Spark

Spark

Fast, Interactive, Language-Integrated Cluster Computing

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

www.spark-project.org

UC BERKELEY

Project Goals

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

» **Iterative** algorithms (machine learning, graphs)

» **Interactive** data mining

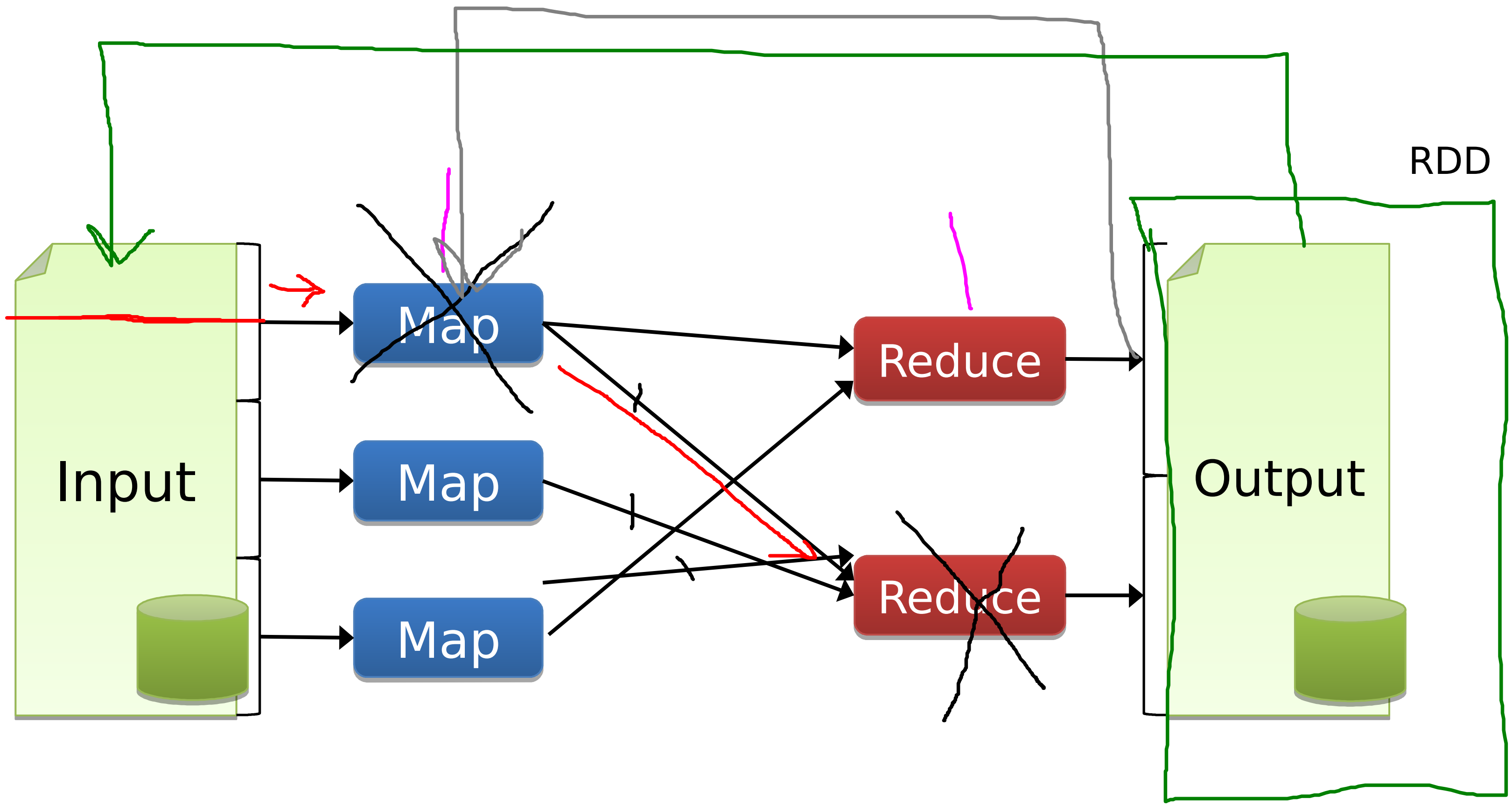
Enhance programmability:

