Using Apache Spark on Hadoop Clusters as Backend for WebLicht Processing Pipelines

Soheila Sahami
NLP Group
Leipzig University, Germany
sahami@informatik.uni-leipzig.de

Thomas Eckart
NLP Group
Leipzig University, Germany
teckart@informatik.uni-leipzig.de

Gerhard Heyer
NLP Group
Leipzig University, Germany
heyer@informatik.uni-leipzig.de

Abstract

Modern annotation tools and pipelines that support automatic text annotation and processing have become indispensable for many linguistic and NLP-driven applications. To simplify their active use and to relieve users from complex configuration tasks, Service-oriented architecture (SOA) based platforms – like CLARIN's WebLicht – have emerged. However, in many cases the current state of participating endpoints does not allow processing of "big data"-sized text material or the execution of many user tasks in parallel. A potential solution is the use of distributed computing frameworks as a backend for SOAs. These systems and their corresponding software architecture already support many of the features relevant for processing big data for large user groups. This submission describes such an implementation based on Apache Spark and outlines potential consequences for improved processing pipelines in federated research infrastructures.

1 Introduction

There are several approaches to make the variety of available linguistic applications – i.e. tools for preprocessing, annotation, and evaluation of text material – accessible and to allow their efficient use by researchers in a service-oriented environment. One of those, the WebLicht execution platform (Hinrichs et al., 2010), has gained significance – especially in the context of the CLARIN project – because of its easy-to-use interface and the advantages of not being confronted with complex tool installation and configuration procedures, or the need for powerful local hardware where processing and annotation tasks are executed.

The relevance of this general architecture can be seen when considering the increasing relevance of "cloud services" in the current research landscape (in projects like the European Open Science Cloud EOSC) and the rising number of alternative platforms. Comparable services like Google's *Cloud Natural Language*, *Amazon Comprehend*, *GATE Cloud* (Gate Cloud, 2018), or the completed *AnnoMarket* project are typically tight to some form of business model and show the significance – including a commercial one – of those applications. It has to be seen how a platform like WebLicht that is mostly driven by its participating research communities can compete with those offerings. However, some of the shortcomings that could be reasons to use alternative services may be reduced in the context of the CLARIN infrastructure as well. Potential problems may include the following areas:

- Support of processing large amount of text material (so called "big data") without loosing the mentioned benefits of a service-oriented architecture.
- Efficient use of parallelization, including the parallel processing of large document collections and the support of large user groups.

Soheila Sahami, Thomas Eckart and Gerhard Heyer 2019. Using Apache Spark on Hadoop Clusters as Back-end for WebLicht Processing Pipelines. *Selected papers from the CLARIN Annual Conference 2018*. Linköping Electronic Conference Proceedings 159: 188–195.

- Open accounting of used resources (ranging from used hardware resources to financial costs) for enhancing user acceptance of services and workflows by making hidden costs more transparent.

Using parallel computing approaches to improve the performance and workload on available hardware is a common topic in computer science. Several approaches have been established over time, including a variety of libraries, distributed computing frameworks, and dedicated computing hardware for different forms of parallelization. This submission proposes using the Apache Spark[1] framework on Hadoop clusters as backend for a WebLicht processing endpoint to address the aforementioned issues. A first prototypical implementation suggests the benefits of this approach.

The following two sections will describe the technical details of this demonstrator. In section 4 some of the general outcomes – like potential consequences for improving the performance, user satisfaction and clarity of the tasks – will be discussed and are followed by a brief summary of this contribution in section 5.

2 Synchronous Communication and Interaction in a SOA

Service-oriented architectures (SOAs) are an architectural approach that supports a group of services – as discrete units of functionality – to communicate with each other. The basic principles of SOAs are their independence of users, products and technologies. Some of SOA-based applications have several limitations such as the inability to develop highly interactive or completely customizable applications or lack of supporting synchronous interactions (Papazoglou, 2003), (Erl, 2005).

CLARIN's WebLicht[2] is a SOA-based processing environment that provides chains of language resources and tools (LRT) as distributed and independent services and covers a wide range of linguistic applications and several languages (Hinrichs et al., 2010). It facilitates the users' processes and relieves them from complicated configuration tasks. In our working prototype, the atomic feature of the implemented tools make them appropriate to integrate into WebLicht and other SOAs as well.

