Journal of Visual Languages and Computing 31 (2015) 130–138

Contents lists available at ScienceDirect

Journal of Visual Languages and Computing

journal homepage: www.elsevier.com/locate/jvlc

A hadoop based platform for natural language processing of web pages and documents

Paolo Nesi [*,1], Gianni Pantaleo [1], Gianmarco Sanesi [1]

Distributed Systems and Internet Technology Lab, DISIT Lab, Department of Information Engineering (DINFO), University of Florence, Firenze, Italy

A R T I C L E I N F O

Available online 30 October 2015

Keywords:
Natural language processing
Hadoop
Part-of-speech tagging
Text parsing
Web crawling
Big Data Mining
Parallel computing
Distributed systems

A B S T R A C T

The rapid and extensive pervasion of information through the web has enhanced the diffusion of a huge amount of unstructured natural language textual resources. A great interest has arisen in the last decade for discovering, accessing and sharing such a vast source of knowledge. For this reason, processing very large data volumes in a reasonable time frame is becoming a major challenge and a crucial requirement for many commercial and research fields. Distributed systems, computer clusters and parallel computing paradigms have been increasingly applied in the recent years, since they introduced significant improvements for computing performance in data-intensive contexts, such as Big Data mining and analysis. Natural Language Processing, and particularly the tasks of text annotation and key feature extraction, is an application area with high computational requirements; therefore, these tasks can significantly benefit of parallel architectures. This paper presents a distributed framework for crawling web documents and running Natural Language Processing tasks in a parallel fashion. The system is based on the Apache Hadoop ecosystem and its parallel programming paradigm, called MapReduce. In the specific, we implemented a MapReduce adaptation of a GATE application and framework (a widely used open source tool for text engineering and NLP). A validation is also offered in using the solution for extracting keywords and keyphrase from web documents in a multi-node Hadoop cluster. Evaluation of performance scalability has been conducted against a real corpus of web pages and documents.

1. Introduction

Present day information societies deals with very large repositories of data (in the form of public and private data, including both human and automatically generated content). A statistic reported in the 2014 by the International Data Corporation (IDC), sponsored by MC Digital Universe (EMC) asserts that the digital universe is doubling every two years, and will reach a size of 40 zettabytes (i.e., 40 trillion gigabytes) by 2020, from 4.4 zettabytes in 2013 [1]. Due to the increasing expansion of the web, the Internet of Things and related information technologies, the ICT world and consequently digital users' experience have been largely influenced by data-driven methods for extracting and managing knowledge. Access to large amounts of data has opened new perspectives and challenges. Actually, the term Big Data does not only refer to a matter of size, but implies also other aspects (summarized in the Big Data 4 V paradigm: Volume, Variety, Velocity and Veracity) such as data reliability and consistency, as well as data encoding which comes in many heterogeneous formats, often leading to interoperability lacks. For all these reasons, current IR tools and applications must be able to scale up to

* Corresponding author.
E-mail addresses: paolo.nesi@unifi.it (P. Nesi), gianni.pantaleo@unifi.it (G. Pantaleo).
[1] http://www.disit.dinfo.unifi.it

http://dx.doi.org/10.1016/j.jvlc.2015.10.017
1045-926X/© 2015 Elsevier Ltd. All rights reserved.

datasets of interest, in most cases up to web scale. We are currently facing growing needs and requirements for efficiently processing such large volumes of data, as the amount of information that can be stored and analyzed is rapidly increasing. Furthermore, about 85% of digital information available over the Internet is in unstructured form [2]. This represents a challenging application field for automatic Information Retrieval (IR) solutions. Actually, annotation and extraction of high level features has become an extremely time consuming and inefficient operation to be done manually. Anyway, the ability to extract information from text is nowadays demanded from the market to improve effectiveness and competitivity.

A feasible approach to handle Big Data processing in a more efficient way is represented by the “Divide and Conquer” concept [3]. The basic idea is to partition a large problem into smaller sub-problems; to the extent that these sub-problems are independent, they can be parallely processed by different threads. This aspect, concurrently with the evolution of multi-processor architectures and network speeds, is leading to the application of distributed architectures and parallel-computing paradigms to Big Data mining and processing. Actually, single CPU-based or even multi core CPU-based algorithms have revealed not to be fast enough for data processing in application areas such as Information Technologies [4]. On the other hand, typical drawbacks of parallel programming are the use of very low level languages as well as the necessity for the programmer to handle communication and synchronization issues [5]. Parallel solutions have been applied also to search engines and indexing systems; for instance, the MOSE (My Own Search Engine) [6] platform was specifically designed to run on parallel architectures, exploiting document partitioning as data-parallel technique. MOSE uses both task-parallel and data-parallel approaches to increase efficiency for data storage and computational resource usage. The Elastic Search[2] is a more recent open source distributed search engine, designed to be scalable, near real-time capable and providing full-text search capabilities [7]. The development of parallel computing models and commodity LAN-connected cluster of computers represents a cost-effective solution to increase time performances for Big Data processing. Among the several proposed solutions, the Apache Hadoop[3] ecosystem has recently gathered a quite diffuse interest. It has been designed to support distributed applications, large-scale data processing and storage, providing high scalability. Data access and management relies on the Hadoop Distributed File System (HDFS), modeled upon the Google File System – GFS [8]. The MapReduce programming paradigm is the core of the Hadoop HDFS file system. It provides an easier and transparent way for programmers to write applications to be executed in parallel on commodity hardware clusters.

