Master Thesis
Computer Science

Scaling UIMA

Simon Gehring

Am Jesuitenhof 3
53117 Bonn
gehring@uni-bonn.de
Matriculation Number 2553262

At the
Rheinische Friedrich-Wilhelms-Universität Bonn

supervised by
Prof. Dr. Heiko Röglin and Dr. Timm Heuss

August 30, 2018

Contents

Abstract

In this thesis, we will evaluate different means of scaling UIMA (Unstructured Information Management Architecture), using modern technologies like Docker, a container virtualization solution, and Apache Spark, a cluster computing framework. We will compare said implementations with the native UIMA-AS (UIMA Asynchronous Scaleout) approach in terms of processor and memory efficiency, ease of implementation and maintainability. The evaluation will be based on a specific scenario, however it will be easily configurable by exchanging very few lines of code.

Update, was wir eigentlich evaluaten, sobald das Evaluation chapter fertig ist.

Chapter 1

Introduction

Natural language is most commonly used to transmit information human-to-human. While most of this interaction takes place orally or written on paper, the digital revolution and the rise of social media increased the amount of digitally stored natural language tremendously. Gantz and Reinsel predicted 2012 that the amount of digital data stored globally will double about every two years until at least the year 2020 [GR12].

https://www.statista.com/statistics/759234/worldwide-natural-languag zeigt das sehr schön für die Zukunft. Brauche einen Premiumzugang.

Many opportunities arise from this amount of digital data, specifically in the field of machine learning. In 2011, IBM's QA (Question Answering) system "Watson" famously outmatched professional players in the quiz show "Jeopardy!" [Fer12, ESI+12]. Kudesia et al. proposed 2012 an algorithm to detect so called CAUTIs[1], common hospital-acquired infections, by utilizing a NLP (Natural Language Processing) analysis with precomputed language models on the medical records of patients [KSDG12].

1.1 Motivation

Platzhalter für motivierende Statistiken. Idee: NLP benötigt große Daten, was im Grunde BigData ist (Größenordnung Peta oder Exabyte.) Muss entsprechend skaliert werden. Daten von Statista finden, eventuell sogar Graphen eventuell sogar Unternehmensdaten(?)

Natural language is inherently unstructured and hardly machine readable. Even a seemingly easy task, like separating a sentence into words is still an ongoing research topic [PT18].

Since many NLP frameworks like Stanford's Core NLPSuite [MSB+14], The Natural Language Toolkit [BL04] and Apache OpenNLP [Apaa] exist, there was a need for generalizing the NLP approach. In 1995, the University of Sheffield began developing GATE (General Architecture for Text Engineering), a graphical development application for generic NLP problems [CMBT02]. Aside of algorithms for typical NLP tasks like tokenization, sentence splitting or part of speech tagging, GATE provides a graphical interface for developing custom algorithms in form of plugins, which can use the

[1] Catheter-associated Urinary Tract Infections

results of previous NLP analyses. However, the possibilities for scaling GATE applications are sparse. In 2012, GATECloud.net launched, a proprietary, cloud based GATE computation interface [TRCB13].

In [FL04], Ferrucci and Lally introduced UIMA, another general purpose NLP framework. Initially developed by IBM, UIMA has been open-source and is maintained by Apache since 2006. UIMA itself provides no built-in NLP analyses, but offers a common analysis structure among its plugins, which is used to combine different NLP approaches into one single framework. A popular implementation example of UIMA is IBM's QA system "Watson", which famously outmatched professional players in the quiz show "Jeopardy!" in 2011 [Fer12, ESI+12]. For this matter, UIMA was configured to run on thousands of processor cores to achieve a feasible reaction time [ESI+12]. Although this example seems to demonstrate that Apache UIMA is indeed very scalable, it has its drawbacks. The native UIMA scaling framework UIMA-AS is configured by XML (Extensible Markup Language) files, which are difficult to maintain. Furthermore it does not work out-of-the-box with other UIMA derivatives like UIMAfit.

In this thesis, we will implement and evaluate a scale-out framework for UIMA, which utilizes modern technology to ensure maintainability and scalability.