WebLicht and comparable environments are mostly based on synchronous communication and interaction between users and the framework. The typical WebLicht workflow relies on users issuing – simple or combined – tasks and waiting for the final results for their export or further analysis. In general, this assumes synchronous interactions which are hardly feasible for big data analysis that may require execution times of several days or more.

However, WebLicht's popular Web-interface does not provide any adequate handling of long-term processes, processing of document collections or (very) large input data in general. Although it should be noted that with *WebLicht as a Service*, a first alternative approach exists for WebLicht, that mitigates some of these problems[3].

3 Technical Approach

In this section we discuss the techniques utilized in our implementation including Apache Hadoop and Apache Spark and their strengths in comparison with other techniques. We also describe the implemented linguistic applications and compare their efficiency with similar tools that use different frameworks.

3.1 Apache Hadoop

Apache Hadoop is a framework to process large-scale data in a distributed computing environment and is a popular framework for applications that deal with massive data and where the response time is significant. Its large ecosystem consists of the Hadoop Kernel, MapReduce, HDFS and some other components such as YARN, Apache Hive and Zookeeper (Apache Hadoop, 2019).

Apache Hadoop has a fault-tolerant distributed storage system called Hadoop Distributed File System (HDFS) that supports storage of massive amounts of data. HDFS is highly fault-tolerant, suitable for applications that support large data sets, easily deployable on low-cost hardware and provides high throughput access to data. It scales up incrementally and is able to handle the failure of

[1] https://spark.apache.org/
[2] https://weblicht.sfs.uni-tuebingen.de
[3] https://weblicht.sfs.uni-tuebingen.de/WaaS/

storage infrastructure without losing data by storing three complete copies of each block of data redundantly on three different servers (Bhosale and Gadekar, 2014).

MapReduce (Dean and Ghemawat, 2004) – which is the processing pillar in the Hadoop ecosystem – is a programming model and associated implementation that allows processing of data up to the size of multiple terabyte in parallel on large clusters (with thousands of nodes) in a reliable, fault-tolerant way. A MapReduce job divides input data into independent blocks. Users define the *Map* functions that process these data blocks in parallel and generate intermediate results as set of key-value pairs. The key-value based approach is utilized for the challenging task of combining processed data in a distributed environment. *Reduce* functions merge the intermediate values based on the same intermediate keys to produce the output. Input and output of the functions are stored in HDFS; the framework supervises and monitors scheduled tasks, and re-executes failed tasks (Bhosale and Gadekar, 2014), (Apache Hadoop, 2019).

Hadoop has some particular properties that make it more eligible (White, 2012). For instance, it can increase access speeds – the rate at which data can be read or written from or to drives – by reading and writing data from and to multiple disks in parallel instead of serial accesses and allows shorter analysis times by parallel execution of tasks. Furthermore, replication and redundant copies of data is a central component of Hadoop to avoid data loss in the event of hardware failure.

In comparison with relational database management systems (RDBMS) Hadoop is more efficient in many respects. RDBMS are able to do large-scale batch analysis on several disks but there are several problematic issues resulting from the used architecture. For example, updating a large portion of data – which often comes with sort or merge operations – is less efficient than using MapReduce. Furthermore, seek time latency – the time required to move the disk's head to a particular place on the disk to read or write data – when processing big data is considerable, especially in unstructured resources like texts. The approach of data access patterns in MapReduce – read, process and write the entire data set at once – decreases this latency.

In High Performance Computing (HPC) and Grid Computing platforms, large-scale data processing is executed using Message Passing Interface (MPI) which is distributing tasks across a cluster of machines and utilizing a shared file system. However, accessing very large data volumes (up to several terabytes) makes network bandwidth a potential bottleneck that can lead to idle computing nodes. MapReduce uses the data locality feature: data is collected by corresponding computing nodes; data is stored locally which improves access times (White, 2012).

3.2 Apache Spark

Apache Spark is also a general-purpose cluster computing framework for big data analysis with an advanced in-memory programming model and upper-level libraries and APIs. It uses a multi-threaded model where splitting tasks on several executors improves processing times and fault tolerance.

