KOSHIK- A Large-scale Distributed Computing Framework for NLP

Peter Exner and Pierre Nugues
Department of Computer Science, Lund University, Lund, Sweden
{*peter.exner, pierre.nugues*}*@cs.lth.se*

Keywords: NLP Framework, Distributed Computing, Large Scale-processing, Hadoop, MapReduce.

Abstract: In this paper, we describe KOSHIK, an end-to-end framework to process the unstructured natural language content of multilingual documents. We used the Hadoop distributed computing infrastructure to build this framework as it enables KOSHIK to easily scale by adding inexpensive commodity hardware. We designed an annotation model that allows the processing algorithms to incrementally add layers of annotation without modifying the original document. We used the Avro binary format to serialize the documents. Avro is designed for Hadoop and allows other data warehousing tools to directly query the documents. This paper reports the implementation choices and details of the framework, the annotation model, the options for querying processed data, and the parsing results on the English and Swedish editions of Wikipedia.

1 INTRODUCTION

In recent years, the typical size of what we call a *large corpus* has grown from one million words to billions of tokens. Such amounts have transformed what we can expect from language technology. Google's knowledge graph (Singhal, 2012) and IBM Watson (Ferrucci, 2012) are two recent examples of what large-scale NLP made possible.

This *big data* era also means that most researchers in the field will have to collect, process, and interpret massive amounts of data. Although extremely promising, this also means that most NLP experiments or applications will no longer be able to rely on a single computer whatever its memory size or processor speed.

KOSHIK[1] is a framework for batch-oriented large scale-processing and querying of unstructured natural language documents. In particular, it builds on the Hadoop ecosystem and takes full advantage of the data formats and tools present in this environment to achieve its task.

Volume, velocity, and variety are three aspects commonly regarded as challenging when handling large amounts of data. KOSHIK tries to address these challenges by:

- Using Hadoop, an infrastructure that is horizontally scalable on inexpensive hardware.
- Having a batch-oriented annotation model that allows for incremental addition of annotations.
- Supporting a wide variety of algorithms (tokenizer, dependency parsers, coreference solver) for different input formats: text, CoNLL, and Wikipedia.

2 RELATED WORK

Early work on NLP frameworks has recognized the importance of component reuse, scalability, and application to real world data in unstructured information processing. Examples of frameworks include MULTEXT, GATE, and UIMA that were used in applications such as document retrieval, indexing, and querying of processed information.

MULTEXT (Ide and Véronis, 1994) adopted the principles of language independence, atomicity, input/output streams, and a unifying data type to create a system where tools can be reused and extended to solve larger and more complex tasks. MULTEXT stores the output from processing modules interleaved in the original document as SGML markup. In contrast, documents in Tipster II (Grishman et al., 1997) remain unchanged. The outputs from the processing modules are separately added as annotations and stored in a dedicated database.

[1]Hadoop-based projects are often named with an elephant or other animal theme. Following this tradition, we named our framework after an Asian elephant, KOSHIK, who can imitate human speech.

GATE (Bontcheva et al., 2004) is an architecture that builds on the work of Tipster by providing a unifying model of annotation that represents the data read and produced by all of the processing modules. Furthermore, GATE provides a uniform access to algorithmic resources (tools, programs, or libraries) through an API. GATECloud (Tablan et al., 2013) acts as a layer for the GATE infrastructure on top of multiple processing servers and provides a parallel analysis of documents.

UIMA (Ferrucci and Lally, 2004) is an infrastructure designed for the analysis of unstructured documents (text, speech, audio, and video). It has components for reading documents, performing analysis, writing to databases or files, and a configurable pipeline. The annotation model in UIMA is based on a hierarchical type system defining the structure of annotations associated with documents. UIMA is compatible with a large set of external NLP tools. These include OpenNLP[2], DKPro Core (Gurevych and Müller, 2008), and JULIE Lab[3]. Scalability of the processing in UIMA is offered through UIMA Asynchronous Scaleout (UIMA AS) and Behemoth[4] for processing within Hadoop.

3 KOSHIK OUTLINE

Rather than creating a new framework for parallel computation, such as UIMA AS or GATECloud, we designed KOSHIK from the ground-up for the Hadoop environment using Hadoop-compatible tools and data formats. One advantage of this approach lies in the ability to reuse other text and NLP processing tools in Hadoop, such as Cloud9[5], for further processing.

