Just-In-Time Delivery for NLP Services in a Web-Service-Based IT Infrastructure

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*—Just-In-Time delivery of resources is a standard procedure in the industrial production of goods. The reasons for introducing this paradigm and its potential benefits are in large parts applicable for the area of web-based Natural Language Processing Services as well. This contribution focuses on prerequisites and potential outcomes of a Just-In-Time-capable infrastructure of Natural Language Processing services using an example service. The benefits of such an endeavour are sketched with a focus on the ongoing development of large scale service delivery platforms like the European Open Science Cloud or similar projects.**

Keywords–NLP Services; Research Infrastructure; Just-In-Time Delivery; Cluster Computing.

I. INTRODUCTION

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 [1].

The use of linguistic 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 [2] or DARIAH [3], or cross-domain projects, like the European Open Science Cloud (EOSC) [4], 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 is 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 [5].

One of the potential outcomes and goals of a more "industrialised" 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, like reduced response times or reduction of required storage facilities [6]. 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 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 [7]. In IT infrastructures, utilising 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 utilised 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 the 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 industrialised IT infrastructures.

In this contribution, we present an example of a Natural Language Processing (NLP) service with a focus on its transparency regarding execution times and required resources. As a result, valid resource configurations can be chosen considering available resources and expected delivery times.

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 technical approaches. Sections IV and V illustrate and discuss the outcomes and results and are followed by a brief conclusion of this contribution in Section VI.

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 [8] 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.

 ISBN: 978-1-61208-706-1

Established requirements deal with all kinds of aspects in the organisational, legal and technical environment of companies and organisations that are involved in the overall process. At least a subset of those requirements, is directly transferable to activities in IT processes and infrastructures [1], 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. [9]).

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 runtime*, 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 [10]).

Many essential requirements for a JIT integration are already handled in existing frameworks – for example CLARIN's WebLicht [11] –, 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 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 catalogue. This contains the identification of all relevant parameters of a tool and well-founded empirical knowledge about their effects on the runtime 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 minimising (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 sketch its benefits.

III. Technologies and tools

In this section, we explain the chosen technologies and their specific features relevant for the context of this contribution. Afterwards, the implemented NLP tools and utilised resources are described.

A. Technical Approaches

For services where response times are a critical subject, the utilised technology should support technical features like fault-tolerance and high availability. Being fault-tolerant relies on the ability of the system to detect a hardware fault and immediately switch to a redundant hardware component. High availability systems refer to architectures that are able to operate continuously without failure during a specific time period. For this contribution, we have selected Apache Hadoop clusters and the Apache Spark execution engine to address the topic of fault tolerance and high availability. Furthermore, this approach supports the parallelisation of tasks to improve response times significantly.

Apache Hadoop is a popular framework to process large-scale data in a distributed computing environment. Its large ecosystem consists of the Hadoop Kernel, MapReduce, HDFS and some other components [12]. Hadoop Distributed File System (HDFS) as a highly fault-tolerant distributed storage system is able to handle the failure of storage infrastructure without losing data by storing three complete copies of each block of data redundantly on three different nodes [13].

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 tasks on several executors improves processing times and fault tolerance. Apache Spark uses Resilient Distributed Dataset (RDD), a data-sharing abstraction, which is designed as fault-tolerant collections and is capable to recover lost data after a failure using the lineage approach: during the construction of an RDD, Spark keeps the graph of all transformations. In the event of a failure, it re-runs all failed operations to rebuild lost results. The RDDs are persisted and executed completely in RAM – In-Memory Databases (IMDB) –, therefore generating and rewriting the recovered data is a fast process [14] [15].

B. 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 [16]. These tools use Apache Hadoop as their framework, Apache Spark as execution engine and HDFS for storing input data and outputs. These tools are atomic services that can be integrated into any SOA-based annotation environment.

In order to have an accurate estimation of execution times, a variety of benchmarks were carried out for all implemented tools. As an example, the duration of sentence segmentation for datasets of German documents with sizes from 1 to 10 Gigabytes was evaluated using different cluster configurations. The cluster configuration varied in the number of assigned executors (1 to 32 nodes) and allocated memory per executor

 ISBN: 978-1-61208-706-1

(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.

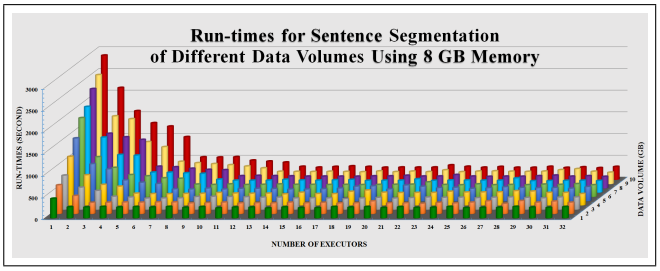

Figure 1. Run-times for segmenting 1 to 10 GB text materials using 1 to 32 executors and 8 GB memory per executor.

