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

Sergio Ramírez-Gallego[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-Gallego), 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.

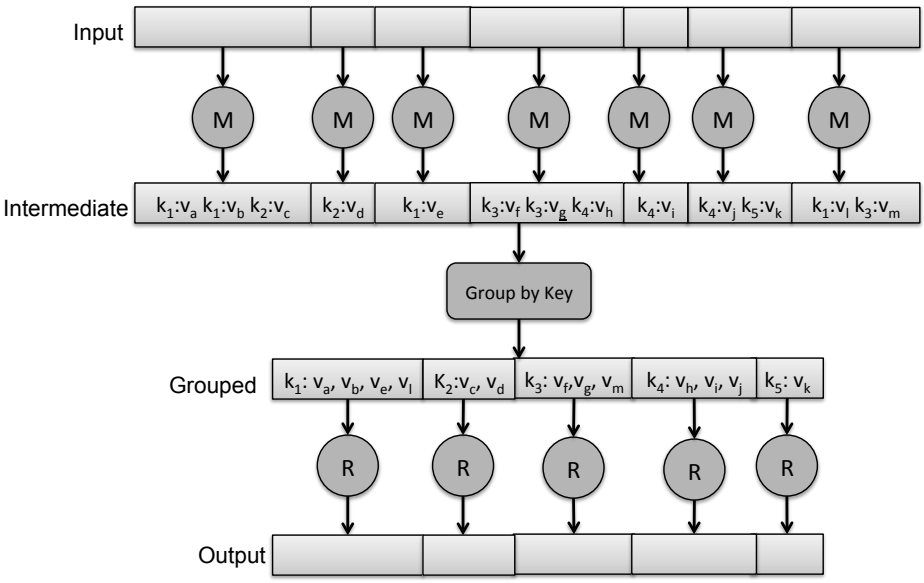

Figure 1: The MapReduce programming model.

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

3.1. Hadoop

TODO

3.2. Spark

Apache Spark Framework [**?**] was born in 2010 with the publication of Resilient Distributed Datasets (RDD) structures [**?**], the key idea behind Spark. Although ASF has a close relationship with many components from Hadoop Ecosystem, Spark provides specific support for every step in the Big Data stack, such as its own processing engine, and machine learning library.

Apache Spark [**?**] can be defined as a distributed computing platform which can process large volume data sets in memory with a very fast response time, thanks to its memory-intensive scheme. It was originally thought to tackle problems deemed as unsuitable for previous disk-based engines like Hadoop. Continued use of disk is replaced in Spark by memory-based operators that efficiently deal with iterative and interactive problems (prone to multiple I/O actions).

The heart of Spark is formed by Resilient Distributed Datasets (RDD), which transparently controls how data are distributed and transformed across the cluster. Users just need to define some high-level functions that will be applied and managed by RDDs. These elements are created whenever data are read from any source, or as a result of a transformation. RDDs consist of a collection of data partitions distributed across several data nodes. A wide range of operations are provided for transforming RDDs, such as: filtering, grouping, set operations, among others. Furthermore RDDs are also highly versatile as they allows users to customize partitioning for an optimized data placement, or to preserve data in several formats and contexts.

In Spark, fault tolerance is solved by annotating operations in a structure called lineage. Spark transformations annotated in the lineage are only performed whenever a trigger I/O operations appears in the log. In case of failure, Spark re-computes the affected brach in the lineage log. Although replication is normally skipped, Spark allows to spill data in local disk in case the memory capacity is not sufficient.

Spark developers provided another high-level abstraction, called DataFrames, which introduces the concept of formal schema in RDDs. DataFrames are distributed and structured collections of data organized by named columns. They can be seen as a table in a relational database or a dataframe in R, or Python

(Pandas). As a plus, relational query plans built by DataFrames are optimized by the Spark's Catalyst optimizer throughout the previously defined schema. Also thanks to the scheme, Spark is able to understand data and remove costly Java serialization actions.

A compromise between structure awareness and the optimization benefits of Catalyst is achieved by the novel Dataset API. Datasets are strongly typed collections of objects connected to a relational schema. Among the benefits of Datasets, we can find compile-time type safety, which means applications can be sanitized before running. Furthermore, Datasets provide encoders for free to directly convert JVM objects to the binary tabular Tungsten format. These efficient in-memory format improves memory usage, and allows to directly apply operations on serialized data. Datasets are intended to be the single interface in future Spark for handling data.

3.2.1. MLlib

MLlib project [**?**] was born in 2012 as an extra component of Spark. It was released and open-sourced in 2013 under the Apache 2.0 license. From its inception, the number of contributions and people involved in the project have been growing steadily. Apart from official API, Spark provides a community package index [**?**] (Spark Packages) to assemble all open source algorithms that work with MLlib.