1.2 Implementation Requirements

The implementation should meet a number of requirements. First and foremost, the underlying framework, Apache UIMA, must not be limited in functionality. Since one of UIMAs strengths is the modularity and ease of plugin development, the scale-out framework to implement is required to work with native UIMA classes without any restriction. While this requirement sounds easy to meet, it is actually quite limiting since UIMA plugins can be arbitrary Java-Code. Without special care, such code is not necessarily thread-safe and thus can not be safely executed multiple times in parallel within one JVM (Java Virtual Machine).

The metrics, the scale-out framework should optimize, include:

- CPU Usage
- RAM Usage
- Document Throughput
- Byte Throughput
- Maintainability
- Implementability
- Code Quality

With CPU and RAM Usage being the percentage of actually used (virtual or real) resources, higher values being more preferable. Even if many of those resources go into

Ich sollte hier mal nach Quellen gucken. Ist zwar meist selbsterklärend, aber ein paar Schaden nicht.

administrative tasks, a scaling framework should use as many of the available resources as possible when faced with a large list of parallel jobs. Thus, a value near one should be aimed for. Obviously, at the same time, resource allocation when idling should be as low as possible.

Kam mir gerade die Idee. UIMA-AS deployed Services, die immer da sind (ie deren Modelle immer im RAM sind). Spark schießt alles sofort wieder ab, sobald es nicht wieder benötigt wird.

A little more obvious metric is the achieved throughput of data. Since collections of input documents can be shaped very differently, both, a high document and byte throughput are aimed for. When handling small documents, maybe even single words or sentences, the byte throughput is very limited to the actual input size and the large number of documents is responsible for the size of the input data collection. In such a situation, a high document throughput would be preferred to a high byte throughput. On the other hand, on larger documents, a high byte throughput is the more accurate metric since it is independent from the individual size of the input documents.

The framework is aimed to work in large academic but also industry compliant environments. Therefore the maintainability is important. It describes the amount of effort to maintain any current usage of the framework, for example changing the underlying NLP algorithm, modifying the hardware setup or making configuration changes. This is inherently more complicated by the genericness of UIMA, which allows for sophisticated plug-in initialization and configuration logic, making on-the-fly changes more difficult.

TODO: Alle plugins in plug-ins ändern

Furthermore the framework should be easy to utilize. Code that already has been written for single threaded execution should be easily reusable. This is especially important for UIMA since large repositories of plug-ins like the DKPro Core (Darmstadt Knowledge Processing Software Repository) [DKP] already exist and are infeasible to be rewritten.

Source? Keine gefunden ;_;

Equally important as ease of utilization is the longevity of the framework. Ensuring a maximum of Code Quality is necessary to avoid large refactorings and API (Application Programming Interface) changes in the near future. In a non-academic environment, applications often have to last for a long time before replacement. Thus, a robust underlying code is aimed for.

Code Quality Analyzers finden und citen

Obviously the last three bullet points, Maintainability, Implementability and Code Quality, are not easily measured, since those are subject to individual perception. However both, Maintainability and Implementability can be measured in LoC (Lines of Code). There are many static code quality analyzers, which output a score, or at least a count of quality issues. Such a score can be used to measure the quality of the frameworks code.

1.3 Outline

First, in Chapter 2, the functionality of UIMA is detailed, especially with focus on distributed computing and scaling. For this matter, both native UIMA scaling frameworks, UIMA-AS and CPE (Collection Processing Engine) are subject in said chapter. Also Apache Spark, a cluster-computing framework, will be briefly explained since it will form the foundation of the frameworks scaling capabilities. Lastly, approaches like Leo and v3NLP will be introduced. Those are also UIMA based scaling solutions, which both suffer from different problems.

Chapter 3 will explain the scaling framework in detail. It starts with the choice of technology and proceeds to the implementation. This includes macroscopic network views and microscopic code details.

Refinen, sobald Implementation geschrieben ist.

Then, Chapter 4 deals with the results of the framework implementation. For this, the requirements given in Section 1.2 are evaluated against the framework, UIMA-AS and the single-threaded approach.

