Big Data Analytics: A Tutorial on Information Process Fusion with MapReduce

Sergio Ramírez[a,*], Alberto Fernández[a], Salvador García[a], Francisco Herrera[a]

[a]*Department of Computer Science and Artificial Intelligence, University of Granada, Granada, Spain*

Abstract

TODO

Keywords: Big Data Analytics, MapReduce, Information Fusion, Spark, Machine Learning

---

1. Introduction

TODO

In order to address all these objectives, this paper is organized as follows. First, Section 2 presents an introduction on the MapReduce programming framework, also stressing some alternatives for Big Data processing. Section 3 includes an overview on those technologies currently available to address Big Data problems from a distributed perspective. Section 4 presents the core of this paper, analyzing the different design options for developing Big Data analytics algorithms regarding how the partial data and models are aggregated. Then, we show a case study in Section 5 to contrast the capabilities regarding scalability of the different approaches previously introduced. Finally, Section 6 summarizes and concludes this paper.

2. MapReduce as Information Process Fusion

INTRO

2.1. MR

The rapid growth and influx of data from private and public sectors has popularized the notion of "Big data [1]". The surge in Big Data has led to the development of custom models and algorithms that are able to extract significant value and insight in different areas such as medical, health care, business, management and so on [2, 3, 4].

---

*Corresponding author. Tel:+34-958-240598; Fax: +34-958-243317

Email addresses: `sramirez@decsai.ugr.es` (Sergio Ramírez), `alberto@decsai.ugr.es` (Alberto Fernández), `salvagl@decsai.ugr.es` (Salvador García), `herrera@decsai.ugr.es` (Francisco Herrera)

In order to to provide robust and scalable solutions, new research paradigms and developmental tools have been made available. These frameworks have been designed to ease both the storage necessities and the processing of Big Data problems [1].

The MapReduce execution environment [5] is the most common framework used in this scenario. Being a privative tool, its open source counterpart, known as Hadoop, has been traditionally used in academia research [6]. It has been designed to allow distributed computations in a transparent way for the programmer, also providing a fault-tolerant execution scheme. To take advantage of this scheme, any algorithm must be divided into two main stages: Map and Reduce. The first one is devoted to split the data for processing, whereas the second collects and aggregates the results.

Additionally, the MapReduce model is defined with respect to an essential data structure: the <key,value> pair. The processed data, the intermediate and final results work in terms of <key,value> pairs. To summarize its procedure, Figure 1 illustrates a typical MapReduce program with its *Map* and *Reduce* steps. The terms $k_i : v_j$ refer to the key and value pair that are computed within each Map process. Then, values are grouped linking them to the same key, i.e. $k_i : v_j, \ldots, v_h$, and feed to the same Reduce process. Finally, values are aggregated with any function within the Reduce process to obtain the final result of the algorithm.

From this initial model, more recent alternative frameworks have arisen to provide more efficient computations for iterative process, which are the basis for any Machine Learning algorithm. Among them, Apache Spark [7, 8] is clearly emerging as a more commonly embraced platform for implementing Machine Learning solutions that scale with Big Data.

2.2. Alternatives

TODO

3. Big Data Technologies for Analytics

TODO

3.0.1. Hadoop

TODO

3.0.2. Spark

Spark[7, 8] [1] has been developed to overcome data reuse across multiple computations. It supports iterative applications, while retaining the scalability and fault tolerance of MapReduce, supporting in-memory processes. It is implemented in the functional programming language Scala and has further programming interfaces in Java, Python and the declarative query language SparkSQL.

