Spark
Cluster Computing with Working Sets

Matei Zaharia, Mosharaf Chowdhury,
Michael Franklin, Scott Shenker, Ion Stoica
Background

MapReduce and Dryad raised level of abstraction in cluster programming by hiding scaling & faults

However, these systems provide a limited programming model: acyclic data flow

*Can we design similarly powerful abstractions for a broader class of applications?*
Spark Goals

Support applications with *working sets* (datasets reused across parallel operations)
  » Iterative jobs (common in machine learning)
  » Interactive data mining

Retain MapReduce’s fault tolerance & scalability

Experiment with programmability
  » Integrate into Scala programming language
  » Support interactive use from Scala interpreter
Programming Model

Resilient distributed datasets (RDDs)
- Created from HDFS files or “parallelized” arrays
- Can be transformed with map and filter
- *Can be cached across parallel operations*

Parallel operations on RDDs
- Reduce, collect, foreach

Shared variables
- Accumulators (add-only), broadcast variables
Example: Log Mining

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

```scala
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
...
RDD Representation

Each RDD object maintains *lineage* information that can be used to reconstruct lost partitions.

```scala
Ex: cachedMsgs = textFile(...).filter(_.contains("error"))
    .map(_.split('"t')(2))
    .cache()
```

![Diagram showing the lineage of RDDs](image)
Example: Logistic Regression

Goal: find best line separating two sets of points
```
val data = spark.textFile(...).map(readPoint).cache()

var w = Vector.random(D)

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

println("Final w: " + w)
```
Logistic Regression Performance

- Running Time (s)
- Number of Iterations

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Hadoop</th>
<th>Spark</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>127 s / iteration</td>
<td>174 s</td>
</tr>
<tr>
<td>Further</td>
<td>6 s</td>
<td>6 s</td>
</tr>
</tbody>
</table>

Graph showing the performance comparison between Hadoop and Spark in terms of running time and number of iterations.
Demo
Conclusions & Future Work

Spark provides a limited but efficient set of fault tolerant distributed memory abstractions
   » Resilient distributed datasets (RDDs)
   » Restricted shared variables

In future work, plan to further extend this model:
   » More RDD transformations (e.g. shuffle)
   » More RDD persistence options (e.g. disk + memory)
   » Updatable RDDs (for incremental or streaming jobs)
   » Data sharing across applications
Related Work

DryadLINQ
» Build queries through language-integrated SQL operations on lazy datasets
» Cannot have a dataset persist across queries
» No concept of shared variables for broadcast etc

Pig and Hive
» Query languages that can call into Java/Python/etc UDFs
» No support for caching a datasets across queries

OpenMP
» Compiler extension for parallel loops in C++
» Annotate variables as read-only or accumulator above loop
» Cluster version exists, but not fault-tolerant

Twister and Haloop
» Iterative MapReduce implementations using caching
» Cannot define multiple distributed datasets, run multiple map & reduce pairs on them, or decide which operations to run next interactively