16th World Congress of the International Fuzzy Systems Association (IFSA)
9th Conference of the European Society for Fuzzy Logic and Technology (EUSFLAT)

Linguistic Aggregation Functions using the MapReduce Paradigm

Patricia Conde-Clemente, Gracian Trivino, Jose M. Alonso

European Centre for Soft Computing
Gonzalo Gutiérrez Quirós s/n, 33600 Mieres, Asturias, Spain
Email: {patricia.conde,gracian.trivino,jose.alonso}@softcomputing.es

Abstract

We explore the possible benefit that provides a linguistic approach to Big Data. The proposal illustrates how implement Linguistic Aggregation Functions using the MapReduce paradigm. The best known paradigm applied to Big Data. The proposal allows several benefits to Big Data e.g., it allows to interpret data in a more intuitive way, reduce data size into different levels of granularity, and manage the imprecision and incompleteness of data. We show the usefulness of the proposed approach with an illustrative example.

Keywords: Big Data, Computing with Perceptions, Fuzzy Logic.

1. Introduction

Humanity creates quintillion of data (10^{18}) per year [1]. This vast quantity of data, widely known as "Big Data", is generated with great velocity and variety of sources and formats. Currently, the challenge is to process these data to extract new knowledge that must be represented in a meaningful manner [2, 3].

There are two main ways to represent the knowledge extracted from data. The first and most common choice is the so called Advanced Data Visualization. It combines data analysis methods with interactive visualization of tables, charts or graphs [4]. A complementary option is the Automatic Text Generation [5]. It involves computer programs that automatically produce texts from input data.

Our work in Linguistic Descriptions of Complex Phenomena (LDCP) collects and interprets data coming from complex phenomena, yielding reports in natural language which are easy to understand even by non-expert users. This technology has been used to automatically generate linguistic descriptions about, e.g., the driving behavior in simulation environments [6], the human gait quality [7] and the beauty of double stars [8].

The advent of Big Data opens new challenges in the research line of LDCP. When we deal with Big Data some issues are of main concern [3, 9]:

- During the analysis process, developers must use scalable algorithms ready to interpret the data efficiently and manage the usual imprecision and incompleteness.
- During the knowledge extraction and representation process, developers must offer data visualization techniques aimed at assisting the users with the intuitive and effective interpretation of the extracted knowledge.

Although as for as we know there is not any scientific publication regarding Automatic Text Generation applied to Big Data, we have found four companies which offer this service. They are Yseop[1] [10], Automated Insights[2] [11], Arria Data2Text[3] [12] and Narrative Science[4] [13].

We have thoroughly evaluated the solutions provided by these companies, paying attention to the information they provide in their websites and related patents. Notice that patents keep hidden the techniques they actually use to process Big Data. Anyway, in the available information there are not any details about how they deal with scalability, efficient processing and management of incomplete and inaccurate data.

In this paper, we explore the possibilities of applying LDCP to Big Data problems. We describe how this new technology allows to design scalable algorithms, to build an efficient computational model for complex applications and to manage the usual imprecision and incompleteness.

Namely, the main contribution in this paper consists of implementing Linguistic Aggregation Functions, a basic element in LDCP, using the MapReduce paradigm [14]. Currently, it is the best known paradigm applied to resolve Big Data problems [15].

We present an illustrative example that shows the usefulness of the proposed approach. The goal is the generation of sentences similar to those produced by the US census bureau[5].

The rest of this paper is structured as follows. Section 2 describes some preliminary concepts. Section 3 describes how to implement Linguistic Aggregation Functions using the MapReduce paradigm and includes an illustrative example about the pro-

[1] http://yseop.com
[2] http://automatedinsights.com/
[3] http://www.arria.com/
[4] http://www.narrativescience.com/
[5] http://www.census.gov/

posed approach. Finally, Section 4 presents conclusions and sketches future work.

