A Parallel R Framework for Processing Large Dataset on Distributed Systems

Hao Lin
Shuo Yang
Samuel P. Midkiff

Nov. 17, 2013

This work is initiated and supported by Huawei Technologies
Rise of Data-Intensive Analytics

Data Sources

Personal data for the internet
- query history
- click stream logging
- tweets
- ...

Machine Generated Data
- Sensor networks
- Genome sequencing
- Physics experiments
- Satellite imaging data
Huge demand for data analysts, statisticians & data scientists

Traditional tools work on summary or sampled data

A good tool for large-scale data:
• Usability: stick to traditional semantics
• Performance: distributed parallelism
• Fault Tolerance: MapReduce
R used by about 30% of data analysts

Why R:
• Data structures
• Functional language
• Rich functionality
• Graphic visualization
• Open source

Why not R:
• Single threaded
• Limited memory

From: Survey by Revolution Analytics
http://r4stats.com/articles/popularity/
Developed at AMP lab, UC Berkeley

Flexible Programming Model
  • DAG job scheduler

Performance
  • In-memory
  • Good for iterative algorithms

Resilient Data Sets
  • Recover from loss and failures
RABID Package Structure

- **R User Analytics Application**
  - Storage Server
  - Spark
  - HDFS/HBase
  - Optimizer, Scheduler
  - Lower-level Ops
  - Matrix Ops
  - DM functions

RABID as an R extension package
Runtime Overview

- **R scripts**
- **Web server**
- **R driver session**
- **Symbol tables**
- **Spark**
- **Scheduler, optimizer**
- **Fault-tolerant**
- **Master**
- **Slave**
- **Worker**
  - **Task**
  - **R worker**
  - **Task**
  - **R worker**

Data shuffle: R scripts → Web server → R driver session → Symbol tables → Spark → Master → Slave → Worker → Task → R worker → Task → R worker → Slave → Slave → ...
Programming Model

R List – Most general data structure
• Collection to store elements of any & different types
  • similar to Python tuples
  • Very general and flexible

BigList – distributed list structure
• Extended R List to be distributed
• Override R list functions to support BigList
• Building blocks for higher level structures and functions
RABID Example (1)

Sample APIs

```r
library("Rabid")

DATA <- rb.readLines("hdfs://...")
DATA <- lapply(DATA, as.numeric, cache=TRUE)

func <- function(a) { ... }
DATA2 <- lapply(DATA, func)

DATA3 <- aggregate(DATA2, by=id, FUN=mean)

centroids <- as.list(sample(DATA, 16))
```

Load the Rabid package

Read text into a BigList

Apply function in parallel

Apply UDF in parallel

Aggregate by user specified keys

Transform back to R list
Sample APIs

```r
library("Rabid")

mat <- rb.read.matrix("hdfs://...", cache=F)

mat1 <- mat + mat
mat2 <- mat %*% mat

rownames(t(mat))
colnames(t(mat))
cor(mat)
```
Further block the data for better efficiency of transferring and processing.
Data Blocking for `aggregate()`

Slave 1

- Hash key 1
- Hash key 1
- Hash key 2

Slave 2

- Hash key 1
- Hash key 1
- Hash key 2

User's key + aggregated values

Data shuffling

---

Slave 1

- Hash key 1
- Hash key 1
- Hash key 1

Slave 2

- Hash key 2
- Hash key 2
- Hash key 2
Distributing Computation

Computations are abstracted as R functions, which are serialized to the nodes and evaluated.

R has a scoping rule for searching free variables in an enclosing environment.

We need to ship the functions together with the values of free variables in its environment.

```r
z <- 1
func1 <- function() {
  y <- 2
  func2 <- function(x) {
    x + y + z
  }
}
```
Two major overheads of each RABID operation:

1) data transferring
2) serialization/deserialization

Merge adjacent deferred operations into one reduces the overheads

What kind of operations can be merged: non-aggregation
Merging Deferred Operations

\[ g(a, b) \rightarrow g[f(x), f(y)] \]
Fault Tolerance

Take the advantage of Spark’s fault tolerance feature at the worker side

Detect user code errors that terminate R worker sessions; catch the error and stop Spark job immediately

Zookeeper at the master side
Applications & Benchmarking

Compare RABID with Hadoop & RHIPE

• RHIPE: R and Hadoop Integrated Programming Environment, developed at Purdue University

Logistic Regression:

• 10 worker nodes
• 1 ~ 100 million records
• RABID uses 1/6 LOC of Hadoop

Movie Clustering (K-Means):

• 10 worker nodes
• 11 ~ 90 million ratings
• RABID uses 1/8 LOC of Hadoop
Logistic Regression Runtime over Data Size

Implementation tools
- Hadoop
- RABID
- RHIPE

Elapsed time in secs

Data size in millions
K-Means Movie Clustering Runtime

![Graph showing the runtime of K-Means clustering using different tools: Hadoop, RHIPE, and RABID. The x-axis represents data size in millions, while the y-axis represents elapsed time in seconds. The graph illustrates the scalability and performance of these tools as data size increases.]
Logistic Regression Runtime over # nodes

![Bar chart showing runtime over number of nodes for different implementation tools: Hadoop, RHIEPE, and RABID. The chart illustrates that RHIEPE has the longest runtime followed by Hadoop, with RABID having the shortest runtime.](Image)
Logistic Regression Runtime on iterations

Implementation tools
- Hadoop
- RHIPE
- RABID
- RABID with node down in 3rd iteration

Elapsed time in secs

iterations

1
2
3
4
Conclusions

- RABID provides R users with a familiar programming model that scales to large cloud based clusters, allowing larger problems sizes to be efficiently solved.
- Preliminary results show RABID outperforms Hadoop and RHIPE on our benchmarks
- RABID is cloud-ready to be used as a service
Future Work

- Optimizing data transferring between R session and Spark
- Trade-off between fault tolerance and performance
- Benchmarking more applications and at a larger scale
Thank You!

Also thank Prof. Michael Franklin and Matei Zaharia at UC Berkeley, for discussing the ideas over Spark project

Contact: haolin@purdue.edu
         shuo.yang@huawei.com
         smidkiff@purdue.edu

Nov 17, 2013