Document processing is implemented as MapReduce jobs, that through Hadoop allow for horizontal scaling of computational power. KOSHIK supports a full pipeline of NLP multilingual tools including prefilters, tokenizers, syntactic and semantic dependency parsers, and coreference solvers. To our knowledge, this is the first framework to support a full pipeline with a semantic layer in the Hadoop environment.

KOSHIK's annotation model resembles that of ATLAS (Laprun et al., 2002) and Tipster II, which it extends to support the variety of output models from the processing tools. Data serialization of documents and annotation is made using Avro[6], a binary language-independent serialization format. Avro allows the serialization of complex structures to data files that can be queried through other Hadoop tools, most notably Hive (Thusoo et al., 2009) and Pig (Olston et al., 2008).

The rest of the paper is structured as follows. We introduce the KOSHIK architecture in Sect. 4 and we discuss how we chose a distributed computing platform. We outline KOSHIK's implementation. Section 5 gives an overview of the annotation model and shows how we represent different document structures and annotations from the parsers. We discuss how annotated data can be queried using the tools in Hadoop in Sect. 6. In Sect. 7, we discuss applications of our framework and give an overview of results from parsing the English and Swedish editions of Wikipedia. Finally, we conclude with a discussion in Sect. 8.

4 ARCHITECTURE

KOSHIK supports a variety of NLP tools implemented atop of the Hadoop distributed computing framework. The requirements on KOSHIK were driven by the desire to support scalability, reliability, and a large number of input formats and processing tasks. Figure 1 shows an overview of the architecture. The following sections detail the implementation choices and engineering details.

4.1 Distributed Computing Framework

The design of a distributed computing framework has a critical influence in the processing speed of large corpora. At first hand, distributed processing consists in sharing the processing of documents over multiple computing nodes. This involves among other things the division and distribution of a collection of documents, scheduling of computing tasks, and retrieval of computing outputs. At the very core, the nature of this task is communication and coordination oriented.

We first experimented with the Message Passing Interface (MPI) communications protocol. MPI allows a program in a distributed environment to share information and coordinate in a parallel task by providing communication and synchronization functionalities between processes. While using MPI solved the problem of distributing computational power, many other factors such as reliability caused by hardware failures and rescheduling of failed tasks remained unsolved.

After the initial attempt with MPI, we built KOSHIK to run on Hadoop (White, 2012). Hadoop is

[2]http://opennlp.apache.org/
[3]http://www.julielab.de/Resources/Software/NLP_Tools.html
[4]https://github.com/DigitalPebble/behemoth
[5]http://lintool.github.io/Cloud9/
[6]http://avro.apache.org/