[1] `http://spark.apache.org/`

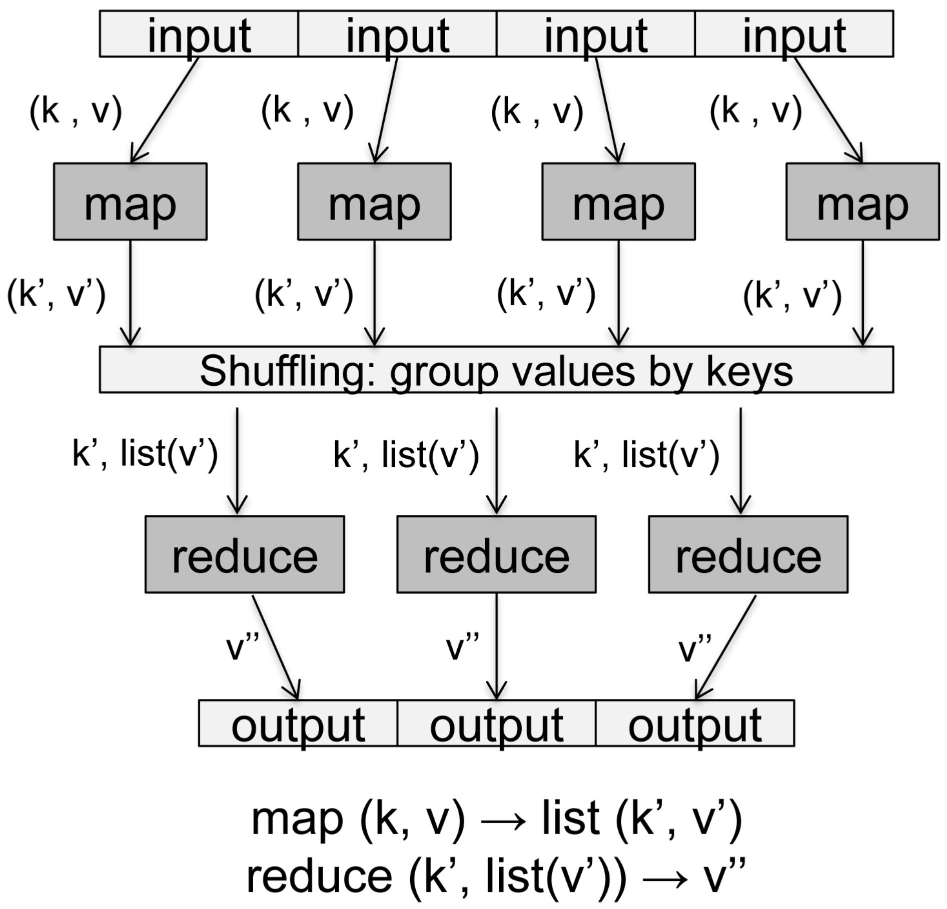

Figure 1: The MapReduce programming model.

Spark rests upon two main concepts: resilient distributed datasets (RDD), which hold the data objects in memory, and transformations or actions that are performed on the datasets in parallel.

RDDs are a collection of distributed items, with read-only-mode and fault-tolerant. As stated above, these data structures allow users making intermediate results to persist in memory, as well as controlling their partitioning to optimize data placement, and also manipulating them using a rich set of operators.

These operators are based on coarse-grained transformations. Specifically, there are 20 transformations and 12 actions usable. The difference between the two kinds of operators is that transformations are functions to generate and manipulate RDDs out of input files or other RDDs, whereas actions can only be used on RDDs to produce a result set in the driver program or write data to the file system[7].

In order to manage data objects within the algorithms developed under Spark, two API are available in current version 1.6.0. On the first hand, DataFrame API, which is clearly the most stable option and therefore it offers the best performance. On the other hand, an API simply named as Dataset may be used also. However, being the most recent add to this framework, it does not yet leverage the additional information it has, and it can be slower than RDDs. It is expected to be fully developed and improved for Spark version 2.0.

It is worth to point out that Spark implements a Machine Learning library known as MLlib[2], included within the MLBase platform [9]. MLlib currently supports common types of machine learning problem settings, as well as associated tests and data generators. It includes binary classification, regression, clustering and collaborative filtering, as well as an underlying gradient descent optimization primitive. Finally, Mahout [10], maybe the most well-known Machine Learning library for Big Data, has recently announced its migration to Spark, due to the good properties of this framework for the execution of this type of algorithms. Neither of them support any fuzzy modeling algorithm yet.

3.0.3. Flink

The Flink framework[3] has its origins in the Stratosphere project [11, 12]. The main idea behind this model is the optimization of execution plans, which is based on basic technologies from relational database systems. It is written in Java, but it also supports Scala and Python languages. It avoids the use of a declarative query language with aims at simplifying the development of applications for those users with less programming experience.

The Apache Flink framework currently provides 17 functions for data transformation, which are known as PACT operators, which are quite similar to that of the Spark framework. The main difference is based on two unique iteration operators, namely bulkIteration and deltaIteration. The former operator employ the whole dataset in every run, whereas the latter divides the data in a workset and solution set. One of the advantages of these iterators is that fewer data has to be computed and sent between nodes [13]. In accordance to the former, these are better suited for an efficient implementation of both machine learning and data mining algorithms, supporting also data stream processing.