NLP technologies allow to automatically extract machine readable information and knowledge from unstructured natural language data, which is at the basis of many application areas, e.g.: comprehension and supervised classification of text documents [9], content extraction [10], design of recommendation tools and Decision Support Systems, query expansion [11], Question–Answer frameworks and Sentiment Analysis. This last category is capturing a growing interest recently, since it represents one of the most widely used technological approach for understanding users' behavior in social media monitoring, opinion mining and target-marketing. NLP is therefore a data-intensive application area with large computational needs and growing complexity, which can benefit of distributed frameworks and parallel programming paradigms, assuming that fault tolerance to machine failures is provided. Actually, since NLP algorithms can be very time consuming to produce results, it is of fundamental importance to recover from failures in order not to lose intermediate computational results. NLP routines are generally executed in a pipeline where different plugins and tools are responsible for a specific task. One of the most intensive tasks is text annotation, which is defined as the process of adding structured linguistic information (related to grammatical, morphological and syntactical features of annotated words and phrases) to natural language data [12]. Text annotation is at the basis of higher level tasks such as extraction of keywords and keyphrases, which is defined as the identification of a set of single words or small phrases that can describe the general topic of a document [13]. Keywords annotation is widely used for content extraction and summarization, in order to produce machine-readable corpora, as well as to build content-based multi-faceted search queries. Currently, a large portion of web documents still does not have any keywords or keyphrases assigned. However, it is necessary to design and implement efficient and scalable automated solutions, since manual annotation results to be a highly time-consuming and inefficient process. Current integrated NLP architectures are often prone to problems related with information congestion and losses [14]. Several studies in literature show how existing NLP tools and frameworks are not well suited to process very large corpora, since their primary design focus was not oriented to scalability [15].

This paper proposes an extended and improved version of the work presented in [16]. The proposed system allows the execution of general purpose NLP applications, through the use of the open source GATE[4] APIs [17] executed via MapReduce on a multi-node Hadoop cluster. The paper is organized as follows: Section 2 provides an overview of related work and state of the art for the main research areas involved; in Section 3, the architecture of the proposed system is described; in Section 4, a validation of the system is reported, performed on real corpora retrieved online. Finally, Section 5 is left for conclusions and future work considerations.

2. Related work

The task of automatic keyword extraction deals with automated annotation of relevant, topical words and

[2] https://www.elastic.co/products/elasticsearch
[3] http://hadoop.apache.org/
[4] https://gate.ac.uk/

phrases from the text body of a document [18]. This activity, which is deeply connected with NLP methods, has been extensively studied in recent literature. Existing solutions are typically divided into four categories, depending on the processing approach: statistic, linguistic, machine learning and mixed approaches [19]. Statistical methods are commonly based on estimation of simple features (e.g., terms position, frequency, POS-tag, co-occurrence, IR measures of relevance, such as TF-IDF [20]) and more complex models (e.g., Bayesian Networks, K-Nearest Neighbor, Expectation-Maximization [21]). NLP techniques are at the foundation of linguistic-based solutions, which generally provide more accurate results with respect to statistical methods, even though they are computationally more expensive [22]. Linguistic methods include lexical analysis, syntactic analysis, exploiting also semantic features [23]. Machine learning methods treat keyword extraction task as a supervised learning problem; they employ different techniques, such as naive Bayes algorithms [24], genetic algorithms, least square support vector machines (LS-SVM) [25]. Mixed approaches rely on a combination of both the previously mentioned solutions, possibly with the addition of heuristic knowledge like annotated lists, gazetteers, blacklists [26]. POS-tag based filtering [27]. Training datasets and corpora can be jointly used with external resources, for instance lexical knowledge repositories (Wikipedia [28], DBpedia etc.). Graph-based approaches typically extract a graph from input documents and use a graph-based ranking function to determine the relevance of the nodes as key terms [29]. They have been proving to effectively model structural information and relationships in natural language data. Neural networks have been used to design keywords [21] and keyphrases extraction systems [30]. Topic-based clustering is used in content extraction and text summarization methods; this usually involves grouping the candidate keyphrases into topics or domains [31,32].

Parallel computing applications for NLP tasks have been studied since the 90 s: Chung and Moldovan [33] proposed a parallel memory-based parser called Parallel, implemented on a Semantic Network Array Processor (SNAP). Van Lohuizen [34] proposed a parallel parsing method relying on a work stealing multi-thread strategy in a shared memory multi-processor environment. Hamon et al. realized Ogmios [15], a platform for annotation of specialized domain documents within a distributed corpus; in this case the scaling capabilities are provided by distributing data (as an alternative approach to distributed processing). The system provide NLP functionalities such as word and sentence segmentation, named entity recognition, POS-tagging and syntactic parsing. Jindal et al. [5] developed a parallel NLP system based on Learning Based Java (LBJ) model [35] (which is a platform for developing NLP applications), and using Charm++ [36] as a parallel programming paradigm. The Koshik [37] multi-language NLP platform has been designed for large scale-processing and querying of unstructured natural language documents distributed upon a Hadoop-based cluster. It supports several types of algorithms, such as text tokenization, dependency parsers and co-reference solver. The advantage of using the Hadoop distributed architecture and its MapReduce programming model is the capability to efficiently and easily scale by adding inexpensive commodity hardware to the cluster.

