

# Spark



## Distributed Memory Abstractions for Cluster Computing

Matei Zaharia, Mosharaf Chowdhury, Justin Ma, Michael J. Franklin, Scott Shenker, Ion Stoica

### Motivation

- MapReduce & Dryad have been very successful, but use acyclic data flow model that is not suitable for all apps
- *Can we provide similarly powerful abstractions for a broader class of apps?*

### Spark Goals

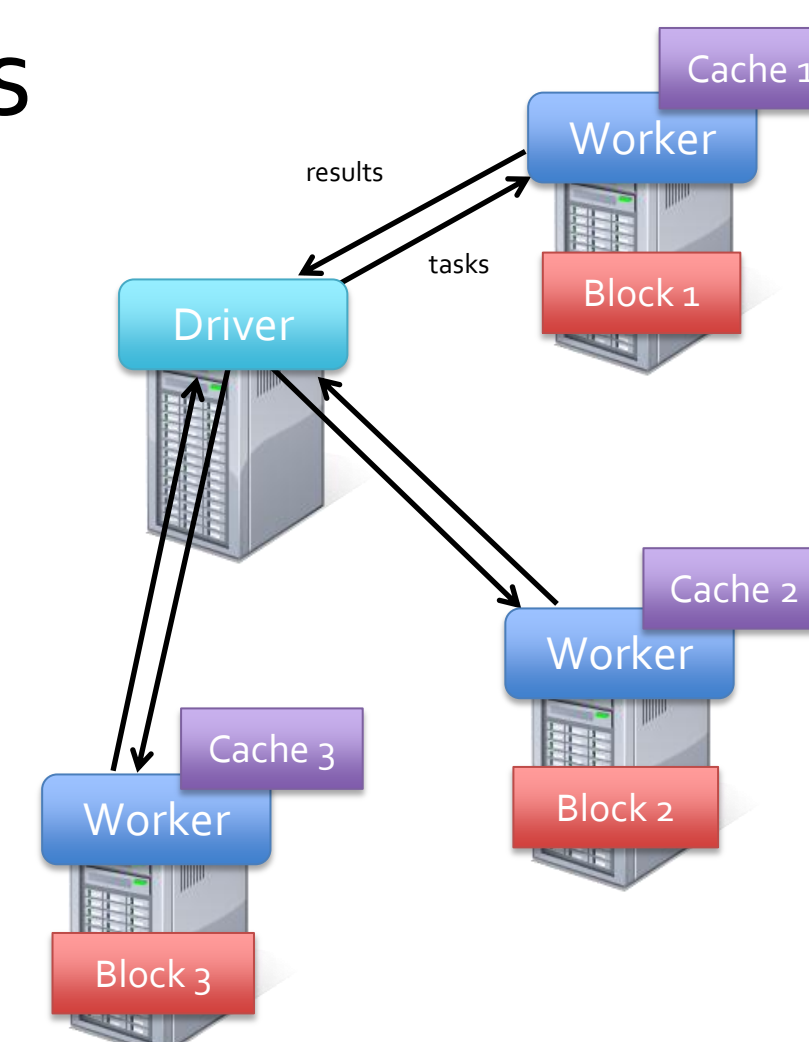
- Support cluster applications that reuse *working sets* of data, including:
  - Iterative algorithms
  - Interactive data mining
- Provide automatic fault tolerance and load balancing similar to MapReduce
- Ease of programming through integration into Scala language

### Programming Model

- *Resilient Distributed Datasets* (RDDs)
  - Collections of objects stored across cluster nodes that can be rebuilt on failure
  - Created by applying transformations (e.g. map) to data in stable storage
  - Can be explicitly cached for reuse
- *Parallel operations* on RDDs (reduce, etc)
- *Restricted shared variables* (broadcast variables and accumulators)

