

Spark

In-Memory Cluster Computing for
Iterative and Interactive Applications

Matei Zaharia, Mosharaf Chowdhury, Tathagata Das,
Ankur Dave, Justin Ma, Murphy McCauley, Michael Franklin,
Scott Shenker, Ion Stoica

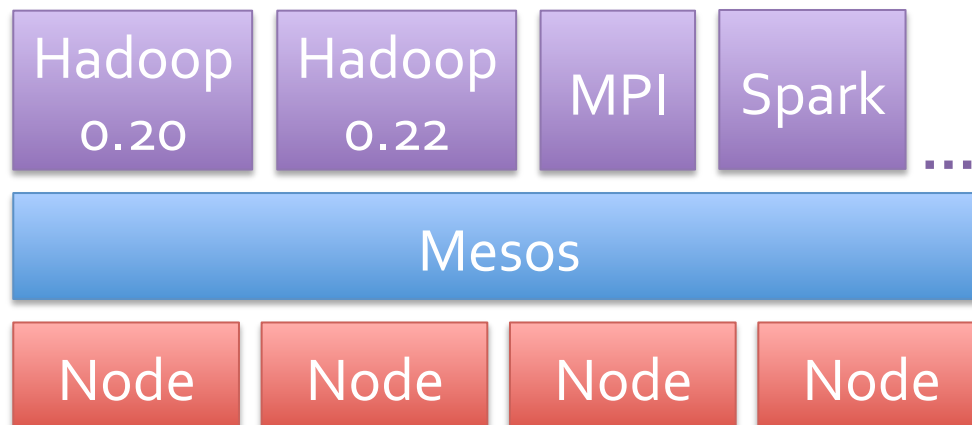
University of California, Berkeley



Background: Mesos Project

Rapid innovation in cluster computing frameworks
(MapReduce, Dryad, Pregel, Dremel, ...)

Goal: let multiple frameworks efficiently *share* a cluster
and make it easier to develop *new* ones



Mesos Features

Scalable to 10,000s of nodes, 100s of jobs,
1000s of scheduling decisions per second

High availability using ZooKeeper

Isolation using Linux Containers

Supports Hadoop, MPI, Torque, Spark

www.mesosproject.org



Spark Goals

Extend the MapReduce model to better support two common classes of analytics apps:

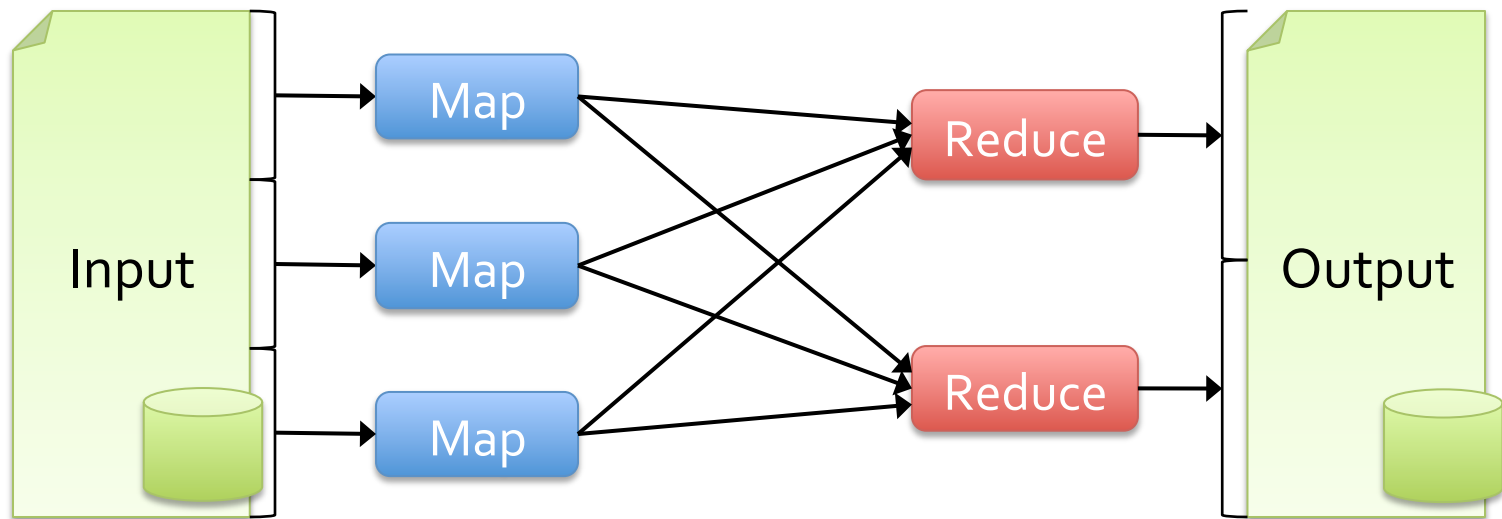
- » **Iterative** algorithms (machine learning, graphs)
- » **Interactive** data mining