Commercial tools have also been proposed: InfoTech Radar[5] is a software solution implementing NLP and Sentiment Analysis on the Hortonworks Sandbox Hadoop distribution. Beemoth[6] is an open source platform for large scale document processing based on Apache Hadoop, employing third party NLP tools (including GATE, Tika and UIMA). GATECloud [38] is an adaptation of the GATE software suite to a cloud computing environment using the PaaS paradigm.

Other parallel computing environments have been proposed: the Spark framework [39] has been originally developed at Berkeley UC to support applications implementing acyclic data flow models (such as iterative algorithms and applications which require low-latency data sharing processes). Spark introduces programming transformations on Resilient Distributed Datasets (RDD) [40], which are read-only collections of objects distributed over a cluster of machines. Fault tolerance is provided by rebuilding lost data relying on lineage information (without requiring data replication). RDD allows to store data on memory, as well as to define the persistence strategy. Gopalani and Arora [41] have compared Spark and Hadoop performances on K-means clustering algorithms, showing that Spark outperforms Hadoop on different cluster configurations.

3. System architecture

The proposed system aims at executing a generic NLP application on real corpora in a distributed architecture. The open source GATE Embedded Java APIs are used to perform the NLP tasks. In this paper, the specific design case of POS-annotated keywords and keyphrases extraction from unstructured text is presented. However, any other GATE application can be executed, with very minimal modification to the code, in order to properly set up the desired output features that may change according to different use cases. The open source Apache Hadoop framework has been chosen for the realization of an efficient and scalable solution, in order to improve performances with respect to a single node architecture, providing also data integrity and failures handling. The HDFS file system has been installed on a multi-node commodity cluster (more details will be provided in Section 4). Typically, a cluster is composed by a master Namenode, which assigns tasks to the different clients (Datanodes), monitoring their execution progress and also handling data failures. Datanodes are moreover responsible for data storage. An overview of the proposed system architecture is depicted in Fig. 1.

The MapReduce programming paradigm is used to parallelize the crawler work, the execution of NLP tasks

[5] http://www.itcinfotech.com/software-product-engineering/solutions/RADAR.aspx

[6] https://github.com/DigitalPebble/behemoth/wiki/tutorial

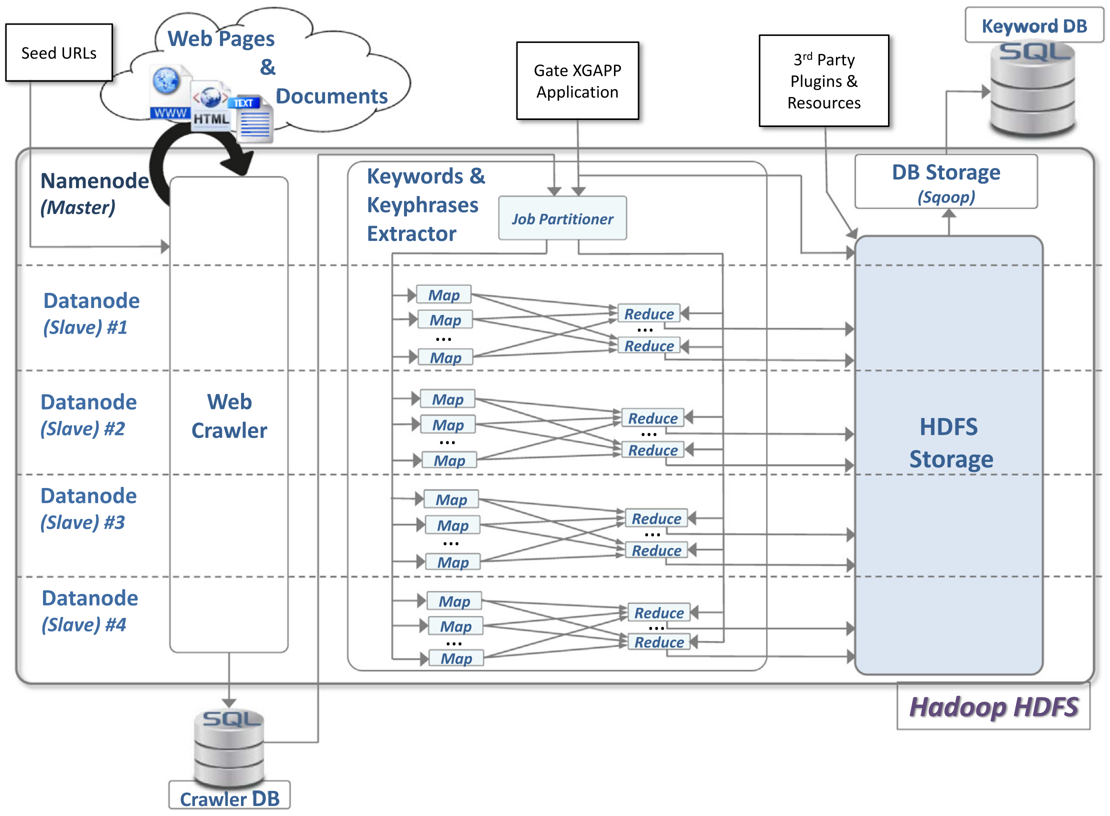

Fig. 1. Overview of the proposed system architecture. Map and Reduce functions in the Keyword & Keyphrase Extractor module can be customized by the user/porgrammer, depending on the desired NLP actions to be performed upon input text documents. In our case, the Map function produces a key/value pair composed by the web domain of each analyzed document and the corresponding parsred text. The Reduce function, in turn, fulfills the execution of the GATE application and the computation of TF-IDF relevance metric.