2. Preliminary Concepts

2.1. Linguistic Descriptions of Complex Phenomena

LDCP is based on the Computational Theory of Perceptions [16, 17]. This theory provides a framework to develop computational systems with the capacity of computing with the meaning of natural language expressions, i.e., with the capacity of computing with imprecise descriptions of the world in a similar way how humans do it. In the Computational Theory of Perceptions, a granule is a clump of elements which are drawn together by indistinguishability, similarity, proximity or functionality. The boundary of a granule is fuzzy. Fuzziness of granules allows us to model the way in which human concepts are formed, organized and manipulated in an environment of imprecision, uncertainty, and partial truth [18].

Granular Linguistic Model of Phenomena (GLMP) is the core of LDCP. It implements the Granular Computing paradigm [19], which operates with information granules to build efficient and human-centric views of the modeled world. Granular Computing deals with abstraction, summarization and condensation of information.

GLMP is a network of granules that represent the monitored phenomenon with several levels of granularity. In consequence, the network is scalable in vertical and horizontal ways. Vertically, we can add more levels of abstraction that are to be executed in sequential way. Horizontally, we can divide problems into a set of more manageable and smaller sub-tasks. They can be executed in parallel way.

Figure 1 shows an example of GLMP. It allows us to compare the number of inhabitants, men and women in US, with two levels of granularity: state and country. GLMP has two main elements: Perception Mappings (PM) and Computational Perceptions (CP). Each PM receives a set of input CPs and they are aggregated in a single CP that is transmitted upwards. Notice that the output of one PM can act as input of others. A CP represents the whole linguistic domain of the information unit of the modeled phenomenon. In short, each output CP is explained by the related PM, using a set of input CPs.

A CP is a couple (A,W) where:

A $=\{a_{ij}\}$ is a matrix, with one or more dimensions, of linguistic expressions (words or sentences in natural language) that represents the whole linguistic domain in CP.

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}$$

W $= \{w_{ij}\}$ is a matrix of validity degrees $w_{ij} \in [0, 1]$ assigned to each a_{ij} in the specific context. The sum of all validity degrees must be $\sum w_{ij} = 1$.

A PM is a tuple (U,y,g,T) where:

U $= (u_1, u_2, .., u_n)$ is a vector of n input CPs u_i = (A_{ui},W_{ui}). In the special case of the first level of perception mappings, u_i are the inputs to the network. They are values z $\in$ R being provided either by sensors or obtained from a database.

y = (A_y,W_y) is the output CP.

g is an aggregation function employed to calculate $W_y = g(W_{u1},W_{u2},...,W_{un})$ from the input CPs. In Fuzzy Logic, many different types of aggregation functions have been developed. For example, g might be implemented using a set of fuzzy rules. In the case PMs in the first level, g is built using a set of membership functions.

T is a text generation algorithm which allows generating the sentences in A_y. In simple cases, T is a linguistic template, e.g., "California has {few | some | many} inhabitants", but it can be customized according to user preferences, mood, etc.

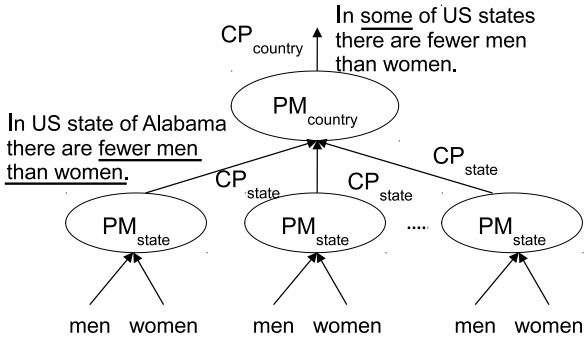

Figure 1: GLMP about the US inhabitants.

In Figure 1 there are several PMs that represent different levels of granularity. The first level of granularity is related to states (e.g., Alabama), while the second level represents the country (e.g., US).

