Performance Catalogs for Just-In-Time Delivery of Web-based Natural Language Processing Services

Soheila Sahami
Natural Language Processing Group
University of Leipzig, Germany
Email: sahami@informatik.uni-leipzig.de

Thomas Eckart
Natural Language Processing Group
University of Leipzig, Germany
Email: teckart@informatik.uni-leipzig.de

Abstract—**The modern production of industrial goods is often based on Just-In-Time delivery of required resources. This paradigm makes a variety of demands on the infrastructure in which production takes place or in which services are provided and consumed. The reasons for introducing this strategy and its potential benefits are in large parts applicable to the area of Web-based Natural Language Processing services. This contribution focuses on prerequisites and potential outcomes of a Just-In-Time-capable infrastructure of Natural Language Processing services using examples in the context of real-world research projects. The benefits of this endeavor are sketched with a focus on the ongoing development of large scale service delivery platforms like the European Open Science Cloud, CLARIN, or similar projects. As a major outcome, the creation of "performance catalogs" containing extensive information about the run-times and performance of every involved tool is seen as an essential precondition for these environments.**

Keywords–Just-In-Time Delivery; Performance Catalog; Natural Language Processing Services; Research Infrastructure; Cluster Computing.

I. Introduction

This article is an extended version of the conference publication "Just-In-Time Delivery for NLP Services in a Web-Service-Based IT Infrastructure" presented at the *Adaptive Workflow Management for NLP Data Processing workshop* at ADAPTIVE 2019 [1]. This article provides a more detailed discussion of relevant parameters defining performance profiles of services. These parameters, which may impact the performance of services, such as language and quality of input material are defined and an extended set of experiments depending on different hardware configurations and tool parameters are conducted.

In industrial production environments, providing resources immediately before they are required in the context of a larger production chain – typically called *Just-in-Time Delivery* (JIT delivery) – is a standard procedure for many decades now. The transfer of this concept into the area of information technology offers a new competitive opportunity that promises significant advancements, such as faster responses, improved quality, flexibility, and reduced storage space [2].

The use of Natural Language Processing (NLP) applications – i.e., tools for preprocessing, annotation, and evaluation of text material – is an integral part for a variety of applications in scientific and commercial contexts. Many of those tools are nowadays available and actively used in service-oriented environments, where – often complex – hardware and software configuration is hidden from the user. In the context of large research infrastructures, like CLARIN [3] or DARIAH [4], or cross-domain projects, like the European Open Science Cloud (EOSC) [5], one of the key goals is to facilitate the use of services which, are seen as integral and indispensable building blocks of a modern scientific landscape.

These research infrastructure projects can be seen as driving forces for current trends in the dissemination and delivery of tools and services. However, in many respects, they are undergoing a similar development as already completed in many commercial areas where delivery and use of services are performed in an industrial scale. Systematic assessment and improvement of quality, measurement of throughput times or other criteria are prerequisites for the use of services even for time-critical applications [6].

One of the potential outcomes and goals of a more "industrialized" infrastructure could be a just-in-time delivery of services, providing the benefits – while requiring comparable prerequisites – already accustomed in the industrial production of goods. They include reduced response times, reduction of required storage facilities [7], and more. However, those topics are hardly addressed in today's text-oriented research infrastructures. Some of the missing preliminary work that is required to offer JIT delivery of linguistic services – like the transparency of the process and its sub-processes, deep knowledge about required resources and execution times – are addressed in this contribution.

One of the important challenges in JIT delivery is the applied strategy to address the reliability and predictability of services [8]. In IT infrastructures, utilizing fault-tolerant techniques is one of the solutions to improve the reliability of an application. Parallelised implementations using cluster-based processing architectures are technologies that are utilized to decrease run-times and to enable the processing of large scale resources. Furthermore, they provide a helpful means to configure processes in a dynamic manner. This allows suggesting several configurations based on the available resources of the service provider or temporal requirements of the user. Clear information about potential expenses and the estimated delivery time for each configuration gives users a means to select a suitable service (or service chain) or service configuration that fits their needs best.

This also helps users to have a clear strategy for data storage, duration of data retention, and delivery time. These features have the potential to enhance the user's satisfaction and provide added values that lead to a stronger position in competitive industrialized IT infrastructures.

In this contribution, we present examples of Natural Language Processing services with a focus on their transparency regarding execution times and required resources. As a result, valid resource configurations can be chosen considering available resources and expected delivery times. It should be pointed out that multiple NLP tools have been implemented to prove the suggested approach and more tools and the other methods – such as machine-learning-based or hybrid approaches – can be contemplated as the future extension of this research.

The following Section II gives more details about the parallelism of just-in-time delivery of IT services and their industrial counterpart. Section III describes the used methodology and its characteristics, technical approaches and tools. Section IV describes the research infrastructures that can be used as the base for these tools and compares them with commercial counterparts. Section V explains the chosen parameters followed by Section VI showing the outcomes of the experiments that are performed by assigning various cluster and tool configurations varying every individual parameter. Sections VII and VIII illustrate and discuss the outcomes and results and are followed by a brief conclusion of this contribution and a short outlook in Section IX.

II. Just-in-time Delivery of IT-Services

Just-in-time delivery (or just-in-time manufacturing) is a management concept that was introduced in the Japanese automotive industry [9] and was adopted for many other areas of production and delivery of goods since. Based on experiences and best practices, catalogues were developed that contain extensive lists of requirements that make the usage of JIT delivery chains manageable and trustworthy.

Established requirements deal with all kinds of aspects in the organizational, legal and technical environment of companies and organizations that are involved in the overall process. At least a subset of those requirements is directly transferable to activities in IT processes and infrastructures [2], including the more recent deployment, provisioning, and use of services in complex Service-Oriented Architectures (SOAs). This contains procedures and guidelines like the strict use of a "pull-based" system, process management principles with a focus on flow management, adequate throughput, and continuous assessment of quality and fitness of used processes and outcomes. Its obvious benefits have made the underlying policies also a cornerstone of modern agile management principles (c.f. [10]).

There is some research about transferring the JIT concept and its principles to service-oriented environments, like the ones gaining momentum in the area of NLP applications. In the context of such IT services chains, the term *just-in-time* can be understood in different ways. It is often referring to the specific decision for a set (or chain) of services – out of a potentially large inventory of compatible services from different providers – as part of the typical discovery/bind-process *at run-time*, i.e., without a fixed decision for specific providers or even prior knowledge about the current inventory of available services. This is sometimes called "just-in-time integration" of services (for example in IBM's developer documentation [11]).

Many essential requirements for a JIT integration are already handled in existing frameworks – for example CLARIN's WebLicht [12] –, like compatibility-checks of all services regarding their input parameters and generated output, a systematic monitoring of all participating service providers of the federation, or – in parts – even adherence to legal constraints.

A different approach for services-based JIT delivery focuses on the estimated time of arrival (ETA) of the required results for a specific service chain. This is especially important considering the growing amount of text material that is required to be processed. Most academic providers of NLP services are not able to guarantee acceptable processing times – or the completion of large processing jobs at all – with their current architectures for (very) large data sets in the context of SOAs. However, this kind of functionality is required to reach new user groups and to make them competitive offerings in comparison with the other (including commercial) service providers. This aspect is hardly addressed in previous and current projects of the field but gains significance in current attempts to make scientific working environments more reliable and trustworthy with a strong focus on cloud-based solutions (like the European Open Science Cloud EOSC).