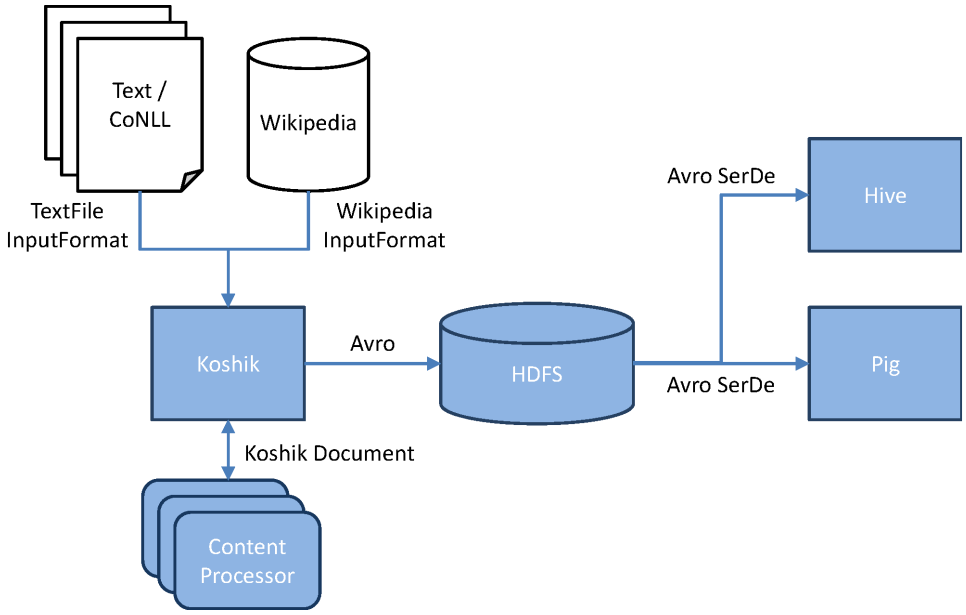

Figure 1: An overview of the KOSHIK architecture. Input from text, CoNLL, and Wikipedia dump XML files are imported using a corresponding InputFormat. The imported documents are analyzed using a pipeline of content processors and serialized to Avro binary format. The serialized documents can be queried using Hive or Pig.

an infrastructure for running large-scale data processing tasks in a distributed environment. It offers a programming model, MapReduce (Dean and Ghemawat, 2008), through which processing workflows, referred to as MapReduce jobs, can be expressed. Using this model, Hadoop abstracts implementation details such as network communication and reading/writing to disk. Its execution engine automatically schedules jobs, performs load balancing, monitors and reruns failed tasks. Data stored on the Hadoop Distributed File System (HDFS) is replicated across nodes for reliable storage. This also has the benefit of offering data locality for processing task, thereby lowering the network traffic between nodes.

While Hadoop has native support for Java and support for many high-level programming languages (Python, Ruby etc.) through Hadoop streaming, processing of data is further simplified by tools that remove the necessity to write MapReduce jobs; the most notable ones being Pig and Hive. Pig has a scripting language, called Pig Latin, that can express workflows which read and transform large data-sets. Hive offers an SQL-like language, called HiveQL, to express queries that are transformed into MapReduce jobs. These tools leverage the power of querying over distributed datasets while offering a simplified querying language familiar to RDBMS analysts.

Hadoop scales horizontally over inexpensive off-the-shelf hardware and is offered in many distributions: Cloudera[7], Hortonworks[8], MapR[9], and many more. It can also be run on a computing cloud services such as Amazon EC2 (Elastic Compute Cloud)[10].

4.2 MapReduce

Designed by Google, MapReduce is a programming model inspired by the map and reduce primitives present in Lisp and other functional programming languages. Users specify a map and a reduce function that both receive and output key value pairs. Map functions receive key value pairs based on the input data submitted with a MapReduce job. The key value pairs output from mappers are sorted by the keys and partitioned into groups that are sent as input to reducers. Output from reducers are written to the HDFS filesystem in Hadoop.

While the number of map tasks is governed by how Hadoop splits the input data, the number of reduce tasks can be explicitly specified by the user. This knowledge has guided our implementation of Koshik as we have chosen to place all processing in reduce tasks. In doing so, we allow the user to retain control over the number of simultaneous tasks running on each node. This is an advantage especially when

[7] http://www.cloudera.com/
[8] http://hortonworks.com/
[9] http://www.mapr.com/
[10] http://aws.amazon.com/ec2/

an algorithm is performing a memory intensive computation that cannot be divided into more fine grained tasks. Typically, these algorithms can be found in syntactic and semantic dependency parsers that require large training models.

4.3 Processing Pipeline

KOSHIK currently supports the input of data from regular text files, CoNLL-X (Buchholz and Marsi, 2006), CoNLL 2008 and 2009 (Surdeanu et al., 2008), and Wikipedia dump files. These are converted by map tasks into KOSHIK documents. Processing of documents is done by specifying a pipeline of annotators, called content processors. Through pipelines, processing can be expressed as a linear workflow. Each content processor implements a simple interface that specifies one process function that takes a document as input, enriches it by adding layers of annotations, and outputs the document. Thus, integrating an NLP tool into KOSHIK is performed by including the library of the tool and implementing a process method. The implementation of the process method is aided by the annotation model, described in Section 5, which provides a set of objects representing the common output from tokenizers, syntactic and semantic parsers, and coreference solvers. By keeping the interface lightweight and the annotation model simple, we believe that the barrier for porting tools from other toolkits to KOSHIK is lowered.