Enhance programmability:

- » Integrate into Scala programming language
- » Allow interactive use from Scala interpreter

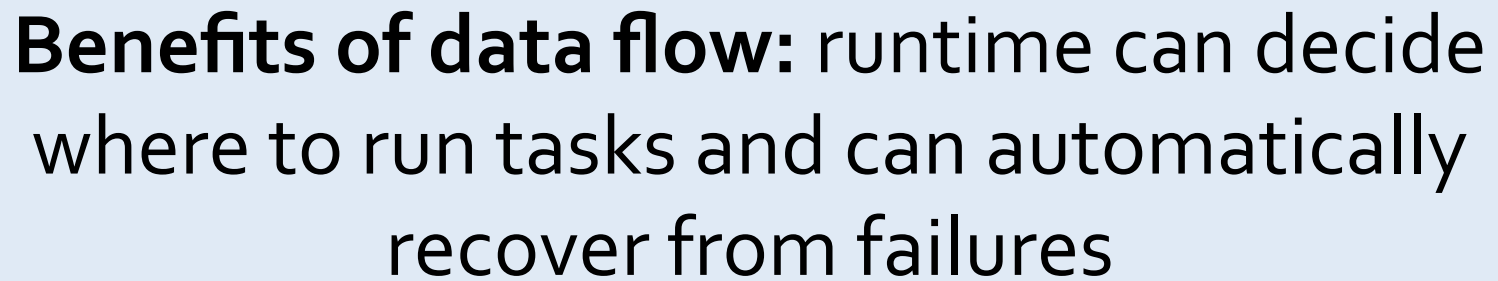
Motivation

Most current cluster programming models are based on *acyclic data flow* from stable storage to stable storage



Motivation

Most current cluster programming models are based on *acyclic data flow* from stable storage to stable storage



Benefits of data flow: runtime can decide where to run tasks and can automatically recover from failures

Motivation

Acyclic data flow is inefficient for applications that repeatedly reuse a *working set* of data:

- » **Iterative** algorithms (machine learning, graphs)
- » **Interactive** data mining tools (R, Excel, Python)

With current frameworks, apps reload data from stable storage on each query

Solution: Resilient Distributed Datasets (RDDs)

Allow apps to keep working sets in memory for efficient reuse

Retain the attractive properties of MapReduce
» Fault tolerance, data locality, scalability

Support a wide range of applications

Outline

Spark programming model

Applications

Implementation

Demo

Programming Model

Resilient distributed datasets (RDDs)

- » Immutable, partitioned collections of objects
- » Created through parallel *transformations* (map, filter, groupBy, join, ...) on data in stable storage
- » Can be *cached* for efficient reuse

Actions on RDDs

- » Count, reduce, collect, save, ...