and the final output writing on external SQL database. The proposed architecture receives as input two absolute file paths: the first points to the location on the local file system where the GATE application and required plugins are stored; the other points at a text file containing the seed URLs for the crawling module. The different modules work asynchronously; actually, the crawling phase and the keywords/keyphrases extraction process can be scheduled and executed independently. Finally, a dedicated procedure stores extracted keywords and keyphrases in an external SQL database. The whole system architecture is implemented in Java. In the next subsections, a description will follow of the main modules constituting the proposed framework, which are listed as following:

- The *Web Crawler* module is responsible for crawling web pages and documents, starting from an input text file containing user defined seed URLs.
- The *Keywords & Keyphrase Extractor* module is in charge of two main operations. The first is the execution of the *GATE Application* implemented in a MapReduce environment and the subsequent is the storage of extracted keywords and keyphrases in the HDFS. The second operation is the relevance estimation of keywords and keyphrases within their corresponding domain corpora, by computing the *TF-IDF Relevance* function, which is a metric widely adopted in Information Retrieval.
- The *DB Storage* module finally stores extracted keywords, keyphrases and corresponding metadata into an external SQL database.

We are planning to soon release the source code of the proposed system, with open source licence, at: https://github.com/disit.

3.1. MapReduce

Let us first to recall the basic principles of the MapReduce programming paradigm. For a more comprehensive and detailed reference, see [42,43]. Inheriting the basic concepts of functional programming, the idea at the basis of MapReduce is to divide the process to be executed into smaller jobs, undertaken by the Mapper and Reducer computational primitives. Mappers and Reducers implement the map and reduce functions, respectively, which define the specific atomic tasks to be executed on input data.

Hadoop partitions input data, splitting into fixed-size parts, according to the HDFS block size (which is set to 64 MB by default). A MapReduce job assigns one map task for each split. The basic data structure under MapReduce is represented by key-value pairs. Actually, input and output of both the map and the reduce functions are formed by key-value pairs, which can be arbitrarily defined as

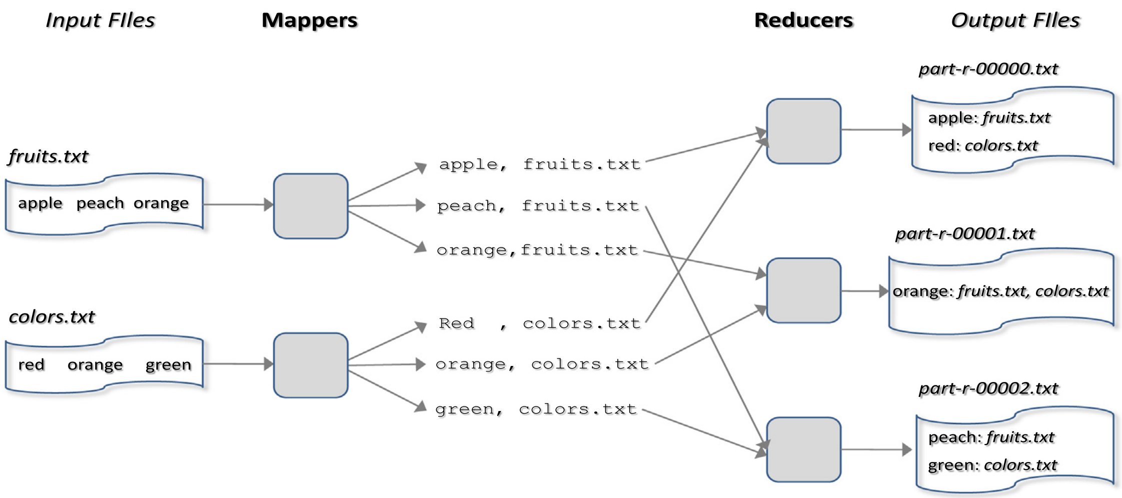

Fig. 2. Example of an inverted word index creation in MapReduce.

numerical values, strings, or even more complex objects, such as arrays, lists, etc. The *Map* function generates, as an intermediate output, key/value pairs representing logical records from the input data source (as an example, it could be a single text line in case the input is a text file, or a row in case it is a database). Subsequently, a *Shuffle and Sort* process merges all intermediate values associated with the same key. The sorted pairs form the input of the *Reduce* function/phase (which is called once per unique map output key) that performs the required processing on the splitted input data. The whole process can be summarized by the following notation [36]:

Map: (k1, v1) → list{k2, v2}
Shuffle & Sort: list{k2, v2} → (k2, list{v2})
Reduce: (k2, list{v2}) → list{k3, v3}

The MapReduce design introduces an abstraction of the complexity involved in the management of common parallel frameworks. This allows the programmer to focus only on properly defining the map and reduce functions, while lower level constrain (like parallelization, work distribution and communication issues among cluster nodes) are completely transparent and automatically managed by the Hadoop ecosystem. Anyway, it is assumed that the whole computational process to be executed can be considered as a problem which can be treated through a key/value approach.

