Knowledge-Based Systems 79 (2015) 36–42

Contents lists available at ScienceDirect

Knowledge-Based Systems

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

Big data for Natural Language Processing: A streaming approach

Rodrigo Agerri [*], Xabier Artola, Zuhaitz Beloki, German Rigau, Aitor Soroa

IXA NLP Group, University of the Basque Country (UPV/EHU), Donostia-San Sebastián, Spain

A R T I C L E I N F O

Article history:
Received 30 March 2014
Received in revised form 22 October 2014
Accepted 8 November 2014
Available online 20 November 2014

Keywords:
Natural Language Processing
Distributed NLP architectures
Big data
Storm
NLP tools

A B S T R A C T

Requirements in computational power have grown dramatically in recent years. This is also the case in many language processing tasks, due to the overwhelming and ever increasing amount of textual information that must be processed in a reasonable time frame. This scenario has led to a paradigm shift in the computing architectures and large-scale data processing strategies used in the Natural Language Processing field. In this paper we present a new distributed architecture and technology for scaling up text analysis running a complete chain of linguistic processors on several virtual machines. Furthermore, we also describe a series of experiments carried out with the goal of analyzing the scaling capabilities of the language processing pipeline used in this setting. We explore the use of Storm in a new approach for scalable distributed language processing across multiple machines and evaluate its effectiveness and efficiency when processing documents on a medium and large scale. The experiments have shown that there is a big room for improvement regarding language processing performance when adopting parallel architectures, and that we might expect even better results with the use of large clusters with many processing nodes.

1. Introduction

Professionals in any sector need to have access to accurate and complete knowledge to be able to take well-informed decisions. This is getting more and more difficult due to the sheer size of data they need to process. This also means that the knowledge and information of professionals is quickly getting out of date. However, their decisions have an even bigger impact in today's highly-interconnected world. Thus, professional decision-makers are involved in a constant race to stay informed and to respond adequately to any changes, developments and news. However, the volume of news and documents provided by major information brokers has reached a level where state-of-the-art tools are no longer adequate to provide a solution.

Processing huge amounts of textual data has become a major challenge in the Natural Language Processing (NLP) research area. As the majority of digital information is present in the form of unstructured data such as web pages or news articles, NLP tasks such as cross-document coreference resolution, event detection or calculating textual similarities often require processing millions of documents in a timely manner. For example, the main goal of the Newsreader project[1] is to perform multilingual real-time event detection and extract from text what happened to whom, when and where, removing duplication, complementing information, registering inconsistencies and keeping track of the original sources. The project foresees an estimated flow of 2 million news items per day and the complex linguistic analysis of those documents needs to be done in a reasonable time frame (one or few hours). Therefore, the project faces an important challenge with respect to the scalability of the text processing.

This overwhelming flow of textual data calls for a paradigm shift in the computing architecture and large scale data processing. For instance, Singh et al. [27] process a corpus comprising news articles published during the last 20 years. McCreadie et al. [20] present a distributed framework for event detection that is capable of effectively processing thousands of twitter posts every second. These challenges fall into a new class of the so called "Big Data" tasks, requiring large scale and intensive processing which have to be able to efficiently scale up to huge amounts of data [23,27,20].

This paper presents a new distributed architecture and technology for scaling up text analysis to keep pace with the rate of current growth of news streams and collections. We designed and deployed a complete chain of NLP modules within virtual machines (VMs). We also present the twelve NLP modules included

* Corresponding author.
E-mail addresses: rodrigo.agerri@ehu.es (R. Agerri), xabier.artola@ehu.es (X. Artola), zuhaitz.beloki@ehu.es (Z. Beloki), german.rigau@ehu.es (G. Rigau), a.soroa@ehu.es (A. Soroa).

[1] http://www.newsreader-project.eu/.

http://dx.doi.org/10.1016/j.knosys.2014.11.007
0950-7051/© 2014 Elsevier B.V. All rights reserved.

into the virtual machines, of which a subset, the IXA pipes tools, are the ones used to do most of the experimentation [1].[2] We also provide empirical performance results when applied to realistic volumes of news within hard time constraints.

The rest of the paper is organized as follows. Section 2 presents and discusses alternative big data frameworks. Section 3 presents the text analysis modules of the English and Spanish pipelines. Section 4 describes the distributed architecture and deployed solution for massive processing of streams of texts. Section 5 describes the experiments carried out to evaluate the performance of the proposed solution. Finally, in Section 6 we discuss some concluding remarks and planned future research work.

2. Big data frameworks

Processing massive quantities of data requires designing solutions that are able to run distributed programs across a large cluster of machines [30]. Besides, issues such as parallelization, distribution of data, synchronization between nodes, load balancing and fault tolerance are of paramount importance. *Apache Hadoop*[3] is a framework designed to perform large scale computations that is able to scale to thousands of nodes in a fault-tolerant manner. It is probably the most widely used framework for large scale processing on clusters of commodity hardware. Hadoop implements *MapReduce*, a programming model for developing parallel and distributed algorithms that process and generates large data sets. Hadoop is the basis for a large number of other specific processing solutions such as Mahout[4] for machine learning or Giraph[5] for graph processing, to name but a few.