» Integrate into Scala programming language

» Allow interactive use from Scala interpreter

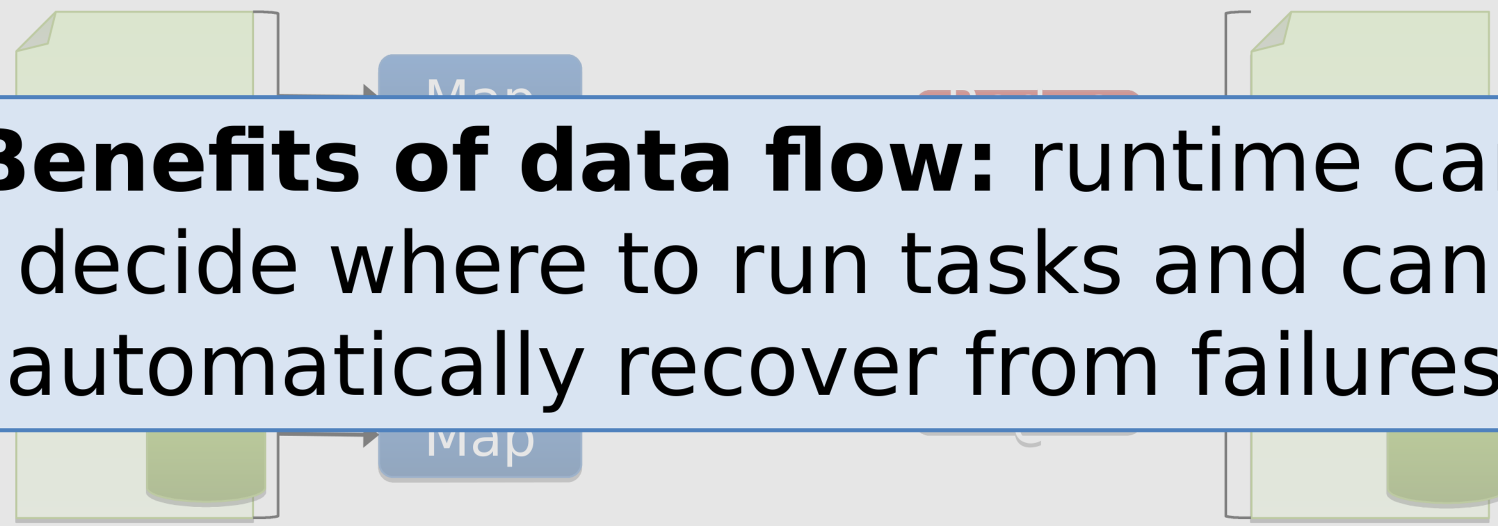
Motivation

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

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, …

Spark Operations

Transformations
(define a new RDD)

map
filter
sample
groupByKey
reduceByKey
sortByKey

flatMap
union
join
cogroup
cross
mapValues

Actions
(return a result to driver program)