The services in charge of managing and assigning tasks and data blocks to the different nodes are the *JobTracker* and the *TaskTracker*. The JobTracker daemon determines which files to process and which node to assign to each task; it also monitors all tasks while they are running. TaskTrackers run atomic tasks and send back reports about task progress to the JobTracker. If a task fails, the JobTracker reschedules its assignation on a different Tasktracker, up to a limit number of attempts defined in the configuration environment. The example of a simple inverted index creation, shown in Fig. 2, could be useful to clarify the outlined concepts.

A fundamental aspect behind MapReduce is the use of data locality optimization principle. In order to scale out and improve computational performances by minimizing the use of cluster bandwidth, the JobTracker assigns MapReduce jobs in a way such that TaskTrackers on each node would run the map reduce code only on data blocks locally present on that node, when it is possible.

3.2. *Web crawler*

The crawling engine of the proposed system is based on the open source Apache Nutch[7] tool. It has been initialized with a set of seed URLs of commercial companies, services and research institutes.

The Nutch workflow follows different phases: the first is the *Inject* phase, in which some seed URLs are injected for initial bootstrapping. The *Generate* phase define the set of URLs that are going to be fetched. Next, the *Fetch* phase deals with fetching the previously generated set of URLs into segments. The *Parse* phase is dedicated to the parsing of fetched segments content; the *Update* phase populates an external SQL Database with navigated URLs and corresponding parsed segment contents. All of these operations are carried on by single MapReduce Jobs, and the outcome of each phase is stored in the local HDFS. Finally, Apache Solr[8] is used to index all collected documents, providing also a search interface. The Solr index is ultimately installed onto the master Namenode only.

3.3. *Keywords/keyphrases extractor*

This module takes as input the parsed text content associated to each web URLs ingested during the crawling phase and retrieved by querying the Solr index. The present module is in charge of defining the MapReduce model, that is, of properly designing the map and reduce

[7] http://nutch.apache.org/
[8] http://lucene.apache.org/solr/

functions for the execution of the GATE application. In the proposed case, Map and Reduce functions are defined in a proper way for our goals: since we are interested in annotating keywords and keyphrases at single web-domain level, we designed a Map function that associates key/value record pairs where the *key* is the domain extracted from the web URL of the single page/document, and the *value* is the text previously parsed by the Nutch-based crawler. The Reduce function, in turn, fulfills the configuration and execution of a multi-corpora GATE application (each corpus containing text documents and pages belonging to a single web domain), as well the subsequent estimation of extracted keywords/keyphrases relevance at web domain level (as later described in Section 3.5).

3.4. GATE application

This block is in charge of executing the GATE application in the MapReduce environment, according to the input configuration parameters and the pipeline defined in an input file (usually a.xgapp or.gapp file). This is an XML-based file which is considered, by extension, as the effective GATE application file, containing file paths and references to all the Processing Resources and plugins used, in addition to the definition of the processing pipeline. As previously addressed, the proposed system allows the execution of a generic GATE application. In this specific case, the ANNIE (*A Nearly-New Information Extraction System*) plugin, more specifically the *Sentence Splitter* and the *Tokenizer* tools, have been used to segment into tokens the text content of crawled documents, while the *TreeTagger* plugin has been used for POS-tagging. Finally, the Java written *JAPE* (*Java Annotation Pattern Engine*) plugin syntax has been employed to define custom rules for filtering undesired, noisy parts of speech and user defined stop-words. These rules are contained in a dedicated jape file. Common nouns and adjectives are then annotated as potential keywords candidates. Next, candidate keyphrases are identified as contiguous phraseological combinations and patterns of candidate keywords. The described GATE pipeline is depicted in Fig. 2.

The following strategy has been followed to run the GATE application in MapReduce: the Namenode loads, at run time, a zip archive containing all the required GATE APIs, configuration and application files, libraries and plugins, in the HDFS Distributed Cache. By this way our application will copy in memory and extract the necessary files only once, and the allocated content will be accessible by all the Datanode. This has been considered as the most efficient solution, since there is no need to install required plugins and libraries on each single node. As mentioned earlier, an additional advantage of this approach is represented by fact that any generic GATE application can be potentially executed in the proposed architecture (taking benefits of all the NLP features and capabilities offered), providing to embed the xgapp application file, the.jape file for annotation rules and patterns, as well as all the required resources and plugins in the input zip file.