As stated at the beginning, one of the most interesting features of the Flink framework is known as the PACT optimizer. It creates several semantically equivalent execution plans by reordering the PACT operators based on code analysis and data access conflicts [11, 12]. The physical optimization chooses efficient strategies for data transport and operator execution on specific nodes. A cost-based strategy selects the most efficient plan based on criteria like network or storage I/O costs.

Finally, the Flink ecosystem also provides a Machine Learning library, FlinkML[4], which includes several data preprocessing, supervised learning, and recommender systems. It is still at a preliminary stage, but it is expected to add new algorithms in a short future.

[2]`http://spark.apache.org/docs/latest/mllib-guide.html`
[3]`http://flink.apache.org/`
[4]`https://ci.apache.org/projects/flink/flink-docs-release-0.10/libs/ml/`

4. Big Data Analytics on Fusion Process

"Synchronizing flags" in every distributed algorithmic approach is undoubtedly the most challenging part during the design process. This issue is not an exception in the MapReduce scheme. In this particular case, developers must take two significant decisions. On the one hand, the components selected for the key-value representation in both the Map and Reduce input and outputs. On the other hand, how the list of values are aggregated in the Reduce step.

For Big Data applications in Machine Learning, these choices are of extreme importance for the behavior of the final models. In this section, we want to analyze in detail this characteristic of the MapReduce programming environment by introducing a taxonomy to organize current state-of-the-art approaches regarding how partial models from the Map task are aggregated. We will determine how this fact may also affects the actual scalability of the whole system.

Specifically, we distinguish among three different type of models. First, those that carry out a direct fusion of partial models, thus becoming and ensemble system (Section 4.1). Second, those that create a partial model within different Maps, and combine them to obtain a single output (Section 4.2). Finally, we identify those designs which distribute both data and sub-models to iteratively build the final system (Section 4.3).

4.1. Direct fusion of models: ensemble approach

In every MapReduce framework, the system partitions the input training data to feed the Map functions. This computational paradigm reminds of the traditional *bootstrap aggregating* (bagging) approach; in this case with no replacement. Therefore, it is straightforward to obtain an ensemble-based classifier [**?**] by directly joining every single model obtained in a different Map task.

The premise for this type of methodologies is simple yet effective. Each Map task works independently on its chunk of data. Depending on the user's requirements, each Map function is devoted to build a one or more models. For example, a pool of 100 classifiers to be obtained from 5 Maps will result on 20 classifiers per Map, each one of them built from a 20% of the original dataset. The "key" is shared by all values given as output from the Map. This fact makes the logic of the Reduce phase to stress for its simplicity. Every single classifier is directly joined into the final system.

One of the first approaches to build an ensemble of classifiers with MapReduce was the Parallel Learner for Assembling Numerous Ensemble Trees (PLANET) in 2009 [14]. More recent proposals include a Random Forest for Hadoop [15, 16], whose implementation can be found at Mahout-MapReduce Machine Learning library [17, 18].

FS: [19] es ensemble o approximate? Sergio please encargate de tu propio modelo.

There are two main advantages for this type of design. On the one hand, we have stressed its direct synergy with the MapReduce programming model, easing the programmer implementation. On the other hand, we must refer to the degree of diversity of those models obtained from different Maps, being a crucial characteristic for the success of these types of methods [20].

In spite of the previous goodness, there is one significant problem: there is a limit in the scalability degree. It must be acknowledge that there is a strong relationship between the number of Maps and the required number of models. In other words, we cannot decrease the total running time by adding a larger number of Maps, as this may result in "void" functions with a data input but no output.

4.2. Approximate fusion of models: aggregation of partial systems

Modelos que funcionan directamente.

This way, the most straightforward way to design an scalable algorithm is trying to avoid dependencies among parallel tasks.

FRBCS,

Another interesting approaches from the boosting perspective [21] were AdaBoost.PL, LogitBoost.PL and MultiBoost.PL [22]. These include and ensemble of ternary classifiers [23], or an ensemble of ensembles. The methodology in this case is exactly the same than for XXX.

IS, Clustering kNN,

4.3. Exact fusion for scalable models: distributed data and models' partition

TODO

5. Practical Study on XXX

TODO

6. Concluding Remarks

TODO

Acknowledgments

This work have been partially supported by the Spanish Ministry of Science and Technology under projects TIN2014-57251-P and TIN2015-68454-R; and the Foundation BBVA project 75/2016 BigDaP-TOOLS.

[1] A. Fernández, S. Río, V. López, A. Bawakid, M. J. del Jesus, J. Benítez, F. Herrera, Big Data with cloud computing: An information sciencesight on the computing environment, MapReduce and programming framework, WIREs Data Mining and Knowledge Discovery 4 (5) (2014) 380–409.