MLlib is a Spark library geared towards offering distributed machine learning support to Spark engine. This library includes several out-of-the-box algorithms for alike tasks, such as: classification, clustering, regression, recommendation, even data preprocessing. Apart from distributed implementations of standard algorithms, MLlib offers:

- Common Utilities: for distributed linear algebra, statistical analysis, internal format for model export, data generators, etc.
- Algorithmic optimizations: from the long list of optimizations included, we can highlight some: decisions trees, which borrow some ideas from PLANET project [**?**] (parallelized learning both within trees and across them); or generalized linear models, which benefit from employing fast C++-based linear algebra for internal computations.
- Pipeline API: as the learning process in large-scale datasets is tedious and expensive, MLlib includes an internal package (*spark.ml*) that provides an uniform high-level API to create complex multi-stage pipelines that connect several and alike components (preprocessing, learning, evaluation, etc.). *spark.ml* allows model selection or hyper-parameter tuning, and different validations strategies like k-fold cross validation.
- Spark integration: MLlib is perfectly integrated with other Spark components. Spark GraphX has several graph-based implementations in MLlib, like LDA. Likewise, several algorithms for online learn-

ing are available in Spark Streaming, such as online k-Means. In any case, most of component in the ASF stack are prepared to effortlessly cooperate with MLlib.

3.3. Flink

Apache Flink [**?**] is a distributed processing component focused on streaming processing, which was designed to solve problems derived from micro-batch models (Spark Streaming). Flink also supports batch data processing with programming abstractions in Java and Scala, though it is treated as a special case of streaming processing. In Flink, every job is implemented as a stream computation, and every task is executed as cyclic data flow with several iterations.

Flink provides two operators for iterations [**?**], namely, standard and delta iterator. In standard iterator, Flink only works with a single partial solution, whereas delta iterator utilizes two worksets: the next entry set to process and the solution set. Among the set of advantages provided these iterators is the reduction of data to be computed and sent between nodes [9]. According to the authors, new iterators are specially designed to tackle machine learning and data mining problems.

Apart from iterators, Flink leverages from a PACT optimizer which analyzes the code and the data access conflicts to reorder operators and create semantically equivalent execution plans [10, 11]. Physical optimization is then applied on plans to boost data transport and operators' execution on nodes. Finally, PACT optimizer selects the most resource-efficient plan, regarding network and storage.

Furthermore, Flink provides a complex fault tolerance mechanism to consistently recover the state of data streaming applications. This mechanism is generating consistent snapshots of the distributed data stream and operator state. In case of failure, the system can fall back to these snapshots.

FlinkML is aimed at providing a set of scalable ML algorithms and an intuitive API to Flink users. Until now, FlinkML provides few alternatives for some fields in machine learning: SVM with CoCoA or Multiple Linear regression for supervised learning, k-NN join for unsupervised learning, scalers and polynomial features for preprocessing, Alternating Least Squares for recommendation, and other utilities for validation and outlier selection, among others. FlinkML also allows users to build complex analysis pipelines via chaining operations (like in MLlib). FlinkML pipelines are inspired by the design introduced by sklearn in [**?**].

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 [12]. More recent proposals include a Random Forest for Hadoop [13, 14], whose implementation can be found at Mahout-MapReduce Machine Learning library [15, 16].

FS: [17] 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 [18].

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.

One final drawback that must be stressed is related to the inner concept of ensemble. Being composed of a "large" set of individual models, two facts must be taken into account. On the one hand, since all of them are considered for the classification step, it is hard to understand the actual reason for the labeling of the queried data. On the other hand, the robustness of the whole ensemble system somehow depends on the individual quality of its components, as well as their diversity. Both properties are not easy to hold, and impose a restriction in the building of the ensemble, for both standard and Big Data applications.

4.2. Approximate fusion of models: aggregation of partial systems

As it was pointed out in the previous section, one straightforward way to design an scalable algorithm is trying to compute independent parallel tasks, i.e. to avoid data exchange between nodes. By carrying out separate computations within each Map task, complete models are obtained, all of which could work directly over the test data, as they maintain no a priori relationship among them.

To empower the recognition ability of these single models, their combination into an ensemble system has been suggested. However, we have also emphasized several some problems related to this design. In order to address or at least alleviate them, one possible solution is trying to aggregate these partial systems into a single model.

The hitch in this case is how to design the Reduce function in order to combine the individual models coming from each Map. In order to have an insight on the possible alternatives to overcome this problem, some significant proposals given in the specialized literature will be described next.