3.5. TF-IDF relevance estimation

The TF-IDF metric (*Term Frequency – Inverse Document Frequency*) is calculated for each candidate keywords and keyphrases as a measure of their relevance with respect not only to the single document in which it has been extracted, but rather to the whole corpus, represented by all documents and pages belonging to a single web domain. A simple thresholding upon computed TF-IDF values allows discarding the most common stop words and non-relevant terms, such as articles or conjunctions which usually have a high occurrence in natural language texts, although they do not contain significant information. TF-IDF is given by the product of two functions: the *Term Frequency* (TF) which provides a direct measure about how frequently a term occurs in a certain document, and the *Inverse Document Frequency* (IDF) which acts as balance term, showing higher values for terms having lower occurrence in the whole corpus (which are supposed to have a higher specificity for the specific domain context). The TF-IDF value for a candidate keyword or keyphrase k in a document d contained in a corpus D is given by:

$$(TF-IDF)_k = TF_k \cdot IDF_k,$$

where:

$$TF_k = \frac{f_k}{n_d}, \qquad IDF_k = \log \frac{N_d}{N_k}$$

being f_k the number of occurrences of the candidate keyword k in the document d, n_d the total number of terms contained in document d, N_D the total number of documents in the corpus D and N_K the total number of documents within the corpus in which the candidate key k appears. Candidate keywords and keyphrases having a TF-IDF value above a defined threshold are annotated as final keywords/keyphrases, while the other are pruned. Extracted keywords and keyphrases are finally stored in the Hadoop HDFS file system, together with their corresponding TF-IDF values and source web domain URL.

3.6. External DB storage

Once the output of the *Keywords & Keyprhase Extractor* module has been written on the Hadoop file system, a final processing step is required to populate an external SQL database which allows external access to the extracted information. For this purpose, the Apache Sqoop[9] open source tool has been used, which has been specifically designed for data transfer between Hadoop HDFS and relational datastores. The Sqoop tool has been installed on the master Namnode only and the export feature has been used to insert HDFS resident data into an external SQL database. In order to successfully accomplish the data export, full read/write privileges on the database have been granted to all the machines on the cluster. Each database record is populated with an extracted keyword or keyphrase and its corresponding metadata, that is POS-tag (or a different custom tag

9 http://sqoop.apache.org/

if it is a keyphrase), the TF-IDF value, the source web domain and the crawling timestamp.

4. Evaluation

As an improvement of the work presented in [16], the evaluation of performances scalability of the proposed system has been conducted in a way similar to the former validation. However, we extended the dataset from 10,000 to 20,000 web page and documents and we evaluated the processing time for the execution of the Keyword Extractor Module on the text content of considered dataset which had been previously parsed by the Nutch-based crawler. In this way, no more external network access is required for keywords and keyphrase extraction. By this way we minimize bottlenecks and other issues which not depend from the parallelization of the process. Thanks to these general optimizations to the code and to the test configuration, the proposed solution has shown several improvements, with respect to the version proposed in [16], both in terms of time performances and scalability. The Hadoop cluster architecture used for tests has been assessed on different configurations, ranging from 2 to 5 nodes. Each node is a Linux 8-cores workstation with Hadoop HDFS installed. In order to avoid data integrity errors and failures due to decommission and re-commission of cluster nodes, Hadoop allows to perform a rebalance of stored blocks among the active nodes of the cluster, if necessary.

The MapReduce model supplies speculative execution of tasks, and it is designed to provide redundancy in order to handle fault tolerance. By this way, it may happen that the JobTracker has to reschedule failed or killed tasks, and this can affect the execution time of the whole process. Therefore, for performance comparison, the best processing times have been selected among several test instances that have been conducted for each node configuration. By this way, the number of attempts for re-executing failed or killed tasks is supposed to be minimized. For the whole test dataset, containing about 20,000 documents, a total of nearly 9 million keywords and keyphrases have been extracted. Time processing results for the different tested node configurations are shown in Table 1. As a term of comparison, running the same GATE application on the same corpora dataset on a single non-Hadoop workstation took approximately 115 h. A possible explanation to this significant performance gap can be the fact that the Java code of our standalone GATE application is not optimized for multi-threading, while the MapReduce adaptation executed in Hadoop can benefit of MapReduce configuration parameters, which define the maximum number of map and reduce task slots to run simultaneously (exploiting multi-core technology even on a single-node cluster). The resulting speed-up curve for our test data is shown in Fig. 3. The present solution outperforms the previous version proposed in [16].

As it can be noticed, the scaling capabilities of the proposed system confirm the nearly linear growth trend of the Hadoop architecture, and a significant improve can be already noticed with a small numbers of computational nodes (see Fig. 4).

Table 1
Evaluation results: time performances assessed for different cluster configurations.

Configuration	Processing time (hh:mm:ss)	Speed-up
HDFS-single node	06:55:46	–
HDFS-2 nodes	04:35:39	1.51
HDFS-3 nodes	03:12:26	2.16
HDFS-4 nodes	02:21:01	2.95
HDFS-5 nodes	01:56:11	3.58

5. Conclusions and future work

In this paper, a distributed system for crawling web documents and extracting keywords and keyphrases has

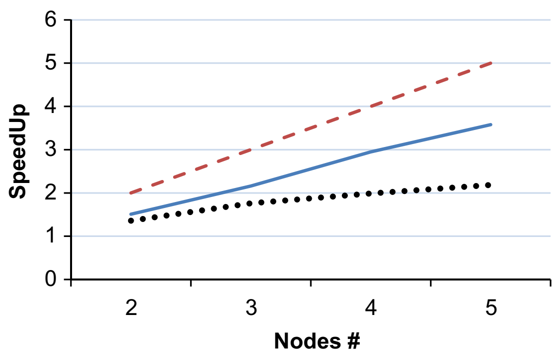

Fig. 4. Processing time performances depicted for distributed extraction of Keywords and Keyphrases (solid curve), performed with different cluster configurations, against the ideal linear trend (dashed curve) and test results (dotted curve) obtained in our previous work presented in [16].