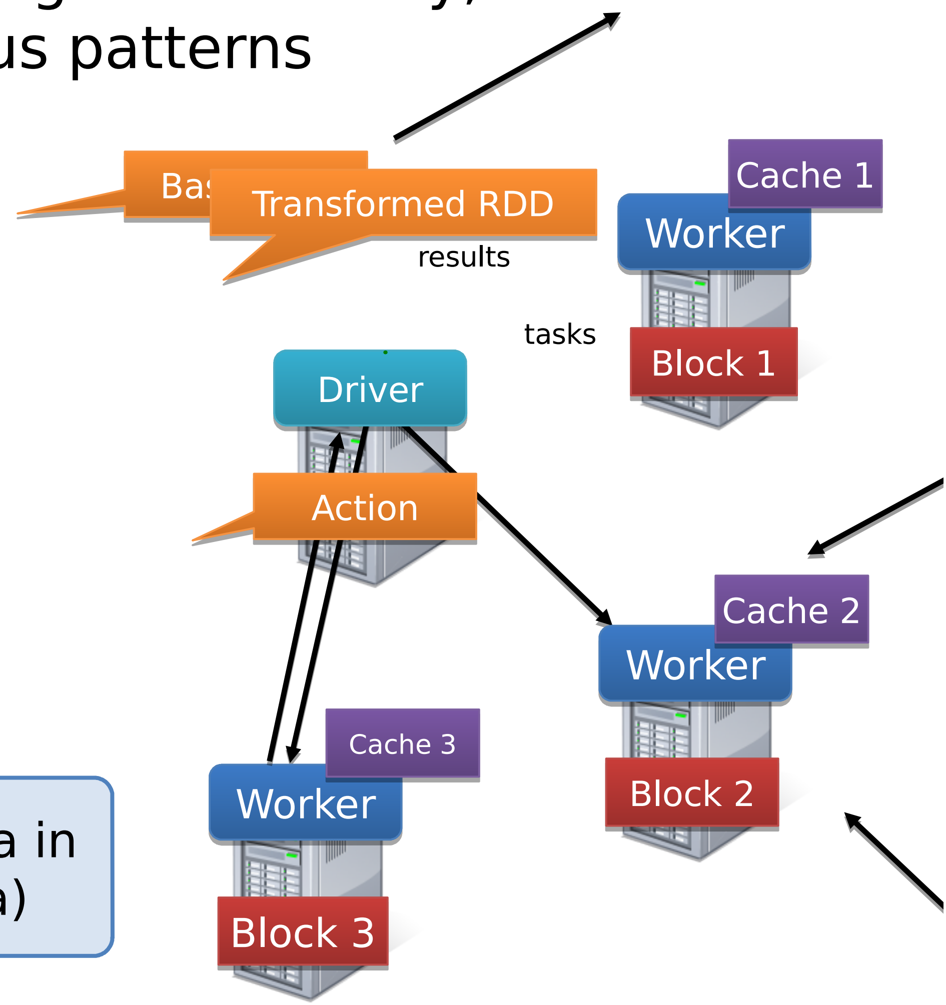
collect
reduce
count
save
lookupKey

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
. . .
```



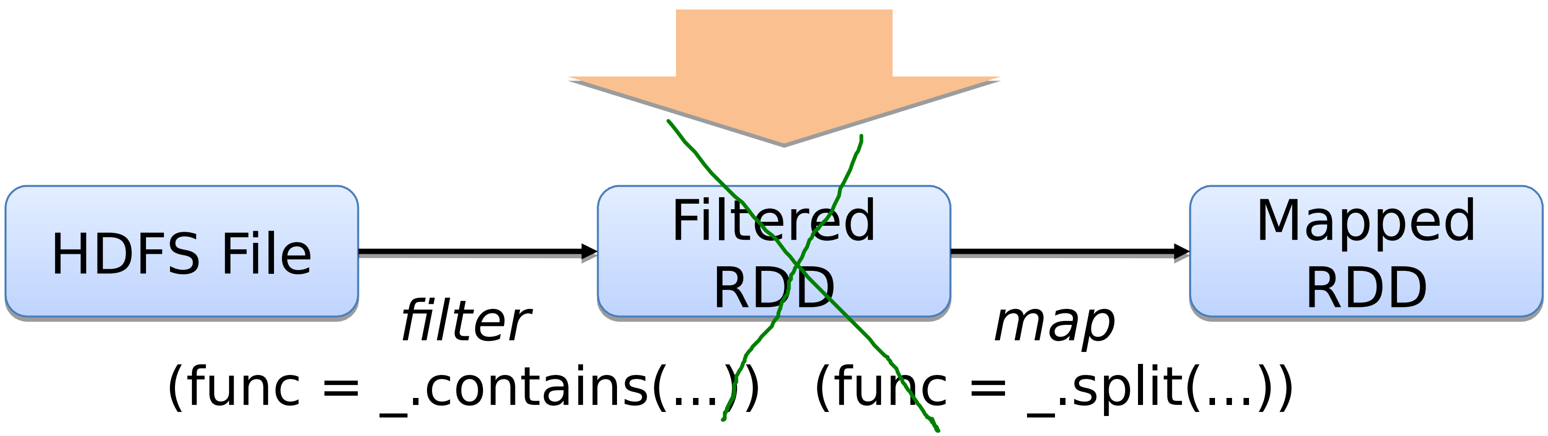