Compared to MapReduce, Apache Spark uses a data-sharing abstraction called Resilient Distributed Dataset (RDD); individual operations (i.e. Map and Reduce) are similar. RDDs are In-Memory Databases (IMDB) which are designed to run completely in RAM. Using this extension, Apache Spark is able to perform processing workloads easier and more efficient and provides large speedups. RDDs are transient: every time they are used they are recomputed. If data is used more often, users can persist individual RDDs in memory for a faster reuse.

One of the key properties of RDDs is their design as fault-tolerant collections, which are capable to recover lost data after a failure occurs and can be processed and manipulated in parallel. In other distributed computing frameworks, fault tolerance is achieved by data replication or check pointing while Spark uses a different approach called lineage. When building an RDD, the graph of used transformations is kept and if any failure occurs, the operations are re-run to rebuild lost results. As the RDDs are stored in memory, rewriting recovered data is faster than writing operations over the network. Thus, lineage-based recovery can save both execution time and storage space ((Zaharia et al., 2016), (Hamstra and Zaharia, 2013), (Salloum et al., 2016)).

3.3 Scalability

In parallel computing, beside the performance that increases by distributing executions over several processing cores, the efficiency and scalability of applications are also desirable attributes.

Scalability relates to the capability of a process to deal with a growing amount of data. As one of the performance metrics for parallel implementation is the runtime, the scalability is measured by the speedup that is the ratio of runtime using one executor to n executors. Efficiency is calculated by the speedup divided by the number of executors. In the ideal case, the highest value for efficiency shows a linear growing speedup. In reality and for typical use cases, a sub-linear speedup also shows an improvement in efficiency ((Hill, 1990), (Bondi, 2000)).

3.4 Tools

In the context of our implementation, a variety of typical NLP tools – including sentence segmentation, pattern-based text cleaning, tokenizing, language identification, and named entity recognition – were implemented[4]. These tools use Hadoop as their framework, Apache Spark as execution engine and store the input data and outputs on HDFS. The tools are atomic services that have the potential to be integrated in any SOA-based annotation environment.

In order to execute these tools, we have used a cluster provided by Leipzig University Computing Center (Meyer et al., 2018). Table 1 illustrates hardware and configuration of this cluster (Lars-Peter Meyer, 2018).

Number of nodes	CPUs	Hard drives	RAM	Network
90	6 cores per node	>2 PB in total	128 GB per node	10 Gbit/s Ethernet

Table 1: Cluster Characteristics

During the execution, as the first step, input text files are read from HDFS and loaded in RDDs. Those RDDs are distributed over the allocated cluster hardware and are processed by several executors in parallel. The results are provided by merging processed RDDs and the finalized outputs again are stored on HDFS.

Optimum hardware configuration for each job can be set dynamically considering volume and type of input data as well as the selected processing pipeline which may consist of a single or even multiple tools. The specific configuration is determined automatically based on empirical values taken from previous runs and takes the current workload of the underlying cluster into account.

For the subset of these tasks that is supported by WebLicht's text corpus format (TCF) (Heid et al., 2010) (i.e. tokenization and sentence segmentation) converters between TCF and the RDDs were written. As a result, the endpoint is structured as depicted in Figure 1.