### Architecture

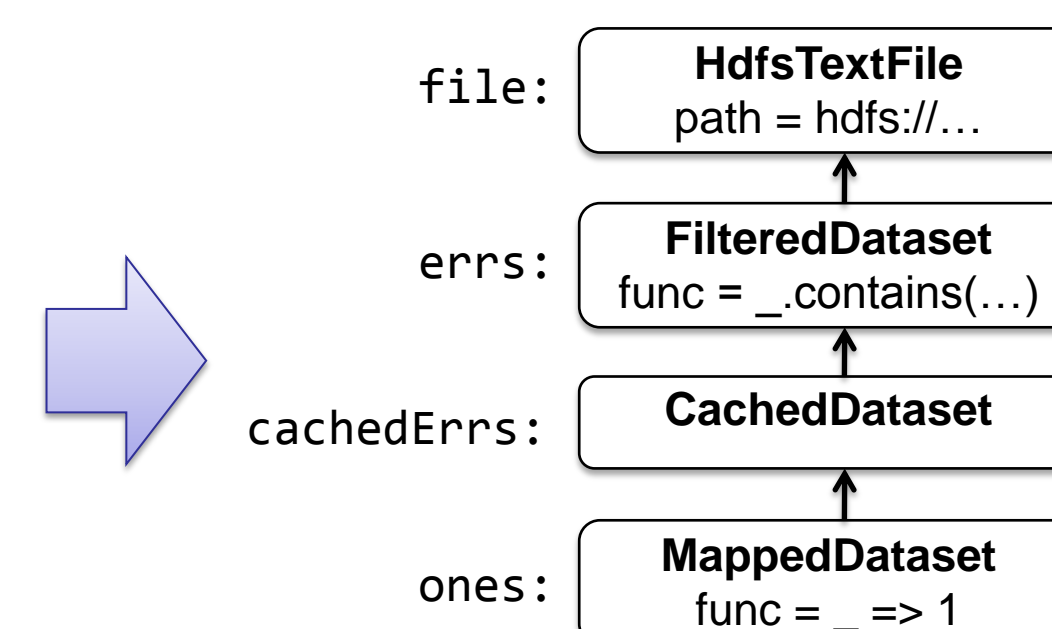
- Nodes cache slices of RDDs when requested by user
- Fault tolerance achieved through *lineage*
  - RDD handles contain enough info to rebuild from source data



### RDD and Lineage Example

```
// Cache an RDD containing all the
// lines with "ERROR" in a log file
file = spark.textFile("hdfs://...")
errs = file.filter(_.contains("ERROR"))
cachedErrs = errs.cache()

// Count errors using the cached RDD
ones = cachedErrs.map(_ => 1)
count = ones.reduce(_+_)
```



### Broadcast Variables

```
// Define a broadcast variable
bv = spark.broadcast(someBigObject)

// Use it in a parallel operation
dataset.foreach(element => {
  doStuff(bv.value)
})

// Use bv in a 2nd parallel operation;
// cached copy on each node is reused
dataset.foreach(element => {
  doOtherStuff(bv.value)
})
```

### Accumulators

```
// Define accumulator initialized to 0
accum = spark.accumulator(0)

// Use it in a parallel operation
dataset.foreach(element => {
  accum += doStuff()
})

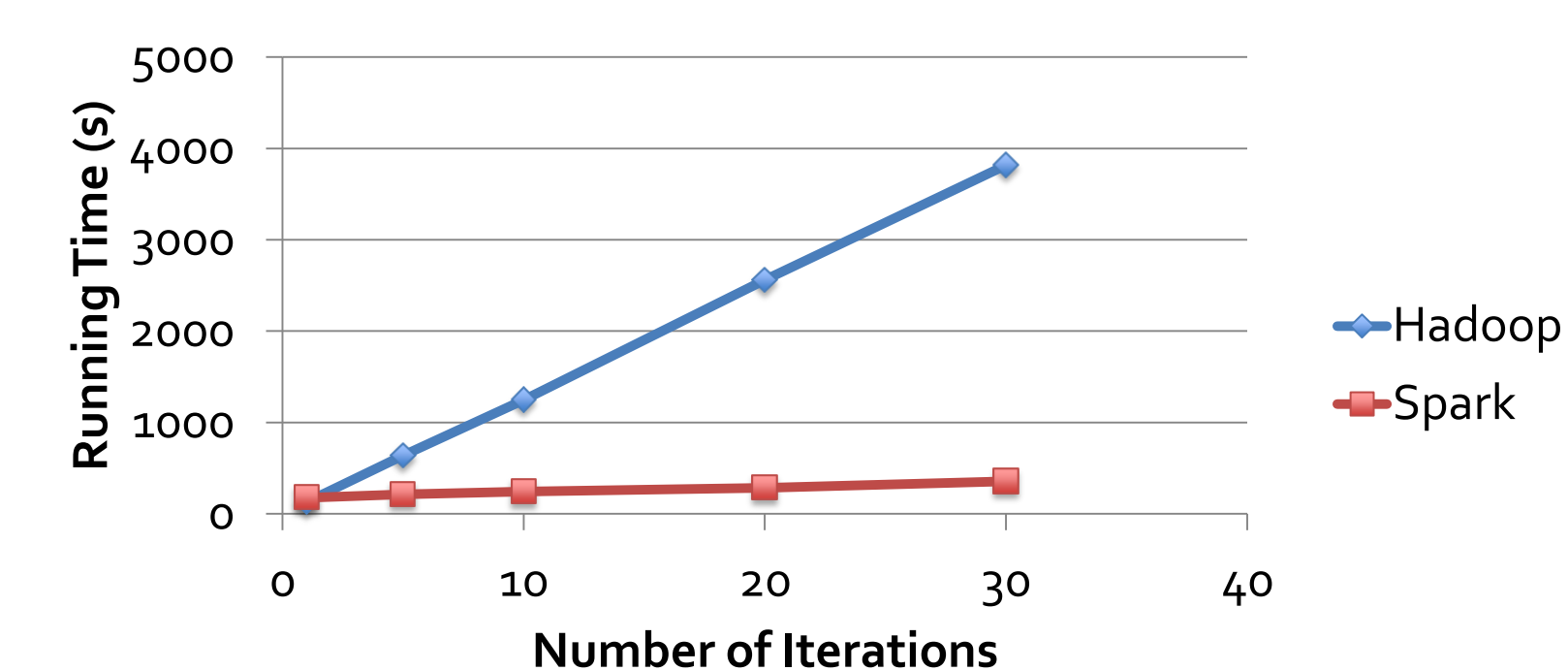
// Read value in driver program
println(accum.value)
```