One of the main problems of using the Hadoop framework is that it requires casting any computation as a MapReduce job. In the case of the NLP pipeline presented in this work, these solutions would require a complete reimplementation of each NLP module, which is clearly impractical. *Apache SPARK* [31] overcomes this problem by extending Hadoop with new workloads like streaming, interactive queries and learning algorithms.

Hadoop follows a *batch* processing model, where computations start and end within a given time frame. In a *streaming computing* scenario [8], however, the processing is open-ended. Thus, the program is designed to process documents forever while maintaining high levels of data throughput and a low level of response latency.

Storm[6] is an open source, general-purpose, distributed, scalable and partially fault-tolerant platform for developing and running distributed programs that process continuous streams of data. Storm is agnostic with respect to the programming model or language of the underlying modules, and, thus, it is able to integrate third party tools into the framework.

The main abstraction structure of Storm is the *topology*, a top level abstraction which describes the processing node that each message passes through. The topology is represented as a graph where nodes are processing components, while edges represent the messages sent between them. Topology nodes fall into two categories: the so called *spout* and *bolt* nodes. *Spout* nodes are the entry points of a topology and the source of the initial messages to be processed. *Bolt* nodes are the actual processing units, which receive incoming text, process it, and pass it to the next stage in the topology. There can be several instances of a node in the topology, thus allowing actual parallel processing.

The data model of Storm is a *tuple*, namely, each *bolt* node in the topology consumes and produces tuples. The tuple abstraction is general enough to allow any data to be passed around the topology.

In Storm, each node of the topology may reside on a different physical machine; the Storm controller (called *Nimbus*) is the responsible of distributing the tuples among the different machines, and of guaranteeing that each message traverses all the nodes in the topology. Furthermore, *Nimbus* performs automatic re-balancing to compensate the processing load between the nodes.

Section 4 describes our solution to big data processing using *virtualization*, Apache Storm and a set of NLP tools organized in a data-centric architecture. The description of such NLP tools is the subject of the next section.

3. NLP pipeline

Many Natural Language Processing (NLP) applications demand some basic linguistic processing (Tokenization, Part of Speech (POS) tagging, Named Entity Recognition and Classification (NERC), Syntactic Parsing, Coreference Resolution, etc.) to be able to further undertake more complex tasks. Generally, NLP annotation is required to be as accurate and efficient as possible and existing tools, quite rightly, have mostly focused on performance. However, this generally means that NLP suites and tools usually require researchers to perform complex compilation/installation/configuration procedures in order to use such tools. At the same time, in the industry, there are currently many Small and Medium Enterprises (SMEs) offering services that one way or another depend on NLP annotations.

In both cases, in research and industry, acquiring, deploying or developing such base qualifying technologies is an expensive undertaking that redirects their original central focus. In research, much time is spent in the preliminaries of a particular research experiment trying to obtain the required basic linguistic annotation, whereas in an industrial environment SMEs see their already limited resources taken away from offering products and services that the market demands. In order to address this issue, we have developed a set of NLP tools which we refer to as the IXA pipes tools [1].[7] The IXA pipes tools consist of *ready to use modules* to perform efficient and accurate linguistic annotation while allowing users to focus on their original, central task.

3.1. IXA pipes

The aim of the IXA pipes tools is to provide multilingual NLP tools that are simple and ready to use, portable, modular, efficient, accurate and distributed under a free license. As in Unix-like operating systems, the IXA pipes consists of a set of processes chained by their standard streams, in a way that the output of each process feeds directly as input to the next one. The Unix pipeline metaphor has been applied for NLP tools by adopting a very simple and well-known data-centric architecture, in which every module/pipe is interchangeable for another one as long as it reads and produces the required data format. The IXA pipes are designed to minimize or eliminate any installation/configuration/compilation effort and are distributed under the Apache 2.0 license, which is free and commercially friendly [1].

The data-centric architecture of the IXA pipes relies on a common interchange format in which both the input and output of the modules needs to be formatted to represent and filter linguistic annotations: the NLP Annotation Format (NAF[8] [16]). NAF has evolved from the KYOTO Annotation Framework (KAF [6]) and it is

[2] http://ixa2.si.ehu.es/ixa-pipes.
[3] http://hadoop.apache.org/.
[4] https://mahout.apache.org/.
[5] http://giraph.apache.org/.
[6] http://storm.incubator.apache.org/.
[7] http://ixa2.si.ehu.es/ixa-pipes.
[8] http://wordpress.let.vupr.nl/naf/.

compliant with the Linguistic Annotation Format (LAF [18]). NAF is a standoff layered representation of the analysis of a whole series of NLP modules ranging from tokenization, part-of-speech tagging, lemmatization, dependency parsing, named entity recognition, semantic role labeling, event and entity-coreference to factuality and opinions. NAF is a document based representation.