At state level, PM_{state} evaluates the difference between the number of men and women by state. It is defined by the tuple (U,y,g,T):

U is a vector with two elements: the number of men and women in the state $U = (men, women)$.

y is the output CP_{state}. It describes in a linguistic form the difference between the number of men and women by state. It is expressed by A_{state} = {fewer men than women, similar number of men and women, more men than women} and their corresponding degrees of validity.

g is the aggregation function in charge of computing the difference (D) between the number of men and women by state, applying the following formula:

$$D = \frac{men - women}{men + women} x100(\%)$$

D is the input of a set of membership functions composed of the following triangles and trapezoids (see Figure 2): {fewer men than women $[-\infty, -\infty, -20\%, 0]$, similar number of men and women $[-20\%, 0, +20\%]$, more men than women $[0, +20\%, \infty, \infty]$.

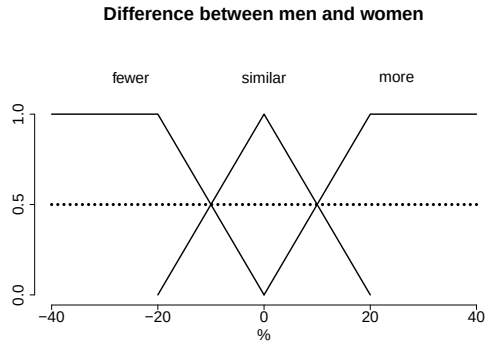

Figure 2: Strong fuzzy partitions for PM_{state}.

T is a text generation template: "In *name-country* state of *name-state* there is/are {fewer men than women | similar number of men and women | more men than women}". Where *name-country* and *name-state* are parameters with the name of the country and state respectively.

The definition of PM_{state} is the same for all the 50 US states. Thus, it is executed 50 times, one for each state and gets 50 CP_{state} as output. These CP outputs will be taken as inputs in the second level of granularity.

At country level, $PM_{country}$ evaluates the difference between the number of men and women in the whole country. It is defined by the following tuple (U,y,g,T):

U is a vector with 50 CP_{state} (one for each state).

y is the output $CP_{country}$. It has the following set of possible sentences $a_{country_{ij}}$, where $i \in \{1, 2, 3\}$ are related to {few, some, most} and $j \in \{1, 2, 3\}$ are related to {fewer men than women, similar number of men and women, more men than women}, e.g.,

$a_{country_{11}}$ → *"In few of US states there are fewer men than women."*

g is the aggregation function. It is based on the $\alpha-$cuts method proposed in [20]. In [6] and [21] the interested reader can find further examples of the use of this method to generate quantified sentences.

T is a text generation template: "In {few | some | most} of *name-country* states there is/are {fewer men than women | similar number of men and women | more men than women}". Where *name-country* is a parameter with the name of the country.

In the LDCP architecture, after interpreting data with the GLMP, we execute the Report Template that allows us to select the most suitable sentences to the final user (see, e.g., [6], [7] and [8]).

2.2. Big Data

Big Data is usually defined by three dimensions or 3Vs: Volume, Velocity and Variety [22]. Volume dimension involves processing large amounts of data (more than terabytes). Velocity dimension indicates that data are often generated at high speed and they need to be processed in real or near real time, in batch or as streams. Variety dimension indicates that data come from a great variety of sources and thus can be structured, unstructured or semi-structured. In [23], Demchenko introduced two more dimensions, namely, Value and Veracity. Value dimension means the added-value (or new knowledge) that the collected data can bring to the intended process or activity. Veracity dimension includes two aspects: data consistency and data trustworthiness.

Currently, the standard framework *de facto* in both industry and academia to solve Big Data problems is Apache Hadoop[6]. It is based on an implementation of the MapReduce programing paradigm [15] which is enhanced for processing large data sets [14]. This paradigm facilitates the creation of parallel and distributed systems written in a functional style. As a result, they are ready to be automatically parallelized and executed on a large cluster of machines [14].

The paradigm is mainly based on two user-defined functions *map* and *reduce*. These functions are executed in parallel on multiple machines, with a total of m *map* and r *reduce* tasks. The number of *map* tasks (m) corresponds with the number of partitions of the input data. The number of *reduce* tasks (r) corresponds with the number of intermediate keys.