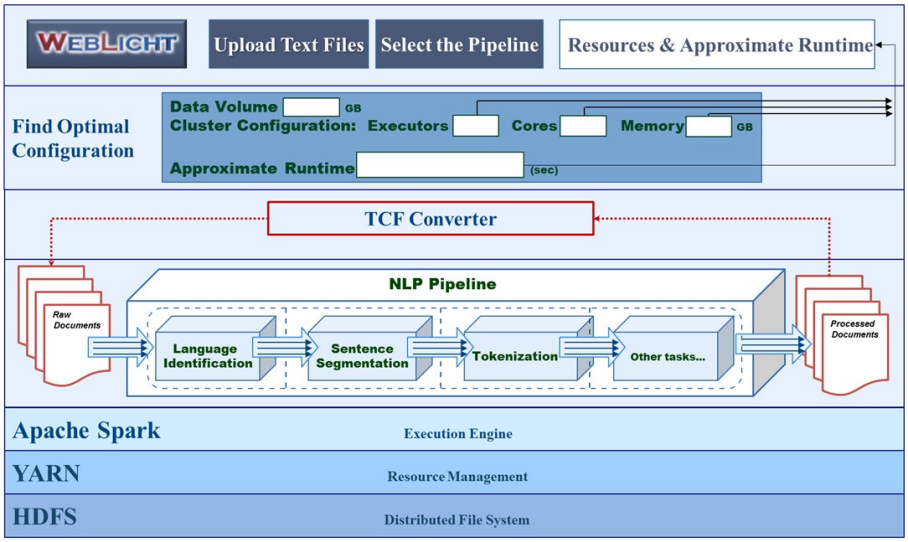

Figure 1: WebLicht with a Spark-based Backend.

[4] http://hdl.handle.net/11022/0000-0007-CA50-B

For the evaluation of the implemented solution benchmarks were carried out. The benchmarks were executed to show the impact of parallelization for every task. The diagrams in Figures 2 and 3 show runtimes for various data volumes with comparable characteristics using different cluster configurations. They illustrate the effect of configuration variables on concrete process runtimes and especially the impact of parallelization (i.e. the number of executors). Using these results, for every batch of input data a cluster configuration can be estimated that constitutes an acceptable trade-off between allocated resources and the expected runtime.

4 Results

The implemented tools already provide added value to the WebLicht platform. In this section, we will describe some of the more general outcomes.

4.1 Performance Improvements

The central advantage of the described implementation is its ability to use state-of-the-art cluster computation technology to process input material in massive scale. Parallelization in general has the potential to achieve major performance improvements; using Apache Hadoop and Spark – on the basis of large scale clusters – is a promising approach in this area. As a result, we were able to process huge amounts of data with a significant decrease in resources compared with sequential approaches.

Table 2 compares the runtimes for processing 6.5 GB input data for the subset of tools that can be currently integrated into WebLicht, using our parallel implementation and an existing sequential application. Both tests used the same hardware configuration with 8 GB memory and one executor. The measured runtimes illustrate that parallelization using Apache Spark can generate results in significantly less amount of time when the same input texts were processed using equal resources (memory and CPU).

	Segmenting	Tokenizing
Existing Sequential Implementation	8,200 sec	19,860 sec
Parallel Implementation	357 sec	781 sec

Table 2: Runtime for processing 6.5 GB input text using sequential and parallel implementations

The diagrams in Figure 2 and 3 show the scalability of the tools. As expected, by increasing the number of executors, runtime declines and efficiency improves. It is worth mentioning, that the number of executors cannot grow infinitely and exceeding available resources will lead to negative effects on the efficiency.

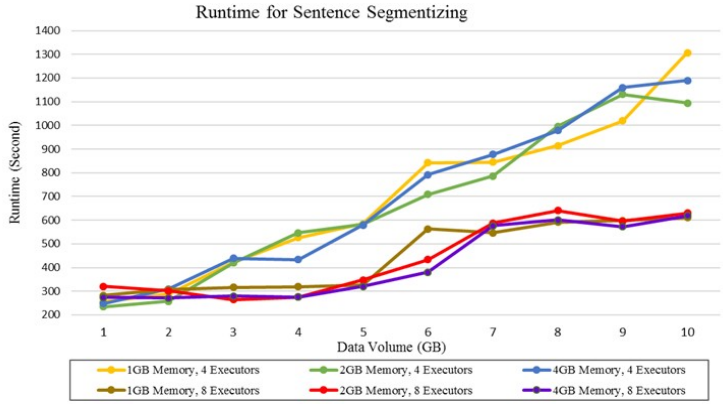

Figure 2: Segmentation 1 to 10 GB Text Data using 4 or 8 Executors and 1, 2 or 4 GB RAM.