Lastly, Chapter 5 summarizes the results, including possible limitations of the implementation. It also gives an outlook on how to extend and publicize the framework to the general public.

Chapter 2

Basics

Es gibt haufenweise Konzepte, die ich für das Framework, allerdings noch eher für das Testen und Benchmarken verwendet habe. Diese muss ich natürlich alle (oder zumindest die meisten) vorstellen.

Ganz oben dabei ist natürlich UIMA, wobei das eigentlich schon extrem in der Introduction benutzt wird. Ich mag es nicht jetzt erst auf UIMA einzugehen. Aber was will man machen? Irgendeine Struktur braucht man ja.

Danach kommt Spark. Da ist auf dieser Seite eher B̈enutzeräls Developer bin, werde ich vermutlich nicht besonders tief in die Materie eingehen. Ich denke ich werde nicht bis zum Map-Reduce kommen. Und wenn dann werde ich das nur anschneiden und auf Quellen verweisen. Die internen Spark-Konzepte sind nunmal nicht wirklich wichtig für die Implementierung.

Kommen wir zu dem Versuchsaufbau, kommt natürlich Docker ins Spiel. Docker spielt natürlich im Development (schnelles Deployment, etc.) eine Rolle, hat aber bei mir außerdem die Rolle des VM-Ersatzes übernommen. Wie wichtig Docker für den gesamten Versuch war, lässt sich leicht in den Docker- und Composefiles lesen. Diese garantieren darüber hinaus natürlich Vergleichbarkeit.

Trotzdem sind die underlying Konzepte des Docker-Universums nicht allzu wichtig für das Framework an sich. Deployed wird es (da es sich ja nur um eine library handelt und nicht standalone arbeitet) sowieso nur über Maven. Ich plane daher auch eine mögliche Docker-Section nicht allzu groß zu gestalten.

Ich habe es zwar nicht benutzt, aber eine kleine Einführung in HDFS könnte nützlich sein. Ich werde häufiger (im Kontext von BigData) anmerken, dass Daten vermutlich von einem verteiltem Dateisystem, etwa einem HDFS kommen und dahin geschrieben werden.

2.1 UIMA-Family

Unbedingt bilder hinzufügen und establishen was eine Pipeline ist!!1!

2.1.1 Apache UIMA

Apache UIMA is one of few general approaches to implement NLP solutions. With a very modular architecture, UIMA is a popular tool that can easily be applied to a majority of NLP problems. A large part of the popularity of UIMA stems from the large DKPro Core collection of components, containing hundreds of analysis modules and precomputed language models [EdCG14], which are easily imported into existing Java projects with the build automation tool Apache Maven [DKP].

A common problem with UIMA is scaling [DCR+15, ESI+12, RBJB+10]. UIMA itself provides two distinct interfaces to analyze larger collections of unstructured data, with one being UIMA-AS and the other being the more dated and less flexible CPE [FLVN09].

2.1.2 UIMAfit

2.1.3 UIMA-CPE

UIMA CPE ist der Vorgänger von UIMA-AS und basiert im Grunde darauf, dem User vollständige Macht zu geben um die Skalierung zu bewerkstelligen. Das geht natürlich vollkommen nach hinten los, da Dinge wie Cas Initializers nicht trivial zu konfigurieren sind. Außerdem musste sich der User selbstständig um Dinge wie reconfigure() und typeSystemInit() kümmern. Natürlich basiert CPE auch auf XML-Deskriptoren, für die kein UIMAfit/Leo existiert.

2.1.4 UIMA-AS

Das bisherige non+ultra. Man deployed pipelines (oder einzelne engines) als Services, die sich am broker registrieren. Anfragen werden an den Broker geschickt, der sie dann weiter sendet. Je nach Broker-Implementierung kann man hier sehr schön resilience zu bauen. Es gibt hier ein paar Dinge zu beachten, was thread-safety angeht. Außerdem ist fraglich ob UIMA-AS tatsächlich uneingeschränkt skalierbar ist, da der CollectionReader möglicherweise einen bottleneck darstellt. Problematisch ist in jedem Fall die Tatsache, dass das gesamte CAS wieder zurück geschickt wird. Das ist möglicherweise gar nicht gewollt, da in der Pipeline bereits consumer bereit stehen, die Dinge in Datenbanken etc schreiben.