As it can be appreciated in Figure 3, the execution of the MapReduce paradigm involves four bottom-up steps. We will explain them through an

[6]http://hadoop.apache.org

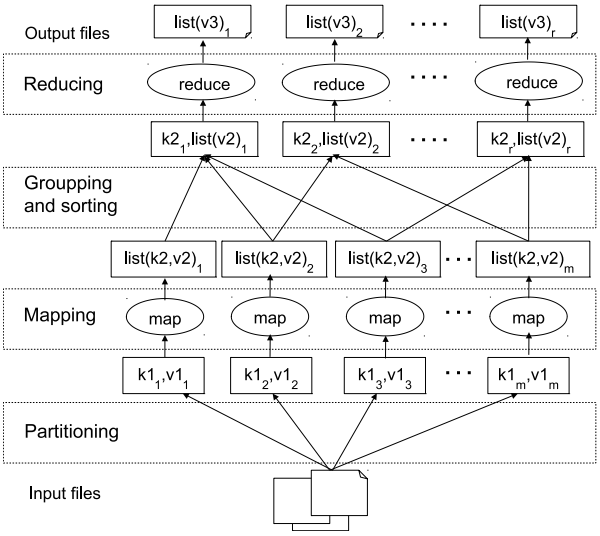

Figure 3: MapReduce paradigm. Circles represent processes and rectangles represent data structures.

example (similar to the previous one) regarding the number of inhabitants, men and women, from every US state.

1. **Partitioning:** The input is a set of files that contain the information about the US census. Each line or record contains the data of one person. These data include the sex and state in which the person lives. The input files are split into m pieces with smaller size. The programmer indicates the number of pieces m along with the names that identify them.
2. **Mapping:** Each *map* function has as input a (key, value) pair ($k1_i$, $v1_i$) where $i \in \{1, 2, \ldots, m\}$. The key $k1_i$ corresponds with the partition name and the value $v1_i$ with all records contained in the piece. During the execution, the *map* function processes each record contained in $v1_i$. The *map* function associates the key k2 with the state (e.g., k2= Alabama) and the value v2 with the sex of the person (e.g., v2 = man). The output of this function is a list of intermediate (key, value) pairs list$(k2,v2)_i$, containing the sex of all people included in the partition (e.g., list((Alabama, man), (Colorado, woman), (Alabama, man),..., (California, woman))$_1$).

$$map(k1_i, v1_i) \rightarrow list(k2, v2)_i \quad (1)$$

3. **Grouping and sorting:** When the m *map* tasks have ended, we have a total of m lists key/value list$(k2,v2)_i$. The aim of this step is sorting and grouping all values v2 with the same key k2, i.e., the aim is the creation of pairs ($k2_j$,list$(v2)_j$) with $j \in \{1, 2, \ldots, r\}$, where key is unique (e.g., $k2_1$ = Alabama) and the list contains all people of each State (e.g., list$_1$=[man, woman, man, man,..., woman]).
4. **Reducing:** Then, r *reduce* tasks are executed, one for each key $k2_j$. The *reduce* function receives as inputs a key along with its corresponding list of values. The function aggregates the input list into a smaller number of values. In our example, the function counts the number of inhabitants, men and women, by state. To do this, the function examines each item in the list and counts the number of men and women. Then, it saves the results v3 in the output list.

$$reduce(k2_j, list(v2)_j) \rightarrow list(v3)_j \quad (2)$$

The output list contains three elements: the name of the evaluated state, the number of men, and the number of woman (e.g., list(v3)=[Alabama, 2320188, 2459548]). Notice that each output of the *reduce* function is saved in an output file. Typically, all the r outputs files (the global MapReduce output) can be taken as input by another MapReduce process, in another distributed application, or sent to the visualization process.

3. How to implement Linguistic Aggregation Functions using the MapReduce paradigm

In this section, we explain how to implement Linguistic Aggregation Functions using the MapReduce paradigm. The focus is set in exploring the benefits that it can bring to solve Big Data problems.

Among other features GLMP is a scalable algorithm, that allows us to model the information in a granular way and manage the uncertainty of the information.