The IXA pipes currently provide the following linguistic annotations for both English and Spanish: sentence segmentation, tokenization, part-of-speech (POS) tagging, lemmatization, Named Entity Recognition and Classification (NERC), constituent parsing and coreference resolution. To avoid duplication of efforts, the supervised probabilistic algorithms used to train the POS tagger, NERC tagger and Constituent parser are those implemented by the Apache OpenNLP machine learning API.[9]

We now describe the modules so far developed. We also present their empirical evaluation. Additionally, in Section 3.3 we describe other third-party NLP tools which have been added to the pipeline in the context of the Newsreader project.[10]

ixa-pipe-tok provides rule-based Sentence Segmentation and Tokenization. The rules are originally based on the Stanford English Tokenizer,[11] but with substantial modifications and additions. These include tokenization for other languages such as French and Italian, normalization according to the Spanish Ancora Corpus [28], paragraph treatment, and more comprehensive gazetteers of non breaking prefixes. The tokenizer performs at peaks of around 250 K words per second.

ixa-pipe-pos is a POS tagger and lemmatizer for English and Spanish. *Perceptron* [10] models for English have been trained and evaluated on the WSJ treebank using the usual partitions, as explained in [29]; ixa-pipe-pos scores 97.07% vs 97.24% obtained by the Stanford POS tagger [29]. For Spanish, *Maximum Entropy* models have been trained and evaluated using the Ancora corpus, which was randomly divided in 90% for training and 10% for testing. We obtain a performance of 98.88% (the corpus partitions are available for reproducibility). Giménez and Marquez [17] report 98.86% whereas Freeling [22] report around 97%, although they train and test on a different subsets of the Ancora corpus. Piping ixa-pipe-tok and ixa-pipe-pos annotates around 5500 words/s.

ixa-pipe-nerc provides Named Entity Recognition (NERC) for English and Spanish trained on a variety of datasets. The experiments performed in this paper use models trained on the CONLL 2002[12] and 2003[13] tasks for four types of named entities: persons, locations, organizations, and names of miscellaneous entities that do not belong to the previous three groups. The experiments described in Section 4 use two very fast language-independent models using a rather simple set of local features (e.g., similar to that of [9], except POS tag features). For English, Perceptron models have been trained using the CoNLL 2003 dataset. We currently obtain 84.52 F1 which is coherent with other results reported with these features [9,24]. Other, better performing models trained with external knowledge are available (87.11 F1) but at the cost of slower speed performance. The best Stanford NERC model reported on this dataset achieves 86.86 F1 [15], whereas the best system on this dataset achieves 90.80 F1 [24], using non-local features and substantial external knowledge. For Spanish we currently obtain best results training Maximum Entropy models. Our best model using only local features obtains 80.20 F1 vs 81.39 F1 [7], the best result so far on this dataset. Their result uses external knowledge and their system obtains 79.28 F1 without it.

ixa-pipe-parse provides statistical constituent parsing for English and Spanish. Maximum Entropy models are trained to build shift-reduce bottom-up parsers [25] as implemented by the Apache OpenNLP API. Parsing models for English have been trained using the Penn Treebank and for Spanish using the Ancora corpus [28]. For English we obtain 87.21 parseval F, a little bit lower than other more fine-tuned parsers [11]. As far as we know, and although previous approaches exist [12], *ixa-pipe-parse* provides the first publicly available statistical parser for Spanish, obtaining around 88.10 parseval F.

ixa-pipe-coref performs coreference resolution. The algorithm is loosely based on the Stanford Multi Sieve Pass system [19]. So far we have evaluated our module on the CoNLL 2011 dev set and we are 2 CoNLL F scores behind the Stanford's system (57.8 vs 59.3 CoNLL F1), the best on that task [19].

3.2. Related NLP toolkits

We believe that among the free, commercially friendly and multilingual toolkits, our NLP pipeline performs competitively in terms of evaluation and performance, but also that, together with the third-party tools added, provides one of the most, if not the most, comprehensive ready-to-use NLP pipelines currently available under a free and permissive license such as Apache License 2.0.

Other NLP toolkits provide similar functionalities to the IXA pipes tools, although not many of them provide multilingual support. GATE [13] is an extensive framework supporting annotation of text. GATE has some capacity for wrapping Apache UIMA components,[14] so it should be able to manage distributed NLP components. It also provides cloud services for big data processing via the GateCloud.[15] However, GATE is a very large and complex system, with a corresponding steep learning curve.

Freeling [22] provides multilingual processing for a number of languages, including Spanish and English. As opposed to the IXA pipes, Freeling is a monolithic toolkit written in C++ which needs to be compiled natively. The Stanford CoreNLP[16] is also a monolithic suite, which makes it difficult to integrate other tools in its chain.

Due to the data-centric architecture of IXA pipes, it is virtually trivial to replace or extend the toolchain with a third-party tool. The only requirement is for a tool to write and read NAF format, which the *kaflib* library[17] makes extremely easy.

