A Parallel R Framework for Processing Large Dataset on Distributed Systems*

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 in both enterprise and scientific institutions. Statistical languages provide rich functionality and ease of use for data analysis and modeling and have a large user base. R [3] is one of the most widely used of these languages, but is limited to a single threaded execution model and to the memory on a single node that limits the size of problems that can be solved. We propose a framework for a highly parallel R called R Analytics for BIg Data (RABID). We achieve the goal of providing data analysts with the easy-to-use R interface that effectively scales to clusters by integrating R and the MapReduce-like distributed Spark [15] runtime. We provide preliminary experimental results showing the promise of our system.

Keywords

Distributed computing, big data analytics in cloud, R, Data Mining

1. INTRODUCTION

The R [3] language and system was specifically designed for data analysis computations and graphics. It has become increasingly popular over time and is the top software tool in the data analysis community according to some surveys [12]. Relative to tools such as SAS, SPSS, Python, etc., R is an excellent candidate for parallel data analytics since it was developed for sequential data analytics, is a cross-platform and open source system and has a rich collection of add-on packages. Despite its benefits, R suffers from the severe restrictions that its execution model is single threaded and memory is limited to what is available on a single node. Computational science and engineering and web applications have led to very large datasets becoming common. For example, CERN stores 24 Peta-bytes of data each year which must be analyzed and interrogated. With the growth of cloud-based infrastructures there are increasing amounts of data in logs tracking user behavior and system performance. At the same time, cloud services are providing opportunities to tackle problems of storing and processing large datasets. Allowing R to process these large amounts of data will allow the large R user-community to take advantages of widespread parallel resources such as the cloud.

Several other projects have focused on allowing R to run on larger systems and tackle larger datasets. Two parallel R packages are `Snow` [14] and `Snowfall` [10]. Their underlying distributed runtime framework is MPI and they only provide access to limited APIs that support parallel functions. The `pbdMPI` package [13] allows a user to program with low-level message-passing calls, which are very different from the current R programming model. `pbdDMat` [13] provides a higher level distributed matrix abstraction based on MPI and primarily targets linear algebra. `RHIPE` [2] provides an R programming framework on top of Hadoop's MapReduce [1], but requires users to learn and use map-reduce. `Ricardo` [6] also uses Hadoop to parallelize data computation, but requires users to embed queries in R scripts to enable Hadoop computation. All of these systems require R users to deviate significantly from standard R programming. None of the MPI based systems supports fault tolerance.

Distributed computing frameworks designed for iterative workloads seem, from a purely technical perspective, to be the right way to support large scale data mining and statistical modeling in the cloud. In practice R programmers, used to its functional and data analytic oriented programming model, must overcome a steep learning curve to use these systems. RABID integrates Spark [15] and R to provide a distributed data computing framework that runs efficiently on clusters in the cloud. Instead of writing hundreds of lines of non-R code, data analysts need only to produce several or tens of lines of R-style scripts to create a work-flow data process.

This paper makes the following contributions:

1. It describes the RABID system, its compatible distributed data structures, API operators and runtime system that allow R to scale to distributed systems and provide fault tolerance;

2. It describes the blocking data reorganization and the runtime operation merging optimization that reduces data movement and improves system performance;

*This work was supported by Huawei Technologies, US R&D Center, as a gift research funding

```
DATA <- lapply(rb.readLines('hdfs://...') as.numeric(a), cache=T)
set.seed(1000)
centroids <- as.list(sample(DATA, 16))
func <- function(a) { ... }
while(cond < threshold) {
    newCen <- as.list(
            aggregate(
                    x=lapply(DATA, func),
                    by=id,
                    FUN=mean))
    tempDist <- sum(mapply(function(x,y) dist(x[2],y[2]),
                    centroids, newCen))
    centroids <- newCen
}
```

Figure 1: K-Means Clustering as an example of a using RABID. API functions are in bold font, RABID distributed data structures are in all capital letters.

3. It shows RABID's performance on two benchmarks relative to RHIPE and Hadoop.

This paper is organized as follows. We briefly go over the background of the R language and related work in Section 2. We describe the architecture of RABID in Section 3 and details of system design in Section 4. Application cases and evaluations are presented in Section 5. Finally, we draw conclusions and discuss the future work.