KOSHIK currently supports a wide range of filters, tokenizers, taggers, parsers, and coreference solvers for a wide number of languages. Multilinguality is provided by each NLP tool through a language specific model. The supported tools include:

- Filtering of Wiki markup[11].
- OpenNLP, sentence detector and tokenizer.
- Mate Tools, part-of-speech tagger, lemmatizer, syntactic and semantic dependency parser (Björkelund et al., 2009; Bohnet, 2010).
- Stanford CoreNLP, including named entity tagger (Finkel et al., 2005), syntactic parser (Klein and Manning, 2003) and coreference solver (Lee et al., 2011).
- Stagger (Östling, 2012), a part-of-speech tagger for Swedish.
- MaltParser (Nivre et al., 2007), a dependency parser.

[11] http://en.wikipedia.org/wiki/Wikimarkup/

5 ANNOTATION MODEL

In many ways, the core of our framework lies in the annotation model. With this model, the content processors only need to care about the input and output of annotated documents. This allows for the free interchange of content processors in a pipeline. Therefore, it is important that the annotation model is designed to support a wide variety of document structures and output from content processors, such as taggers and parsers. It is also essential to create a schema structure such that annotations can easily be queried once serialized.

Our annotation model is similar to ATLAS and Tipster II in that we associate the regions of the original document with metadata. Rather than interleaving annotation such as in XML, we append layers of annotations to documents. In this way, we separate the annotations and leave the original document unmodified. This approach makes it possible the incremental addition of information where content processors increasingly enrich documents by adding layers of annotation. It also supports a pipeline where content processors can be mixed-and-matched as each content processor finds the layer of annotation needed for its algorithm.

Unlike UIMA, we focus our analysis on text documents. This restriction makes the development of our content processors simpler since they only have to handle one type of document and can make assumptions about the document and annotation features. Furthermore, serialization and subsequent querying of processed data is also simplified since it becomes possible to determine the expected document and annotation attributes.

The base of the annotation model is represented by a document and an annotation type. This model is then extended by subtypes to represent tokens, edges, spans, and other features. Figure 2 shows an overview of the annotation model and Figure 3 shows how the model can be used to annotate a document.

The following subsections discuss the annotation model and how they can represent output of morphosyntactic, semantic, and discourse analysis.

5.1 Documents

The KOSHIK document model has attributes to preserve the original content together with the version, language, source, and indexing. In addition, the model supports a number of metadata descriptors. Each document has a set of associated annotations attached.

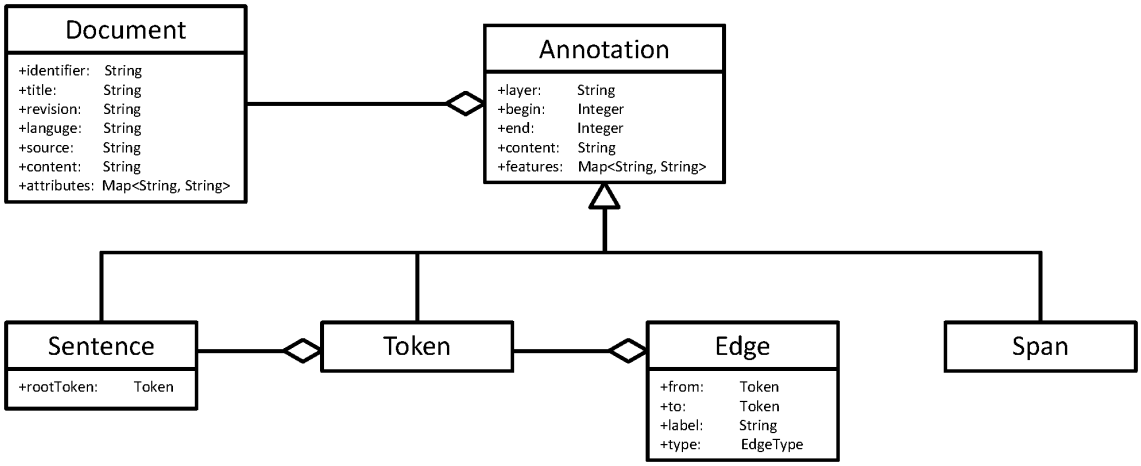

Figure 2: KOSHIK uses an annotation model for representing document metadata, structure and other annotations from parsers.

