RABID : A Distributed Parallel R [*]

Hao Lin
Purdue University
West Lafayette, IN
haolin@purdue.edu

Shuo Yang
Huawei R&D Center
Santa Clara, CA
shuo.yang@huawei.com

Samuel P. Midkiff
Purdue University
West Lafayette, IN
smidkiff@purdue.edu

Abstract

Large-scale data mining and deep data analysis are increasingly important for both enterprise and scientific applications. Statistical languages provide rich functionality and ease of use for data analysis and modeling and have a large user base. R[2] is one of the most widely used of these languages, but is limited to a single threaded execution model and problem sizes that fit in a single node. This paper describes highly parallel R system called RABID (R Analytics for BIg Data) that maintains R compatibility, leverages the MapReduce-like distributed Spark[6] and achieves high performance and scaling across clusters. Our experimental evaluation shows that RABID performs up to 5x faster than Hadoop and 20x faster than RHIPE on two data mining applications.

Keywords Distributed computing, Big data analytics, R, Data mining

1. Introduction

Domain experts are interested in what can be discovered from data, not learning new programming languages, and so providing parallel computing tools that are compatible with widely used languages is important. R is the top software tool in the data analytics community[5], and its popularity is growing. R, as a sequential language, is easy to use but is limited to single core performance and the memory on a single node. An R execution engine that allows users to easily execute their R programs on parallel systems with high performance would allow analysts to efficiently target large datasets without the distraction and pain of learning a new language. RABID is such a system.

Other efforts are summarized in Figure 1. RABID is an R system that allows R users to implement efficient analyses (both iterative and non-iterative) of datasets on parallel systems without needing to learn new programming models, languages or parallel frameworks. RABID's combination of high-performance and R-compatibility places it in the upper right corner of the chart in Figure 1. RABID accomplishes this in three ways. First, RABID is implemented on the Spark framework. Spark outperform Hadoop by 20x[6] on iterative jobs and provides fault tolerance and high availability, which RABID uses. This alone is insufficient for great parallel R performance. Second, RABID uses distributed data structures that act like regular R data structures and a serialization strategy that is transparent to users and is compatible with the Renjin[1] R execution engine utilized by RABID. This enables R compatibility and reduces the memory footprint of a RABID job. Third, R performs optimizations that reduce the communication overhead using three strategies. RABID uses a static analysis to determine the subset of data visible within a function that is actually needed by the function, reducing the communication volume of data sent to an R function, and the memory footprint by 80% of the original. Next, RABID performs *operation merging* to dramatically reduce the communication of intermediate values by 60% on average. Finally, RABID uses communication pipelining to allow communication and computation to proceed in parallel, and increase performance by 20% on average in our applications. These techniques together allow RABID to support the familiar R programming model while providing users with high performance parallel executions.

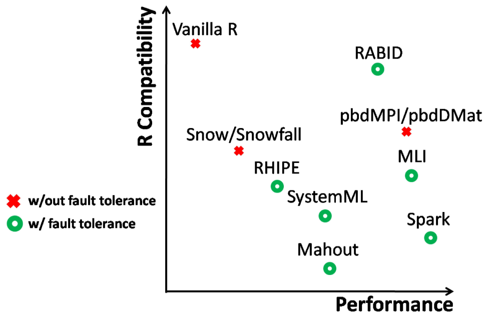


Figure 1. Different frameworks compared by R compatibility and performance.

2. RABID Overview

The RABID system allows high performance parallel executions of R on clusters. It does this by bridging the gap between R and and the parallel data engine. By targeting Spark we can leverage its efficiency in handling iterative computations and its fault tolerance mechanisms. Detailed system design can be referred in [4].

RABID provides distributed data structures and low-level and high-level operations on those data structures. The low-level operations target RABID's distributed R list and the high-level operations target distributed data frames and matrices as well as providing parallel data mining functions. We also support access to the Hadoop Distributed File System (HDFS) from R programs. The *R task* coordinates distributed R code hosted by either the Renjin or GNU R interpreters on server nodes and the *optimizer and scheduler*, as the name implies, performs optimizations described in Section 3 and schedules distributed R functions.

R scripts, written by a RABID user, are submitted through a web server to the RABID (and Spark) master that runs the user's command. An R session process (*R driver*) on the master runs the user's script with RABID support. It keeps dataset variables in symbol tables, schedules DAG structured jobs[6] and maintains

[*] This work was supported by Huawei Technologies, US R&D Center in a gift to Purdue University. This material is also based upon work supported by the National Science Foundation under Grant No. 1216809. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of Huawei Technologies or the National Science Foundation.

user defined R functions (UDFs). By default, the R code is executed using the Renjin[1] R virtual machine, which is written in Java. Renjin was chosen because it, like Spark, is implemented in Java, and consequently can be better integrated with Spark.

