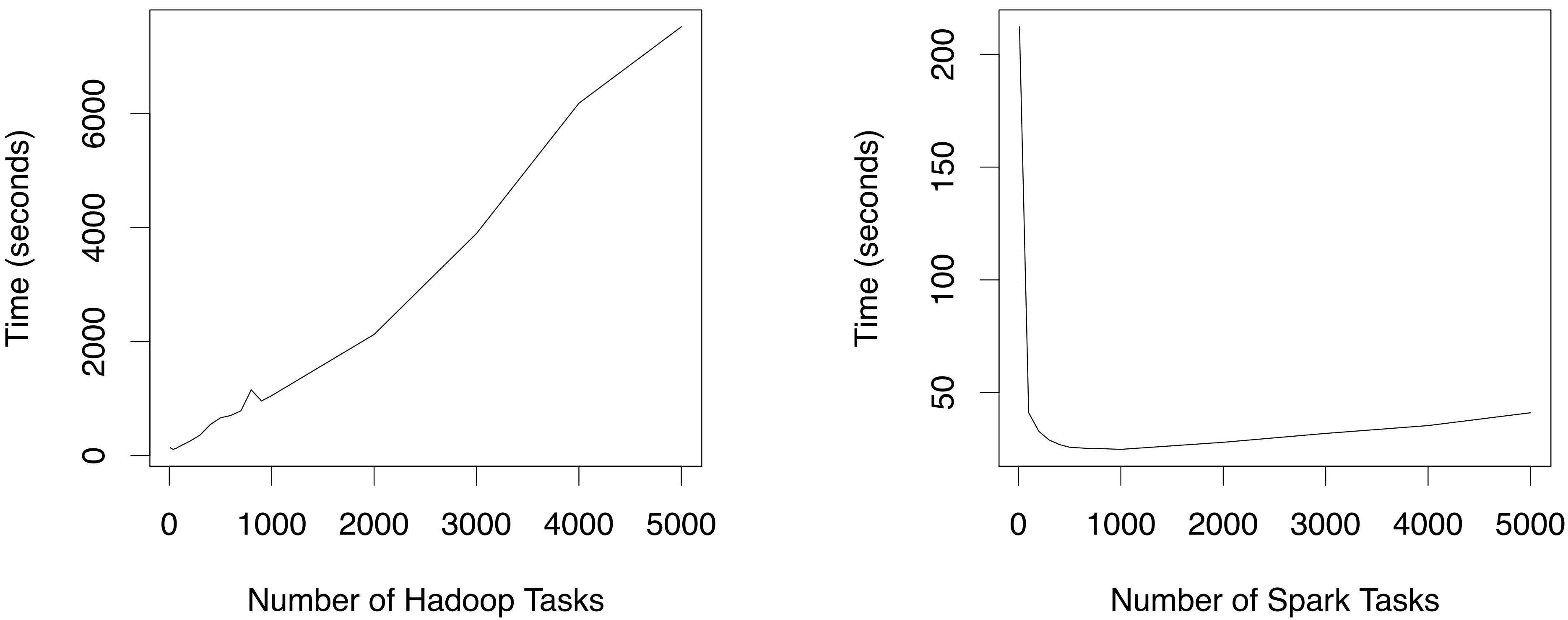
Spark uses an event-driven architecture and can launch tasks in 5ms.

» better parallelism

» easier straggler mitigation

» elasticity

» multi-tenancy resource sharing


Time (seconds)
0
2000
4000
6000
0
1000
2000
3000
4000
5000
Number of Hadoop Tasks
Time (seconds)
50
100
150
200
0
1000
2000
3000
4000
5000
Number of Spark Tasks