Additionally, every IXA pipe also offers the possibility of easily training new models with your own data for POS tagging, NERC and constituent parsing. The IXA pipes are already being used for big data processing in several FP7 European projects: OpeNER,[18] NEWSREADER, QTLEAP,[19] LIMOSINE[20] among others. In addition to the performance results stated in Section 3.1 and Table 2 provides statistics about the individual performance of each of the processing modules for two different datasets.

In the Newsreader project the final goal is to discover and structure the hidden knowledge in large amount of texts about what happened to whom, when and where. Thus, the linguistic annotation provided by the IXA pipes is not enough, as information about factuality, semantic roles, etc. is required. To this aim, we exploit the data-centric design and modularity of the NLP pipeline to easily chain other linguistic processors. As it has been already mentioned, the only requirement to integrate new tools in the

9 http://opennlp.apache.org.
10 http://www.newsreader-project.eu/.
11 http://www-nlp.stanford.edu/software/tokenizer.shtml.
12 http://www.clips.ua.ac.be/conll2002/ner/.
13 http://www.clips.ua.ac.be/conll2003/ner/.
14 http://uima.apache.org/.
15 https://gatecloud.net/.
16 http://nlp.stanford.edu/software/corenlp.shtml.
17 https://github.com/ixa-ehu/kaflib.
18 http://www.opener-project.org.
19 http://qtleap.eu.
20 http://limosine-project.eu/.

pipeline is for any new tool to read and write NAF via its standard input/output streams. The result is, to our knowledge, the most comprehensive multilingual NLP pipeline out there.

3.3. Additional modules

We have integrated seven additional tools to the pipeline to add the following linguistic annotations for the Newsreader project: recognition of temporal expressions, word-sense disambiguation (WSD), named entity disambiguation (NED), semantic role labeling (SRL), factuality recognition, opinion mining, and resolution of event coreference.

Temporal Expression Recognition: *Timepro*[21] is a supervised probabilistic tagger for the recognition of temporal expressions (date, duration, set, time). Their TempEval3 trained model obtains 72.36 F score.

Word Sense Disambiguation: The *svm_wsd* implements a machine learning Word Sense Disambiguation system based on Support Vector Machines.[22]

Named Entity Disambiguation: *DBpedia Spotlight* [21] is a Wikification tool for automatically annotating mentions of DBpedia resources in text, providing a solution for linking unstructured information sources to the Linked Open Data cloud through DBpedia. DBpedia Spotlight recognizes that names of concepts or entities have been mentioned (e.g. "Michael Jordan"), and subsequently matches these names to unique identifiers (e.g. the machine learning professor[23] or the basketball player[24]). We have created a NED client to query the DBpedia Spotlight server for the Named entities detected by the *ixa-pipe-nerc* module.

Semantic Role Labelling: The Mate tools[25] provide a state-of-the-art dependency parser and semantic role labeler [5]. The tools are language independent, provide a very high accuracy and are fast. The dependency parser had the top score for German and English dependency parsing in the CoNLL shared task 2009.

Factuality: We use a tool to detect factuality which has been developed within the NewsReader project.[26] The tool aims at classifying whether an linguistic expression or event has actually happened. The module has been trained on the main resource for factuality detection, namely, the FactBank [26].

Opinion miner: An opinion miner based on machine learning. The opinion mining task is divided into two steps: detection of opinion entities (holder, target and expression) using Conditional Random Fields and Opinion entity linking (expression < target and expression < holder) using binary Support Vector Machines.[27]

Event Coreference resolution: We use the tool developed by [14] to perform intra and across documents event coreference resolution.

4. Big data processing

In this section we describe the different configurations created with the NLP tools described in the previous section. The aim is to configure a linguistic pipeline that is able to automatically extract or discover knowledge in huge amounts of texts by using big data processing with the tools that will be presented in this section.

Scalable NLP processing requires parallel processing of textual data. The parallelization can be effectively performed at several levels, from deploying copies of the same linguistic processor (LP) among servers to the reimplementation of the core algorithms of each module using multi-threading, parallel computing. This last type of fine-grained parallelization is clearly out of the scope of the present work, as it is unreasonable to re-implement all the modules needed to perform such a complex task as mining events. We rather aim to processing huge amount of textual data by defining and implementing an architecture for NLP which allows the parallel processing of documents.

4.1. A distributed pipeline for NLP processing

Deploying Linguistic Processors (LPs) often requires pre-installing a large set of common software modules on the same machine, which must be accessible to the LP. The capacity to reproduce experiment results is a crucial instrument in any scientific endeavor and thus, we aim at building a NLP pipeline which analyzes the documents in a reproducible manner: an LP module applied to a particular input text has to produce the same output regardless of the software framework (machine, operating system, etc.) on which it is installed. Therefore, special care has to be taken to guarantee that the same version of the LP modules, along with the exact same dependencies, are deployed among the different machines.

