BISHOP – Big Data Driven Self-Learning Support for High-performance Ontology Population

Daniel Knoell[1], Martin Atzmueller[2], Constantin Rieder[1], and Klaus Peter Scherer[1]

[1] Karlsruhe Institute of Technology
D-76344, Eggenstein-Leopoldshafen, Germany
firstname.lastname@kit.edu

[2] University of Kassel, Research Center for Information System Design
Wilhelmshöher Allee 73, 34121 Kassel, Germany
atzmueller@cs.uni-kassel.de

Abstract. Self-learning support systems are already being successfully used to support sophisticated processes. For the widespread industrial use, there are still challenges in terms of accessibility with respect to the process and the scalability in the context of large amounts of data.
This paper provides an example-driven view on the Bishop project for Big Data driven self-learning support for high-performance ontology population. We outline workflows, components and use cases in the context of the project and discuss methodological as well as implementation issues.

1 Introduction

Linked Enterprise Data requires the effective and efficient learning of ontologies. Typically, only large data sources provide the means for obtaining results with sufficient quality. Therefore, methods that work at large-scale are necessary, e. g., using high performance methods, resulting in increasing efforts concerning Big Data processing and management. In addition, typically specialized infrastructure needs to be set-up and configured, which is usually complicated and costly. Therefore, both accessibility and scalability of the applied methods and techniques need to be increased.

This paper presents the Bishop project that addresses these issues in order to provide a systematic approach towards large-scale self-learning support systems. We present an example-driven approach on the project and discuss specific workflows, components and use cases supported by appropriate tools. Hence, the remainder of the paper is structured as follows: We first provide provide an overview on the Bishop project, putting it into the context of related work, and discuss exemplary workflows and components in Section 2, before we present a set of of use cases that are elaborated in a requirements engineering step in order to identify first measures and process forces. Overall, These use cases are used as a reference for the different architectural variants, e. g., in the context of natural language processing methods for self-learning from texts. Furthermore, we discuss suitable tool support in that context. Finally, we conclude with a summary and outlook on further steps in Section 3.

Preprint of: Daniel Knoell, Martin Atzmueller, Constantin Rieder, and Klaus-Peter Scherer (2016) BISHOP – Big Data Driven Self-Learning Support for High-performance Ontology Population. Proc. LWDA 2016 (FGWM Track), Potsdam, Germany. (In Press).

2 BISHOP by Example: Workflow and Components

BISHOP is part of the APOSTLE project, which is the acronym for "Accessible Performant Ontology Supported Text Learning". While learning ontologies from text is not a novel approach and is e. g., used to learn the concept hierarchy out of web data [10], the Bishop project tackles the efficient and effective self-learning of ontologies for large data. The TELESUP Project [9], for example also deals with the automatic ontology population by using textual data, however, it does not consider Big Data.

In a first step, a conceptual framework is derived from the requirements that captures the decisions for integrating self learning methods into high performance environments. For that, different Big Data frameworks like Map/Reduce (Hadoop), Spark and Flink need to be investigated, in order to estimate the performance in the scope of the targeted data. Then a test scenario for the comparison of the results will be defined. After the set-up of the big data infrastructure, it will be evaluated with different persistence strategies. In parallel, it is necessary to find an easy way for the set-up of the big data environment and the deploying of existing Java applications. An additional parallel task is to find and efficient way for storing and querying huge amounts of semantic structures. Here, also intelligent mechanisms for persistence, distribution and parallelization will be devised.

By optimizing the accessibility and scalability, significant efficiency improvements in technical services for the creation of self-learning systems, such as expert systems and knowledge-based support systems are enabled.

2.1 Exemplary Workflows

The project consists of different parts which lead to different workflows. These workflows are processed in parallel and are described in the following subsections.

Calculating a Thesaurus The automatic generation of a thesaurus requires the steps, described in Figure 1.

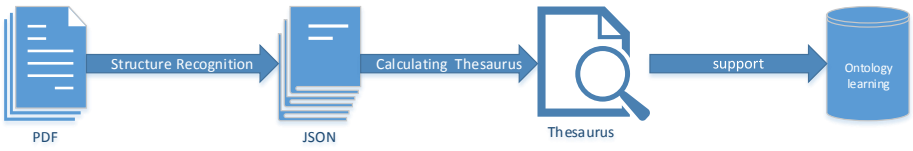

Fig. 1: Workflow: Calculating a Thesaurus

The data format in industry is often PDF. So in the first step the PDF files need to be converted to a more structured format like JSON. This is an important step, which is also necessary for other areas of the processing and is detailed in section 2.3. The JSON files serve as input for the application which calculates the thesaurus. A possible application for this task could be "JoBimText" [15]. A further description is given in section 2.3. Finally, as last step, the thesaurus can be used for ontology learning.