Das ist ein wenig die Verteidigung für mein System, bei dem es absolut grauenvoll ist, wenn man die CAS wieder zurück schickt :D

Hier bietet sich auch ein Schaubild an.

2.2 Apache Spark

2.3 Related Work

Hier im Grunde alles was es an "Vorarbeiten" bzw. konkurrierende Ansätze gibt.

Ich bin sehr unzufrieden mit der related work in den Basics. Es passt hier absolut nicht rein. Ich werde das vermutlich bald wieder in das erste Kapitel Introduction schieben.

2.3.1 Leo

Leo ist für UIMA-AS was UIMAfit für UIMA ist. Leider leidet Leo unter einer sehr schwachen und verletzlichen Programmierung. Dies zeigen zum Einen die Inkompabilitäten zu UIMAfit, zum Anderen aber auch statische Code Analyse. Alles in allem aber ein brauchbares Tool und mein Go-To, sollte ich UIMA-AS noch einmal aufsetzen.

2.3.2 v3NLP

v3NLP ist auch ein scaling framework für NLP, was auf UIMA aufbaut. Es wurde damals speziell für cTAKES und MetaMap programmiert.

Chapter 3

Implementation

In this chapter we will first discuss the choice of Apache Spark as a distribution technology. Afterwards the framework implementation details will be documented with a special focus on data distribution, namely Serialization and Compression.

3.1 Technologies

Erkläre die Entscheidung warum auf Spark gesetzt wurde als Distributionsframework. Das möglicherweise mit Kruchten 4+1 (`https://de.wikipedia.org/wiki/4%2B1_Sichtenmodell`) aufhübschen.

Ich sollte auch ein wenig mit den generischen Ansätzen Compression & Serialization angeben. War schließlich nicht trivial die scheiße zu entwickeln.

Timm schlägt ein Schichtendiagramm vor, das zeigt wo ich zwischen UIMA und Spark stehe. Klingt gut, aber auch recht kompliziert in der Umsetzung. Ich denke das werde ich als letztes machen.

Ich denke aus diesen Bildern kann sich recht kanonisch eine Gliederung für das Kapitel entwickeln. Hier mal eine Liste von Abbildungen, die ich mir bisher vorstelle

Framework schematisch in Netzwerksicht (Also welchen Weg Dokumente so in meinem FW gehen)

3.2 Implementation

3.3 Data Distribution

Since all the input data, in form of documents, and output data, in form of analysis results, must be transmitted over a network, be it virtual or real, the serialization of larger Java objects play a role in performance. Since both, the input and the output, are stored inside a CAS (Common Analysis System) object it suffices to find a suitable serialization algorithm for those. However, finding an optimal algorithm is not trivial and usually even depends on the input data. Larger documents produce larger CAS,

which in turn need a longer time to be deployed to the corresponding Apache Spark workers. However, small documents still are no guarantee for small CAS sizes, since analysis results can be of arbitrary size and number, depending on the UIMA pipeline.

Furthermore it can be useful to compress serialized data, depending on the network setup and the serialization algorithm. Most native UIMA serializations produce XML files, which are very verbose and well compressible. Compression algorithms specifically designed for XML files achieve packing ratios of up to 80 % [GS05, MPC03, Sak09]. However, such algorithms often come at the price of a relatively high runtime. This is especially undesirable if the transmitted data is small or the serialization sparse and the expected compression ratio is low.

Since an optimal choice for both, serialization and compression, is not possible for the general case the framework exposes two interfaces, namely `CasSerialization` and `CompressionAlgorithm`.

3.3.1 Serialization

In [ESI+12] Epstein et al. explain how serialization of CAS was an important bottleneck and a problem to solve. They configured UIMA-AS in several ways to serialize only the parts of the CAS object that are needed for further analysis. Obviously this can not be done in the general case when the underlying analysis algorithms are unknown, which is why the framework takes an instance of `CasSerialization` as an optional parameter.