A key idea is the incremental creation and adaptation of "performance profiles" for all elements of a provider's service catalog. This contains the identification of all relevant parameters of a tool and well-founded empirical knowledge about their effects on the run-time of every single NLP task for all kinds of plausible inputs. This requires a processing architecture that is able to dynamically allocate resources for each assigned job while minimizing (or even eliminating) the effects of other jobs that are executed in parallel.

In the following, we will describe a concrete example of such a service performance profile depending on the assigned hardware configuration and workflow arrangement and sketch its benefits.

III. Approach

In this section, the essential features of a JIT delivery in IT systems are explained. This is followed by a discussion of the chosen technologies and their specific features relevant to the context of this contribution. Finally, the implemented NLP tools and utilized resources are described.

A. Essential Features

Generally speaking, the degree of user acceptance in regard to JIT delivery in IT systems relies on several parameters such as diversity of the tools, state-of-the-art technologies, support for complex requests, and quality of the services. However, reliability and predictability of services [8] are two of the most important parameters that have large effects on the success of JIT delivery in IT systems. Accordingly, it is essential to choose technologies that support these two parameters.

1) Reliability: The international standard ISO/IEC/IEEE 24765:2017 has defined reliability as the "degree to which an object or an object's services provide agreed or expected functionality during a defined time period under specified conditions." A highly reliable application performs the expected functionality and is able to avoid, detect, and repair the failures in a way that users do not notice them. In other words, the application should be enabled to continue normal operation in the presence of faults, or be fault-tolerant. Fault tolerance is defined in ISO/IEC/IEEE 24765:2017 as the "degree to which a system, product or component operates as intended despite the presence of hardware or software faults." [13]

In the context of this article, fault tolerance refers only to the ability of a system to detect a hardware fault and immediately switch to a redundant hardware component. To increase the degree of fault tolerance and provide more reliable applications, distributed systems are implemented using cluster-based processing architectures [14] [15].

2) Predictability: In general, a predictable system is a system that has agreed or expected functionality for the possible states. Predictability is the degree to which a system or a component of the system behaves as expected in different situations. In the field of information technology and computer science, this criterion is known as the availability of an application. Availability refers to the probability that the system will operate continuously without failure during a specific time period [16]. High availability, as a critical feature, is addressed by cluster computation technology. This technology uses redundancy in order to improve the degree of availability [17].

B. Technical Approach

For services, where response times are critical, used technology should support technical features like fault tolerance and high availability. In addition, the technology should be able to process large scale data in a feasible time to satisfy the demand for the processing and analyzing of a rapidly growing amount of text data.

For this contribution, Apache Hadoop clusters and the Apache Spark execution engine are used to address the topics of fault tolerance and high availability. This approach supports the distribution and parallelization of tasks and significantly improves execution and response times.

Apache Hadoop is a popular framework to store and process large-scale data in a distributed computing environment that – with its built-in mechanisms – provides a high degree of parallelism, robustness, reliability, and scalability. One beneficial approach in this distributed processing technology is to run processes wherever the data is located. This means Hadoop initially distributes the data to multiple machines in the cluster and then assigns computation tasks based on the locality of data. This location-based assignment reduces the communication overhead between participating nodes [18]. Apache Hadoop's large ecosystem consists of the Hadoop Kernel, MapReduce, Hadoop Distributed File System (HDFS), Apache Spark, and some other components [19].

In the context of Big Data, partitioning the data across a number of separate machines is obligatory. HDFS, as the storage layer of Apache Hadoop, is a distributed file system that provides access to the data across the Hadoop clusters. HDFS is a highly fault-tolerant distributed storage system that is able to handle the failure of storage infrastructure without losing the data. The application data – the file contents – is split into large blocks with a default size of 128 MB, and each block of the file is independently replicated at multiple machines, where the size of the data blocks and the number of the replicas can be defined by the user on a file-by-file basis. Given the importance of the file system namespace, HDFS makes it resilient to failure by providing a backup of the file system and also periodically creating a copy of the list of blocks belonging to each file. Duplicating the copies of data blocks multiple times in the individual data nodes increases the reliability, allowing the system to tolerate node failure without suffering data loss. In the event of failure, this information facilitates the recovery of data [20].

Apache Spark is also a general-purpose cluster computing framework for big data analysis with an advanced In-Memory programming model. It uses a multi-threaded model where splitting the tasks on several executors improves processing times and fault tolerance. Apache Spark uses the Resilient Distributed Dataset (RDD), a data-sharing abstraction that is designed as a fault-tolerant collection and is capable of recovering lost data after the failure using the lineage approach. In this approach, Apache Spark keeps a graph of the data transformations during the construction of an RDD. In the event of failure, it reruns all failed operations to rebuild the lost results. The RDDs are persisted and executed completely in RAM – In-Memory Databases (IMDB) –, therefore generating and rewriting the recovered data are performed as fast processes [21] [22].

C. NLP Tools and Resources

Using Hadoop-based cluster computing architecture, a variety of typical NLP tools were implemented, including sentence segmentation, pattern-based text cleaning, tokenizing, language identification, and named entity recognition [23]. These tools use Apache Hadoop as their framework, Apache Spark as execution engine and HDFS as file system and storage manager. Furthermore, their atomic design facilitates to integrate them into SOA-based annotation environments.

In order to have an accurate estimation of execution times, a variety of benchmarks were carried out for the implemented tools. For these benchmarks, we have used a cluster provided by Leipzig University Computing Center [24]. Table I illustrates the characteristics of this cluster [25]. In this cluster, Apache Hadoop 2.7.3 is used as framework to process the data in the distributed environment and store the data on HDFS. Apache Spark 2.3.0 is used as the execution engine and YARN is configured as the resource manager.

TABLE I. Cluster Characteristics

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

As an example, one of these benchmarks evaluates the duration of sentence segmentation for datasets of German documents with sizes from 1 to 10 Gigabytes using different cluster configurations. The cluster configuration varies in the number of assigned executors (1 to 32 nodes) and allocated memory per executor (8 or 16 GB). Each test was repeated three times; average execution time over all three runs was used for the following statistics. Figures 1 and 2 show these execution times for sentence segmentation from 1 to 10 GB of input data with different resource configurations.

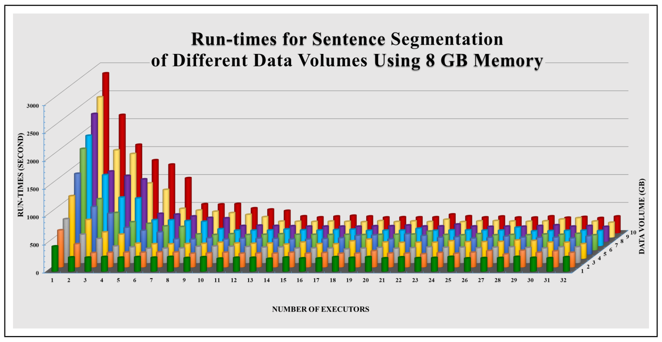


Figure 1. Run-times for segmenting 1 to 10 GB text materials using 1 to 32 executors and 8 GB memory per executor.