Easy Deployment The set up of an **infrastructure** to fulfill necessary tasks can be very difficult and time consuming. Furthermore it should be noted that each user has a different work equipment. Therefore, an easy deployment of such an infrastructure is needed to start the data processing quickly and platform independent keeping the frustration level low by avoiding installing, configuring and setting up activities.

A modern technology facing these restrictions could be a **container based approach** of deployment. One possible solution considering these limitations could be the open-source project Docker that provides suitable features by deploying applications inside software containers. The so called docker images are providing the applications which are running in the docker containers and accelerating the distribution and deploying efforts. By deploying a ready to start configuration with a preset environment and set of applications the expectation is a more user friendly set up process that allows a quick start.

In addition to the prepared configurations and on the basis of the above a further important step is to design a set of conventions to reduce the complexity of mandatory configurations. One possible solution could be the design of an appropriate configuration and **set up wizard** that guides the user through the complex processes. This kind of support could be a helpful extension because it has been in use for decades (e. g. classic installer wizards) and has proven its worth.

A second point of the easy deployment is how to get an existing **Java application** running on the big data environment, see Figure 2. Here appropriate conversion methodologies need to be developed.

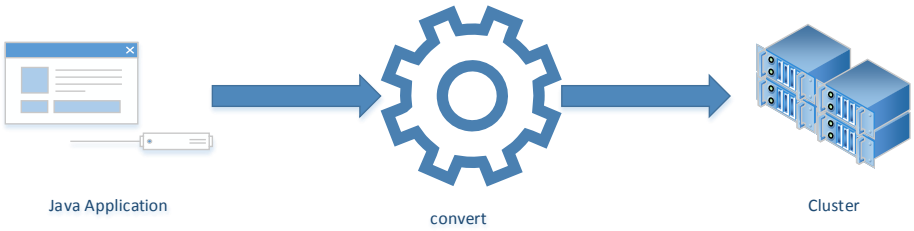

Fig. 2: Workflow: deploying a single computer application on a cluster

Storing and Querying Semantic Structures According to the current state of the art, the management of huge amounts of semantic data (ontologies) is still inefficient. For larger amounts of data the current solution is to merely use larger amounts of main memory. However, if the limit of the currently used memory exceeds the volume of the semantic data, there is at the moment no effective solution. Therefore, an problem-solving approach, which can handle the requirements of huge ontologies is necessary. An approach is intended, which retains the advantages of current solutions as well as possible and fixes the weaknesses in dealing with large amounts of data. For this, innovative methods needs to be developed and integrated into the overall approach.

2.2 Components

In Figure 3 the components of the whole project are illustrated. Via *Business Services* all components of the system are loosely connected. This also includes the *Self-learning Methods* which are evaluated under the aspects of parallelizability and scalability. These methods can either be in the field of text learning [6], for example NLP, and in the field of learning with structured data. After the evaluation follows a review and then based on the results an adaptation.

Various forms of parallelization are implemented and evaluated. The component *Persistence* allows the permanent storage of the documents. The aim is to develop a library which offers different options how to store the documents. It detects depending on the required scaling and the used system which persistence method is to be used. In the the first step the documents are stored on the local file system. After that the implementation of additional storage capabilities, such as the Hadoop Distributed File System (HDFS) or MapR-DB, is done. These can be automatically selected when storing large amounts of data.

Therefore the scalability of the infrastructure has to be evaluated. The results of the evaluation allow the construction (or potentially refinement) of rules for decision-making for the storage strategy used. Parameters such as size and type the data are also considered. The last component is the *Ontology Proxy* which enables the storing and accessing of the semantic representation of the documents. Therefore various existing solutions are evaluated and adjusted substantially or completely redeveloped. Furthermore, it is examined whether techniques from the database environment can be applied and whether these yield performance improvements.