This have led us to adopt virtual machine (VM) technologies for deploying the LP modules. Virtualization is a widespread practice that increases the server utilization and addresses a variety of dependencies and installation requirements. Besides, virtualization is a 'de facto' standard in cloud computing solutions, which offers the possibility of installing many copies of the virtual machines on commodity servers.

Specifically, we create one VM per language and pipeline configuration so that a full processing chain in one language can be run on a single VM. This approach allows us to scale horizontally (or scale out) as a solution to the problem of dealing with massive quantities of data. We thus scale out our solution for NLP by deploying all the NLP modules into VMs and making as many copies of the VMs as necessary to process an initial batch of documents on time.

Table 1 shows the modules installed into the English VM. We defined two pipelines for event extraction, each one comprising different modules (last column in the table).

Inside each VM the modules are managed using the Storm framework for streaming computing, where each LP module is wrapped as a *bolt* node inside the Storm topology (c.f. Section 2). When a new tuple arrives, the *bolt* node calls an external command sending the tuple content to the standard input stream. The output of the LP module is received from the standard output stream and passed to the next node in the topology. Each module thus receives a NAF document with the (partially annotated) document and adds

Table 1
LP modules installed on the VMs. The last column shows the pipeline version where the modules were used.

Module	Description	Pipeline
ixa-pipe-tok	Tokenizer, sentence splitter	(1,2)
ixa-pipe-pos	POS tagger	(1,2)
ixa-pipe-parse	Constituency parser	(2)
TimePro	Time expression recognition	(1,2)
ixa-pipe-nerc	Named Entity Recognition	(1,2)
WSD	Word Sense Disambiguation	(1,2)
dbpedia-spotlight	Named Entity Disambiguation	(1,2)
ixa-pipe-coref	Coreference resolution	(2)
MATE	Dependency parser and Semantic Role Labeling	(1,2)
opinion miner	Opinion detection and Opinion holders to targets	(2)
factuality	Factuality	(1,2)
eCoref	Event coreference	(1,2)

[21] http://textpro.fbk.eu/docs.html.
[22] https://github.com/cltl/svm_wsd.
[23] http://dbpedia.org/page/Michael_I._Jordan.
[24] http://dbpedia.org/page/Michael_Jordan.
[25] http://code.google.com/p/mate-tools/.
[26] https://github.com/newsreader/Factuality-Classifier.
[27] https://github.com/cltl/opinion_miner_deluxe.

Table 2
Total time (in s) and percentage taken by each module in the *car* and *wikinews* datasets. Note that we use two different versions of the pipeline on each dataset.

Module	*Car* dataset			*Wikinews* dataset		
	Time (s)	%	# Elems.	Time (s)	%	# Elems.
ixa-pipe-tok	12,152	0.41	35,187,862	3954	0.29	5,919,406
opinion-miner	–	–	–	13,584	0.99	66,771
ixa-pipe-pos	45,352	1.55	34,527,492	14,802	1.08	5,906,089
WSD	78,080	2.66	595,874	19,454	1.42	81,797
ixa-pipe-parse	–	–	–	22,780	1.66	9,716,924
MATE-DEP	121,092	4.13	31,943,943	25,688	1.87	5,611,177
dbpedia-spotlight	75,086	2.56	1,960,604	27,819	2.03	3,02,731
factuality	91,960	3.14	4,327,233	27,694	2.02	871,844
eCoref	73,098	2.49	3,747,382	31,183	2.27	812,787
ixa-pipe-nerc	112,643	3.84	2,475,062	43,750	3.19	355,284
TimePro	–	–	–	69,514	5.07	81,599
ixa-pipe-coref	–	–	–	297,627	21.71	188,361
MATE-SRL	2,322,472	79.21	5,246,967	773,055	56.39	1,033,657
Total	2,931,935	100.00	120,012,419	1,370,904	100.00	30,948,427

new annotations into it. The tuples in our Storm topology consist of two elements, a document identifier and the document itself, which is encoded as a string with the XML serialization of the NAF document.

If one module fails to produce a valid NAF document, the input document is moved to a specific directory and a log entry is created. The processing of this particular document is stopped at this point, and the system starts processing the next document in the input directory.

Inside the VM there is an initial *spout* which scans for a particular directory. When a new document arrives, the *spout* passes the document to the first node in the pipeline, which in turn will pass its output to the next stage, and so on. This setting is similar to a standard pipeline architecture but it has a main advantage: when a module finishes its processing, it passes the annotated document to the next step, and starts processing the next document. Therefore, in this setting there are as many documents processed in parallel as stages in the pipeline.

Note that in this approach Storm is used within a single VM and that this setting is not ideal nor the type of architecture for which the Storm framework is meant to be used for. However, using Storm as the controlling backbone of the LP modules installed within each VM has many advantages: First, inside each VM the Storm topology is able to run many LP modules in parallel. Second, having implemented this *batch* approach using Storm, it is straightforward to adopt a fully *streaming* architecture, as described in Section 5.1, where LP modules reside on several distributed VMs. Finally, the initial experiments performed using the batch approach will give valuable insights as to identifying those language processors which require more resources than others.