5.2 Annotations

Annotations associate the region of a document with some metadata. Regions are unidimensional and are bounded by a beginning and an end. Metadata can either consist of a string or represent multiple features, as key-value pairs in a dictionary. By restricting the structure and values of metadata, serialization and subsequent querying of data becomes simplified. As an example, using Hive, the dictionary holding the metadata can easily be decomposed into rows, one for each dictionary entry. As complex relational joins are avoided, this simplifies the expression of queries.

5.3 Tokens

The token type represents the tokens in a sentence. Each token has fields to represent morphosyntactic features such as form, part-of-speech, and lemma. All fields from the CoNLL-X and CoNLL 2008 shared task data format are available as default. In addition, the token type can be extended with metadata to store parser precision as a feature in the annotation metadata dictionary.

5.4 Spans

Spans are used to model mentions in coreference chains, named entities, and other entities that span over several tokens. Spans can also model the output from shallow parsers such as syntactic chunkers.

5.5 Edges

Typically, edges model relations between tokens resulting from syntactic and semantic dependency parsers. Edges are saved as features of tokens and rebuilt during deserialization.

5.6 Serialization

All documents and annotations in KOSHIK are serialized in order to retain structure and to be stored in a compact format. The fields of both simple and complex types, such as strings, integers, and maps, are directly mapped to corresponding types in Avro. Our choice to serialize to Avro was governed by the fact that many tools in the Hadoop environment have built-in capabilities to deserialize and read the Avro binary format. For instance, Hive can directly query complex structures such as arrays and maps using an Avro Serializer and Deserializer (SerDe).

6 QUERYING ANNOTATED DATA

Many analyses require querying parsed data, be it on a specific document or a collection of documents. Such queries may range from counting the number of tokens to calculating the number of arguments for a certain predicate. One possible approach to querying data in Hadoop, is to implement a query by writing a MapReduce job in a programming language. However, implementing even the most trivial query might represent a technical hurdle to many. Even so, advanced queries become more complicated as they need to be written as flows of multiple MapReduce jobs. To overcome this problem and to offer interactive analysis of data, Hive and Pig offer simple yet powerful query languages by abstracting MapReduce jobs.

Ideally, the querying should be performed directly

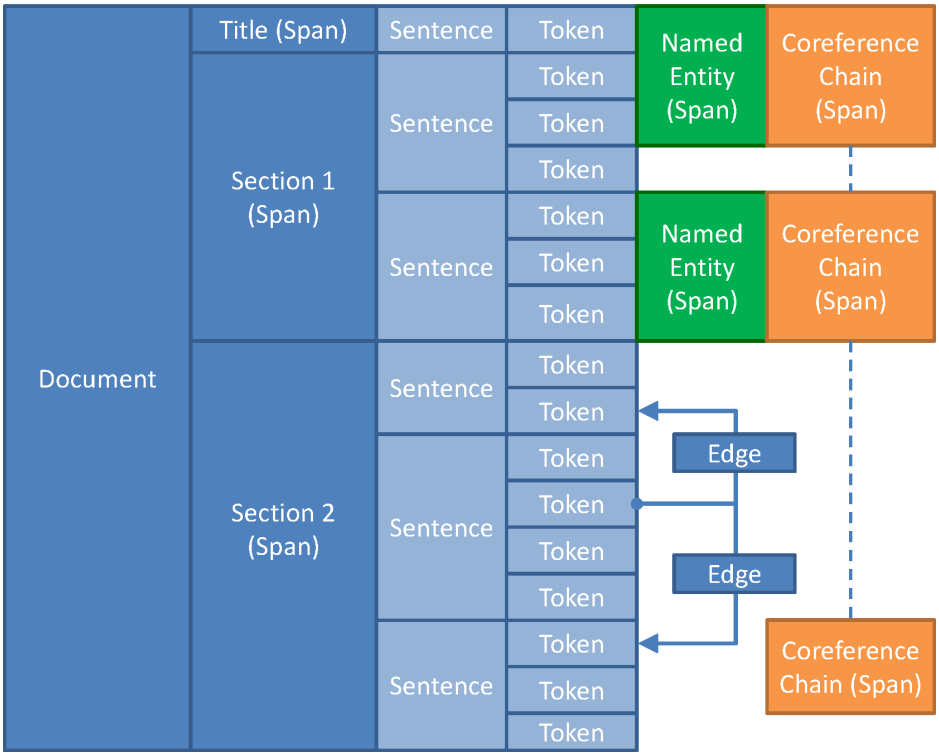

Figure 3: Applying the KOSHIK annotation type to a document structure. The figure shows the various annotation layers added by a pipeline of NLP tools to a text document. An annotation type associates a region of the document with some type of metadata. This metadata can be a part of the original text itself or, as in the case of the token type, it can be a part-of-speech tag, lemma, etc.