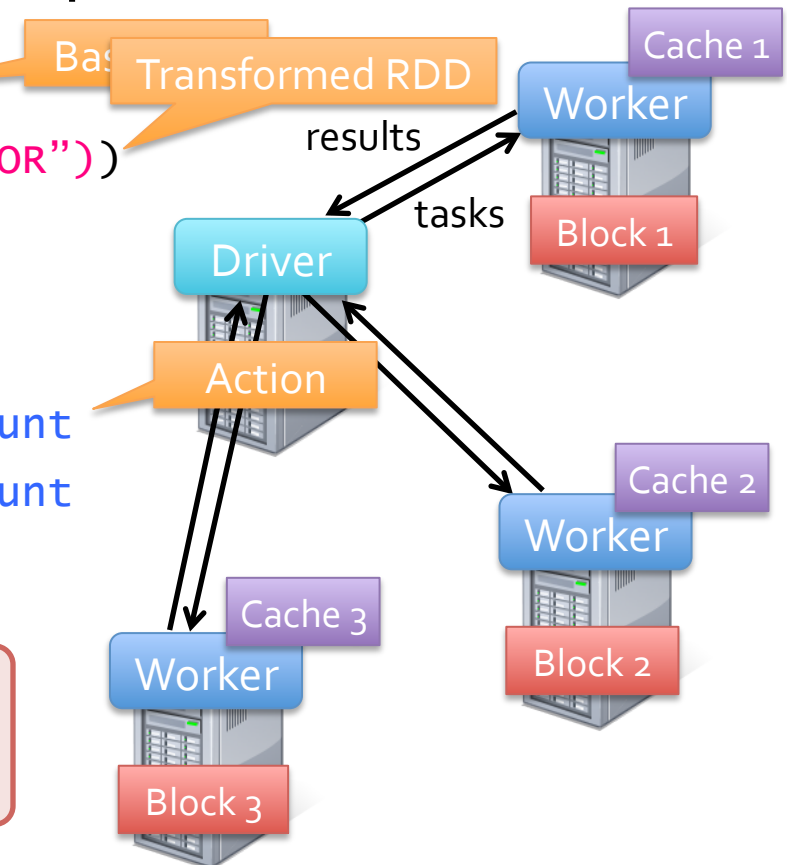
Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
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
. . .
```

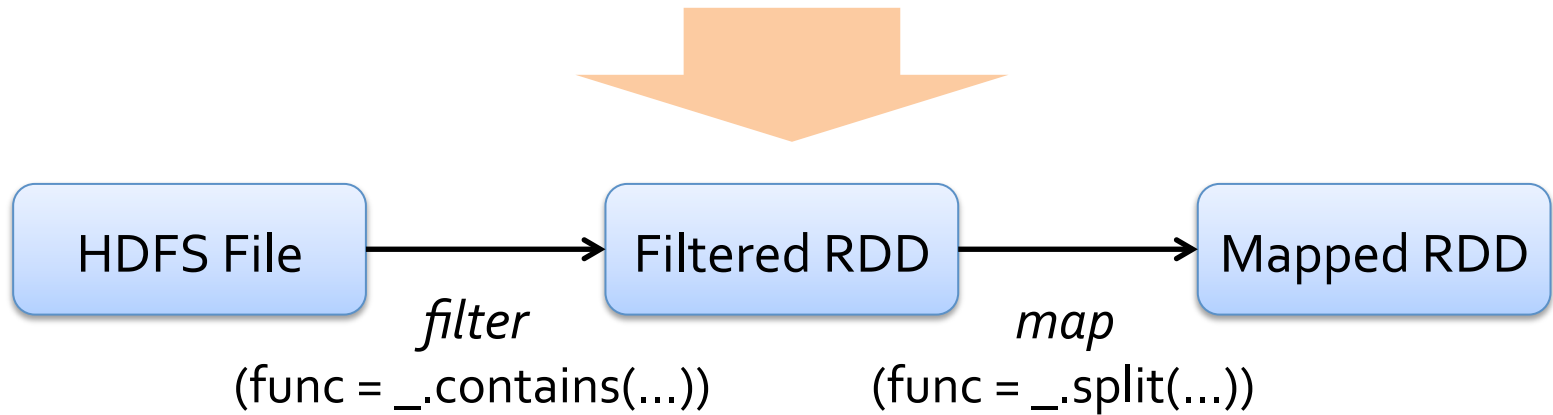
Result: scaled to 1 TB data in 5-7 sec