On the one hand, PM is a linguistic aggregator which takes as input a list of CPs and generates as output a single CP. On the other hand, *reduce* function aggregates inputs (previously prepared by the *map* function) in a reduced output. In order to implement linguistic aggregators using the MapReduce paradigm: 1) the *map* function must first preprocess the given PM inputs (a set of CPs or z values $\in$ R) and 2) the *reduce* function must implement a PM which takes the given list of inputs and generates a single CP output. In Figure 4, we show a representation of the proposed implementation. For simplicity, the picture focuses on the reducing step, it illustrates only a partial view of the entire paradigm. The *reduce* function implements PM_{kcp} which takes as input the key *kcp* and its corresponding list of values $list(cp)$, and it generates a single CP output. Then, *reduce* function saves all the results v3 in the output list.

In the remaining of this section, we show the usefulness of the proposed approach with two illustrative examples.

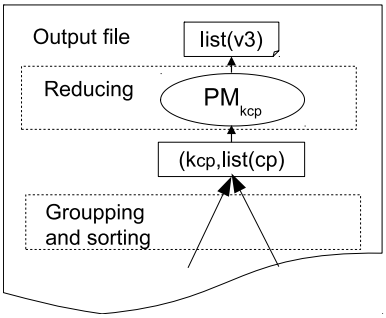

Figure 4: Linguistic Aggregation Functions using the MapReduce paradigm.

3.1. Illustrative Example about US Inhabitants

The aim of this first example is the generation of sentences that compare the number of inhabitants, men and women, in US. With this purpose, we merge the two examples introduced in Section 2. In Figure 5 we show a partial representation of the MapReduce paradigm. It implements the GLMP about the US inhabitants (see Figure 1). The picture depicts one MapReduce task which it is in charge of aggregating the inhabitants at "State level". Then, the top order perception aggregates the inhabitants at "Country level". We describe the detail of the two steps, as follows:

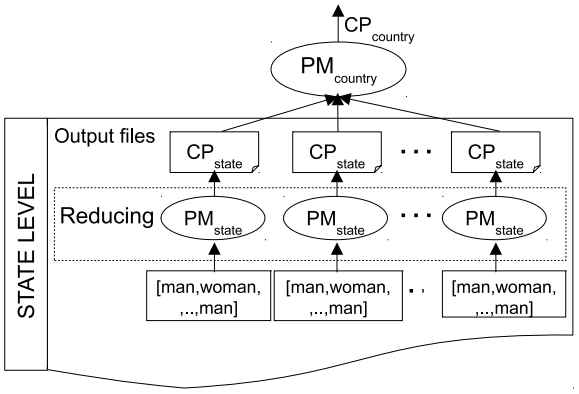

Figure 5: Partial representation of the MapReduce paradigm which implements the GLMP about the US inhabitants.

State level of US inhabitants. 1) Partitioning, 2) Mapping and 3) Grouping steps are the same as defined in section 2.2. The first change is introduced in the 4) Reducing step. We execute one *reduce* function for each state (k2) in order to aggregate the list of men and women (list(v2)). Once the function computes the number of inhabitants (men and women), it calls the PM_{state}. As a result, the output list contains two elements, the name of the evaluated state and CP_{state}.

Country level of US inhabitants. The input, to obtain the top order perception, is made up of 50 files (one per state) with only one line. Data size have been reduced in the MapReduce step, for this reason, this step does not need be implemented in parallel way.

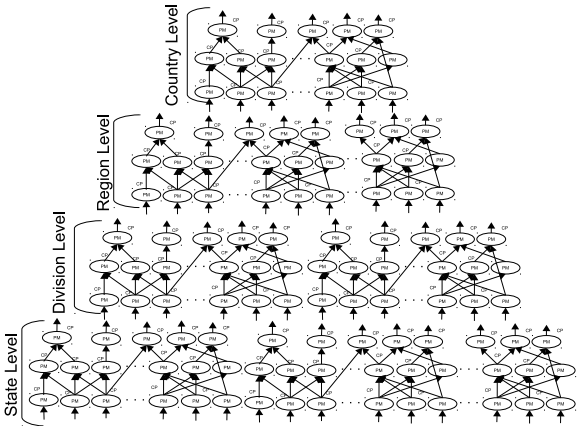

Figure 6: Extending the GLMP about the US inhabitants.