on the data output from KOSHIK, without any need of transformation or duplication of the data. This means that if a large amount of data is analyzed, it becomes unfeasible and unscalable to offload it as a whole into another cluster or data warehouse. In essence, such an action would duplicate the data. To avoid unnecessary duplication of data and to query it using tools within the Hadoop environment, KOSHIK serializes data to the Avro format. By doing so, Hive is able to directly query the data from KOSHIK without creating any unnecessary duplication.

Hive offers an SQL-like query language called HiveQL. Both simple types, integers, strings, etc., and complex ones, structs, maps, arrays, are supported by the type system in Hive. HiveQL supports primitive operations from relational algebra including projections, selection, and joins. More complicated queries are made possible by creating Hive User-Defined Functions (UDF). As an example, the dictionary of tokens holding morphosyntactic information, part-of-speech, form, lemma, etc., are easily decomposed into separate rows using the `explode()` UDF in Hive and allows for the querying of the distribution of part-of-speech tags.

7 APPLICATIONS AND RESULTS

The typical scenario for using KOSHIK consists of the following steps:

1. Import of corpora to the Koshik document model.
2. Analysis of documents using a NLP pipeline.
3. Querying or conversion of annotated documents to a desired output format.

To evaluate KOSHIK, we constructed a component pipeline to extract predicate–argument structures from the English edition of Wikipedia and solve coreferences. In addition, we present the results of a syntactic dependency analysis of the Swedish edition. The experiment was performed on 12-node Hadoop cluster; each node consisting of a PC equipped with a 6-core CPU and 32GB of RAM.

For the English semantic analysis, KOSHIK uses a state-of-the-art graph-based dependency parser (Björkelund et al., 2009; Bohnet, 2010). KOSHIK uses the Stanford CoreNLP multi-pass sieve coreference resolver to resolve anaphoric expressions (Lee et al., 2011). For Swedish, we used the Stagger part-of-speech tagger (Östling, 2012) and the MaltParser

dependency parser (Nivre et al., 2007). In Table 1, we show the results from the English analysis and in Table 2, the results from the Swedish analysis.

Table 1: English Wikipedia statistics, gathered from the semantic and coreference analyses.

Corpus size	7.6 GB
Articles	4,012,291
Sentences	61,265,766
Tokens	1,485,951,256
Predicates	272,403,215
Named entities	148,888,487
Coreference chains	236,958,381
Processing time	462 hours

Table 2: Swedish Wikipedia statistics, gathered from the syntactic analysis.

Corpus size	1 GB
Articles	976,008
Sentences	6,752,311
Tokens	142,495,587
Processing time	14 minutes

8 CONCLUSIONS

In this paper, we have described a framework, KOSHIK, for end-to-end parsing and querying of documents containing unstructured natural language. Koshik uses an annotation model that supports a large set of NLP tools including prefilters, tokenizer, named entity taggers, syntactic and semantic dependency parsers, and coreference solvers. Using the framework, we complete the semantic parsing of the English edition of Wikipedia in less than 20 days and the syntactic parsing of the Swedish one in less than 15 minutes. The source code for Koshik is available for download at https://github.com/peterexner/KOSHIK/.

9 FUTURE WORK

While many high precision NLP tools exist for the analysis of English, resources for creating tools for other languages are more scarce. Our aim is to useKoshik and create parallel corpora in English and Swedish. By annotating the English corpus semantically and the Swedish corpus syntactically, we hope to find syntactic level features that may aid us in training a Swedish semantic parser. We will also continue to expand the number and variety of tools and the language models offered by Koshik.

ACKNOWLEDGEMENTS

This research was supported by Vetenskapsrådet, the Swedish research council, under grant 621-2010-4800 and has received funding from the European Union's seventh framework program (FP7/2007-2013) under grant agreement 230902.