2. RELATED WORK

We discussed related R-based systems in the introduction.

The open source Hadoop [1] ecosystem (an open source MapReduce [7], Distributed File System (HDFS), etc.) and Dryad [9] are widely used. SystemML [8] is a matrix-based Hadoop extension framework for machine learning that uses a domain specific language. Hadoop's design, however, precludes it from efficiently solving iterative problems because of job setup overheads and unnecessary disk I/Os [5, 15]. Data mining and statistical analysis, the foundation of deep analysis, are iterative problems with analysts exploring data of interest and building a model based upon some assumptions or hypothesis. The Spark [15] framework solves iterative data processing problems, which enables 20X performance improvements over Hadoop for some workloads that need in-memory iterative analysis, while retaining features like fault tolerance and high availability. Our work provides a familiar interface to the R community and spares these data analysts from learning to program C++, Java or Scala.

3. OVERVIEW OF THE RABID PROGRAMMING MODEL AND SYSTEM

3.1 RABID Programming Model

R is, at its core, a functional language that passes data through a series of operators (often represented as functions) that repeatedly transform the data. Iterative computations often involve some invariant data serving as input to a series of operations in a loop, with a transformed version of this data serving as input to the next iteration of the loop. The series of operators that transform some input data D to a new set of data D' is the *lineage* of D'. Loop bodies are often a lineage.

We illustrate the R (and RABID) programming model using the K-Means Clustering algorithm script shown in Figure 1. In the example script, the RABID API *lapply()* applies an R function *as.numeric* to each line read by *rb.readLines()*. *as.numeric()* converts each record to a numeric vector. The *cache* parameter requests R to cache the numeric working set in RAM if feasible. Next, a set of randomly picked *centroids* is defined as a list. In the iterative optimization loop, both *lapply()* and *aggregate()* (a reduce-like function that groups records by user-specified keys) are repeatedly invoked to compute the centroids closest to each record in *data* and to update the new centroids with the mean of points in the same cluster (aggregated with same key *id*), respectively. These transformations are using the user defined R functions *func* and the built-in function *mean*. The loop continues until the convergence condition is reached. We note that the lineage of the result of each loop iteration is the two `lapply` and one `aggregate` operation.

RABID enables parallelism and the ability to handle large datasets in two ways. First, it provides a set of R compatible distributed data types that allow data to be distributed across multiple servers in a cluster. Second, it provides parallel implementations of standard R functions that operate on these distributed data types. In the example, the RABID versions of *lapply()* and *aggregate()* are data parallel, and *as.list()* is used to collect distributed data into an R list on the master machine. Except for *rb.readLines()*, all other RABID functions override standard R and provide users with signatures that differ only in optional parameters.

3.2 Distributed Data Types and APIs

RABID provides several R-compatible data types that are "drop-in" replacements for the corresponding standard R data types. A key data type for R (and other functional languages) is the *list*. R lists differ from arrays in that all elements of an array should have a common type, but different list elements can have different types. RABID supports the `BigList` distributed type. R and RABID also provides data types such as matrices and data frames (well-structured data with rows of records and columns of attributes, similar to database data tables.)

Each distributed dataset is accessed as an instance of a "Big*" class. These objects are descriptors, or *promise objects*, i.e., instead of actually containing the data they store the necessary information to obtain the data. Thus distributed datasets are lazily evaluated and operations are executed only when their values are needed, providing opportunities for exploring performance optimizations. A UML view of the structure of the RABID data types is shown in Figure 2.

More low-level APIs can be referred in [11]. Table 1 shows high-level RABID APIs that manipulate the BigMatrix and BigDataFrame data types. The APIs take optional tuning parameters to control communication granularity but are otherwise identical to vanilla R APIs, making RABID easy for R programmers to use. Figure 4 shows that these APIs are part of the RABID package.