### Interactive Spark

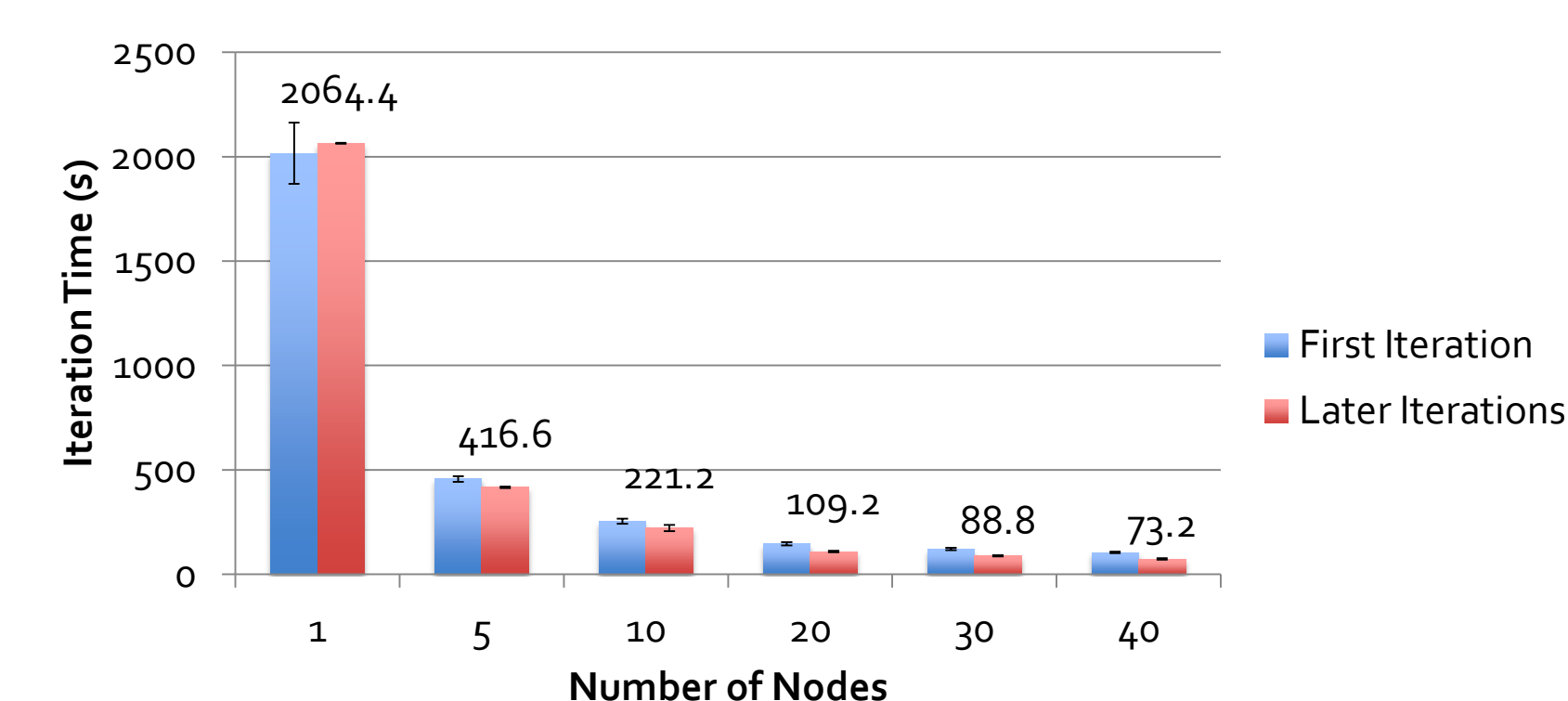
- Modified Scala interpreter to enable interactive use of Spark
  - Bytecode analysis & modified class generation strategy to capture dependencies for each input line
  - Remote class loading for workers