An instance of said interface implements two methods with the signatures shown in Listing 3.1..

```
1 public byte[] serialize(CAS cas);
2 public CAS deserialize(byte[] data, CAS cas);
```

Listing 3.1: CasSerialization method signatures

While the signature of the `serialize` method is intuitive, this does not immediately apply to the `deserialize` function. Here, a previously created CAS object is given as a parameter for two reasons. First, UIMA allows for the configuration of a custom `CasInitializer`, which can alter the CAS object immediately after creation. Although the usage of `CasInitializers` has been deprecated since at least 2006, it is still a feature of UIMA and must therefore be taken care of [Apad]. By creating a new CAS on the target JVM, the framework first executes the `CasInitializers` and then passes the resulting CAS to the `deserialize` function. The second reasons for this additional parameter is to pass the current UIMA type system. The serialized data might include annotations of types that are unknown to the native UIMA type system and therefore must be defined before deserialization. Although a parameter `TypeSystem` would have been sufficed, the first reason implies the requirement of a complete CAS parameter. Since the created CAS already includes the full type system description, available by `cas.getTypeSystem()`, the framework abstains from passing another parameter to the `deserialize` method. If

CasInitializers get removed from UIMA, this might be a feasible change in the future.

The framework already ships with two implementations of the CasSerialization interface, namely XmiCasSerialization and UimaCasSerialization. The XmiCasSerialization creates complete XMI (XML Metadata Interchange) files, containing the SofA (Subject of Analysis), all analysis results and even the used type system description. To accomplish this, it uses the UIMA XmiCasSerializer class. Thus, the XmiCasSerialization implementation of CasSerialization acts as a mere wrapper. The second serialization algorithm UimaCasSerialization also just wraps around the native UIMA class Serializer, which is the same serialization algorithm UIMA-AS uses to distribute and retrieve CAS objects.

3.3.2 Compression

Since the compression results are very dependent on the use case, data size and serialization algorithm, the framework provides the user with a CompressionAlgorithm interface. An implementation of said interface exposes two methods with signatures as shown in Listing 3.2.

```
1 public byte[] compress(final byte[] input);
2 public byte[] decompress(final byte[] input);
```

Listing 3.2: CompressionAlgorithm method signatures

Completely abstracted from any UIMA concept, this interface simply expects two functions, compress and uncompress to behave such that for every input byte[] X holds that X = decompress(compress(X)). While this is the only technical requirement for this interface, it is usually desired to have |X| > |compress(X)|. Since both methods act UIMA unaware, reducing the object size by omitting parts of the CAS is not possible without deserializing the CAS first, a step that is defined in the CasSerialization interface and not accessible from this context.

The framework ships with two implementations of the CompressionAlgorithm interface. If defaults to the NoCompression class, simply implementing the identity with X = compress(X), effectively disabling any kind of compression. This is useful if network delay is negligible, especially in virtual networks inside a single machine. A compression algorithm would need computation time to process all transmitted CAS, while saving only a minimum of transfer time. Secondly, the class ZLib implements the DEFLATE compression, which is a general purpose lossless compression algorithm, commonly used in ZIP files.

Chapter 4

Evaluation

Die Struktur hier sollte der Section Implementation Requirements ähneln.

Außerdem müssen wir noch begründen warum wir jetzt Framework gegen Single Threaded und UIMA-AS laufen lassen.

Was wir definitiv vergleichen können ist Extensibility und Maintainability, also weiche Metriken. Memory Consumption hab ich nicht mit geloggt, das empfand ich als zuviel Aufwand für zuwenig return. Hintergrund ist, dass sowohl UIMA-AS, als auch Spark nur kleinen Overhead haben. Wir reden hier von <1GB. Jede halbwegs erwachsene Pipeline, mit mindestens einem oder zwei Modellen übertrifft das. Somit bin ich grundsätzlich einfach davon ausgegangen, dass Speicher "genug" da ist, also nichts geswappt wird (und offensichtlich nichts in den OOM-Killer läuft). Ich denke das ist sinnvoll so. Hat man einen Rechner, auf dessen RAM die Pipeline, inklusive aller Modelle, passt, dann passt da auch noch Spark/UIMA-AS drauf. Passt die Pipeline nicht, hat man sowieso Pech.