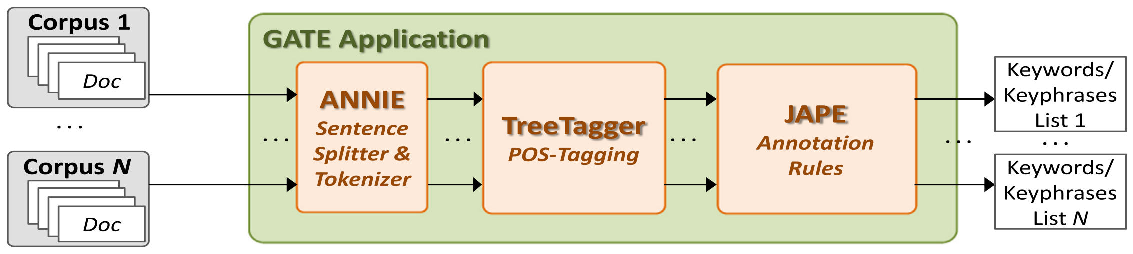

Fig. 3. The GATE pipeline used in the proposed framework. It is worthy to remark that the proposed system allows the execution of a generic GATE application.

been presented. The parallel architecture is provided by implementing the Apache Hadoop platform, while text annotation and key features extraction rely on the NLP opens source GATE platform. The main contributions offered by our work is the capability of executing general purpose GATE applications (including a wide range of NLP activities) in a distributed design (exploiting the benefits of scaling performances, especially for very large text corpora) with minimal code update and without the need for programmers to care about parallel computing constraints, such as task decomposition, mapping and synchronization issues. Evaluating processing performances on different cluster configurations (from 2 to 5 nodes) has showed a nearly linear scalability of the system, which is an encouraging result for future assessments on even larger datasets and cluster configurations. These actually represents open issues for future work. Moreover, it could be interesting to implement our keywords/keyphrases extraction module on other parallel computing environment, such as the cited Spark. Furthermore, in order to improve the quality of key features extraction, external knowledge resources could be used, especially Semantic repositories and frameworks, to allow the annotation of semantic features and relations.