### Results

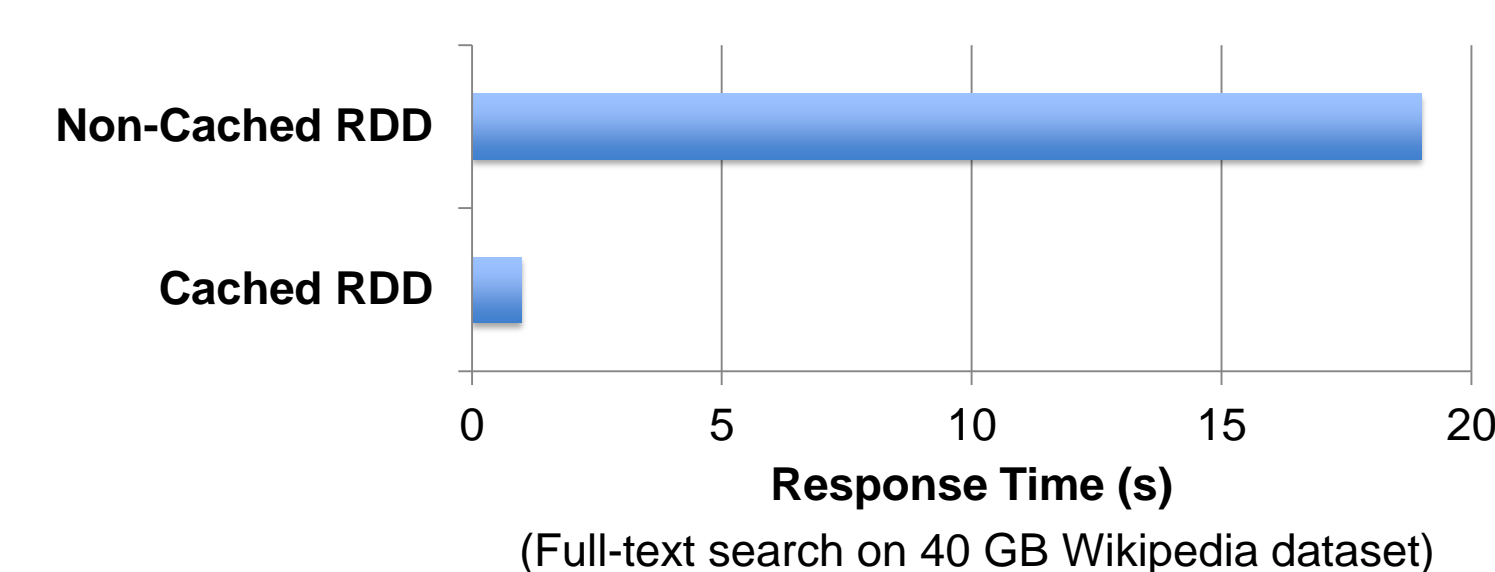
Logistic Regression Performance



Alternating Least Squares Performance



Interpreter Response Time



### Future Work

- Other memory abstractions that can be efficiently supported in clusters (e.g. updatable datasets, streams)
- More RDD storage options (e.g. caching on disk, replication, control over partitioning)
- Debugging tools that leverage lineage to replay portions of jobs