3. Optimizations

Five challenges faced by any parallel and distributed R systems are (1) Maintaining the R programming model; (2) Enabling efficient iterative parallel R execution; (3) Providing fault tolerance; (4) Minimizing the memory footprint and (5) Minimizing communication overheads during the execution of an R program. The first challenge is to maintain usability and RABID does this by maintaining R compatibility, as briefly described in Section 2. RABID meets the second and third challenges by being implemented on top of the Spark framework, which allows for efficient execution of iterative computations and provides fault tolerance. RABID meets the fourth and fifth challenges by applying the optimizations including:

- Reducing the memory footprint. On the driver side, we collect only useful free variables that are serialized to worker nodes. On the worker side, Spark workers share the datasets with the Renjin interpreter.
- Optimizing communications. Datasets are blocked in chunks to reduce the serialization overheads and the data transmission is pipelined.
- Merging operation. Adjacent *non-shuffling* data operations are merged together to reduce the data transmission.

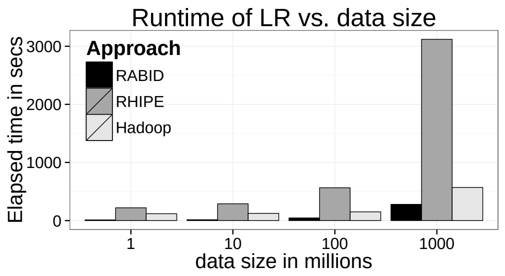
4. Evaluation

We use two workloads, an R implementation of logistic regression (which we refer to as LR) that implements a gradient descent algorithm to compute the weight of 1 billion 10-D data points and an R implementation of K-means that performs a clustering on movie ratings. Both are widely used in data mining applications. Our experiments show the R scripts running on RABID provide improved performance compared to the implementations in Hadoop 0.20.205 and RHIPE 0.7. First, we show that RABID scales well with these workloads and that in the data streaming mode our optimizations give significantly performance improvements.

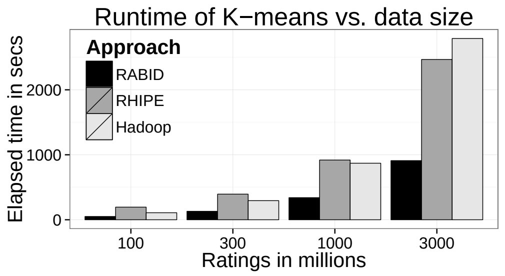
LR and K-means are both iterative algorithms. LR is run using a synthetic dataset with 1 billion 10-D data points. K-means uses the movie dataset with 3 billion ratings from [3]. We use the default block size for data communication unless noted otherwise. Experiments were conducted in a 26-node Linux cluster: each node has 8 cores and 16 GB RAM. The cluster is running RHEL 6.5, Linux Kernel 2.6. Figure 2 shows our evaluation of RABID performance.

5. Conclusions and Future Work

RABID provides R users with a familiar programming model that scales to large clusters, allowing larger problem sizes to be efficiently handled. Unlike other systems that require R programmers to use unfamiliar languages or programming models, RABID users can write R scripts to create a data analysis job. RABID is implemented on Spark and uses operation merging, data pipelining and analysis of the environment variables needed by a UDF to further improve performance. RABID outperforms Hadoop and RHIPE on our benchmarks. Development continues on RABID to support more high-level functions and to implement further optimizations. RABID is cloud-ready and future work will target cloud systems.



(a) Runtime in seconds for LR over 3 iterations on different data sizes in 26-node cluster.



(b) Runtime in seconds for K-means over 3 iterations on different data sizes in 26-node cluster.

Figure 2. Runtime in seconds for LR and K-means (lower is better).

References

[1] Renjin: The R programming language on the JVM. http://www.renjin.org/.

[2] The R project for statistical computing. http://www.r-project.org.

[3] Faraz Ahmadand, Seyong Lee, Mithuna Thottethodi, and T. N. Vijaykumar. PUMA: Purdue MapReduce Benchmarks Suite. Technical Report Purdue ECE Tech Report TR-12-11.

[4] Hao Lin, Shuo Yang, and Samuel P. Midkiff. RABID: A Distributed Parallel R for Large Datasets. In *Big Data (BigData Congress), 2014 IEEE International Congress on*, pages 725–732, June 2014.

[5] Robert A. Muenchen. The Popularity of Data Analysis Software. http://r4stats.com/articles/popularity/.

[6] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster Computing with Working Sets. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10.