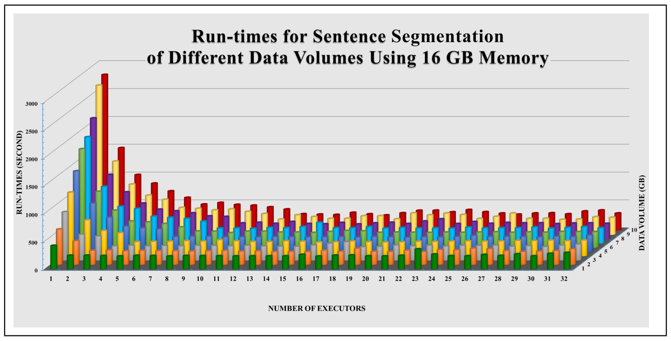


Figure 2. Run-times for segmenting 1 to 10 GB text materials using 1 to 32 executors and 16 GB memory per executor.

A brief explanation is given below for each of the tools.

1) Text Cleaner: This tool is a parallel implementation of a pattern-based text cleaner that uses sets of rules to detect invalid character patterns in text documents. *General rules* – being language-independent – contain generic unlikely patterns in written language. *Language-based rules* consist of individual rules for each of the defined languages to detect language specific ill-formed sentences. *Genre-based rules* are defined to specify additional patterns typically occurring in different sources of origin, including Web, Newspapers and Wikipedia.

2) Sentence Segmentizer: The implemented sentence segmentizer is a distributed version of a rule-based sentence segmentation tool that uses multiple of rule sets and lists of tokens to identify the sentence boundaries in a text. Disambiguation rules consider preceding and succeeding tokens of any potential sentence boundary symbol for their decision. The lists of tokens include *typical sentence boundary marks*, *abbreviations* for different languages, and a set of tokens which are not indicators of a sentence boundary, like file extensions (.com, .pdf) or top-level domains in URLs (.com, .de, .org).

3) Tokenizer: The implemented tokenizer identifies token boundaries using a set of rules and tokens. The tool relies on lists of typical token boundaries and punctuation characters, lists of abbreviations, and a list of known phrases and multi-word units that should be treated as single tokens. The implementation relies on sentences as input material. In cases where the input is not already separated into sentences, the aforementioned sentence segmentizer is called in advance.

4) Language Identification: The language identification tool utilizes sets of high-frequency words in different languages and their frequency to calculate the probability of a sentence belonging to a specific language.

5) Named Entity Recognition: The named entity recognition (NER) tool provides two main functions: the training of new NER models and the actual annotation task. New machine-learning-based models can be created and trained using fully annotated documents as input. The trained models are used to recognize instances of named entities in new documents. This functionality can be run on a cluster to annotate documents in parallel.

IV. Application

This section gives a short overview of research and infrastructure projects relevant to the context of the presented work. The sketched applications are already in use in some of them. Furthermore, a short overview of commercial providers of NLP services gives a broader picture of the current NLP-services landscape.

A. *Leipzig Corpora Collection*

The *Leipzig Corpora Collection* (including its subproject *Deutscher Wortschatz* focusing on the German language) uses a complex crawling infrastructure to continuously gather text material based on freely-available Web resources. This can include the acquisition of more than one terabyte of raw data per day (mostly based on HTML documents). The processing pipeline to convert this vast size of input documents to statistically and linguistically annotated text corpora contains a variety of tools with language and – sometimes – genre-specific configurations. Figure 3 gives a short summary of the used toolchain, that includes many of the aforementioned tools. For more details, cf. [26].

When starting the processing of Web material in the early 2000s, a single-threaded processing pipeline with mostly single-threaded tools was sufficient to convert raw material. Over time, the amount of gathered material, and thus requirements for the processing of this data has increased significantly.

As a consequence, most parts of the processing pipeline were parallelized and different approaches and system architectures were tested. Nevertheless, adequate handling of crawled material is still problematic and can lead to unprocessed heaps of data when incoming bandwidth exceeds the processing capability of subsequent processing and storage solutions.

B. *Commercial and Academic Cloud-based NLP Frameworks*

As mentioned before, the increase in the importance of NLP and NLU-based tools in both commercial and scientific contexts, has strengthen the demand for easy-to-use interfaces suitable for result-oriented users without deep knowledge of the underlying algorithms. As a consequence, a variety of platforms were created that target both user groups and focus on their specific demands.

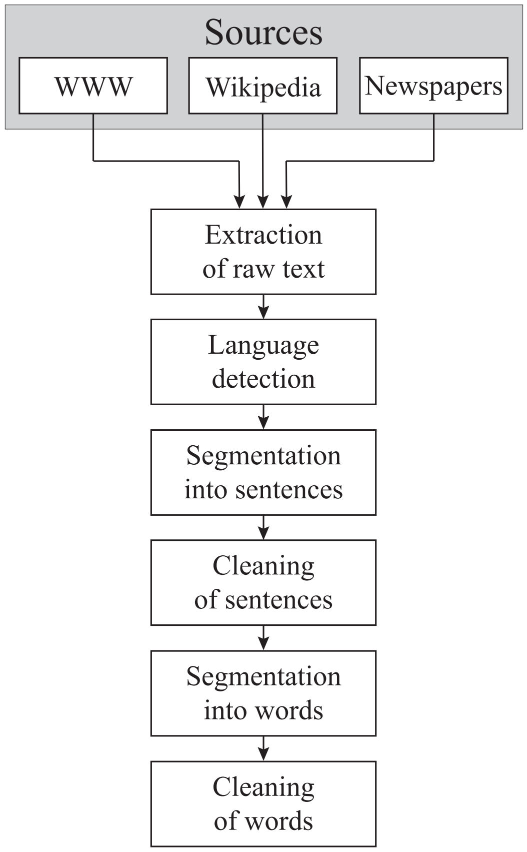


Figure 3. General overview of the processing pipeline of the Leipzig Corpora Collection.

Noteworthy examples of commercial platforms are *Amazon Comprehend* [27], *Google Cloud Natural Language Processing* [28], and IBM's *Watson Natural Language Understanding* [29]. Relevant academic or academic-oriented platforms include CLARIN's *WebLicht* [30](which will be discussed in more detail in the next subsection), *Language Applications (LAPPS) Grid* [31] or *GATE Cloud* [32].

Comparisons between these competitors may be based on a variety of evaluation criteria [33], like :

- Quality of documentation
- Openness of the platform
- Scalability
- Responsiveness
- Extent of supported tools
- Supported programming languages
- Supported natural languages (including specific dialects or language registers)
- Quality of results
- Costs

Evaluation of available platforms shows significant differences especially when comparing commercial systems with their academic counterparts. Typical advantages of the latter include support of more languages (including languages having lesser commercial interests), a broader landscape of provided tools, more options to participate as a service provider, and a lesser focus on financial gain. On the other hand, commercial platforms typically excel in most of the usability-related and technical aspects: high-quality documentation, easy-to-use (Web) interfaces, support of a variety of programming interfaces and, furthermore, often a built-in strategy for scalability aspects that allows the processing of large and very large data sets inside their system. This is especially obvious for companies like Amazon and Google being global providers of cloud computing services (*Amazon Web Services* AWS, *Google Cloud*).