REFERENCES

Björkelund, A., Hafdell, L., and Nugues, P. (2009). Multilingual semantic role labeling. In *Proceedings of CoNLL-2009*, pages 43–48, Boulder.

Bohnet, B. (2010). Very high accuracy and fast dependency parsing is not a contradiction. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 89–97. Association for Computational Linguistics.

Bontcheva, K., Tablan, V., Maynard, D., and Cunningham, H. (2004). Evolving gate to meet new challenges in language engineering. *Natural Language Engineering*, 10(3-4):349–373.

Buchholz, S. and Marsi, E. (2006). Conll-x shared task on multilingual dependency parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*, pages 149–164. Association for Computational Linguistics.

Dean, J. and Ghemawat, S. (2008). Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113.

Ferrucci, D. and Lally, A. (2004). Uima: an architectural approach to unstructured information processing in the corporate research environment. *Natural Language Engineering*, 10(3-4):327–348.

Ferrucci, D. A. (2012). Introduction to "This is Watson". *IBM Journal of Research and Development*, 56(3.4):1:1 –1:15.

Finkel, J. R., Grenager, T., and Manning, C. (2005). Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 363–370. Association for Computational Linguistics.

Grishman, R., Caid, B., Callan, J., Conley, J., Corbin, H., Cowie, J., DiBella, K., Jacobs, P., Mettler, M., Ogden, B., et al. (1997). Tipster text phase ii architecture design version 2.1 p 19 june 1996.

Gurevych, I. and Müller, M.-C. (2008). Information extraction with the darmstadt knowledge processing software repository (extended abstract). In *Proceedings of the Workshop on Linguistic Processing Pipelines*, Darmstadt, Germany. No printed proceedings available.

Ide, N. and Véronis, J. (1994). Multext: Multilingual text tools and corpora. In *Proceedings of the 15th conference on Computational linguistics-Volume 1*, pages 588–592. Association for Computational Linguistics.

Klein, D. and Manning, C. D. (2003). Accurate unlexicalized parsing. In *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1*, pages 423–430. Association for Computational Linguistics.

Laprun, C., Fiscus, J., Garofolo, J., and Pajot, S. (2002). A practical introduction to atlas. In *Proc. of the 3rd LREC Conference*, pages 1928–1932.

Lee, H., Peirsman, Y., Chang, A., Chambers, N., Surdeanu, M., and Jurafsky, D. (2011). Stanford's multi-pass sieve coreference resolution system at the conll-2011 shared task. In *Proceedings of the Fifteenth Conference on Computational Natural Language Learning: Shared Task*, pages 28–34. Association for Computational Linguistics.

Nivre, J., Hall, J., Nilsson, J., Chanev, A., Eryigit, G., Kubler, S., Marinov, S., and Marsi, E. (2007). Maltparser: A language-independent system for data-driven dependency parsing. *Natural Language Engineering*, 13(2):95.

Olston, C., Reed, B., Srivastava, U., Kumar, R., and Tomkins, A. (2008). Pig latin: a not-so-foreign language for data processing. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1099–1110. ACM.

Östling, R. (2012). Stagger: A modern pos tagger for swedish. In *The Fourth Swedish Language Technology Conference*.

Singhal, A. (2012). Introducing the knowledge graph: things, not strings. Official Google Blog.

Surdeanu, M., Johansson, R., Meyers, A., Màrquez, L., and Nivre, J. (2008). The CoNLL 2008 shared task on joint parsing of syntactic and semantic dependencies. In *CoNLL 2008: Proceedings of the 12th Conference on Computational Natural Language Learning*, pages 159–177, Manchester.

Tablan, V., Roberts, I., Cunningham, H., and Bontcheva, K. (2013). Gatecloud. net: a platform for large-scale, open-source text processing on the cloud. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 371(1983).

Thusoo, A., Sarma, J. S., Jain, N., Shao, Z., Chakka, P., Anthony, S., Liu, H., Wyckoff, P., and Murthy, R. (2009). Hive: a warehousing solution over a map-reduce framework. *Proceedings of the VLDB Endowment*, 2(2):1626–1629.

White, T. (2012). *Hadoop: The definitive guide*. O'Reilly Media, Inc.