[2] K. Kambatla, G. Kollias, V. Kumar, A. Grama, Trends in Big Data analytics, Journal of Parallel and Distributed Computing 74 (7) (2014) 2561–2573.

[3] C. P. Chen, C.-Y. Zhang, Data-intensive applications, challenges, techniques and technologies: A survey on Big Data, Information Sciences 275 (2014) 314–347.

[4] X. Wu, X. Zhu, G.-Q. Wu, W. Ding, Data mining with Big Data, IEEE Transactions of Knowledge Data Engineering 26 (1) (2014) 97–107.

[5] J. Dean, S. Ghemawat, MapReduce: simplified data processing on large clusters, Communications of the ACM 51 (1) (2008) 107–113.

[6] T. White, Hadoop: The Definitive Guide, 4th Edition, O'Reilly Media, 2015.

[7] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauly, M. J. Franklin, S. Shenker, I. Stoica, Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing, in: Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12), USENIX, San Jose, CA, 2012, pp. 15–28.

[8] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, I. Stoica, Spark: Cluster computing with working sets, in: HotCloud 2010, 2010, pp. 1–7.

[9] T. Kraska, A. Talwalkar, J.Duchi, R. Griffith, M. Franklin, M. Jordan, Mlbase: A distributed machine learning system, in: Conference on Innovative Data Systems Research, 2013, pp. 1–7.

[10] S. Owen, R. Anil, T. Dunning, E. Friedman, Mahout in Action, 1st Edition, Manning Publications Co., 2011.

[11] A. Alexandrov, R. Bergmann, S. Ewen, J.-C. Freytag, F. Hueske, A. Heise, O. Kao, M. Leich, U. Leser, V. Markl, F. Naumann, M. Peters, A. Rheinlnder, M. J. Sax, S. Schelter, M. Hger, K. Tzoumas, D. Warneke, The stratosphere platform for big data analytics, International Journal on Very Large Databases 23 (6) (2014) 939–964.

[12] F. Hueske, M. Peters, M. Sax, A. Rheinlnder, R. Bergmann, A. Krettek, K. Tzoumas, Opening the black boxes in data flow optimization, PVLDB 5 (2012) 1256–1267.

[13] S. Ewen, K. Tzoumas, M. Kaufmann, V. Markl, Spinning fast iterative data flows, PVLDB 5 (11) (2012) 1268–1279.

[14] B. Panda, J. S. Herbach, S. Basu, R. J. Bayardo, Planet: Massively parallel learning of tree ensembles with mapreduce, Proceedings of the VLDB Endowment 2 (2) (2009) 1426–1437, cited By :122.

[15] J. Assuncao, P. Fernandes, L. Lopes, S. Normey, Distributed stochastic aware random forests - efficient data mining for big data, in: Proceedings - 2013 IEEE International Congress on Big Data, BigData 2013, 2013, pp. 425–426, cited By :7.

[16] Y. Wang, W. Goh, L. Wong, G. Montana, Random forests on hadoop for genome-wide association studies of multivariate neuroimaging phenotypes, BMC Bioinformatics 14 (SUPPL16), cited By :24.

[17] S. Owen, R. Anil, T. Dunning, E. Friedman, Mahout in Action, Manning, 2011.

[18] The Apache Software Foundation, Mahout, an open source project which includes scalable machine learning algorithms (2017).
URL `http://mahout.apache.org/`

[19] S. Ramrez-Gallego, I. Lastra, D. Martnez-Rego, V. Boln-Canedo, J. M. Bentez, F. Herrera, A. Alonso-Betanzos, Fast-mrmr: Fast minimum redundancy maximum relevance algorithm for high-dimensional big data., International Journal of Intelligent Systems 32 (2) (2017) 134–152.

[20] L. I. Kuncheva, Diversity in multiple classifier systems, Information Fusion 6 (1) (2005) 3–4.

[21] R. E. Schapire, A brief introduction to boosting, in: IJCAI, 1999, pp. 1401–1406.

[22] I. Palit, C. K. Reddy, Scalable and parallel boosting with mapreduce, IEEE Transactions on Knowledge and Data Engineering 24 (10) (2012) 1904–1916, cited By :42.

[23] R. E. Schapire, Y. Singer, Improved Boosting Algorithms Using Confidence-Rated Predictions, Machine Learning 37 (3) (1999) 297–336.