BigDataFrame	BigMatrix
rb.data.frame() creates a distributed "data frame" dataset	*rb.matrix(...)* creates a distributed matrix dataset
'$' return a BigList column of specified attribute	*'+', '-', '*', '/', '%*%', '^', '%%'* basic matrix arithmetic
aggregate(x, by, FUN) aggregated by specified attribute	*'t', 'cor', ...* more matrix and statistical computation
adply/apply(X, MARGIN, FUN) apply functions to row or column marginal in the data frame/matrix	

Table 1: Sample APIs for BigDataFrame and BigMatrix.

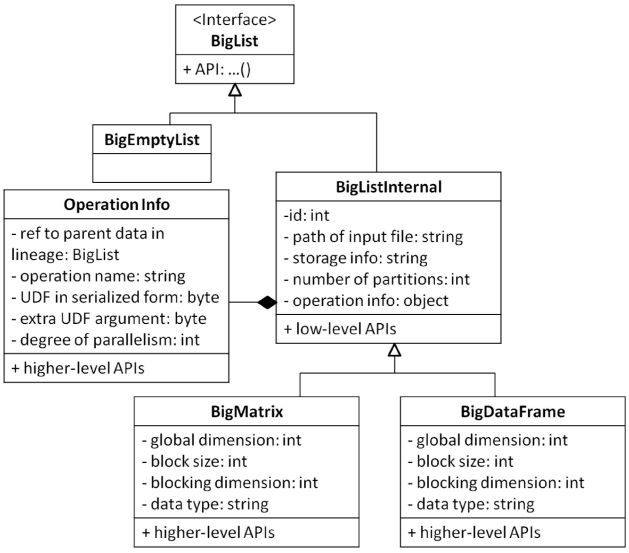


Figure 2: UML model for Big* data type structures.

3.3 The RABID Runtime

RABID is designed to be a cloud-based software service that translates a user's R scripts into Spark jobs in a cluster. Figure 3 gives the runtime overview of our proposed system. The RABID user focuses on developing R scripts while the web server communicates with the RABID (and Spark) master in the cloud to run the user's command. The R session running on the master with RABID support is called the R driver script. It keeps dataset variables in symbol tables, schedules DAG structured jobs [15] and maintains user defined functions (UDFs). Each task worker on a slave directs tasks to serialize data into parallel R sessions, which carry out UDF computations.

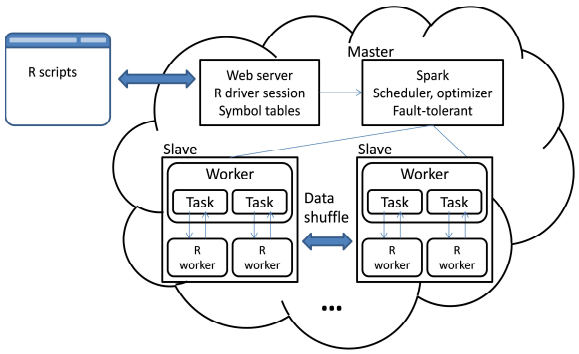


Figure 3: RABID Framework Runtime Architecture.

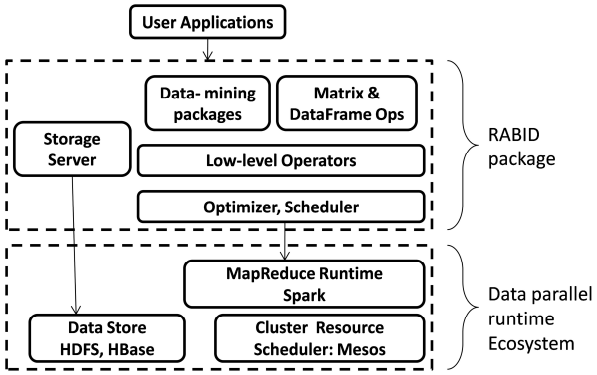


Figure 4: RABID Framework Architecture.