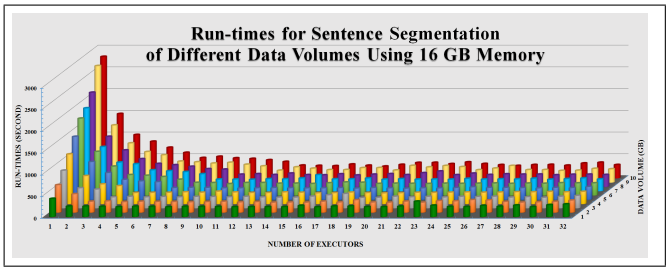

Figure 2. Run-times for segmenting 1 to 10 GB text materials using 1 to 32 executors and 16 GB memory per executor.

These tests were carried out using a cluster provided by the Leipzig University Computing Center [17]. The cluster consists of 90 nodes each with 6 cores and 128 GB RAM. The cluster provides more than 2 PB storage in total and is connected via 10 Gbit per second Ethernet [18].

IV. Results

As Figures 1 and 2 illustrate, run-times vary for different job configurations. As expected, using only a single executor – therefore executing the job without any parallelisation on the cluster – shows the maximum run-time for every data volume. The outcomes of all conducted tests indicate that execution times can be improved with extended hardware resources (i.e. more executors). This improvement complies with the expected behaviour of parallel processing: a sharp decrease in execution time by increasing the assigned resources, followed by a smoother reduction and finally no significant improvement when adding more resources to the job does not improve the speedup anymore. The results show this consistent behaviour for different data volumes using various cluster configurations. Figure 3 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.

Figure 4 shows the results for sentence segmentation of 10 GB text material which requires 2860 seconds using 8 GB RAM and 2795 seconds using 16 GB RAM on a single node, where adding a second executor decreases the run-time to 2115 respectively 1480 seconds.

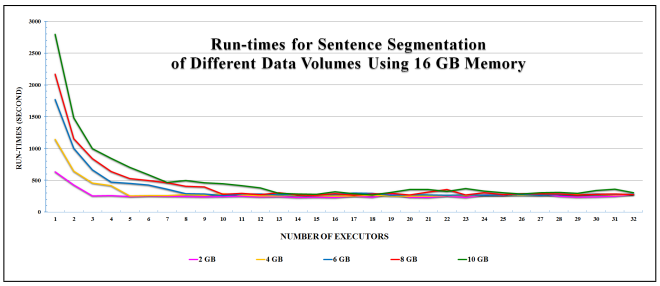

Figure 3. Run-times for different number of executors and data volumes using 16 GB memory per executor

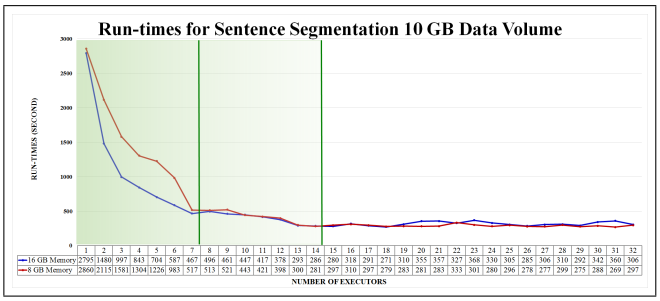

Figure 4. Run-times for different numbers of executors, illustrating different ”speedup areas”

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.

V. Discussion

Execution times are valuable information that can be utilised for the estimation of times of arrival for annotation jobs in NLP toolchains. Measured execution times give the opportunity to configure the 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, they are also relevant for estimating an ”optimal” resource allocation for each 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 optimised. 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 optimised 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

 ISBN: 978-1-61208-706-1

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 4 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 4 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).

VI. CONCLUSION

In this contribution, we described some prerequisites for providing JIT delivery in service-oriented research infrastructures using a typical NLP task as an example. We have utilised 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 that is useful for estimating delivery times and potential expenses for each individual task.

Naturally, providing and maintaining such resources and tools lead to actual financial costs. In commercial platforms, like Amazon Comprehend [19] or Google Cloud NLP [20] 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 each 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

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

REFERENCES

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

[2] 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

[3] 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

[4] EOSC, "EOSC European Open Science Cloud," Online, 2019, Date Accessed: 3 Apr 2019. URL https://www.eosc-portal.eu.

[5] 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.

[6] H. Wildemann, Das Just-in-time-Konzept: Produktion und Zulieferung auf Abruf, 3rd ed. gfmt, 1992.

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

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

[9] 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

[10] IBM, "Web services architecture overview," Online, 2019, Date Accessed: 3 Apr 2019. URL https://www.ibm.com/developerworks/web/library/w-ovr/.

[11] 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: 3 Apr 2019. URL http://www.aclweb.org/anthology/P10-4005.

[12] ApacheHadoop, "Apache Hadoop Documentation," Online, 2019, Date Accessed: 3 Apr 2019. URL http://hadoop.apache.org.

[13] H. S. Bhosale and D. P. Gadekar, "A review paper on Big Data and Hadoop," International Journal of Scientific and Research Publications, vol. 4, no. 10, 2014, pp. 1–7.

[14] 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.

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

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

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

[18] "the galaxy cluster."

[19] Amazon, "Amazon Comprehend - Pricing," Online, 2019, Date Accessed: 4 Apr 2019. URL https://aws.amazon.com/comprehend/pricing/.

[20] Google, "Pricing - Natural Language API - Google Cloud," Online, 2019, Date Accessed: 4 Apr 2019. URL https://cloud.google.com/natural-language/pricing.

 ISBN: 978-1-61208-706-1