A first example is the clustering procedure by means of a standard k-means [19]. Each Map process is devoted to compute the nearest samples (from its input chunk of data) to each of the k centroids (initially set at random). The output key-value is the centroid "id" and the attributes of the nearest sample respectively. Then, each Reducer recalculates the new centroids coordinates by averaging the values of those samples given as "list of values" for each key (centroid). The whole procedure is then iterated in order to refine the centroids, thus improving their robustness. It must be observed the importance of the Reduce step for the global combination of the partial information computed within each Map.

k-Nearest Neighbor (kNN) classifier [27] has also some points in common with clustering, as it is based on distances among examples. In this case, the Map stage is devoted to obtain the k nearest training instances to each query input. To carry out an exact computation, the whole test set should be given as input to each Map, allowing to obtain as key-value the index of the test instance and a list of the k closest distances with their corresponding training class labels. Then, the Reducer only needs to find, among all M sets of k-nearest neighbors (with M the number of Maps) the one with lowest value to finally assign the output class.

Prototype Reduction [20] is also a significant example in this context. First, each Map process works with a different chunk of data by applying any available reduction procedure [21]. Then, selected prototypes

are fed to a single reducer that is devoted to eliminate redundant ones. This implies a clear fusion of data in order to obtain a final model, in this case merging similar examples.

Another interesting approaches come from the boosting perspective [22], which are AdaBoost.PL, LogitBoost.PL and MultiBoost.PL [23]. These include an ensemble of ternary classifiers [24], or an ensemble of ensembles. Although it can be wrongly associated with those methods introduced in the previous section, these have been included in this type of methodology regarding the functionality implemented within the Reduce step. Specifically, the whole boosting procedure, i.e. T iterations, is carried out within each Map, so M ensemble of T classifiers are obtained, being M the number of maps. Then, the T classifiers are arranged according to their score, i.e. the weighted error, and then emitted to the Reducers using their index as "key" and the model as value. Finally, each Reduce process takes all classifiers with the same index (key) and compute an average weight for this ensemble. At classification time, each "sub-ensemble" provides a vote for its class, whose value is exactly the aforementioned weighted average score.

Fuzzy Rule Based Classification Systems [25] are probably one of the clearest type of models in this category. The pioneer implementation was based on the Chi et al approach [26]. Each Map task comprises a complete fuzzy learning procedure to obtain an independent Rule Base (RB). To do so, all examples from the input data chunk are iterated deriving a single rule per example, while merging those with the same antecedent and consequent. Then, rule weights (RWs) are computed based on a fuzzy confidence value from the input examples, also allowing to determine the output class. We must acknowledge that at this stage every single RB might be used for classification purposes. However, the system can be further enhanced by combining all partial models for every Map, as stated in the beginning of this section. To do so, the Map writes as key-value pair the antecedent and consequent (including class and RW) respectively. Reducers then merges those rules with the same antecedent (key) by averaging the values of the RWs for the same class consequents, and then the class with the highest RW is finally maintained.

Throughout this section we have presented a short overview on those MapReduce methods that are based on fusion of partial models. As such, they allow to reinforce the abilities of those individual systems, leading to possibly more robust approaches. However, two issues must be taken into account:

1. The key-value representation to link the Map and Reduce processes. It must be very carefully selected as it has a strong influence in the design of the Map function. Although the procedure is usually implemented to be independent among Maps, a common point should be given in order to allow the final combination.
2. The aggregation function in the Reducer. It has a clear dependence on the previous item, so that these must be designed as a whole to ensure a robust workflow.

Both points will determine the scalability and exactness of the output system. We refer to the changes in behavior when increasing the number of Maps. Specifically, the efforts must be mostly focused on the

development of the partial models, regarding the influence of the lack of data and/or whether the knowledge acquired from different subsets of the problem space are able to be accurately merged.

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.

Bibliography

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

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

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

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

[17] S. Ramírez-Gallego, I. Lastra, D. Martínez-Rego, V. Bolón-Canedo, J. M. Benítez, 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.

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

[19] W. Zhao, H. Ma, Q. He, Parallel K-means clustering based on MapReduce, Vol. 5931 of Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2009.

[20] I. Triguero, D. Peralta, J. Bacardit, S. García, F. Herrera, Mrpr: A mapreduce solution for prototype reduction in big data classification., Neurocomputing 150 (2015) 331–345.

[21] I. Triguero, J. Derrac, S. García, F. Herrera, A taxonomy and experimental study on prototype generation for nearest neighbor classification., IEEE Transactions on Systems, Man, and Cybernetics, Part C 42 (1) (2012) 86–100.

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

[23] 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.

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

[25] A. Fernandez, C. Carmona, M. del Jesus, F. Herrera, A view on fuzzy systems for big data: Progress and opportunities, International Journal of Computational Intelligence Systems 9 (1) (2016) 69–80.

[26] S. del Río, V. López, J. M. Benítez, F. Herrera, A mapreduce approach to address big data classification problems based on the fusion of linguistic fuzzy rules., International Journal of Computational Intelligence Systems 8 (3) (2015) 422–437.

[27] J. Maillo, S. Ramírez-Gallego, I. Triguero, F. Herrera, knn-is: An iterative spark-based design of the k-nearest neighbors classifier for big data., Knowledge-Based Systems 117 (2017) 3–15.