(vs 170 sec for on-disk data)



RDD Fault Tolerance

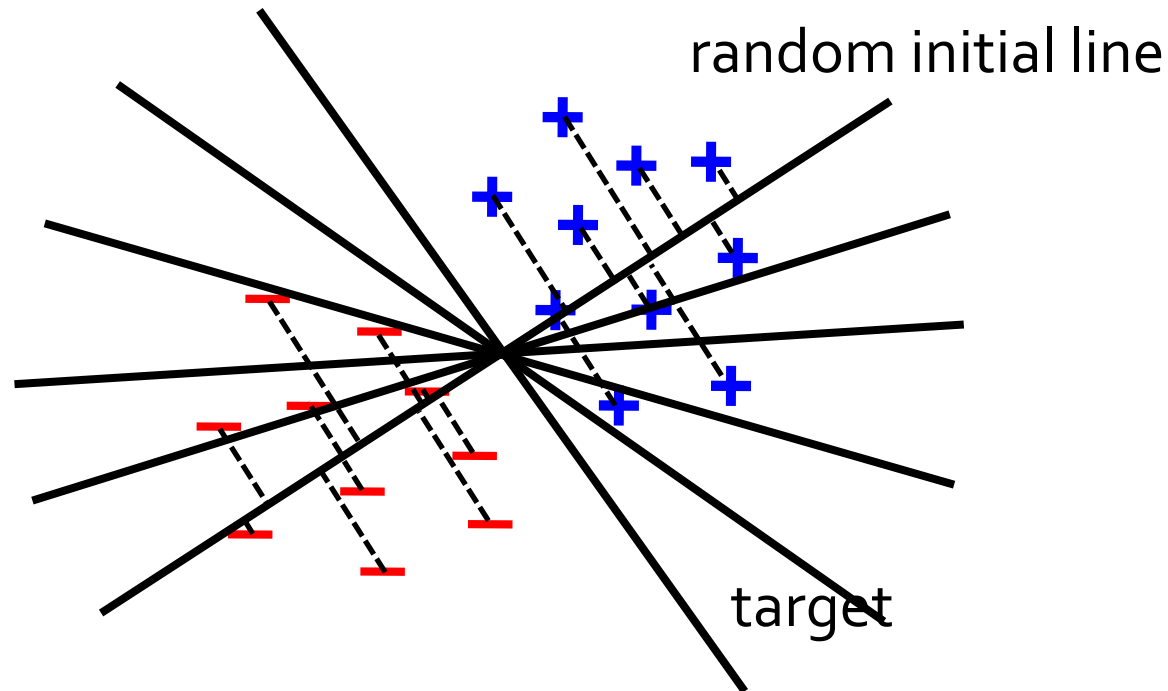
RDDs maintain *lineage* information that can be used to reconstruct lost partitions

Ex: `messages = textFile(...).filter(_.startsWith("ERROR")).map(_.split('\t')(2))`