4.1 Setup

Auf jeden Fall den gesamten Versuchsaufbau beschreiben, inkl. allem Docker-Gedöhns. Bilder:

- Versuchsaufbau UIMA-AS (evtl. inkl. Dockercontainern? Oder eher davon wegabstrahiert?)
- Versuchsaufbau Spark (s.o.)
- Framework in HDFS-Umgebung
- Single-Threaded in HDFS-Umgebung
- UIMA-AS in HDFS-Umgebung (hey, hier kann mein Framework punkten, da mach ich ein Bild zu :D)

4.2 Computation Speed

Relativ kanonisch in byte per second. Sollten die Analysedaten hier etwas hergeben, bietet sich ne Tabelle mit Urdaten und entsprechend nen Diagramm oder 20 an. Sollten die Daten kaputt sein, können noch rudimentäre Zeitabstände aus den Logdateien gelesen werden. Wie sexy das ist, ist allerdings fraglich.

4.3 Extensibility

Die extensibility ist hier zweiseitig zu betrachten, weil wir zum einen UIMA haben, was durch das Annotator-Plugin-System sehr extensible ist, was NLP-Funktionalität angeht. Das steht zumindest im Gegensatz zu v3NLP, was zwar auch Plugins zulässt, allerdings nicht die nativen UIMA-Dinger frisst (soweit ich weiß, muss ich noch bestätigen).

Zum anderen stellt sich die Frage inwiefern Spark-Konzepte weiter auf das Framework geworfen werden können. Es punktet zwar dadurch, dass es mit BigData umgehen kann, verliert allerdings durch das POJO, das der User zurückbekommt, an Spark-Funktionalität. Diese ist erweiterbar, allerdings nur wenn man den Quellcode selbst umschreibt (ie. das Projekt forked). Das ist zwar auch änderbar, ich will jetzt allerdings keien neuen features mehr zum FW hinzufügen, die nicht nur die Benchmarks invalidieren, sondern auch Fehler beinhalten können.

Interessant wäre vielleicht noch zu erwähnen, dass mein FW ein Serialization- und Compression- Interface anbietet, durch das der User diese beiden Aspekte quasi selbst einstellen kann. Beides macht einen großen Leistungsunterschied, besonders wenn man Network vs Localhost-Verkehr betrachtet. UIMA-AS bietet die Möglichkeit die Serialization selbst zu definieren, allerdings nicht die Compression. Der Serializer kann btw. natürlich auch dazu verwendet werden um Daten zu prunen. Das ist aber vom Anwendungsfall abhängig und hier nicht wirklich relevant, evtl sollte ich es allerdings trotzdem mal erwähnen.

4.4 Maintainability

Im Gegensatz zu UIMA-AS punktet hier natürlich auch mein FW. Ich sag nur XML-Dateien. Mein FW (ich hab dem noch gar keinen Namen gegeben) setzt auf die Spark-Infrastruktur. Damit ist es genauso Maintainable wie dieses, was auch immer das heißen mag. Ich gehe davon aus, dass services wie AWS sowas übernehmen.

4.5 Scalability

Ein bisschen seltsam, das als Metrik hinzuzunehmen, aber trotzdem sollte man sich Gedanken darum machen, was passiert wenn wir ZU bigData haben. Bei UIMA-AS würde als erstes vermutlich der Broker streiken, weil es keinen Broker-Broker gibt. Bei Spark kann es mehrere Master in einem Netzwerk geben. Wie das geregelt wird, muss ich noch herausfinden.

Chapter 5

Summary

Hier ist ein wunderhübscher Ort um nochmal alles zusammenzufassen. Daher hat dieser Ort auch jene Überschrift bekommmen :3

5.1 Limitations

Eventuell sogar schon in die Implementierung?