In the context of this paper, the aspect of scalability obviously stands out as one the most relevant issues and is seen by the authors as one of the reasons that hinder the wide application and general success of open platforms in the field. This is especially problematic considering the sub-optimal availability of unique tools for a variety of languages which do not have a commercial focus because of rather small speaker groups or low commercial relevance. This can be seen as problematic considering today's diverse and broad cultural landscape. Closing the gap regarding scalability aspects can be helpful to reduce the identified divergence.

C. *CLARIN WebLicht*

A concrete example of an academic-based annotation framework is the WebLicht platform [12] of the CLARIN project [3]. WebLicht is an execution environment for automatic annotation of text corpora and provides currently more than 400 services in a federated and service-oriented environment. The number of supported tools and languages exceeds those of commercial alternatives and its restriction for scientific applications makes its use free of charge for the targeted user group. Its openness towards new service providers makes it a suitable candidate to evaluate potential benefits of improved scalability in services endpoints of such a federated infrastructure.

A WebLicht-compatible endpoint was created that uses the sketched processing architecture as a back-end (for more details, cf. [34]). Figure 4 shows the structure of this endpoint in the WebLicht environment. User input is handled by WebLicht's Web interface where input documents are uploaded and pipelines are built and started by the user. The second layer contains the implemented tools and shows an example of a user-defined workflow including the reading, annotating, and returning of processed documents. The format conversion is also performed in this layer if required. All actual processing is based on the execution framework providing a file system, resource manager, and execution engine as depicted above.

V. Parameters

In any services-based environment, service catalogs are defined and developed based on demand, experiences, and best practices. These service catalogs contain – among many other aspects – comprehensive lists of requirements that are needed to provide services to the customer. As part of a service catalog management, detailed information about the performance of each item of the service catalog is required. It is an essential part of this task to include and consider all relevant parameters when modifying and improving this data continuously and

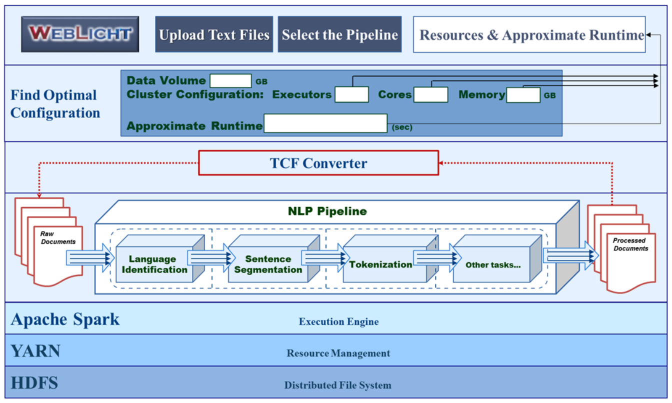


Figure 4. Service-oriented Architecture in WebLicht with Cluster Computation Technology as Backend.

to create meaningful "performance catalogs" for all provided services.

In this contribution, we considered several parameters relevant for performance evaluation and carried out several benchmarks to measure the effect of these parameters on the execution time of the tools. These parameters are categorized into two main categories: *cluster configuration* and *tool configuration*.

1) Cluster Configuration Parameters: These parameters consist of the following hardware resources:

- Executor: The experiments are performed using 8 or 16 worker nodes as executors. On each executor, 5 cores are available allowing 5 concurrent threads on each executor node. The number of cores multiplied by the number of executors defines the degree of parallelization or the maximum number of parallel tasks.
- RAM: Each machine in the cluster is equipped with 128 GB RAM; the allocated memory per executor is set to 8, 16, and 32 GB for most experiments. In general, 1 GB of memory is reserved for Hadoop and its related applications and 7% of the memory is considered for overhead. The rest will be used to process the data.

2) Tool Configuration Parameters: These parameters are selected based on the input documents and the specific workflow and include the following:

- Source of data: The input documents for these experiments are taken from the Leipzig Corpora Collection [26] that are collected from different sources comprising of Wikipedia, Newspapers, and general Web documents.
- Language: The documents are in English or German.
- Data volume: The input document collections are in sizes of 1 to 7 GB.
- Workflow: Workflows in this contribution are based either on the sentence segmentation or tokenization tool, or their combination into a joint workflow.

VI. Experiments

Various experiments were performed to analyze the effects of different cluster and tool configuration parameters on the execution times of the tools. The cluster is configured individually for each experiment to show different degrees of parallelism for all of the tools and origin of documents. Each experiment was repeated three times and the final execution time was measured as the average of these three run-times.

The following depicts the outcomes in diagrams which will be discussed in Section VII.

- Figures 5 to 8 present the run-times for the sentence segmentation task for different data volumes, text types, and cluster configurations.
- Figures 9 to 12 present the run-times for the word tokenization task for different data volumes, text types, and cluster configurations.
- Figures 13 to 16 present the run-times for the combined sentence segmentation and word tokenization task as a joint workflow for different data volumes, text types, and cluster configurations.

VII. Results

As Figures 1 and 2 illustrate, run-times vary for different job configurations significantly. As expected, using only a single executor – therefore, executing the job without any parallelization on the cluster – results in the maximum run-time for every data volume. The outcomes of all tests comply with the expected behaviour of parallel processing: a sharp decrease in execution time by increasing the assigned resources (i.e., executors), followed by a smoother reduction and finally no significant improvement when adding more resources to the job does not improve run-times anymore. The results show consistent behaviour for different data volumes using various cluster configurations.

The statistics also depict other relations: figures 5 to 16 also show that the execution times differ for various tool configuration parameters. Furthermore, the outcomes illustrate that in general, English documents need less time to be processed using these tools. Regarding the source of the documents, newspaper texts are segmented faster than other text sources in both English and German language, where this parameter does not affect the run-times of tokenization tasks significantly.

Another area worth analyzing is the negative impact of overheads in the processing. Figures 17 and 18 depict the execution times of sentence segmentation and tokenizing of 7 GB documents in two scenarios: applying the tools separately or performing the workflow as a combined task. Creating a joint workflow of both tasks reduces some of the redundant steps, as well as some I/O tasks and results in an improved execution time. For instance, processing 7 GB of German newspaper documents using 16 executors and 32 GB memory required 665 seconds when each tool applied separately. This decreases to 552 seconds for a combined toolchain. For English newspaper material and an identical cluster configuration, this leads to a decrease from 492 to 376 seconds.

Figure 19 gives an overview of run-times for data sets from 1 to 10 GB using 1 to 32 executors and 16 GB RAM per executor. It represents a significant reduction in execution times by providing more executors that is followed by a steady state.

Figure 20 shows the results for the sentence segmentation task of 10 GB text material which required 2860 seconds using

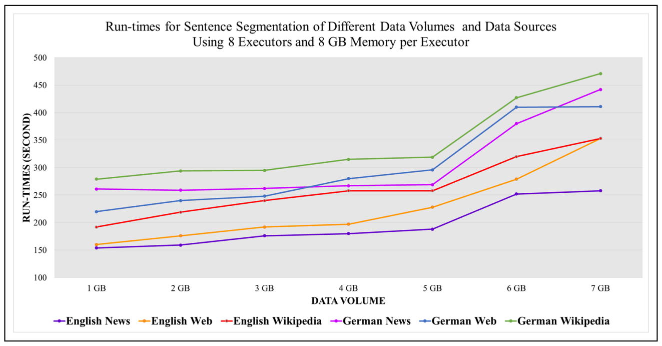


Figure 5. Run-times for segmenting 1 to 7 GB text materials from Newspapers, Wikipedia, and General Web documents in English and German, using 8 executors and 8 GB memory per executor.