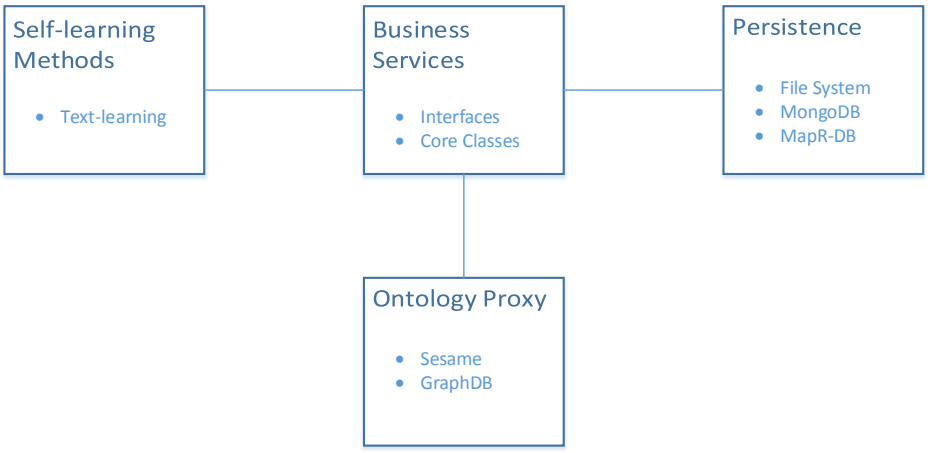

Fig. 3: Components

2.3 Use Cases

This section outlines two use cases in the context of the Bishop project concerning basic techniques for learning from texts, i. e., structure recognition and calculating a Thesaurus. After that, we discuss options for tool support in that scope, considering potential Big Data processing and management methods in the context of processing unstructured, i. e., textual information.

Use Case 1: Support Structure Recognition in PDF Files A common problem in the enterprise environment is, that important data is only available as PDF-Files. It is easy to get content of the PDF-Files as plain text, but that is usually not that helpful, because the structure of the documents gets lost. Without the structure, there is no way to find out if a specific phrase in the document was a heading, a headline, a footnote or even a caption. In the most cases this information is extremely important for the further processing. The recognition of the structure of a PDF-File is a difficult task which is very time consuming and only a few applications are good at it. In combination with a huge amount of PDF-Files, like it occurs for example in the field of the technical documentation, there is a long time waiting for results.

In order to decrease the overall processing time time, it is useful to distribute the application for example on a High Performance Cluster. There are at least two ways for the distribution. The first way is to process every PDF-File on a different node in the cluster. This is expected to be a good solution if there are a lot of files, which are not that big. If there are only a few, but huge PDF-Files it can be helpful to split the files into many parts and distribute the parts of the files. This would be the second way. It has to be evaluated, if the two ways perform as expected, to know which way fits to the correlated case. The optimal behaviour of the resulting application would be, that it picks the right way of distribution, depending on the dataset.

Use Case 2: Calculating a Thesaurus A domain-specific thesaurus can have huge advantages on the task of ontology learning. Especially on the lower layers of the ontology learning layer cake [7], like terms and synonyms layer, it can be useful and improve the results drastically. The problem is, that for the most domains, there is no suitable domain-specific thesaurus available. Furthermore, building up a thesaurus is a time consuming task which needs the involvement and knowledge of experts. However, the time of the experts is typically rare and expensive. These two facts make the manual construction of an domain- specific thesaurus difficult. An automation of this task is also difficult and needs a lot of domain specific documents.

There are approaches like “JoBimText” [15] which can calculate a thesaurus out of huge corpora, but do not take advantage of the structure of the documents. This could have an enormous impact on the quality of the results. The calculation of the thesaurus should also be distributed, because of the huge amount of documents, which need to be processed. Otherwise it would take to much time for the industrial usage.

2.4 Tool Support

According to the *four V* criteria [11] (i. e., velocity, volume, variety, and veracity), big data requires efficient methods to handle the rapidly incoming data with appropriate response time (velocity), the large number of data points (volume), many different heterogeneously structured data sources (variety), and data sources with different quality and provenance standards (veracity). In the context of unstructured and semi-structured data, several challenges have to be addressed, such as information extraction [1] for textual data, as well as integration techniques for the comprehensive set of data sources. For semi-structured data, e. g., rule-based methods [3] or case-based reasoning systems [5] can often be successfully applied. According modeling and indexing techniques can be implemented, e. g., using the Map/Reduce framework [8], see below.