Example: Logistic Regression

Goal: find best line separating two sets of points



Example: Logistic Regression

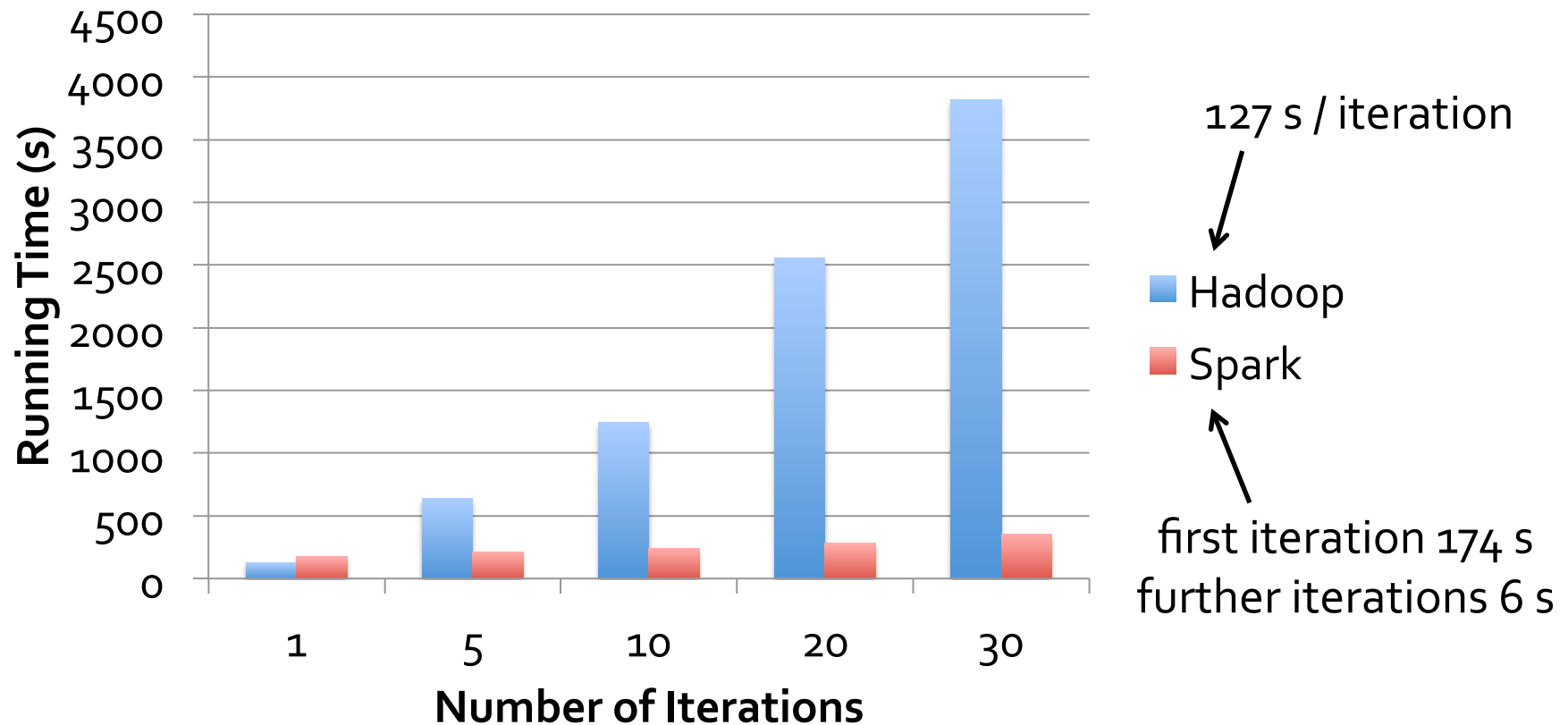
```
val data = spark.textFile(...).map(readPoint).cache()
```