3.2. Extending the GLMP about the US inhabitants

The Census Bureau of US takes into consideration the following population items: sex, age, race, Hispanic or Latino origin, household relationship, household type, household size, family type, family size, and group quarters. Housing items include occupancy status, vacancy status, and tenure (whether a housing unit is owner-occupied or renter-occupied). This population items can be grouped in geography areas: country, regions, divisions, states, counties, county subdivisions, places, metropolitan statistical areas, ZIP Code Tabulation Areas, congressional districts, American Indian and Alaska Native Areas, and Hawaiian Home Lands.

With the aim of generate a model that consider all populations items in all geographical areas, in this section we extend the GLMP about the US inhabitants (see Figure 6). The extended GLMP has hundreds of sub-tasks that interpret the populations items in a horizontal way. The geographical areas are the different levels of granularity that are interpreted in sequential way. Figure 7 shows a partial representation of the MapReduce paradigm that implements the extended GLMP. In this example two MapReduce processes must be executed sequentially. We describe the detail of the two steps, as follows:

State level of US inhabitants This MapReduce step is defined in a similar way than the previous example. Changes on the new model are the increment in the number of PMs (subtasks) for state. The extended model have several PM for state. This implies that each *map* function must prepare the input of each PM

and each *reduce* function must executes a network of PMs.

Division level of US inhabitants The input of the Division level is the output of the State level. Each *map* function must processes each record in and associates the with key, in this case the key is the division. Like the previous MapReduce process, each *map* must prepare data for a large set of PMs. This set of PMs are executed in the Reducing step.

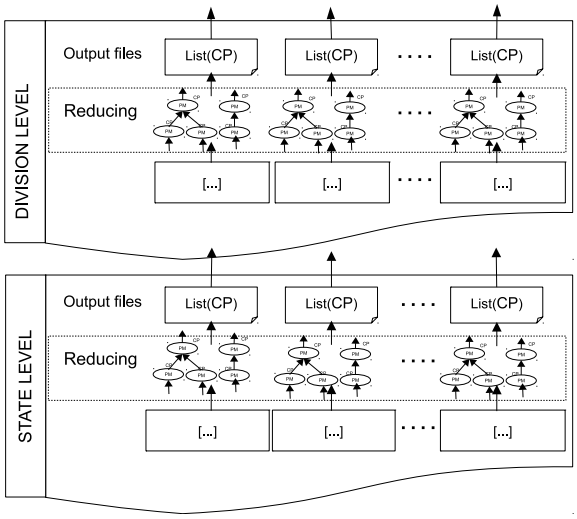

Figure 7: Partial representation of MapReduce paradigm of the extended GLMP about the US inhabitants.

4. Conclusions

We have shown how implement Linguistic Aggregation Functions using the MapReduce paradigm. The *map* function is used to prepare the PM inputs while the *reduce* function deals with implementing a PM which aggregates the PM inputs and generates a single CP output. Thanks to the GLMP features, we are able to generate parallel and distributed systems that offer several benefits in the resolution of Big Data problems: It allows us to 1) interpret data in a more intuitive way, 2) reduce data size into different levels of granularity, and 3) manage the imprecision and incompleteness of data.

Although, there is still much to do in this research line, as applying the proposal in a real case or explore other LDCP features.

However, this paper present a first approach towards to solve Big Data problems using LDCP. This is a promising and surprisingly simple technique to generate linguistic descriptions from Big Data.

Acknowledgments

This work was supported in part by the Spanish Ministry of Economy and Competitiveness under Grant TIN2014-56633-C3-3-R and the Spanish Ministry of Science and Innovation under Grant FPI-MICINN BES-2012-057427.