Before starting with a data processing framework, different questions and requirements need to be clarified, e. g., according to the types, structure and accuracy of data that is to be implemented, cf. [12]. We focus on according tools for Big Data processing, analytics, and management in the following.

Lambda Architecture According to Marz and Warren [14], system properties of a Big data system typically exhibit the following system properties: They should provide a *general* data framework that is *extensible*, enables *ad-hoc queries* with *minimal maintenance*, and *debugging capabilities*. For data storage, this implies mechanisms for handling the *complexity* of data, e. g., for preventing corruption issues and maintenance issues. Further, *robustness and fault-tolerance* should be enforced, as well as *low latency reads and updates*. This also points to *scalability* issues concerning horizontal and vertical scalability, and the option of obtaining *intermediate results* and views, according to some concept of reproducibility.

The lambda architecture incorporates these system principles and especially tackles the concept of reproducibility of results and views for dynamic processing. Essentially, it allows to compute arbitrary functions on arbitrary datasets in real-time [14]. The lambda architecture is structured into several layers briefly, summarized as follows:

- Batch layer: continuously (re-)computes batch views using immutable data records.
- Serving layer: indexes query view, performs updates, and provides access to the dataset. Only batch updates and random reads are supported, no (distributed) writes.
- Speed layer: high-latency updates; fix batch layer lag; needs fast algorithms for incremental updates.
- Complexity isolation: random writes only need to be supported in speed layer. Results are then merged with the precomputed data from the batch layer.

Map/Reduce Map/Reduce[8] is a paradigm for scalable distributed processing of big data, that can be utilized for implementing, e. g., the batch layer. Its core ideas are based on the functional programming primitives *map* and *reduce*. Whereas *map* iterates on a certain input sequence of key-value pairs, the *reduce* function collects and processes all values for a certain key. The Map/Reduce paradigm is applicable for a computational task, if it can be divided into *independent* subtasks, such that there is no required communication between these. Then, large tasks can be split up into subtasks according to a typical divide-and-conquer strategy, e. g., for local exceptionality detection [4].

Map/Reduce is a powerful paradigm for processing big data – with a prominent implementation given by the *Hadoop* framework[3] supported by the HDFS filesystem, and big data databases such as Hive[4] and HBase[5]. Map/Reduce tasks can also be utilized for batch processing in the Lambda architecture discussed above, such that continuous views are (re-)computed by the respective Map/Reduce jobs. These batch tasks can then be complemented by tools for distributed realtime computation like the Storm framework[6], or the Flink[7] platform. This allows a comprehensive data processing pipeline for big data in the Lambda architecture, combining realtime together with Map/Reduce techniques. Alternatives to Map/Reduce, especially considering *in-memory computation* with large datasets include, for example, the Spark[8] [17] and Flink platforms.

Big Data Management NoSQL Databases (Not Only SQL) offer high performance and high availability [16], if no ACID (Atomic, Consistent, Isolated, Durable) transactions are needed. These databases perfectly fit in our Big Data environment. In our case, we use JSON files, which should be stored in the database. A lot of document based NoSQL databases use this format to store the data on the filesystem. So it is quite simple to use a document based database like MongoDB[9], Apache CouchDB[10] or MapR-DB[11]. MongoDB and Apache CouchDB have own solutions for the distribution of the database. MapR-DB is a In-Hadoop NoSQL database that supports JSON document models and wide column data models and can be run in the same cluster as Apache Hadoop and Apache Spark. This has the benefit of an easy integration in the big data environment, which will contain Hadoop and/or Spark. The architecture is shown in Figure 4. The *Databases* and the *File Systems* are connected to the *Interface Layer*, which enables the access of the *Frontend* and the *Big Data Framework*.

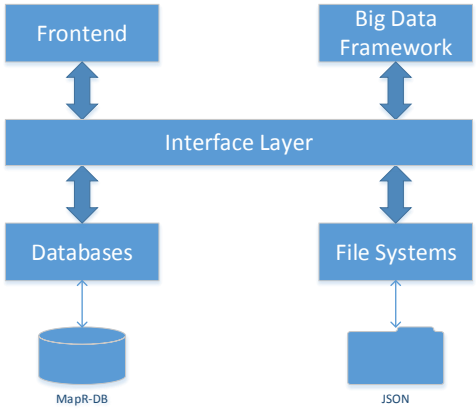

Fig. 4: Big Data Management Architecture based on MapR-DB.