4. SYSTEM DESIGN AND IMPLEMENTATIONS

When running an R script the RABID system has several key goals. Computation, expressed as R functions, and data must fisrt be transferred to R worker sessions on individual nodes from the underlying Spark runtime. Each execution of a UDF in RABID (1) starts a Spark operation and a set of R processes to execute the UDF; (2) starts a pipelined data transmission; and (3) creates a new distributed dataset for the output. This transmission can be optimized by pipelining the delivery of data. Communication can be further optimized by merging operations so that intermediate results computed along a sequence of operations are not communicated but are used on a node within the merged operation. Finally, Spark's underlying fault tolerance mechanism is exploited to allow RABID to respond quickly to both user code failures and system failures.

4.1 Pipelining Data to R Sessions

The underlying HDFS chunks data into *splits* in Spark. RABID, however, would like to do pipelined communication between Spark and R so that processing can begin before all data has arrived at R processes, allowing an overlap between communication and computation. Communicating at split granularity can also cause more data to be sent to a node than can be held in physical memory, resulting in excess paging. To enable pipelining and reduce the memory pressure, RABID breaks splits into smaller *data blocks*. Because data blocks that are too small can also be inefficient, RABID provides an optional user parameter to tune the block size.

As shown in Fig. 5, RABID reads records from a file one-by-one. After a map-like operation (e.g. *lapply()*) is applied to each record, the result is stored in a fixed sized R list that

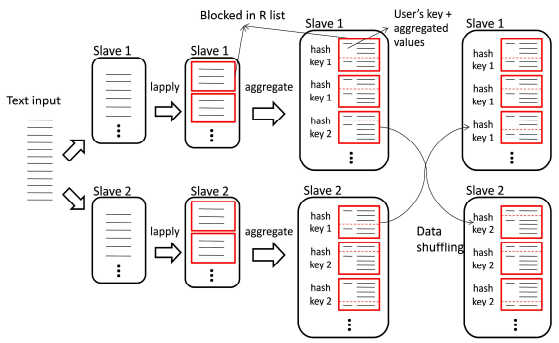


Figure 5: Data blocking.

Algorithm 1 Clear up function environment (closure).

Input: n: an expression node in the R AST; f: the UDF; e: new UDF environment to be serialized
Algorithm: closure (n, f, e)
  if number of node n's children > 1
    for i = 1, 2, ..., number of node n's children
      closure(n[i], f, e) */*recursive over children nodes*/*
  else */*node n has no child*/*
    if n is a variable identifier
      if n is NOT an argument of f */*free variable*/*i
        if n is NOT in the R core packages
          add n into the UDF environment e

is written to Spark when full and refilled until all the data is computed.

4.2 Distributing computation to R

System functions are already present on remote R nodes so only UDFs need to be distributed. R is a dynamically scoped language, i.e., free variables in a function are resolved to the variable with the same name in the environment of the nearest calling function on the runtime stack. Free variables in UDFs to be remotely executed must be identified and added to the environment sent to the remote node. Simply serializing and sending the enclosing dynamic environments to the remote node would communicate large amounts of unneeded data. RABID uses Algorithm 1 to recursively resolve the free variables in a UDF and bind the UDF to the new environment. For better performance UDFs are then compiled with this environment into byte code by the R "compiler" package [3]. R processes for this UDF are created on the remote nodes, and the serialized code and environment are sent to these processes.

Because RABID datasets are promise objects, data accesses by the UDF begin the pipelined communication of needed data to the node and R process. This allows data transmission and UDF execution to be overlapped.

4.3 Merging Operations for Better Performance

The overhead of executing UDFs can be reduced by merging a sequence of RABID operations that are in a lineage, i.e., operations that consume data produced by other operations in the sequence, into a single Spark operation. We call these *merged operations* or *m-ops*. RABID merges adjacent RABID operations that work on the same data split as long as there is no data shuffling or reconstruction of splits as happens with aggregations. The aggregation operation can be considered to work like a barrier. Other operations, like *lapply()*, *Reduce()*, etc., that precede or follow a barrier can be merged into an m-op.

Merging operations has several benefits. First, the merger allows one set of R processes to be created for the entire m-op rather than creating multiple ones for each operation within the m-op. Second, when data is pipelined to the m-op, one set of input data is transmitted to the m-op and all other data needed by constituent operations is produced within the m-op. This is much cheaper than creating a new distributed dataset on Spark and writing results to it with redundant data transmission and serialization, as would be necessary if the operations of the m-op were executed separately. With m-ops there is a single UDF transmission, a single pipelined communication phase and startup, and a single output distributed dataset created and written.