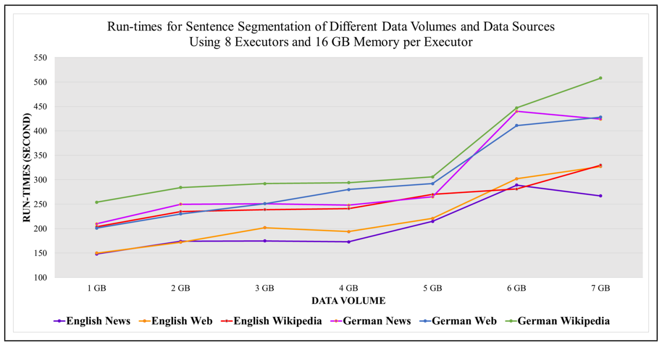


Figure 6. Run-times for segmenting 1 to 7 GB text materials from Newspapers, Wikipedia, and General Web documents in English and German, using 8 executors and 16 GB memory per executor.

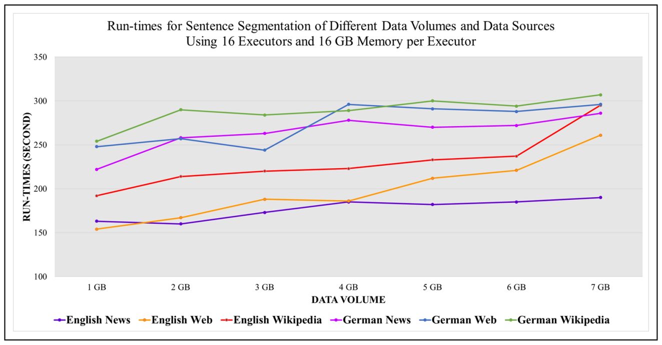


Figure 7. Run-times for segmenting 1 to 7 GB text materials from Newspapers, Wikipedia, and General Web documents in English and German, using 16 executors and 16 GB memory per executor.

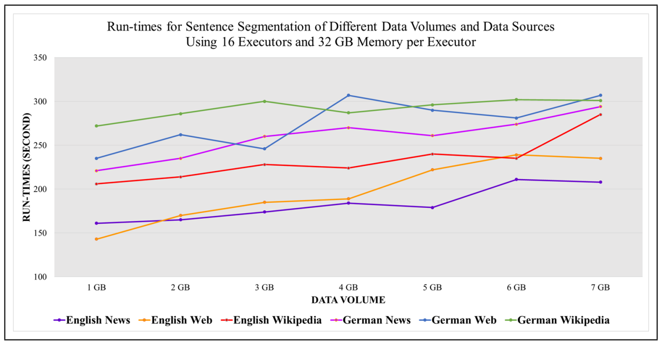


Figure 8. Run-times for segmenting 1 to 7 GB text materials from Newspapers, Wikipedia, and General Web documents in English and German, using 16 executors and 32 GB memory per executor.

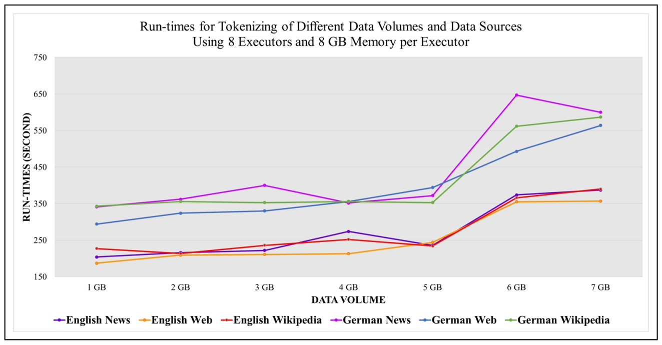


Figure 9. Run-times for tokenizing 1 to 7 GB text materials from Newspapers, Wikipedia, and General Web documents in English and German, using 8 executors and 8 GB memory per executor.

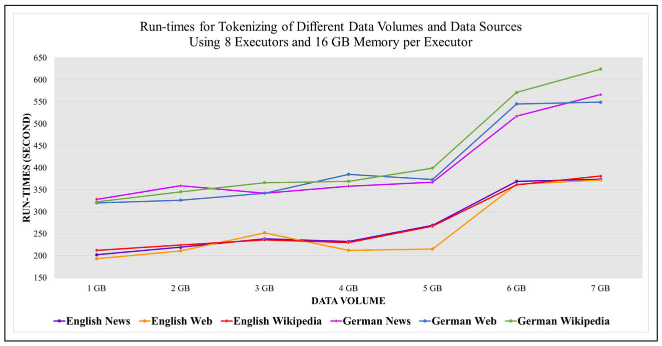


Figure 10. Run-times for tokenizing 1 to 7 GB text materials from Newspapers, Wikipedia, and General Web documents in English and German, using 8 executors and 16 GB memory per executor.

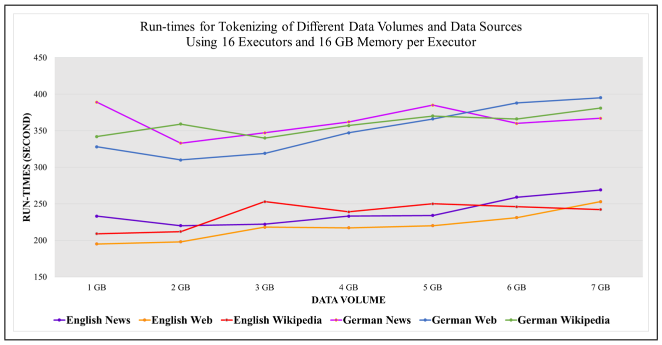


Figure 11. Run-times for tokenizing 1 to 7 GB text materials from Newspapers, Wikipedia, and General Web documents in English and German, using 16 executors and 16 GB memory per executor.

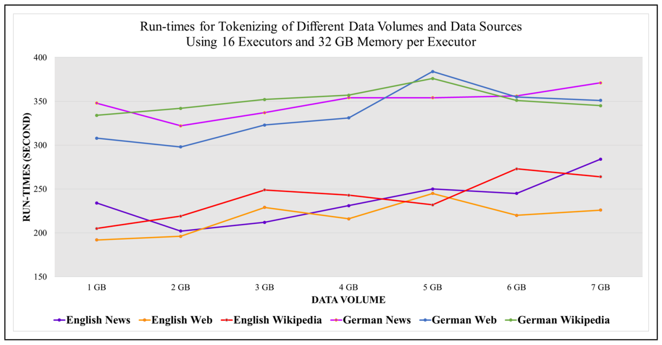


Figure 12. Run-times for tokenizing 1 to 7 GB text materials from Newspapers, Wikipedia, and General Web documents in English and German, using 16 executors and 32 GB memory per executor.