[3] http://hadoop.apache.org/
[4] http://hive.apache.org/
[5] http://hbase.apache.org/
[6] http://storm.apache.org/
[7] http://flink.apache.org/
[8] http://spark.apache.org/
[9] https://www.mongodb.com/
[10] http://couchdb.apache.org/
[11] https://www.mapr.com/products/mapr-db-in-hadoop-nosql

3 Conclusions

This paper presented the Bishop project that investigates methods for Big Data driven large-scale self-learning support for high-performance ontology population. In an example-driven approach we discussed workflows, components, use cases, and tools.

For future work, we will investigate the proposed Big Data frameworks and develop corresponding data processing and analytics methods, also aiming at a methodology for easy cluster set up. Other interesting future directions are given by efficient (distributed) information extraction, e. g., [13] and refinement methods, e. g., [2], for advancing high-performance approaches for ontology population using self-learning support systems.

Acknowledgements. The work described in this paper is funded by grant ZIM-KOOP ZF4170601BZ5 by the German Federal Ministry of Economics and Technology (BMWI).

References

1. Adrian, B.: Information Extraction on the Semantic Web. Ph.D. thesis, DFKI (2012)
2. Atzmueller, M., Baumeister, J., Puppe, F.: Introspective Subgroup Analysis for Interactive Knowledge Refinement. In: Proc. FLAIRS. pp. 402–407. AAAI, Palo Alto, CA, USA (2006)
3. Atzmueller, M., Kluegl, P., Puppe, F.: Rule-Based Information Extraction for Structured Data Acquisition using TextMarker. In: Proc. LWA. University of Würzburg, Germany (2008)
4. Atzmueller, M., Mollenhauer, D., Schmidt, A.: Big Data Analytics Using Local Exceptionality Detection. In: Enterprise Big Data Engineering, Analytics, and Management. IGI Global, Hershey, PA, USA (2016)
5. Bach, K.: Knowledge Acquisition for Case-Based Reasoning Systems. Ph.D. thesis, Dr. Hut Verlag München (2012)
6. Buitelaar, P., Cimiano, P., Magnini, B.: Ontology Learning from Text: Methods, Evaluation and Applications. IOS Press (2005)
7. Cimiano, P.: Ontology Learning and Population from Text: Algorithms, Evaluation and Applications. Springer, New York, N.Y. and London (2006)
8. Dean, J., Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters. Communications of the ACM 51(1), 107–113 (Jan 2008)
9. Furth, S., Baumeister, J.: TELESUP - Textual Self-Learning Support Systems. In: Proc. LWA 2014 (FGWM Workshop). RTWH Aachen University, Aachen, Germany (2014)
10. Karthikeyan, K., Karthikeyani, V.: Ontology Based Concept Hierarchy Extraction of Web Data. Indian Journal of Science and Technology 8(6), 536 (2015)
11. Klein, D., Tran-Gia, P., Hartmann, M.: Big Data. Inform. Spektrum 36(3), 319–323 (2013)
12. Klöpper, B., Dix, M., Schorer, L., Ampofo, A., Atzmueller, M., Arnu, D., Klinkenberg, R.: Defining Software Architectures for Big Data Enabled Operator Support Systems. In: Proc. IEEE International Conference on Industrial Informatics. IEEE, Boston, MA, USA (2016)
13. Kluegl, P., Atzmueller, M., Puppe, F.: Meta-level information extraction. In: Proc. KI. LNCS, vol. 5803, pp. 233–240. Springer, Berlin / Heidelberg, Germany (2009)
14. Marz, N., Warren, J.: Big Data: Principles and Best Practices of Scalable Realtime Data Systems. Manning Publishers, Shelter Island, NY, USA, 1. edn. (2013)
15. Riedl, M., Biemann, C.: Scaling to Large Data: An Efficient and Effective Method to Compute Distributional Thesauri. In: EMNLP. pp. 884–890 (2013)
16. Tudorica, B.G., Bucur, C.: A Comparison between Several NoSQL Databases with Comments and Notes. In: International RoEduNet Conference. pp. 1–5. IEEE (2011)
17. Zaharia, M., Chowdhury, M., Franklin, M.J., Shenker, S., Stoica, I.: Spark: Cluster Computing with Working Sets. In: Proc. USENIX. HotCloud, Berkeley, CA, USA (2010)