References

[1] IBM Study: Digital Era Transforming CMO's Agenda, Revealing Gap In Readiness, October 2011. online at http://www-03.ibm.com/press-/us/en/pressrelease/35633.wss.

[2] C. L. Philip Chen and C.-Y. Zhang. Data-intensive applications, challenges, techniques and technologies: A survey on big data. *Information Sciences*, 275:314–347, 2014.

[3] R. T. Kouzes, G. A. Anderson, S. T. Elbert, I. Gorton, and D. K. Gracio. The changing paradigm of data-intensive computing. *Computer*, 42(1):26–34, 2009.

[4] P. Russom. Big data analytics. Technical report, TDWI Best Practices Report, Q4 2011.

[5] E. Reiter and R. Dale. Building applied natural language generation systems. *Natural Language Engineering*, 3(1):57–87, 1997.

[6] L. Eciolaza, M. Pereira-Fariña, and G. Trivino. Automatic linguistic reporting in driving simulation environments. *Applied Soft Computing*, 13(9):3956 – 3967, 2013.

[7] A. Alvarez-Alvarez and G. Trivino. Linguistic description of the human gait quality. *Engineering Applications of Artificial Intelligence*, 26(1):13 – 23, 2013.

[8] L. Arguelles and G. Trivino. I-struve: Automatic linguistic descriptions of visual double stars. *Engineering Applications of Artificial Intelligence*, 26(9):2083 – 2092, 2013.

[9] A. Labrinidis and H. V. Jagadish. Challenges and opportunities with big data. *Proceedings of the Very Large Data Bases (VLDB) Endowment*, 5(12):2032–2033, August 2012.

[10] A. Kaeser. Patent: Methods and apparatus for processing grammatical tags in a template to generate text. *Yseop SA*, United States(US8150676 B1), November 2008.

[11] R. C. Allen. Patent: Systems for dynamically generating and presenting narrative content. *Automated Insights, Inc.*, United States(US8515737 B2), August 2013.

[12] E. Reiter. Patent: Method and apparatus for situational analysis text generation. *Arria Data2Text Limited*, United States(US8762134 B2), June 2014.

[13] L. A. Birnbaum, K. J. Hammond, N. D. Allen, and J. R. Templon. Patent: System and method for using data to automatically generate a narrative story. *Narrative Science Inc.*, United States(US8688434 B1), April 2014.

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

[15] J. Dittrich and J.-A. Quiané-Ruiz. Efficient big data processing in hadoop mapreduce. *Proceedings of the Very Large Data Bases (VLDB) Endowment*, 5(12):2014–2015, August 2012.

[16] L. A. Zadeh. From computing with numbers to computing with words. From manipulation of measurements to manipulation of perceptions. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 46(1):105–119, 1999.

[17] L. A. Zadeh. Toward a perception-based theory of probabilistic reasoning with imprecise probabilities. *Journal of statistical planning and inference*, 105(1):233–264, 2002.

[18] L. A. Zadeh. Fuzzy sets and information granularity. In George J. Klir and Bo Yuan, editors, *Fuzzy Sets, Fuzzy Logic, and Fuzzy Systems*, pages 433–448. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 1996.

[19] A. Bargiela and W. Pedrycz. *Granular Computing: An Introduction*. Springer Science & Business Media, 2003.

[20] M. Delgado, D. Sánchez, and M.A. Vila. Fuzzy cardinality based evaluation of quantified sentences. *International Journal of Approximate Reasoning*, 23(1):23 – 66, 2000.

[21] D. Sanchez-Valdes, A. Alvarez-Alvarez, and G. Trivino. Linguistic description about circular structures of the mars' surface. *Applied Soft Computing*, 13(12):4738 – 4749, 2013.

[22] D. Laney. 3-D Data Management: Controlling Data Volume, Velocity and Variety. *Application Delivery Strategies*, 949:1–3, 2001.

[23] Y. Demchenko, P. Grosso, C. de Laat, and P. Membrey. Addressing big data issues in Scientific Data Infrastructure. In *International Conference on Collaboration Technologies and Systems (CTS)*, pages 48–55. IEEE, 2013.