5. Experiments

As explained above, we ran two versions of the pipeline in our experiments and each version was used to process a different set of documents. The first dataset (the *car* dataset) consists of 64,540 documents which were selected by first performing a query on a news collection by selecting interesting documents describing events which involve two or more car companies. The second dataset (the *wikinews* dataset) is a collection of 18,886 documents extracted from the Wikinews website.[28] In order to process the documents, each dataset was split into batches, each one containing 3000 documents, and each batch was sent to the LP pipeline for linguistic processing. In total, we used 8 VMs running in parallel for the experiments.

Table 2 shows the total time spent by each module, the total percentage of the time, and the number of elements extracted from both pipelines and datasets. The elapsed times shown in the table are calculated by summing the elapsed time spent by each module. Therefore, the times shown in the table do not consider that many of those documents are actually processed in parallel. The table shows that if the *car* and *wikinews* documents were processed sequentially, they would require 33.9 and 15.8 days to process, respectively. Having 8 VMs running on parallel, these documents were actually processed in around 5 and 2 days.

Table 2 also suggests an unbalance regarding the time spent by each module of the pipeline. In the *car* dataset the *MATE-SRL* module takes almost the 80% of the whole processing time and is by far the module needing more time to complete its task. In the *wikinews* dataset, *MATE-SRL* takes the 56% of the overall elapsed time, followed by the *ixa-pipe-coref* module which spends the 21% of the overall time. These results suggest that both modules are good candidates for parallelization; thus, if we were able to execute several instances of those modules in parallel, the overall performance of the linguistic processing would boost considerably.

5.1. Running modules in parallel

The Storm framework allows several instances of each topology node, thus allowing actual parallel processing. We thus performed a separate experiment with the aim of measuring the expected performance gain when executing time consuming modules in parallel [4]. The experiments were performed on a PC machine with an Intel Core i5-3570 3.4 GHz processor with 4 cores and 4 GB RAM, running on Linux.

For this experiment we wanted to mimic the pipeline described in the previous section. Specifically, we wanted one single module to consume most of the resources required. In this sense, we would be able to measure the performance boost when running many instances of this demanding module in parallel. The previous section showed that *MATE-SRL* is such a resource demanding module. However, running many *MATE-SRL* processes on a single machine turned out to be a rather difficult task, which consumed all the machine available RAM. Therefore, and to keep the experiment manageable in terms of resources and time, we created a small NLP pipeline comprising four modules: *ixa-pipe-tok*, *ixa-pipe-pos*, *ixa-pipe-nerc* and *UKB*, a tool for performing graph-based Word Sense Disambiguation (WSD) using a pre-existing knowledge base [3,2].

[28] http://en.wikinews.org.

Initially the four modules are executed following a pipeline architecture, namely, each module running sequentially one after the other. This setting is the baseline system and the starting point of our analysis.

In a second experiment we implement a Storm topology following again a pipeline approach. This setting is similar to the baseline system but has an advantage: when a module finishes the processing, it passes the annotated document to the next step, and starts processing the next document. Therefore, in this setting there are as many documents processed in parallel as there are stages in the pipeline. Given that the pipeline consists of 4 modules, it will be able to process 4 documents concurrently. Finally, we experiment creating many instances of some selected *bolt* nodes, therefore allowing the parallel execution of them.

We experimented processing 1000 documents, each one containing an average of 1200 words and 50 sentences. We performed experiments with a subset of 100 documents (138,803 words, 5416 sentences) and with the complete set of 1000 documents (1,185,933 words, 48,746 sentences).

Table 3 shows the time elapsed in processing the documents. The first six rows correspond to the processing of 100 documents and the last six rows to the processing of 1000 documents. As Table 3 shows, the baseline system runs at a performance of about 100 words per second. The simple Storm topology yields a performance gain of less than 13%, which is less than expected. The 96% of the processing time is spent by the UKB WSD module, which is by far the module needing more time to complete its task. Although the Storm topology can in principle multiply the performance by a factor of four, in practice all the computing is concentrated in one single node, which severely compromises the overall performance gain.

With these points in mind, we experimented with four alternatives (named *Storm*$_2$, *Storm*$_4$, *Storm*$_5$, and *Storm*$_6$), with respectively 2, 4, 5 and 6 instances of the UKB WSD module running in parallel. The results in Table 3 show that running multiple instances of the UKB WSD does increase the overall performance significantly. The biggest gain is obtained with five instances of WSD, with an increase of 63% in the overall performance. Therefore, more WSD instances do not help improving the results, which is expected given the fact that the machine used for the experiments has 4 CPU cores.

Summarizing, this initial experiments have shown that there is large room for improvement regarding NLP processing performance. A careful identification of the most time and resource consuming NLP modules would allow creating parallel topologies which will yield much better performance. With the use of large multi-node clusters, we can expect a significant boost in the performance.