Result: full-text search of Wikipedia in <1 sec (vs 20 sec for on-disk data)

RDD Fault Tolerance

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

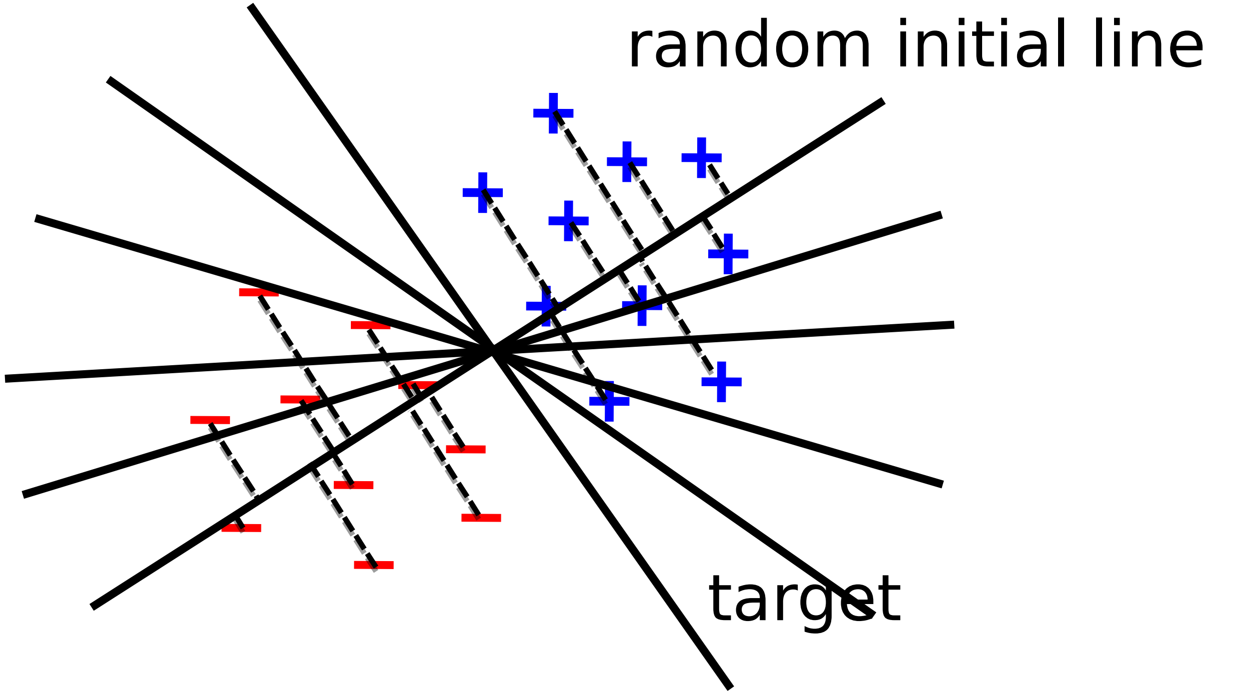
Ex:

```
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)
```

Spark Applications

In-memory data mining on Hive data (Conviva)

Predictive analytics (Quantifind)

City traffic prediction (Mobile Millennium)

Twitter spam classification (Monarch)

Collaborative filtering via matrix factorization

...

Frameworks Built on Spark

Pregel on Spark (Bagel)

- » Google message passing model for graph computatior
- » 200 lines of code

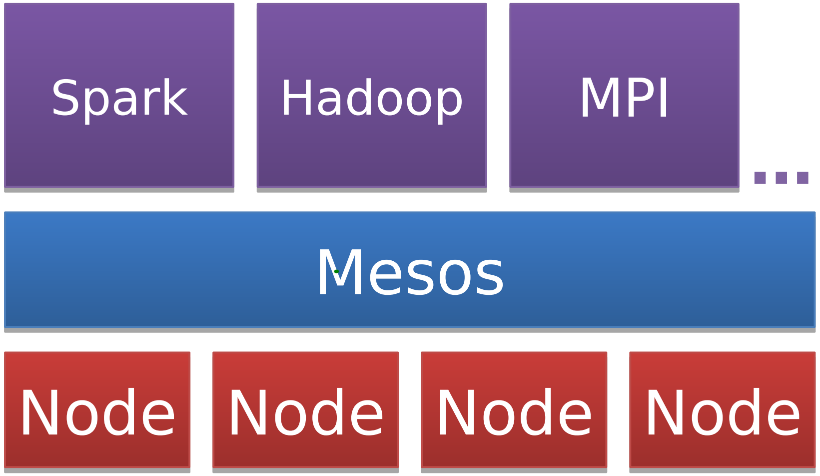
Hive on Spark (Shark)

- » 3000 lines of code
- » Compatible with Apache Hive
- » ML operators in Scala

Implementation

Runs on Apache Mesos to share resources with Hadoop & other apps

Can read from any Hadoop input source (e.g. HDFS)



No changes to Scala compiler

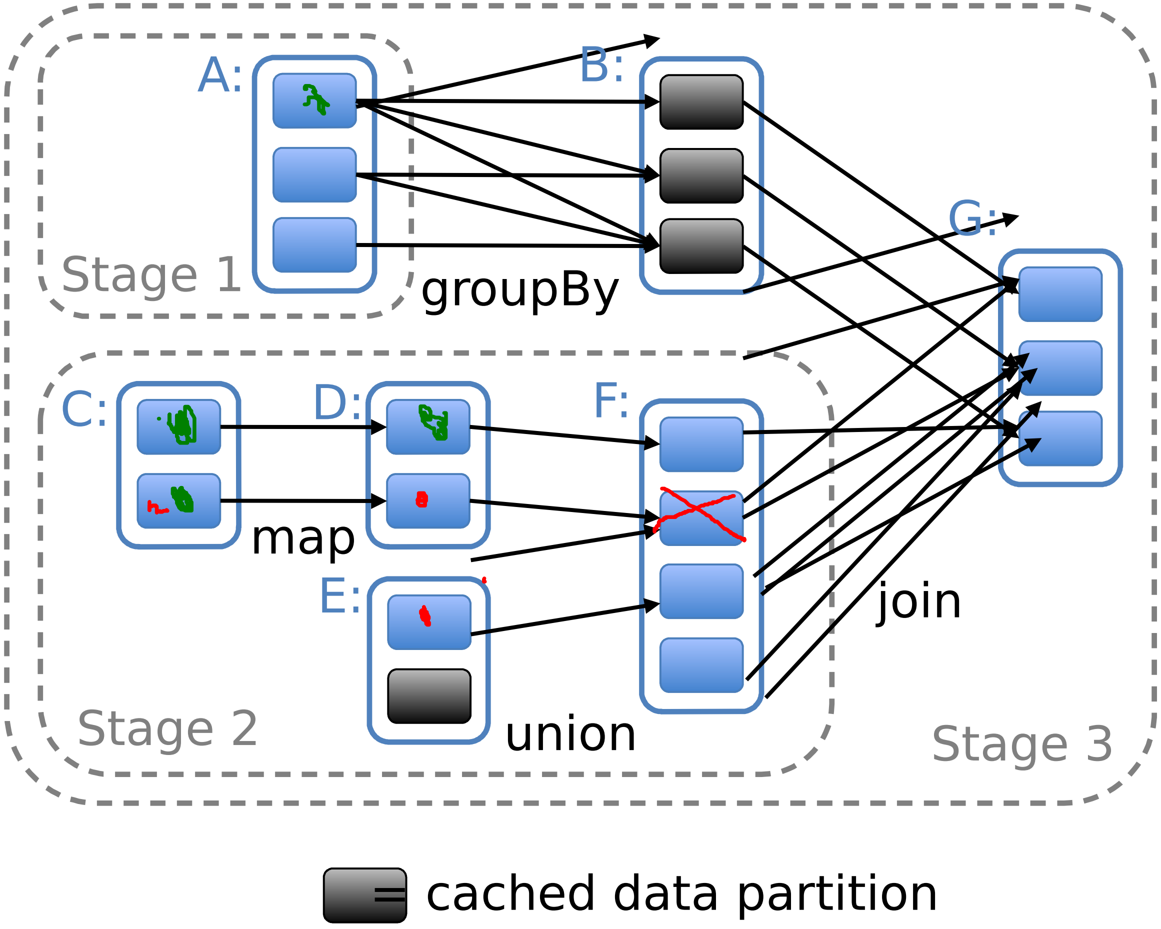
Spark Scheduler

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

Related Work

DryadLINQ, FlumeJava

» Similar "distributed collection" API, but cannot reuse datasets efficiently *across* queries

Relational databases

» Lineage/provenance, logical logging, materialized views

GraphLab, Piccolo, BigTable, RAMCloud

» Fine-grained writes similar to distributed shared memory

Iterative MapReduce (e.g. Twister, HaLoop)

» Implicit data sharing for a fixed computation pattern

Caching systems (e.g. Nectar)

» Store data in files, no explicit control over what is cached

Behavior with Not Enough RAM