More Information

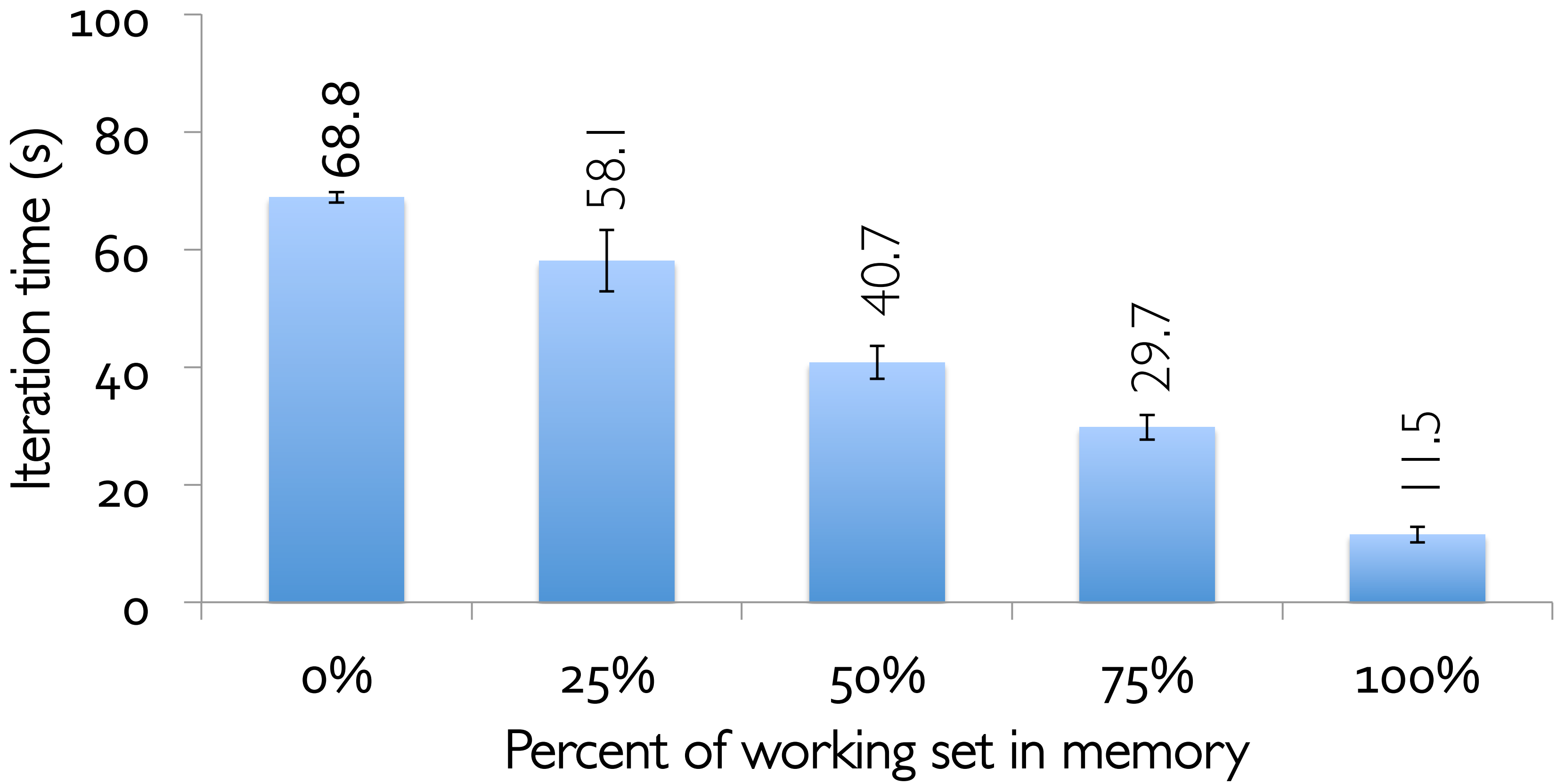
Download and docs: www.spark-project.org