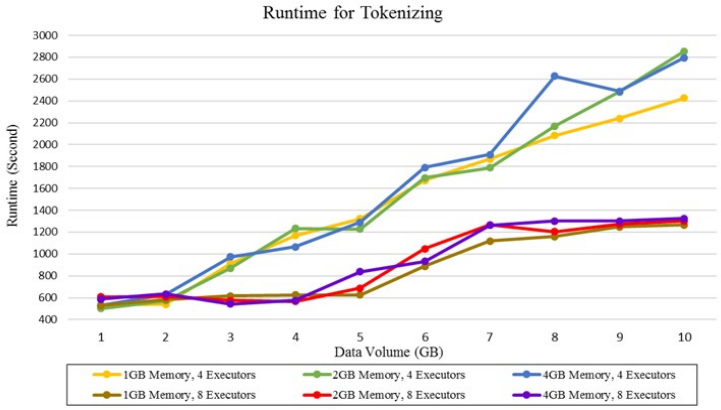

Figure 3: Tokenizing 1 to 10 GB Text Data using 4 or 8 Executors and 1, 2 or 4 GB RAM.

It is also noteworthy that the total runtime for the pipeline including sentence segmentation and tokenizing as a combined task depends on the most time-consuming sub-task – tokenizing in this experiment – regardless of the volume of documents and the allocated resources. That is because combined tasks save I/O time for writing the segmented documents which is the main part of the overall sentence segmentation runtime.

4.2 Benefits of Asynchronous Workflows

The aforementioned focus on synchronous communication patterns is problematic when dealing with very large input data and resulting growing execution times of scientific workflows. In those cases, a direct interaction with the user can not be expected and alternatives have to be evaluated. One of the possible approaches is the automated execution of pipelines as it is supported by *WebLicht as a Service*. However, this already requires some deeper technical knowledge from the end user and contradicts partially the seamless integration of pipeline results in a federated Web-based infrastructure.

For a systematic support of big data processing in the context of WebLicht pipelines, changes in the default workflows and user interfaces might be helpful. This may comprise an improved support for the processing of document collections – in contrast to a more document-centric approach – and a stronger focus on data storage platforms that support workspaces for individual users like B2DROP. This infrastructure component has the ability to function both as a primary means of storage for individual users and user groups, but also as a central entry point to start new workflows in a data-oriented environment. Its function would include both being a potential provider of input data for WebLicht but also as the default storage space of intermediate and final results. User information about status and outcome of scheduled processing jobs can be transfered via Email or job-specific status pages[5]. Those status reports should be seen as an important means to inform users about used hardware resources, required runtimes, and relevant process variables. For increasing user acceptance of the overall system, they may also contain information about required financial resources that would have been necessary to perform the same task using a commercial platform.

4.3 Technical Costs for Accountability and User Motivation

Resulting execution times for a specific configuration (i.e. number of used executors, allocated memory, etc.) are valuable information for estimating requirements and runtime behavior of every task. Based on empirical data, runtimes for new tasks can be estimated considering their general characteristics (i.e. size of input data and used tool configuration). This estimate provides several added values to our system that are briefly described in the following.

[5] A functionality that is already supported by other comparable frameworks.

- **Balance between resources and users satisfaction**: This valuable information helps to find an optimal balance between number of parallel user tasks, available hardware configuration, and waiting times that are still acceptable for the users.
- **Processing time**: It provides the users the approximate process time for the requested services, which may help to increase their willingness to wait for results.
- **Parallel users**: Available resources and estimated runtimes for requested tool chains can be used to define the number of parallel users on the system dynamically.
- **Financial costs**: Using services (resources and tools) has financial costs in commercial NLP platforms like Amazon Comprehend[6] and Google Cloud NLP[7]. Prices are defined based on the chosen service and size of the data (like number of characters). Making these potential costs more visible can encourage users to carefully select a proper configuration. This information – next to required resources and run times – can also be taken as a basis to calculate the technical cost for each tool chain individually.

Having this information, the tool chain is not a black box anymore and this transparency can increase the user's motivation and satisfaction.

As an instance, results of sentence segmentation a 5 GB document using 8 executors with 8 GB memory will be ready after 10 minutes and using 2 executors with 4 GB memory after 25 minutes. Administrator of the system can suggest different configuration considering the available resources and the user has the options to decide based on the waiting time and probable technical cost.

5 Summary

In this working prototype, we have presented an alternative approach to use NLP tools for processing large text data and handling large user groups using parallelization in a cluster environment. Using this approach, besides simple runtime improvements – processing more data using affordable resources and less runtime –, several results were achieved.

