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
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
Outline

Spark programming model

Implementation

Demo

User 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, ...
Example: Log Mining

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

```python
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:**
```scala
messages = textFile(...).filter(_.startsWith("ERROR")).map(_.split('\t')(2))
```
Example: Logistic Regression

Goal: find best line separating two sets of points
Example: Logistic Regression

```scala
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

- Running Time (s)
- Number of Iterations

Hadoop vs. Spark
- First iteration: 174 s
- Further iterations: 6 s
- 127 s / iteration
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

...
Conviva GeoReport

Aggregations on many keys w/ same WHERE clause

40$\times$ gain comes from:
» Not re-reading unused columns or filtered records
» Avoiding repeated decompression
» In-memory storage of deserialized objects
Frameworks Built on Spark

**Pregel on Spark (Bagel)**
- Google message passing model for graph computation
- 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

Stage 1

Stage 2

Stage 3

A: = cached data partition
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

Download our open source release:

www.spark-project.org

matei@berkeley.edu
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

<table>
<thead>
<tr>
<th>% of working set in memory</th>
<th>Iteration time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache disabled</td>
<td>68.8</td>
</tr>
<tr>
<td>25%</td>
<td>58.1</td>
</tr>
<tr>
<td>50%</td>
<td>40.7</td>
</tr>
<tr>
<td>75%</td>
<td>29.7</td>
</tr>
<tr>
<td>Fully cached</td>
<td>11.5</td>
</tr>
</tbody>
</table>
Fault Recovery Results

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Time (s)</th>
<th>Failure</th>
<th>6th Iteration</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>119</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>57</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>56</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>58</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>58</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>81</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>57</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>59</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>57</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>59</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Spark Operations

<table>
<thead>
<tr>
<th>Transformations (define a new RDD)</th>
<th>Transformations (define a new RDD)</th>
<th>Actions (return a result to driver program)</th>
</tr>
</thead>
<tbody>
<tr>
<td>map</td>
<td>filter</td>
<td>collect</td>
</tr>
<tr>
<td>filter</td>
<td>sample</td>
<td>reduce</td>
</tr>
<tr>
<td>sample</td>
<td>groupByKey</td>
<td>count</td>
</tr>
<tr>
<td>groupByKey</td>
<td>reduceByKey</td>
<td>save</td>
</tr>
<tr>
<td>reduceByKey</td>
<td>sortByKey</td>
<td>lookupKey</td>
</tr>
<tr>
<td>sortByKey</td>
<td>flatMap</td>
<td></td>
</tr>
<tr>
<td>flatMap</td>
<td>union</td>
<td></td>
</tr>
<tr>
<td>union</td>
<td>join</td>
<td></td>
</tr>
<tr>
<td>join</td>
<td>cogroup</td>
<td></td>
</tr>
<tr>
<td>cogroup</td>
<td>cross</td>
<td></td>
</tr>
<tr>
<td>cross</td>
<td>mapValues</td>
<td></td>
</tr>
</tbody>
</table>