Table 3
Performance of the NLP pipeline in different settings: *pipeline* is the basic pipeline used as baseline; *Storm* is the same pipeline executed as a Storm topology; *Storm*$_2$ represents a Storm pipeline with 2 instances of the WSD module (*Storm*$_4$ has 4 instances, *Storm*$_5$ 5, and *Storm*$_6$ 6).

	Total time	Words/s	Sent/s	Gain (%)
100 documents				
Pipeline	21 m16 s	108.8	4.2	–
Storm	18 m43 s	123.5	4.8	12.0
Storm$_2$	10 m48 s	214.3	8.4	49.3
Storm$_4$	7 m46 s	297.6	11.6	63.5
Storm$_5$	7 m44 s	299.1	11.7	63.7
Storm$_6$	7 m48 s	296.1	11.6	63.3
1000 documents				
Pipeline	3 h15 m16 s	101.2	4.2	–
Storm	2 h50 m21 s	116.0	4.8	12.8
Storm$_2$	1 h40 m37 s	196.5	8.1	48.5
Storm$_4$	1 h14 m25 s	265.6	10.9	61.9
Storm$_5$	1 h10 m45 s	279.3	11.5	63.8
Storm$_6$	1 h11 m37 s	276.0	11.3	63.3

6. Conclusion and future work

In this paper we have presented a new distributed architecture and technology for scaling up text analysis to keep pace with the rate of the current growth of news streams and collections. We designed and deployed a complete chain of NLP modules within virtual machines. We also present the NLP modules included into the virtual machines, most of them coming from the IXA pipes tools[29] [1]. Moreover, we provide empirical performance results when applied on realistic volumes of news within hard time constraints.

The experiments described here follow a pipeline approach, but in principle we could also run them on non-linear topologies, where two modules are processing the same document at the same time. Non-linear topologies require identifying the pre- and post-requisites of each module, thus deducting the indications as to which modules must precede which and which modules may be run in parallel on the same document. They would also need a special *bolt* that would receive the input from many NLP modules *bolts* (each one conveying different annotations on the same document) and would *merge* every source of information producing a single, unified document.

We want also try different levels of granularity. For instance, a POS tagger works at sentence level, the WSD module works at paragraph level, whereas a coreference module works at document level. We want to experiment splitting the input document into pieces of the required granularity, so that the NLP modules can quickly analyze those pieces, thus increasing the overall processing speed.

As we are developing a fully distributed and highly scalable system, several architecture-related issues come out. One of them is the input method that will receive text documents and send them to the pipeline. To accomplish that, we foresee the need of a distributed message queue system as the input. Another issue is the fact that too much data traffic is produced between each NLP module, since a full NAF document with all the annotation layers must be sent from each module for every document to be processed. This could be avoided using a distributed NoSQL database like MongoDB, and retrieving and storing only the annotation layers required and produced by each module.

Acknowledgements

We are grateful to the anonymous reviewers for their insightful comments. This work has been partially funded by the OpeNER (FP7 Grant No. 296451), NewsReader (FP7-ICT 2011-8-316404) and SKaTer (TIN2012-38584-C06-02) projects. Zuhaitz Beloki's work is funded by a Ph.D. grant from the University of the Basque Country (UPV/EHU), Spain.

References

[1] R. Agerri, J. Bermudez, G. Rigau, IXA Pipeline: efficient and ready to use multilingual NLP tools, in: Proceedings of the 9th Language Resources and Evaluation Conference (LREC2014), Reykjavik, Iceland, 2014.
[2] E. Agirre, O. López de Lacalle, A. Soroa, Random walks for knowledge-based word sense disambiguation, Comput. Linguist. 40 (2014) 57–84.
[3] E. Agirre, O.L.D. Lacalle, A. Soroa, Knowledge-based WSD on specific domains: performing better than generic supervised WSD, in: Proceedings of IJCAI 2009, 2009.
[4] X. Artola, Z. Beloki, A. Soroa, A stream computing approach towards scalable NLP, in: Proceedings of the 9th Language Resources and Evaluation Conference (LREC2014), Reykjavik, Iceland, 2014.

[29] http://ixa2.si.ehu.es/ixa-pipes.