```
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)
```

Logistic Regression Performance



Word Count in Spark

```
lines = spark.textFile("hdfs://...")
```

```
counts = lines.flatMap(line => line.split(" "))  
                .map(word => (word, 1))  
                .reduceByKey(_ + _)
```

```
counts.save("hdfs://...")
```


Spark Applications

Traffic prediction using EM (Mobile Millennium)

Twitter spam classification (Monarch)

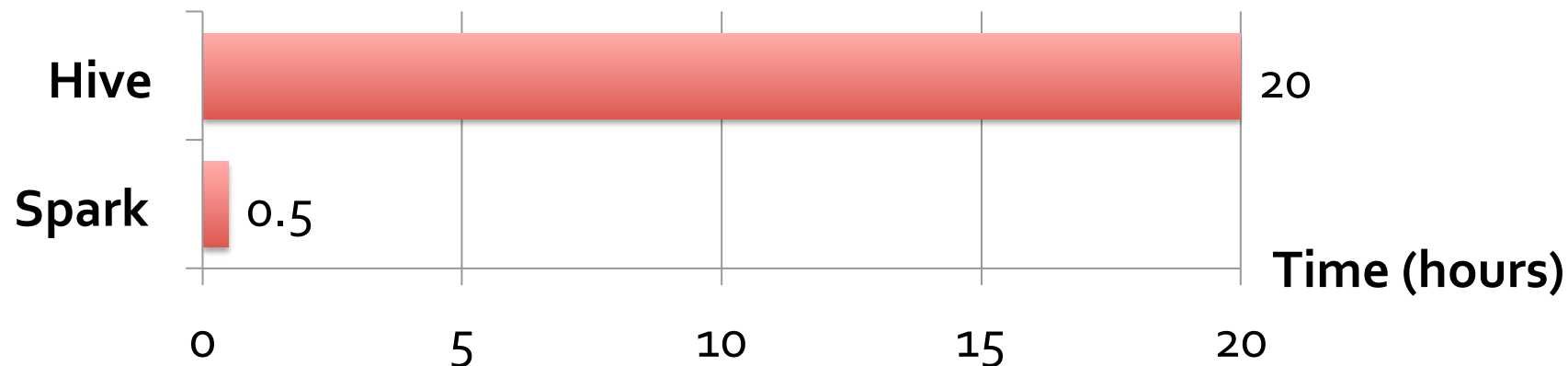
In-memory analytics on Hive data (Conviva Inc.)

K-means clustering

Alternating Least Squares matrix factorization

Network simulation

Conviva GeoReport



Aggregations on many group keys w/ same WHERE clause

40× gain comes from:

- » Not re-reading unused columns
- » Not re-reading filtered records
- » No repeated decompression
- » In-memory storage of deserialized objects

Generality of RDDs

RDDs can efficiently express many proposed cluster programming models

- » **MapReduce** => map and reduceByKey operations
- » **Dryad** => Spark runs general DAGs of tasks
- » **Pregel** iterative graph processing => Bagel
- » **SQL** => Hive on Spark (Hark?)

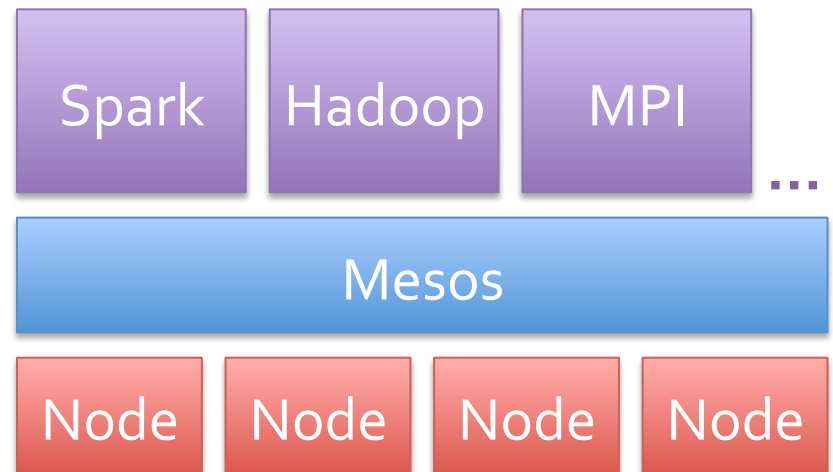
Implementation

Runs on Mesos to share resources with Hadoop

Can read from any Hadoop InputFormat (e.g. HDFS)

No changes to Scala

~10000 lines of Scala code



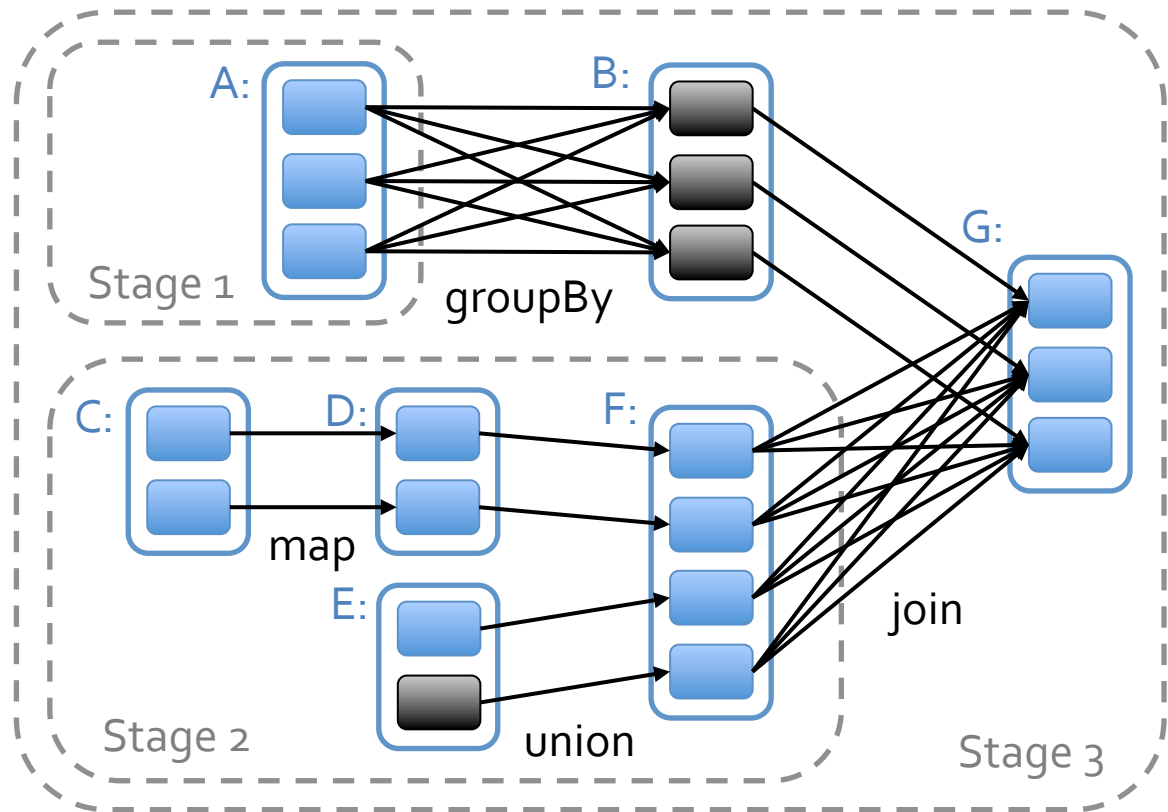
Spark Scheduler

Dryad-like DAGs

Pipelines functions within a stage

Cache-aware work reuse & locality

