A Friendly Critique of SparkR

Michael Sannella (msannell@tibco.com)

TIBCO Spotfire

http://spotfire.tibco.com

Workshop on Distributed Computing in R

HP Labs, January 26-27, 2015

Outline

- Exceptionally quick introduction to Hadoop, MapReduce, Spark, SparkR
- Discuss elements of the SparkR implementation
- Present suggestions for APIs, relevant for any distributed R system

Disclaimers

- This is a *friendly* critique of SparkR
  - I am impressed by Spark and SparkR
  - I want SparkR to succeed, but I think it could be better
  - I think it offers lessons for other distributed R systems
- I am not an expert on Hadoop, Spark, SparkR
  - I have tried small examples with Spark and SparkR
  - I have examined the SparkR code
  - I have not used Spark or SparkR for real problems in production

Intro to Hadoop+MapReduce

- URL: hadoop.apache.org
- A MapReduce job is a distributed computation on a cluster of worker nodes:
  1. Read data from HDFS to multiple workers
  2. Map: Process data on each worker
  3. Reduce: Send data between workers (shuffle) and do additional processing on each worker
  4. Write data to HDFS
- Much effort has gone into implementing more complex operations in terms of MapReduce: SQL queries, etc.
- Practitioners have developed experience
  - Ex: For performance, try to use map operations only

Intro to Spark

- URL: spark.apache.org
- Includes a large set of operations designed to be split among workers: map, filter, flatMap, sample, union, reduceByKey, etc. (superset of map/reduce)
- APIs for Scala, Java, Python
- Console applications for Scala, Python
- Scala code example:

```
val lines = sc.textFile("data.txt")
val lineLengths = lines.map(s => s.length)
val totalLength =
     lineLengths.reduce((a, b) => a + b)
```

Intro to Spark (cont)

- Applications create linked sequences of operations
  - Resilient Distributed Datasets (RDDs)
  - RDDs are lazy: A sequence of transformations is only executed when an action pulls the result
- Stores data in memory between operations
  - Supports iterative processing (iterative machine learning algorithms) and exploration
- Automatic fault-tolerance
  - Automatically rebuilds data if nodes crash
  - Contrast: Require user to explicitly redo operations

Intro to SparkR

- URL: github.com/amplab-extras/SparkR-pkg
  - Developed at Berkeley AMPLab (same as Spark)
- R functions similar to those in Spark Scala API
  - Some exceptions: filterRDD avoids clash w stats::filter
- R code example:

```
library(SparkR)
sc <- sparkR.init(master="yarn-client")
distData <- parallelize(sc, 1:100)
sqData <- map(distData, function(x) x^2)
reduce(sqData,"+")
```

SparkR Implementation Details

- Central controller:
  - R SparkR package uses rJava to call Java/Scala code
- Distributed workers:
  - Scala code spawns Rscript processes
  - Scala communicates with worker process via stdin/stdout, using custom protocol
  - Serializes data via R serialization, simple binary serialization of integers, strings, raw bytes
- SparkR implementation still unpolished, but improving
  - No show methods for RDD, other objects
  - Some functions serialize R objects with ascii=TRUE

Design Issue: Hiding vs Exposing Distributed Operations

- Hiding distributed operations
  - Use same function names for local and distributed computation
  - Allows same code for simple case, distributed case
- Exposing distributed operations
  - Use different function names to emphasize distributed operations
- I believe the API should expose distributed operations, to encourage the programmer to think about performance implications

APIs to Suggest Performance Implications

"We also considered exposing globally distributed linear algebra operations, but explicitly decided against it primarily because global operators would ***hide the computational complexity and communication overhead*** of performing these operations. Instead, by offering linear algebra on subsets (i.e., partitions) of the data, we provide developers with a high level of abstraction while ***encouraging them to reason about efficiency***."

"MLI: An API for Distributed Machine Learning"
Evan Sparks, Ameet Talwalkar, et al.
International Conference on Data Mining (2013)
http://arxiv.org/abs/1310.5426

SparkR: Function Names

- SparkR contains simple function names that don't suggest distributed use: map, reduce, flatMap, count
  - These functions are defined as S4 generics with a single method for class “RDD”
  - I suspect they plan to add methods that work on local data
  - lapply is redefined as an S4 generic with an “ANY” method (normal R lapply) and an “RDD” method (same as “map”)
- My suggestion:
  - Define simple functions whose names e*mphasize* you are using Spark : sparkMap, sparkFilter, sparkReduce, etc.
  - Don't redefine lapply!

Design Issue: Sending Auxiliary Data to Workers

- Many distributed computations require auxiliary data on all workers:
  - Current parameters for iterative model fitting
  - Model objects for prediction
- Spark has a function for broadcasting datasets

```
val broadcastVar = sc.broadcast(1234)
words.map(s => (s,broadcastVar.value)).toArray
```

- Spark collects broadcast objects in function closure

SparkR: Broadcast objects

- SparkR has a similar "broadcast" function:

```
rdd <- parallelize(sc, 1:2, 2L)
broadcastValue <- 1:5
broadcastObj <- broadcast(sc, broadcastValue)
collect(map(rdd, function(x) x+value(broadcastObj)))
```

- Problems:
  - "broadcast" requires variable name as 2nd argument!
  - To serialize functional arguments, it scans function environments and sends all broadcast name variables
  - Issue: It is hard to detect, capture global dependencies of R functions

My Suggestion: Explicitly Assign Global Variables

- Assign global variable at a particular point in a computation
  - A given global variable can be assigned different values at different parts of a computation
  - Use older variable values on redo operations
- Possible API:

```
sparkAssign(sc, "bval", 1:5)
rdd2 <- sparkMap(rdd, function(x) x+bval)
```

- Alternative:

```
rdd2 <- sparkMap(rdd,
    sparkFunction(function(x) x+bval, list(bval=1:5)))
```

Design Issue: Loading Packages on Workers

- Loading code is easy in Java/Scala
  - Set up jar files
  - Code, data resources are loaded as needed
- R: Needs to load packages, setup environment
- SparkR has a function for loading packages:

  `includePackage(sc, pkg)`

My Suggestion: Setup Expressions

- Define expressions which can load packages, as well as any other setup needed
- Allow different setup expressions at different points during a computation

```
sparkSetup(sc,
    {library(Matrix); nums <- rnorm(1000)})
rdd2 <- map(rdd, function(x) x+nums)
```

- Perhaps merge sparkAssign, sparkSetup functions

Design Issue: Developing/Testing Code for Distributed R Processes

- Distributed programs are hard to debug, monitor
- I criticized SparkR above for using S4 generics that could possibly apply to in-memory objects
- However: It would be useful to run/test/debug code on small data in-memory
- Idea: Use Spark/SparkR concept:
  - SparkContext, pointing to cluster of worker nodes

My Suggestion: LocalSparkContext

- LocalSparkContext simulates a cluster of workers on one machine
- Run operations on separate processes, using parallel package fns (makeCluster, clusterEvalQ)
  - Allows checking that R setup, packages are correct
  - Could also save last N processes, to debug them

Summary

- Hadoop, Spark, SparkR are worth looking at
- While examining SparkR, I found some issues of interest when designing any distributed R system
  - Hiding vs exposing distributed operations
  - Sending auxiliary data to workers
  - Loading packages on workers
  - Developing/testing code on distributed R processes