[5] A. Björkelund, L. Hafdell, P. Nugues, Multilingual semantic role labeling, in: Proceedings of the Thirteenth Conference on Computational Natural Language Learning: Shared Task CoNLL '09, Boulder, Colorado, USA, 2009, pp. 43–48.
[6] W. Bosma, P. Vossen, A. Soroa, G. Rigau, M. Tesconi, A. Marchetti, M. Monachini, C. Aliprandi, KAF: a generic semantic annotation format, in: Proceedings of the GL2009 Workshop on Semantic Annotation, 2009.
[7] X. Carreras, L. Marquez, L. Padro, Named entity extraction using AdaBoost, in: Proceedings of the 6th Conference on Natural Language Learning, vol. 20, 2002, pp. 1–4.
[8] M. Cherniack, H. Balakrishnan, M. Balazinska, D. Carney, U. Cetintemel, Y. Xing, S. Zdonik, Scalable distributed stream processing, in: CIDR 2003 - First Biennial Conference on Innovative Data Systems Research, Asilomar, CA, 2003.
[9] S. Clark, J. Curran, Language independent NER using a maximum entropy tagger, in: Proceedings of the Seventh Conference on Natural Language Learning (CoNLL-03), Edmonton, Canada, 2003, pp. 164–167.
[10] M. Collins, Discriminative training methods for hidden markov models: theory and experiments with perceptron algorithms, in: Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing, vol. 10, 2002, pp. 1–8.
[11] M. Collins, Head-driven statistical models for natural language parsing, Comput. Linguist. 29 (2003) 589–637.
[12] B. Cowan, M. Collins, Morphology and reranking for the statistical parsing of Spanish, in: Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing, Association for Computational Linguistics, 2005, pp. 795–802.
[13] H. Cunningham, Gate, a general architecture for text engineering, Comp. Human. 36 (2002) 223–254.
[14] A. Cybulska, P. Vossen, Semantic relations between events and their time, locations and participants for event coreference resolution, in: Proceedings of the International Conference Recent Advances in Natural Language Processing RANLP 2013, INCOMA Ltd., Hissar, Shoumen, Bulgaria, 2013, pp. 156–163.
[15] J.R. Finkel, T. Grenager, C. Manning, Incorporating non-local information into information extraction systems by gibbs sampling, in: Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics, 2005, pp. 363–370.
[16] A. Fokkens, A. Soroa, Z. Beloki, N. Ockeloen, G. Rigau, W.R. van Hage, P. Vossen, NAF and GAF: linking linguistic annotations, in: Proceedings of 10th Joint ACL/ISO Workshop on Interoperable Semantic Annotation (ISA-10), LREC 2014 Workshop, Reykjavik, Iceland, 2014, p. 9.
[17] J. Giménez, L. Marquez, Svmtool: a general POS tagger generator based on support vector machines, in: Proceedings of the 4th International Conference on Language Resources and Evaluation, 2004.
[18] N. Ide, L. Romary, Éric Villemonte de La Clergerie, International standard for a linguistic annotation framework, in: Proceedings of the HLT-NAACL 2003 Workshop on Software Engineering and Architecture of Language Technology Systems (SEALTS), Association for Computational Linguistics, 2003.
[19] H. Lee, A. Chang, Y. Peirsman, N. Chambers, M. Surdeanu, D. Jurafsky, Deterministic coreference resolution based on entity-centric, precision-ranked rules, Comput. Linguist. (2013) 1–54.
[20] R. McCreadie, C. Macdonald, I. Ounis, M. Osborne, S. Petrovic, Scalable distributed event detection for twitter, in: Proceedings of IEEE International Conference on Big Data, 2013.
[21] P.N. Mendes, J, Daiber, M. Jakob, C. Bizer, Evaluating DBpedia spotlight for the TAC-KBP entity linking task, in: Proceedings of the TACKBP 2011 Workshop, 2011.
[22] L. Padró, E. Stanilovsky, Freeling 3.0: towards wider multilinguality, in: Proceedings of the Language Resources and Evaluation Conference (LREC 2012), ELRA, Istanbul, Turkey, 2012.
[23] P. Pantel, E. Crestan, A. Borkovsky, A.-M. Popescu, V. Vyas, Web-scale distributional similarity and entity set expansion, Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing, vol. 2, Association for Computational Linguistics, Stroudsburg, PA, USA, 2009, pp. 938–947.
[24] L. Ratinov, D. Roth, Design challenges and misconceptions in named entity recognition, in: Proceedings of the Thirteenth Conference on Computational Natural Language Learning, 2009, pp. 147–155.
[25] A. Ratnaparkhi, Learning to parse natural language with maximum entropy models, Mach. Learn. 34 (1999) 151–175.
[26] R. Saurí, J. Pustejovsky, FactBank: a corpus annotated with event factuality, Lang. Resour. Eval. 43 (2009) 227–268.
[27] S. Singh, A. Subramanya, F. Pereira, A. McCallum, Large-scale cross-document coreference using distributed inference and hierarchical models, in: Association for Computational Linguistics: Human Language Technologies (ACL HLT), 2011.
[28] M. Taulé, M.A. Martí, M. Recasens, AnCora: multilevel annotated corpora for Catalan and Spanish, in: LREC, 2008.
[29] K. Toutanova, D. Klein, C. Manning, Y. Singer, Feature-rich part-of-speech tagging with a cyclic dependency network, in: Proceedings of HLT-NAACL, 2003, pp. 252–259.
[30] H. Wu, Z. Fei, A. Dai, M. Sammons, D. Roth, S. Mayhew, Illinoiscloudnlp: text analytics services in the cloud, in: Proceedings of (LREC-2014), 2014.
[31] M. Zaharia, N.M.M. Chowdhury, M. Franklin, S. Shenker, I. Stoica, Spark: Cluster Computing with Working Sets. Technical Report EECS Department, University of California, Berkeley, 2010.