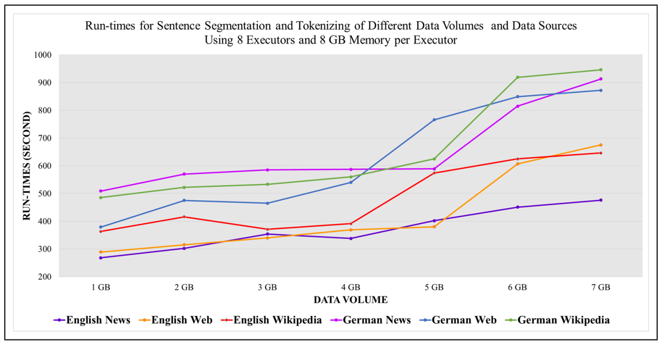


Figure 13. Run-times for segmenting and tokenizing 1 to 7 GB text materials from Newspapers, Wikipedia, and General Web documents in English and German, using 8 executors and 8 GB memory per executor.

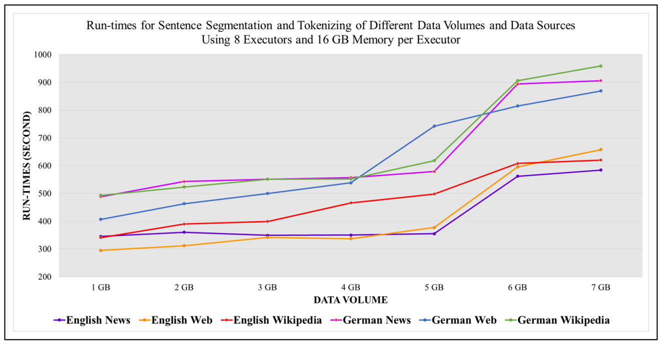


Figure 14. Run-times for segmenting and tokenizing 1 to 7 GB text materials from Newspapers, Wikipedia, and General Web documents in English and German, using 6 executors and 16 GB memory per executor.

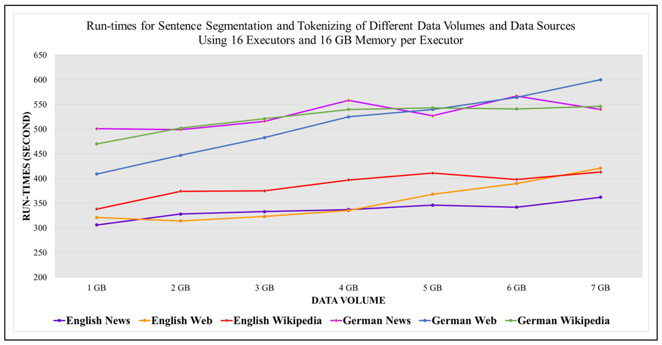


Figure 15. Run-times for segmenting and tokenizing 1 to 7 GB text materials from Newspapers, Wikipedia, and General Web documents in English and German, using 16 executors and 16 GB memory per executor.

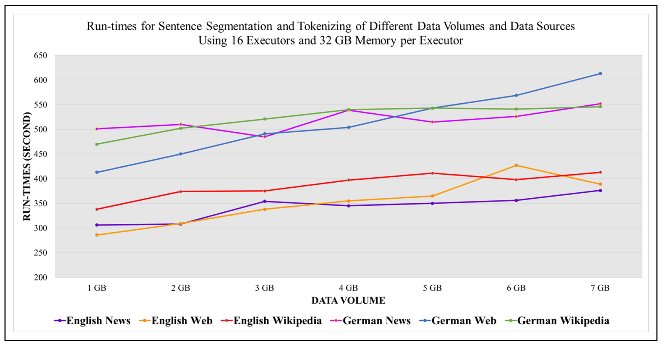
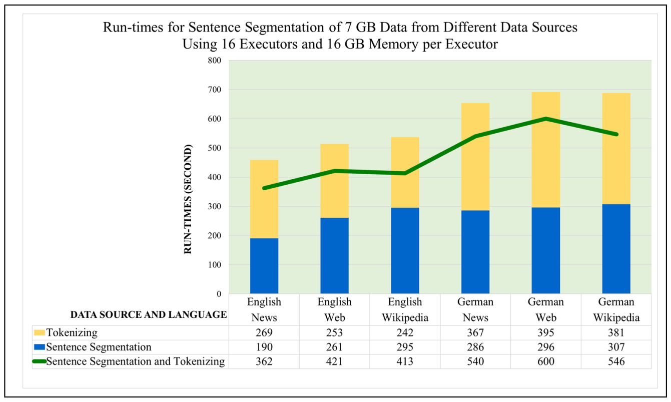
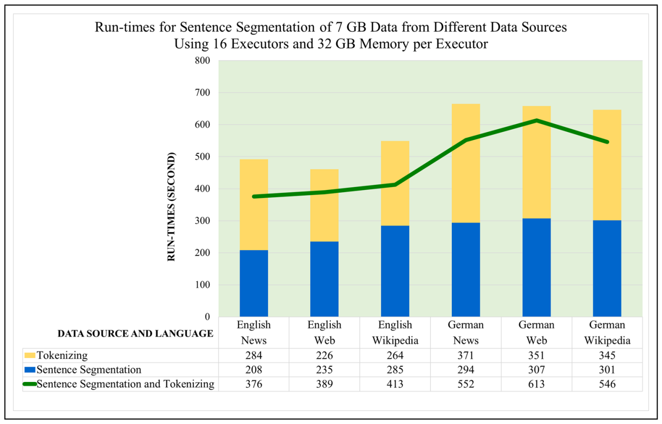


Figure 16. Run-times for segmenting and tokenizing 1 to 7 GB text materials from Newspapers, Wikipedia, and General Web documents in English and German, using 16 executors and 32 GB memory per executor.



DATA SOURCE AND LANGUAGE	English News	English Web	English Wikipedia	German News	German Web	German Wikipedia
Tokenizing	269	253	242	367	395	381
Sentence Segmentation	190	261	295	286	296	307
Sentence Segmentation and Tokenizing	362	421	413	540	600	546

Figure 17. Run-times for applying segmentation, tokenization, and segmentation and tokenization combined as a toolchain on 7 GB text materials from Newspapers, Wikipedia, and General Web documents in English and German, using 16 executors and 16 GB memory per executor.



DATA SOURCE AND LANGUAGE	English News	English Web	English Wikipedia	German News	German Web	German Wikipedia
Tokenizing	284	226	264	371	351	345
Sentence Segmentation	208	235	285	294	307	301
Sentence Segmentation and Tokenizing	376	389	413	552	613	546

Figure 18. Run-times for applying segmentation, tokenization, and segmentation and tokenization combined as a toolchain on 7 GB text materials from Newspapers, Wikipedia, and General Web documents in English and German, using 16 executors and 32 GB memory per executor.

8 GB RAM and 2795 seconds using 16 GB RAM on a single node. Adding a second executor decreases the run-time to 2115 respectively 1480 seconds.

The typical trend can be seen again where run-times decrease significantly up to (around) 7 assigned executors, and with no improvements when allocating 14 executors or more.