Partitioning-aware to avoid shuffles



Interactive Spark

Modified Scala interpreter to allow Spark to be used interactively from the command line

Required two changes:

- » Modified wrapper code generation so that each line typed has references to objects for its dependencies
- » Distribute generated classes over the network

Demo

Conclusion

Spark provides a simple, efficient, and powerful programming model for a wide range of apps

Try our open source release:

www.spark-project.org

github.com/mesos/spark

Related Work

DryadLINQ

- » Build queries through language-integrated SQL operations on lazy datasets
- » Cannot have a dataset persist *across* queries

Relational databases

- » Lineage/provenance, logical logging, materialized views

Piccolo

- » Parallel programs with shared distributed hash tables; similar to distributed shared memory

Iterative MapReduce (Twister and HaLoop)

- » Cannot define multiple distributed datasets, run different map/reduce pairs on them, or query data interactively

Related Work

Distributed shared memory (DSM)

- » Very general model allowing random reads/writes, but hard to implement efficiently (needs logging or checkpointing)

RAMCloud

- » In-memory storage system for web applications
- » Allows random reads/writes and uses logging like DSM

Nectar

- » Caching system for DryadLINQ programs that can reuse intermediate results across jobs
- » Does not provide caching in memory, explicit support over which data is cached, or control over partitioning