» Easy to run locally, on EC2, or on Mesos/YARN

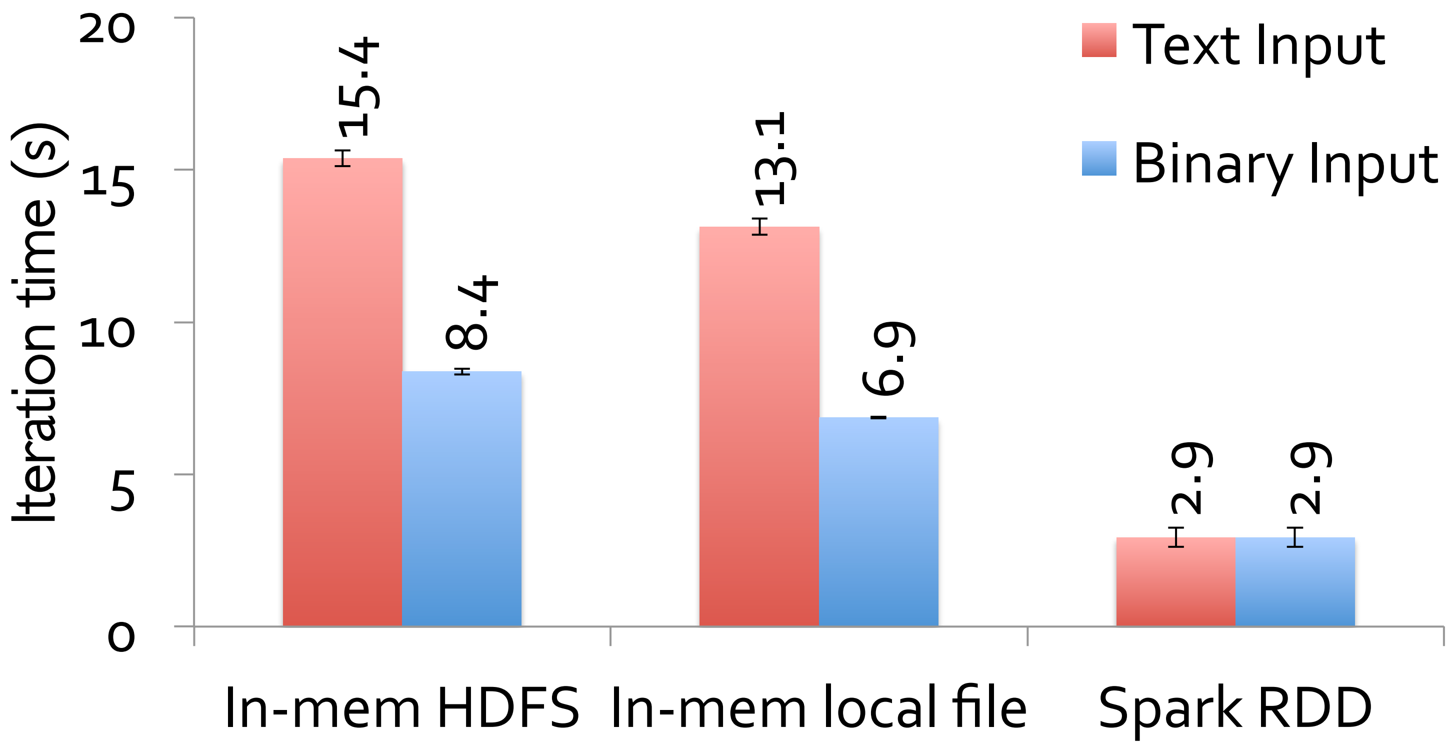
Email: rxin@cs.berkeley.edu

Twitter: @rxin

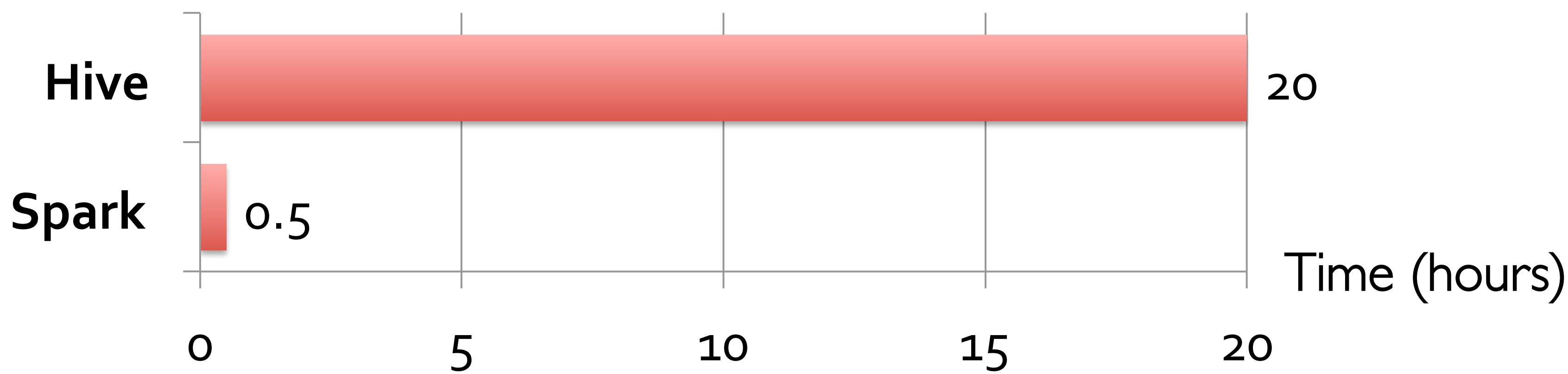
Behavior with Insufficient RAM



Breaking Down the Speedup



Conviva GeoReport



Group aggregations on many keys w/ same filter

40× gain over Hive from avoiding repeated I/O, deserialization and filtering

Example: PageRank

1. Start each page with a rank of 1
2. On each iteration, update each page's rank to

$$\sum_{i \in \text{neighbors}} \text{rank}_i \,/\, |\text{neighbors}_i|$$

```
links = // RDD of (url, neighbors) pairs
ranks = // RDD of (url, rank) pairs

for (i <- 1 to ITERATIONS) {
  ranks = links.join(ranks).flatMap {
    (url, (links, rank)) =>
      links.map(dest => (dest, rank/links.size))
  }.reduceByKey(_ + _)
}
```