5.2 Conclusion

Das hängt ein wenig von den Ergebnissen ab. Ich hoffe auf sowas wie "Ich bin der tollste, UIMA-AS ist Dreck". Das spiegeln die Daten zwar nicht ganz wieder, aber hey :D

5.3 Outlook

Spark-Funktionalitäten können noch in das Framework implementiert werden (sogar extrem leicht, da der return value ein Spark-Objekt ist). Weiterhin ist es natürlich Usern überlassen ob weitere Serializations und Compressions implementiert werden.

Glossary

Apache Spark

Gestohlener Text, muss noch paraphrasiert werden.

Apache Spark is an open-source cluster-computing framework. Originally developed at the University of California, Berkeley's AMPLab, the Spark codebase was later donated to the Apache Software Foundation, which has maintained it since. Spark provides an interface for programming entire clusters with implicit data parallelism and fault tolerance. 1, 3, 8, 9

Application Programming Interface

3

Collection Processing Engine

Collection Processing Engines (CPE) are the first generation of UIMA native scaling solutions. A CPE contains a collection reader, which knows how to read the underlying collection, and CAS Consumers for the final analysis result extraction [Apad]. 3

Common Analysis System

The Common Analysis System is a type of object in the UIMA framework. It contains the subject of analysis, the analysis result and a corresponding type system [Apad]. 8

Darmstadt Knowledge Processing Software Repository

A collection of UIMA components for natural language processing. This includes analysis engines, language models and custom type systems [DKP, EdCG14]. 3

Docker

Docker is a virtualization solution based on containers. By using containers instead of fully fledged virtual machines Docker tries to reduce the system overhead per running application [doc15]. 1

Extensible Markup Language

2

General Architecture for Text Engineering

1

Java Virtual Machine

2

Lines of Code

3

Natural Language Processing

Natural-language processing (NLP) is the discipline of collecting and analysing natural language. This includes for example speech recognition, natural language understanding and generation [Lid01]. 1

Question Answering

Being a subfield of NLP, Question Answering (QA) is about extracting and understanding questions from natural language and answering them accordingly [JM14]. 1

Subject of Analysis

The Subject of Analysis is the document that gets analyzed by a given UIMA application. It is contained in its corresponding CAS [Apad]. 10

UIMA Asynchronous Scaleout

UIMA-AS is the second generation of UIMA native scaling solutions. It is based on a shared queue based service architecture [Apac] 1

Unstructured Information Management Architecture

UIMA is a general purpose framework to extract information from unstructured data [Apab, FLVN09]. Although any data format is supported, natural language texts are the most common one. 1

XML Metadata Interchange

10

Bibliography

[Apaa] The Apache Software Foundation. Apache opennlp. http://opennlp.apache.org/faq.html. Accessed: 2018-08-29.

[Apab] The Apache Software Foundation. Apache uima – apache uima. https://uima.apache.org/. Accessed: 2018-02-26.

[Apac] The Apache Software Foundation. Getting started: Apache uima asynchronous scaleout. https://uima.apache.org/doc-uimaas-what.html. Accessed: 2018-02-26.

[Apad] The Apache Software Foundation. Uima tutorial and developers' guides. https://uima.apache.org/d/uimaj-2.4.0/tutorials_and_users_guides.html. Accessed: 2018-02-26.

[BL04] Steven Bird and Edward Loper. Nltk: the natural language toolkit. In *Proceedings of the ACL 2004 on Interactive poster and demonstration sessions*, page 31. Association for Computational Linguistics, 2004.

[CMBT02] Hamish Cunningham, Diana Maynard, Kalina Bontcheva, and Valentin Tablan. Gate: an architecture for development of robust hlt applications. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 168–175. Association for Computational Linguistics, 2002.

[DCR+15] Guy Divita, M Carter, A Redd, Q Zeng, K Gupta, B Trautner, M Samore, and A Gundlapalli. Scaling-up nlp pipelines to process large corpora of clinical notes. *Methods of information in medicine*, 54(06):548–552, 2015.

