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
in cooperation with the
Fraunhofer-Institut für Kommunikation,
Informationsverarbeitung und Ergonomie
and
PricewaterhouseCoopers GmbH

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

August 23, 2018

Contents

Abstract

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].

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].

Natural language unfortunately tends to be unstructured and hardly machine readable. Even a seemingly easy task, like separating a sentence into words is still an ongoing research topic [PT18]. Apache UIMA (Unstructured Information Management Architecture) 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 (Darmstadt Knowledge Processing Software Repository) 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 in non-academic environments 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 (UIMA Asynchronous Scaleout) and the other being the more dated and less flexible CPE (Collection Processing Engine) [FLVN09].

In this thesis, we will evaluate different means of scaling UIMA, using modern technologies like Docker, a container virtualization solution, Apache Spark, a cluster computing framework, and Apache Kafka, an information stream processing software. We will compare said implementations with the native UIMA-AS and CPE 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.

The following decisions are still to be made:

- On what level of abstraction will be parallelized? (Inter-Annotator, Inter-Pipeline, Inter-JVM, ...)

[1] Catheter-associated Urinary Tract Infections

- What methods of parallelization will be used? (Spark? Native Java Threads? UIMA-AS? Services?)
- What application (and implied data set) is going to be parallelized?
- What metrics are we trying to optimize?
  - CPU Workload (Does the system use all the available processing power?)
  - RAM Usage (How much RAM does the system use? Does it swap?)
  - Throughput (How many documents per second are processed?)
  - Maintainability (Are infrastructure changes or code changes easy to implement?)
  - Extensibility (How easy is it to add more processing power?)
  - Implementability (How simple is the implementation?)

Chapter 1

Introduction

1.1 Motivation

Idee: NLP benötigt große Daten, was im Grunde BigData ist (Größenordnung Peta oder Exabyte.) Muss entsprechend skaliert werden. Das geht einzeln mit verschiedenen NLP Frameworks, aber UIMA ist halt UIMA und generisch. Wär also geil wenn das anständig skalierbar ist.

1.2 Related Work

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

1.2.1 Watson

Watson ist mehr historisch bedingt und zählt eher zu UIMA-AS. Trotzdem interessant, da dort UIMA das erste mal demowürdig skaliert wurde.

1.2.2 v3NLP

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

1.2.3 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.

1.2.4 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.

1.2.5 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

1.3 Outline

Ich outline hier halt die gesamte restliche Thesis. Im Großen und Ganzen ist das hier nur ein unvollständiges und glorifiziertes Inhaltsverzeichnis.

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.

Selbstverständlich kann ich noch weiter ausholen und erst ein wenig über Java, Maven etc. reden, aber ich glaube das ist etwas overkill. Wer meine Arbeit liest, sollte zumindest Java kennen.

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.

Chapter 3

Implementation

Hier bin ich mir auch noch recht unschlüssig. Wichtig ist auf jeden Fall ein Schaubild. Das gilt natürlich für die eigentliche Implementierung des Frameworks, als auch für alle Versuchsaufbauten.

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

- UIMA pipeline (vermutlich schon weit früher, aber spätestens hier sollte klar sein was UIMA eigentlich mit CAS macht)
- UIMA-AS schematisch (evtl. schon oben im Introductionkapitel, aber mindestens ne Referenz dahin)
- Framework schematisch in Netzwerksicht (Also welchen Weg Dokumente so in meinem FW gehen)
- 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)

Chapter 4

Evaluation

Das kommt natürlich ganz auf die gesammelten Daten an, die ich noch genauer analysieren muss. Zum Zeitpunkt an dem ich diesen Text hier schreibe, hab ich ein wenig bammel, dass die Analysedaten teilweise unbrauchbar sind. Das wär natürlich doof.

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 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.

4.2 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 umge-

hen 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.

4.3 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.

4.4 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

Ich gehe davon aus, dass das hier ein kleines Kapitel wird. Zusammenfassung, Fazit und Ausblick bleiben hier noch aus. Viel mehr fällt mir auch nicht ein.

Die Future Work waren einmal ein eigenes Kapitel, ich würde das als Section hier rein stecken. Ich gehe einfach davon aus, dass ich dazu nicht viel mehr als zwei oder drei Seiten schreiben kann. Und anfangen blabla zu erzählen will ich auch nicht.

Glossary

Apache Kafka

Gestohlener Text, muss noch paraphrasiert werden.

Apache Kafka is an open-source stream processing software platform developed by the Apache Software Foundation written in Scala and Java. The project aims to provide a unified, high-throughput, low-latency platform for handling real-time data feeds. 1

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

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 [Apac]. 1

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]. 1

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

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

UIMA Asynchronous Scaleout

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

Unstructured Information Management Architecture

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

Bibliography

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

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

[Apac] 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.

[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.

[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.

[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.

[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.

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