4.4 Fault Tolerance

We take advantage of Spark's existing heartbeat based worker-side fault tolerance mechanisms to handle node failures. Tasks are restarted when the heartbeat indicates a worker task failure. Lineage information for tasks stored in both the R driver process and the Spark master process can allow lost results to be recomputed, reducing checkpointing overheads. Periodic checkpointing will still allow the system to recover faster from failures.

User code errors terminate R worker sessions. In RABID, these errors will be caught and will immediately terminate the job (fast fail) so that Spark's fault-tolerance mechanism does not try and recompute tasks or other useless operations. RABID also collects R's verbose error information from each worker's `stderr` to give users information about the cause of a failure. On the R master, job information such as the symbol table of dataset variables, program execution state and worker information are replicated by the backup by using Apache Zookeeper.

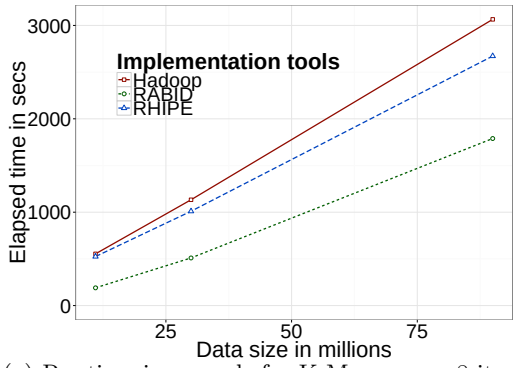
5. EVALUATION

5.1 Experimental Setup

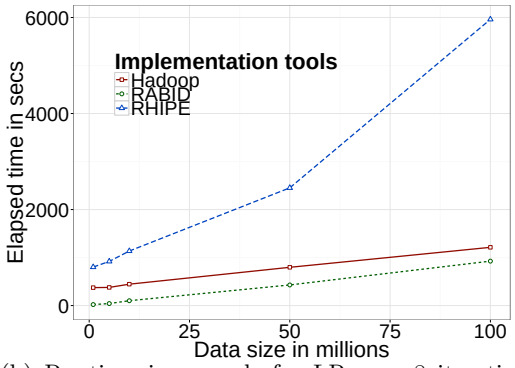
Experiments were conducted in a 12-node Linux cluster cluster: each node has 8 cores and 16 GB RAM. The cluster is running Ubuntu 12.04, Linux Kernel 3.2. We evaluated two data mining algorithms: Regression (LR) and K-means Clustering. We demonstrate that their implementations in RABID provides improved performance compared to those in Hadoop 0.20 and RHIPE 0.7. LR and K-means are both iterative algorithms implemented using the BigList data type. LR is run using a synthetic dataset with from 1 million to 100 million data points. K-Means uses the movie dataset with 100 million ratings from [4].

5.2 Experiment Results

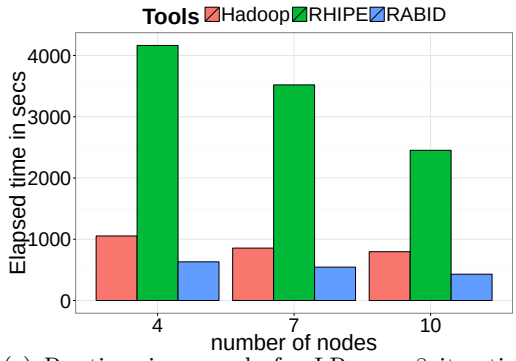
Figure 6(a) gives the time spent for runs of K-means with input movie dataset of 10 million, 30 million and 100 million ratings and RABID again shows significant performance benefits.