[DKP] The DKPro Core Team. Dkpro core™ user guide. https://zoidberg.ukp.informatik.tu-darmstadt.de/jenkins/job/DKProCoreDocumentation(GitHub)/de.tudarmstadt.ukp.dkpro.core$de.tudarmstadt.ukp.dkpro.core.doc-asl/doclinks/6/user-guide.html. Accessed: 2018-02-26.

[doc15] What is docker? https://www.docker.com/what-docker, 2015. Accessed: 2018-02-26.

[EdCG14] Richard Eckart de Castilho and Iryna Gurevych. A broad-coverage collection of portable nlp components for building shareable analysis pipelines. In *Proceedings of the Workshop on Open Infrastructures and Analysis Frameworks for HLT*, pages 1–11, Dublin, Ireland, August 2014. Association for Computational Linguistics and Dublin City University.

[ESI+12] Edward A Epstein, Marshall I Schor, BS Iyer, Adam Lally, Eric W Brown, and Jaroslaw Cwiklik. Making watson fast. *IBM Journal of Research and Development*, 56(3.4):15–1, 2012.

[Fer12] David A Ferrucci. Introduction to "this is watson". *IBM Journal of Research and Development*, 56(3.4):1–1, 2012.

[FL04] David Ferrucci and Adam Lally. Uima: an architectural approach to unstructured information processing in the corporate research environment. *Natural Language Engineering*, 10(3-4):327–348, 2004.

[FLVN09] David Ferrucci, Adam Lally, Karin Verspoor, and Eric Nyberg. Unstructured information management architecture (UIMA) version 1.0. OASIS Standard, mar 2009.

[GR12] John Gantz and David Reinsel. The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east. *IDC iView: IDC Analyze the future*, 2007(2012):1–16, 2012.

[GS05] Marc Georges Girardot and Neelakantan Sundaresan. System and method for schema-driven compression of extensible mark-up language (xml) documents, April 19 2005. US Patent 6,883,137.

[JM14] Dan Jurafsky and James H Martin. *Speech and language processing*, volume 3. Pearson London:, 2014.

[KSDG12] Valmeek Kudesia, Judith Strymish, Leonard D'Avolio, and Kalpana Gupta. Natural language processing to identify foley catheter–days. *Infection control and hospital epidemiology*, 33(12):1270–1272, 2012.

[Lid01] Elizabeth D Liddy. Natural language processing. 2001.

[MPC03] Jun-Ki Min, Myung-Jae Park, and Chin-Wan Chung. Xpress: A queriable compression for xml data. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 122–133. ACM, 2003.

[MSB+14] Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, pages 55–60, 2014.

[PT18] Irina Pak and Phoey Lee Teh. Text segmentation techniques: A critical review. In *Innovative Computing, Optimization and Its Applications*, pages 167–181. Springer, 2018.

[RBJB+10] Cartic Ramakrishnan, William A Baumgartner Jr, Judith A Blake, Gully APC Burns, K Bretonnel Cohen, Harold Drabkin, Janan Eppig, Eduard Hovy, Chun-Nan Hsu, Lawrence E Hunter, et al. Building the scientific knowledge mine (sciknowmine): a community-driven framework for text mining tools in direct service to biocuration. *Language Resources and Evaluation*, page 33, 2010.

[Sak09] Sherif Sakr. Xml compression techniques: A survey and comparison. *Journal of Computer and System Sciences*, 75(5):303–322, 2009.

[TRCB13] Valentin Tablan, Ian Roberts, Hamish Cunningham, and Kalina Bontcheva. Gatecloud. net: a platform for large-scale, open-source text processing on the cloud. *Phil. Trans. R. Soc. A*, 371(1983):20120071, 2013.

Eidesstattliche Erklärung

Hiermit versichere ich, Simon Gehring, dass ich die vorliegende Masterarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Die Stellen meiner Arbeit, die dem Wortlaut oder dem Sinne nach anderen Werken und Quellen, einschließlich Quellen aus dem Internet, entnommen sind, habe ich in jedem Fall unter Angabe der Quelle deutlich als Entlehnung kenntlich gemacht. Dasselbe gilt sinngemäß für Tabellen, Karten und Abbildungen.

Unterschrift: _____

Simon Gehring, Student
Universität Bonn