Dynamic resources configuration was introduced for each individual NLP task that constitutes an acceptable trade-off between allocated resources and expected runtime. The capability of open accounting of required resources – ranging from used hardware resources to financial costs – to make hidden costs more transparent is seen by the authors as a central means to enhance user acceptance of services in an environment like WebLicht, or CLARIN in general. Furthermore, it was outlined how a stronger focus on asynchronous communication in a federated research environment has the potential for seamless integration even in the case of long-term annotation processes or the processing of big data resources.

Acknowledgement

Computations for this work were done with resources of Leipzig University Computing Center.

References

[Apache Hadoop2019] Apache Hadoop. 2019. Apache Hadoop Documentation. Online. Date Accessed: 11 Jan 2019. URL http://hadoop.apache.org/.

[Bhosale and Gadekar2014] Harshawardhan S Bhosale and Devendra P Gadekar. 2014. A review paper on big data and hadoop. International Journal of Scientific and Research Publications, 4(10):1–7.

[Bondi2000] André B Bondi. 2000. Characteristics of scalability and their impact on performance. In Proceedings of the 2nd international workshop on Software and performance, pages 195–203. ACM.

[Dean and Ghemawat2004] Jeffrey Dean and Sanjay Ghemawat. 2004. Mapreduce: Simplified data processing on large clusters. In OSDI'04: Sixth Symposium on Operating System Design and Implementation, pages 137–150, San Francisco, CA.

[Erl2005] Thomas Erl. 2005. Service-oriented architecture: Concepts, Technology, and Design. Prentice Hall PTR.

[6] https://aws.amazon.com/comprehend/pricing
[7] https://cloud.google.com/natural-language/pricing

[Gate Cloud2018] Gate Cloud. 2018. GATE Cloud: Text Analytics in the Cloud. Online. Date Accessed: 11 Apr 2018. URL https://cloud.gate.ac.uk/.

[Hamstra and Zaharia2013] Mark Hamstra and Matei Zaharia. 2013. Learning Spark: lightning-fast big data analytics. O'Reilly & Associates.

[Heid et al.2010] Ulrich Heid, Helmut Schmid, Kerstin Eckart, and Erhard W Hinrichs. 2010. A Corpus Representation Format for Linguistic Web Services: The D-SPIN Text Corpus Format and its Relationship with ISO Standards. In Proceedings of LREC 2010.

[Hill1990] Mark D Hill. 1990. What is scalability? ACM SIGARCH Computer Architecture News, 18(4):18–21.

[Hinrichs et al.2010] Erhard Hinrichs, Marie Hinrichs, and Thomas Zastrow. 2010. WebLicht: Web-based LRT services for German. In Proceedings of the ACL 2010 System Demonstrations, pages 25–29. Association for Computational Linguistics.

[Lars-Peter Meyer2018] Lars-Peter Meyer. 2018. The Galaxy Cluster. Online. Date Accessed: 12 Apr 2018. URL https://www.scads.de/de/aktuelles/blog/264-big-data-cluster-in-shared-nothingarchitecture-in-leipzig.

[Meyer et al.2018] Lars-Peter Meyer, Jan Frenzel, Eric Peukert, René Jäkel, and Stefan Kühne. 2018. Big data services. In Service Engineering, pages 63–77. Springer.

[Papazoglou2003] Mike P Papazoglou. 2003. Service-oriented computing: Concepts, characteristics and directions. In Web Information Systems Engineering, 2003. WISE 2003. Proceedings of the Fourth International Conference on, pages 3–12. IEEE.

[Salloum et al.2016] Salman Salloum, Ruslan Dautov, Xiaojun Chen, Patrick Xiaogang Peng, and Joshua Zhexue Huang. 2016. Big data analytics on Apache Spark. International Journal of Data Science and Analytics, 1(3-4):145–164.

[White2012] Tom White. 2012. Hadoop: The definitive guide. O'Reilly Media, Inc.

[Zaharia et al.2016] Matei Zaharia, Reynold S Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J Franklin, et al. 2016. Apache spark: a unified engine for big data processing. Communications of the ACM, 59(11):56–65.