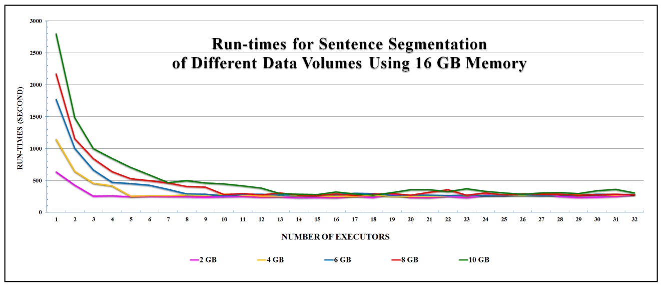
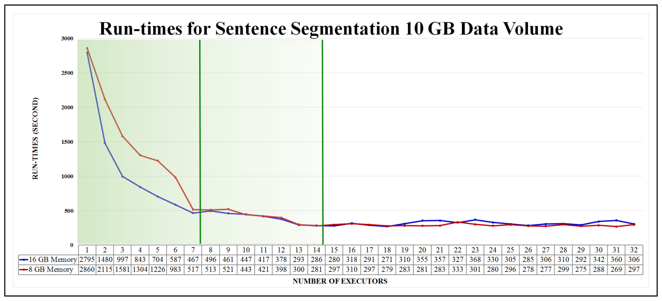


Figure 19. Run-times for different number of executors and data volumes using 16 GB memory per executor for sentence segmentation.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
16 GB Memory	2795	1480	997	843	704	587	467	496	461	447	417	378	293	286	280	318	291	271	310	355	357	327	368	330	305	285	306	310	292	342	360	306
8 GB Memory	2860	2115	1581	1304	1226	983	517	513	521	443	421	398	300	281	297	310	297	279	283	281	283	333	301	280	296	278	277	299	275	288	269	297

Figure 20. Run-times for different numbers of executors, illustrating different ”speedup areas”.

VIII. Discussion

Execution times are valuable information that can be utilized for the estimation of times of arrival for annotation jobs in NLP toolchains. Measured execution times give the opportunity to configure a cluster dynamically based on expected response times, available resources and the current load by a varying number of parallel users or jobs. For instance, if there are x free resources available on the cluster and a processing job requires $x+y$ resources, the new job may be scheduled to be executed after finishing the first running job which, has allocated at least y resources.

Furthermore, execution times are relevant for estimating an ”optimal” resource allocation for every individual tool. In the context of this contribution, these resources include the number of executors and the amount of memory which, can be assigned to each task. Obviously, the term ”optimal” is a very ambiguous one: it depends on the context of which value should be actually optimized. In this context, this may be the overall run-time of a job (i.e., a user-oriented view), the amount of allocated resources (i.e., a cost-oriented view) or a combination of both (by finding some balance between both).

By allocating more executors, execution times can be decreased. At a certain point (which may depend on a variety of parameters), assigning more resources will have no positive effect on execution times anymore. This point can be seen as the optimal configuration for the particular task in respect of optimized run-times, and contains the amount of resources which, are required to generate a result in the shortest possible execution time. In this situation, it is also feasible to generate results by assigning fewer resources – with the drawback of extended processing times – but it is obviously not reasonable to assign more resources to the job. As an example, in Figure 20 the fastest configuration for sentence segmentation of 10 GB text data consists of 14 executors with 16 GB RAM per executor where assigning more resources generates more costs without providing faster execution.

The extracted information helps to provide different resource configurations in accordance with the available hardware resources and desired response times for the user's requested service and input material. For instance, if a user wants to segment 10 GB text material in less than 25 minutes, 3 executors with 16 GB RAM or 4 executors with 8GB RAM would be both suitable configurations. In contrast, for a response time of up to 5 minutes, a configuration consisting of at least 14 executors with 8 or 16 GB RAM would suffice. In an environment where accounting of actual expenses is included, the balance between technical or financial costs and acceptable run-times can also be delegated to the user. In such an environment, a user can choose the desired configuration considering estimated run-times and incurred expenses.

The presented diagrams also show that for particular configuration changes resulting improvements of run-time are only marginal. Especially in case of limited available resources or unexpected usage peaks, these configurations do not have to be available anymore as their effect from the user's perspective are small. For instance, in Figure 20 assigning 7 executors with 16 GB RAM generates the expected result in 467 seconds whereas doubling the number of executors leads only to an execution time of 286 seconds (i.e., a 39% run-time reduction).

The diagrams also illustrate the impact of the language and source of the documents on the response time. In general, for the considered tools and inputs, German texts required more processing time than same-sized English texts. As a more specific outcome, segmenting newspaper texts took less time for both English and German documents compared with other sources of origin.

The reasons are related to different aspects. For example, texts that are published in newspapers typically follow high standards of writing rules using the correct punctuation marks which, are the base for segmenting the data, whereas general web documents and Wikipedia text are less likely to use adequate punctuation marks. Furthermore, the applied rule-based approaches use different numbers of more or less complex rules for different languages and the amount of considered external resources (like the number of multi-word units for a tokenization task) also varies between languages. Language-specific invariants (like the average length of sentences, average number of words per sentence etc.) must also be considered [35].

In addition, the results illustrate that the local combination of tools can decrease execution times significantly. This reduction is achieved by eliminating redundant steps such as reading common rule sets and abbreviation files and, more importantly, removing read/write operations between each of the tools. In our experiments, combining the sentence segmentation and tokenization tasks can decrease the whole execution times by about 25% (Figures 17 and 18). Obviously, this effect is much larger in a distributed environment where format conversion and network transmission times have to be taken into account as well.

IX. CONCLUSION AND OUTLOOK

In this contribution, we described some prerequisites for providing JIT delivery in service-oriented research infrastructures using typical NLP tasks as an example. We have utilized Apache Spark as execution engine on an Apache Hadoop cluster to allow parallel processing of large text collections and to increase the reliability and predictability of the services. An evaluation of required resources for processing different amounts of text offers information about possible hardware configurations in form of a ”performance catalog” that is useful for estimating delivery times and potential expenses for each task.

For more meaningful results, the sketched experiments have to be extended for more languages. In the context of this article German and English documents were used as an example but processing documents in other languages will provide more clear information about the effect of specific languages (or dialects) on resulting response times.

Naturally, providing and maintaining such resources and tools lead to actual financial costs. In commercial platforms, like Amazon Comprehend [27] or Google Cloud NLP [28] these costs are covered by contracts with costumers based on defined parameters (kind of service, required availability, costs of data storage, CPU cycles, etc.). The selected configuration and execution time can be used as a basis for an accounting system which, relies on well-founded expenses for every individual NLP job.

The presented run-times in this abstract can only be a part of a qualified assessment of NLP tasks. Performance profiles require a variety of training cycles to be meaningful and to cover all kinds of input material and their effects on the assessed tool. Furthermore, measuring actual response times for larger toolchains in text-oriented research infrastructures is more complex and needs to take more parameters into account. This is especially relevant for toolchains where multiple service providers are used. Other relevant parameters, like data transfer times between user and service provider or between different services, required format conversions, or similar tasks were not considered here.

Acknowledgement

The evaluation for this paper was performed using resources of Leipzig University Computing Center.