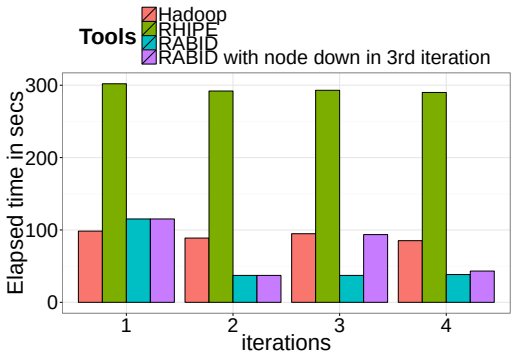
(a) Runtime in seconds for K-Means over 8 iterations on different data sizes



(b) Runtime in seconds for LR over 8 iterations on different cluster sizes



(c) Runtime in seconds for LR over 8 iterations on different cluster sizes



(d) Runtime in seconds for LR on each iteration

Figure 6: Performance results for LR and K-Means.

Figure 6(b) shows the elapsed time of our approach compared to the Hadoop and RHIPE implementations of LR on datasets with 1 million, 10 million and 100 million points over 8 iterations. RABID outperforms RHIPE by an order of magnitude and also Hadoop by a smaller amount. Once the data are cached in the first iteration, following iterations are performed quickly by in-memory computation in RABID. Therefore, if more iterations are needed, RABID benefits more and gives better speedups than Hadoop and RHIPE. Figure 6(c) shows the scalability over the number of nodes. To illustrate fault tolerance, we manually disable one node at the 4th step and see how the system recovers in Figure 6(d).

6. FUTURE WORK AND CONCLUSION

We have presented the RABID system that integrates R and Spark. RABID provides R users with a familiar programming model that scales to large cloud based clusters, allowing larger problems sizes to be efficiently solved. Unlike other systems that require R programmers to use unfamiliar languages or programming models, RABID uses can write R scripts to create a work flow data process. RABID uses operation merging and data pipelining along with optional user tuning of communication parameters to further improve performance and allow RABID to outperform Hadoop and RHIPE on our benchmarks. Development continues on RABID to support more high-level functions and to implement further optimizations, and RABID is cloud-ready to be made as a service.

7. REFERENCES

[1] Apache Hadoop. http://hadoop.apache.org/.

[2] RHIPE. http://www.datadr.org/getpack.html.

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

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

[5] Yingyi Bu, Bill Howe, Magdalena Balazinska, and Michael D. Ernst. HaLoop: Efficient Iterative Data Processing on Large Clusters. *Proc. VLDB Endow.*, 3(1-2):285–296, September 2010.

[6] Sudipto Das, Yannis Sismanis, Kevin S. Beyer, Rainer Gemulla, Peter J. Haas, and John McPherson. Ricardo: Integrating R and Hadoop. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, SIGMOD '10, pages 987–998, New York, NY, USA, 2010. ACM.

[7] Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM*, 51(1):107–113, January 2008.

[8] Amol Ghoting, Rajasekar Krishnamurthy, Edwin Pednault, Berthold Reinwald, Vikas Sindhwani, Shirish Tatikonda, Yuanyuan Tian, and Shivakumar Vaithyanathan. SystemML: Declarative Machine Learning on MapReduce. In *Proceedings of the 2011 IEEE 27th International Conference on Data Engineering*, ICDE '11, pages 231–242, Washington,

DC, USA, 2011. IEEE Computer Society.

[9] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad: Distributed Data-parallel Programs from Sequential Building Blocks. In *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, EuroSys '07, pages 59–72, New York, NY, USA, 2007. ACM.

[10] Jochen Knaus. snowfall: Easier cluster computing. http://cran.r-project.org/web/packages/snowfall/.

[11] Hao Lin, Shuo Yang, and S.P. Midkiff. RABID – A General Distributed R Processing Framework Targeting Large Data-Set Problems. In *Big Data (BigData Congress), 2013 IEEE International Congress on*, pages 423–424, 2013.

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

[13] D. Schmidt, G. Ostrouchov, Wei-Chen Chen, and P. Patel. Tight Coupling of R and Distributed Linear Algebra for High-level Programming with Big Data. In *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion.*

[14] Luke Tierney. snow: Simple Network of Workstations. http://cran.r-project.org/web/packages/snow/.

[15] 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, pages 10–10, Berkeley, CA, USA, 2010. USENIX Association.