References

[1] V. Turner, J.F. Gantz, D. Reinsel, S. Minton, The Digital Universe of Opportunities: Rich Data and the Increasing Value of the Internet of Things, IDC White Paper, 2014.
[2] P. Monali, K. Sandip, A Concise Survey on Text Data Mining, Int. J. Adv. Res. Comput. Commun. Eng. 3 (9) (2014) 8040–8043.
[3] J. Lin, C. Dyer, Data-Intensive text processing with MapReduce, Synthesis Lectures on Human Language Technologies, 177, , 2010.
[4] C.A. Navarro, N. Hitschfeld-Kahler, L. Mateu, A Survey on parallel computing and its applications in data-parallel problems using GPU architectures, Commun. Comput. Phys. 15 (2) (2014) 285–329.
[5] P. Jindal, D. Roth, L.V Kale, Efficient development of parallel NLP applications, Tech. Report of IDEALS (Illinois Digital Environment for Access to Learning and Scholarship), 2013.
[6] S. Orlando, R. Perego and F. Silvestri, Design of a Parallel and Distributed WEB Search Engine, In: Proceedings of the Parallel Computing (ParCo) Conference, Imperial College Press, September, 2001.
[7] O. Kononenko, O. Baysal, R. Holmes, M.W. Godfrey, Mining modern repositories with elasticsearch, In: Proceedings of the 11th Working Conference on Mining Software Repositories, 2014, pp. 328–331.
[8] S. Ghemawat, H. Gobioff, S.T. Leung, The google file system, In: Proceeedings of the 19th ACM Symposium on Operating Systems Principles, Lake George, NY, October, 2003.
[9] F. Colace, M. De Santo, L. Greco, P. Napoletano, Text classification using a few labeled examples, Comput. Hum. Behav. 30 (2014) 689–697.
[10] R. Al-Hashemi, Text summarization extraction system (TSES) using extracted keywords, Int. Arab J. e-Technol. 1 (4) (2010) 164–168.
[11] F. Colace, M. De Santo, L. Greco, P. Napoletano, Weighted word pairs for query expansion, Inf. Process. Manag. 51 (1) (2015) 179–193.
[12] N. Ide., L. Romary, International standard for a linguistic annotation framework, Nat. Lang. Eng. 10 (3–4) (2004) 211–225.
[13] A. Hulth, Improved automatic keyword extraction given more linguistic knowledge, In: Proceedings of the 2003 Conference on Emprical Methods in Natural Language Processing, Sapporo, Japan, 2003.
[14] T. Luis, Parallelization of Natural Language Processing Algorithms on Distributed Systems (master thesis), Information Systems and Computer Engineering, Instituto Superior Técnico, Univ. Técncica de Lisboa, 2008.
[15] T. Hamon, J. Deriviere, Nazarenko, Ogmios: a scalable NLP platform for annotating large web document collections, In: Proceedings of the Corpus Linguistics, Birmingham, United Kingdom, 2007.
[16] P. Nesi, G. Pantaleo, G. Sanesi, A distributed framework for NLP-based keyword and keyphrase extraction from web pages and documents, In: Proceedings of 21st International Conference on Distibuted Multimedia Systems (DMS2015), 2015.
[17] H. Cunningham, D. Maynard, K. Bontcheva, V. Tablan, GATE: a framework and graphical development enviroment for robust NLP tools and applications, In: Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics, ACL '02, Philadelphia, 2002.
[18] P. Turney, Learning algorithms for keyphrase extraction, Inf. Retr. 2 (2000) 303–336.
[19] C. Zhang, H. Wang, Y. Liu, D. Wu, Yi Liao, Bo Wang, Automatic keyword extraction from documents using conditional random fields, J. Comput. Inf. Syst. (2008).
[20] Y. Matsuo, M. Ishizuka, Keyword extraction from a single document using word co-ocuurrence statistical information, Int. J. Artif. Intell. Tools (2004).
[21] A. Azcarraga, M. David Liu, R. Setiono, Keyword extraction using backpropagation neural networks and rule extraction, In: Proceedings of IEEE World Congress on Computational Intelligence (WCCI), Brisbane, Australia, June, 2012.
[22] S. Siddiqi, A. Sharan, Keyword and keyphrase extraction techniques: a literature review, Int. J. Comput. Appl. 109 (2) .
[23] J. Kaur, V. Gupta, Effective approaches for extraction of keywords, Int. J. Comput. Sci. Issues 7 (6) (2010) 144–148.
[24] I. Witten, G. Paynte, E. Frank, C. Gutwin, C. Nevill-Manning, KEA: practical automatic keyphrase extraction, In: Proceedings of the 4th ACM Conference on Digital Library, 1999.
[25] C. Wu, M. Marches, J. Jiang, A. Ivanyukovich, Y. Liang, Machine learning-based keywords extraction for scientific literature, J. Univ. Comput. Sci. 13 (10) (2007) 1471–1483.
[26] Z. Liu, P. Li, Y. Zheng, M. Sun, Clustering to find exemplar terms for keyphrase extraction, In: Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing, 2009, pp. 257–266.
[27] F. Liu, D. Pennell, F. Liu, Yang Liu, Unsupervised approaches for automatic keyword extraction using meeting transcripts, In: Proceedings of the Human Language Technologies: The Annual Conference of the North American Chapter of the Association for Computational Linguistics, 2009, pp. 620–628.
[28] O. Medelyan, E. Frank, I.H. Witten, Human-competitive tagging using automatic keyphrase extraction, In: Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing, 2009, pp. 1318–1327.
[29] K.S. Hasan, V. Ng, Automatic keyphrase extraction: a survey of the state of the art, In: Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, vol. 1, 2014, pp. 1262–1273.
[30] K. Sarkar, M. Nasipuri, S. Ghose, A new approach to keyphrase extraction using neural networks, Int. J. Comput. Sci. Issues vol. 7 (Issue 2) (2010) 16–25. No 3.
[31] M. Grineva, M. Grinev, D. Lizorkin, Extracting key terms from noisy and multitheme documents, In: Proceedings of the 18th International Conference on World Wide Web, 2009, pp. 661–67.
[32] Z. Liu, W. Huang, Y. Zheng, M. Sun, Automatic keyphrase extraction via topic decomposition, In: Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing, 2010, pp. 366–376.
[33] M. Chung, D.I. Moldovan, Parallel natural language processing on a semantic network array processor, IEEE Trans. Knowl. Data Eng. vol. 7 (3) (1995) 391–404.
[34] M.P. van Lohuizen, Parallel processing of natural language parsers, In: Proceedings of the 15th Conference of Parallel Computing, 2000, pp. 17–20.
[35] N. Rizzolo, D. Roth, Learning based java for rapid development of NLP systems, In: Proceedings of the International Conference on Language Resources and Evaluation (LREC), 2010.
[36] L.V. Kale, G. Zheng, Charm++ and AMPI: adaptive runtime strategies via migratable objects, In: M. Parashar, X. Li (Eds.), Advanced Computational Infrastructures for Parallel and Distributed Applications, Wiley Interscience, New York, 2009, pp. 265–282.
[37] P. Exner, P. Nugues, KOSHIK-A large-scale distributed computing framework for NLP, In: Proceedings of the International Conference on Pattern Recognition Applications and Methods (ICPRAM 2014), 2014, pp. 463–470.
[38] V. Tablan, R.I. Cunningham, K. Bontcheva, GATECloud.net: a platform for large-scale, open-source text processing on the cloud, Philos. Trans. R. Soc. 37 (2013).

[39] M. Zaharia, M. Chowdhury, M.J. Franklin, S. Shenker, I. Stoica, Spark: Cluster Computing with Working Sets, Technology Report of UC Berkeley, 2011.

[40] 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: Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12), 2012, pp. 15–28.

[41] S. Gopalani, R. Arora, Comparing apache spark and map reduce with performance analysis using K-means, Int. J. Comput. Appl. 113 (1) (2015) 8–11.

[42] T. White, Hadoop, the Definitive Guide, O'Reilly, Sebastopol, CA, USA, 2012.

[43] A. Holmes (Ed.), Hadoop in Practice, Manning Publications Co., Greenwich, CT, USA, 2012.