References

[1] S. Sahami and T. Eckart, “Just-In-Time Delivery for NLP Services in a Web-Service-Based IT Infrastructure,” in Workshop NLP-adapt: Adaptive Workflow Management for NLP Data Processing at ADAPTIVE 2019: The Eleventh International Conference on Adaptive and Self-Adaptive Systems and Applications, Venice, Italy, 2019, pp. 103–106. [Online]. Available: https://www.thinkmind.org/index.php?view=article&articleid=adaptive_2019_7_20_58002

[2] F. W. McFarlane, Information technology changes the way you compete. Harvard Business Review, Reprint Service, 1984.

[3] E. Hinrichs and S. Krauwer, “The CLARIN Research Infrastructure: Resources and Tools for e-Humanities Scholars,” Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC-2014), May 2014, pp. 1525–1531. [Online]. Available: http://dspace.library.uu.nl/handle/1874/307981

[4] J. Edmond, F. Fischer, M. Mertens, and L. Romary, “The DARIAH ERIC: Redefining Research Infrastructure for the Arts and Humanities in the Digital Age,” ERCIM News, no. 111, Oct. 2017. [Online]. Available: https://hal.inria.fr/hal-01588665

[5] EOSC, “EOSC European Open Science Cloud,” Online, 2019, Date Accessed: 4 Dec 2019. URL https://www.eosc-portal.eu.

[6] C. Kuras, T. Eckart, U. Quasthoff, and D. Goldhahn, “Automation, management and improvement of text corpus production,” in 6th Workshop on the Challenges in the Management of Large Corpora at the 11th Language Resources and Evaluation Conference (LREC 2018), Miyazaki (Japan), 2018.

[7] H. Wildemann, Das Just-in-time-Konzept: Produktion und Zulieferung auf Abruf, 5th ed., ser. TCW. Mnchen: TCW Transfer-Centrum fr Produktions-Logistik und Technologie-Management, 2001, vol. 4.

[8] P. Blais, “How the information revolution is shaping our communities,” Planning Commissioners Journal, vol. 24, 1996, pp. 16–20.

[9] T. Ohno, Toyota Production System: Beyond Large-Scale Production. Taylor & Francis, 1988.

[10] P. Heck and A. Zaidman, “Quality criteria for just-in-time requirements: just enough, just-in-time?” in 2015 IEEE Workshop on Just-In-Time Requirements Engineering (JITRE). Los Alamitos, CA, USA: IEEE Computer Society, aug 2015, pp. 1–4. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/JITRE.2015.7330170

[11] IBM, “Web services architecture overview,” Online, 2019, Date Accessed: 4 Dec 2019. URL https://www.ibm.com/developerworks/web/library/w-ovr/.

[12] E. W. Hinrichs, M. Hinrichs, and T. Zastrow, “WebLicht: Web-Based LRT Services for German,” in Proceedings of the ACL 2010 System Demonstrations, 2010, pp. 25–29, Date Accessed: 4 Dec 2019. URL http://www.aclweb.org/anthology/P10-4005.

[13] “ISO - Popular Standards,” Online, 2019, Date Accessed: 4 Dec 2019. URL https://www.iso.org/standard/71952.html.

[14] S. G. Manikandan and S. Ravi, “Big data analysis using Apache Hadoop,” in 2014 International Conference on IT Convergence and Security (ICITCS). IEEE, 2014, pp. 1–4.

[15] S. Sagiroglu and D. Sinanc, “Big data: A review,” in 2013 International Conference on Collaboration Technologies and Systems (CTS). IEEE, 2013, pp. 42–47.

[16] A. S. Tanenbaum and M. Van Steen, Distributed systems: principles and paradigms. Prentice-Hall, 2007.

[17] F. Wang, J. Qiu, J. Yang, B. Dong, X. Li, and Y. Li, “Hadoop high availability through metadata replication,” in Proceedings of the first international workshop on Cloud data management. ACM, 2009, pp. 37–44.

[18] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth et al., “Apache Hadoop Yarn: Yet another resource negotiator,” in Proceedings of the 4th annual Symposium on Cloud Computing. ACM, 2013, p. 5.

[19] ApacheHadoop, “Apache Hadoop Documentation,” Online, 2019, Date Accessed: 4 Dec 2019. URL http://hadoop.apache.org.

[20] K. Shvachko, H. Kuang, S. Radia, R. Chansler et al., “The Hadoop distributed file system,” in MSST, vol. 10, 2010, pp. 1–10.

[21] M. Zaharia, R. S. Xin, P. Wendell, T. Das, M. Armbrust, A. Dave, X. Meng, J. Rosen, S. Venkataraman, M. J. Franklin et al., “Apache Spark: a unified engine for big data processing,” Communications of the ACM, vol. 59, no. 11, 2016, pp. 56–65.

[22] M. Hamstra and M. Zaharia, Learning Spark: lightning-fast big data analytics. O’Reilly & Associates, 2013.

[23] S. Sahami and T. Eckart, “Spark WebLicht Webservices,” Online, 2019, Date Accessed: 4 Dec 2019. URL http://hdl.handle.net/11022/0000-0007-CA50-B.

[24] L.-P. Meyer, J. Frenzel, E. Peukert, R. Jäkel, and S. Kühne, “Big Data Services,” in Service Engineering. Springer, 2018, pp. 63–77.

[25] L.-P. Meyer, “The Galaxy Cluster,” Online, 2018, Date Accessed: 4 Dec 2019. URL https://www.scads.de/de/aktuelles/blog/264-big-data-cluster-in-shared-nothing-architecture-in-leipzig.

[26] D. Goldhahn, T. Eckart, and U. Quasthoff, “Building Large Monolingual Dictionaries at the Leipzig Corpora Collection: From 100 to 200 Languages,” in Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC’12), 2012.

[27] Amazon, “Amazon Comprehend - Pricing,” Online, 2019, Date Accessed: 4 Dec 2019. URL https://aws.amazon.com/comprehend/pricing/.

[28] Google, “Pricing - Natural Language API - Google Cloud,” Online, 2019, Date Accessed: 4 Dec 2019. URL https://cloud.google.com/natural-language/pricing.

[29] IBM, “Watson natural language understanding,” Online, 2019, Date Accessed: 4 Dec 2019. URL https://www.ibm.com/cloud/watson-natural-language-understanding.

[30] “WebLicht,” Online, 2019, Date Accessed: 4 Dec 2019. URL https://weblicht.sfs.uni-tuebingen.de.

[31] “The language application grid— a web service platform for natural language processing development and research,” Online, 2019, Date Accessed: 4 Dec 2019. URL https://www.lappsgrid.org/.

[32] “Gate cloud: Text analytics in the cloud,” Online, 2019, Date Accessed: 4 Dec 2019. URL https://cloud.gate.ac.uk/.

[33] K. Thießen, A. Al-Ali, and J. Puchta, "Funktionsumfang und Leistungsfähigkeit kommerzieller und wissenschaftlicher NLP-Plattformen im Web," Seminar work, 2019.

[34] S. Sahami, T. Eckart, and G. Heyer, "Using Apache Spark on Hadoop Clusters as Backend for WebLicht Processing Pipelines," in CLARIN Annual Conference 2018 in Pisa, Italy, 2018.

[35] T. Eckart and U. Quasthoff, Statistical Corpus and Language Comparison on Comparable Corpora. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 151–165. [Online]. Available: https://doi.org/10.1007/978-3-642-20128-8_8