Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

Fraunhofer
IAIS

Master of Science in Computer Science

– Wintersemester 2018/2019–

Evaluation von Ansätzen zur skalierten Ausführung von workflowbasierten Anwendungen am Beispiel einer Audio-Mining Anwendung

von

David Luhmer

Erstprüfer:	Prof. Dr. Martin Leischner
Zweitprüfer:	Prof. Dr. Karl Jonas
Betreuer:	Dr. rer. nat. Jens Fisseler / Dr. rer. nat. Christoph Andreas Schmidt
Unternehmen	Fraunhofer-Institut für Intelligente Analyse- und Informationssysteme IAIS - Abteilung NetMedia
Eingereicht am:	20.02.2019

Erklärung

Hiermit erkläre ich an Eides Statt, dass ich die vorliegende Arbeit selbst angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher keiner Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Bonn, den 20.02.2019

Datum Unterschrift

Inhaltsverzeichnis

Tabellenverzeichnis

Listings

Abbildungsverzeichnis

1 Einleitung

Die letzten zwei Jahrzehnte wurden durch das exponentielle Wachstum der zur Verfügung stehenden Daten geprägt. Täglich produzieren Menschen und Maschinen mehr und mehr Daten, die oftmals in verteilten Datenspeichern abgelegt werden. Anwendungsgebiete lassen sich beispielsweise in der Physik und Astronomie finden, wo immense Datenmengen von Teilchenbeschleunigern oder Satelliten erzeugt werden, die gespeichert und verarbeitet werden müssen. Aus diesen Datenmengen können weder vom Menschen direkt noch durch traditionelle Analysemethoden neue Erkenntnisse gewonnen werden. Zur Verarbeitung dieser Datenmassen sind parallele sowie verteilte Datenanalyseverfahren notwendig. [MTT18, NEKH+18]

Datenanalysen werden von Wissenschaftlern genutzt, um neue Erkenntnisse aus Informationen zu gewinnen. Die Extraktion von Informationen ist ein komplexer Prozess, der als Datenanalyse-Workflow dargestellt werden kann. Datenanalyse-Workflows verknüpfen verteilte Datenmengen, verschiedene Anwendungen sowie die benötigten Algorithmen. Workflows ermöglichen auf deklarative Weise eine abstrakte Darstellung von Abhängigkeiten, ohne auf konkrete Details der Anwendung einzugehen. [MTT18, NEKH+18]

Hierbei ist eine Differenzierung des Begriffs Workflow (zu Deutsch „Arbeitsablauf“) vorzunehmen. So wird zwischen Business und Scientific Workflows unterschieden. Eine Definition sowie Abgrenzung beider Begriffe ist in Kapitel 2.2 zu finden.

Besonders im Kontext von wissenschaftlichen Anwendungen stellt, neben der zuverlässigen Ausführung von Workflows sowie automatischen Datenverwaltung, die Ausführung und Reproduzierbarkeit von Experimenten in beliebigen Berechnungsumgebungen eine Herausforderung dar. Laut Novella et al. sind viele existierende Workflow-Engines diesen Anforderungen nur bedingt gewachsen, da Workflows, die speziell für eine Plattform entwickelt wurden, häufig nicht auf anderen Plattformen ausführbar sind. [NEKH+18]

Ein konkretes Beispiel für eine berechnungsintensive Anwendung stellt die, durch die Abteilung „NetMedia“ des Fraunhofer-Institut für Intelligente Analyse- und Informationssysteme IAIS entwickelte Audio-Mining Anwendung, dar. Audio-Mining bezeichnet eine Technik, die es ermöglicht, Audioinhalte nach Wörtern oder Phrasen zu durchsuchen [Wri13]. Zum Extrahieren von Wörtern aus Audioinhalten werden Spracherkennungssysteme verwendet, die gesprochene Wörter in Text überführen [IAI18]. Spracherkennungssysteme für Mediendateien (Radio-, Fernsehsendungen) müssen stets auf dem aktuellsten Stand sein. So treten in Nachrichtensendungen ständig neue Begriffe und Namen auf, die ein Spracherkennungssytem erkennen muss, wie zum Beispiel „Brexit“,

„BAMF“, „Deepfake“ oder Personennamen wie „Carles Puigdemont“ oder „Annalena Baerbock“. Ein einmal trainiertes Spracherkennungssystem veraltet, wenn es nicht ständig aktualisiert wird. Für verschiedene Akzente bzw. sprachliche Ausprägungen müssen weiterhin verschiedene Sprachmodelle trainiert werden. Dabei müssen zum Training der jeweiligen Sprachmodelle unterschiedliche Parameter verwendet werden.

Derzeit stellt die Trainingsdauer für neue Sprachmodelle ein Problem dar. Bislang werden neue Sprachmodelle auf einem Computer in einem einzelnen Thread berechnet. Die Trainingsdauer beträgt im Durchschnitt 6 Tage. Bei der Entwicklung des bisherigen Trainingsprozesses wurden Performance Optimierungen vernachlässigt. Neben der Anforderung, ein Sprachmodell pro Tag trainieren zu können, ist es Ziel dieser Arbeit, den Trainingsprozess zu automatisieren, parallelisieren und eine automatische Fehlerbehandlung zu ermöglichen. Weiterhin sollen in Zukunft mehrere Sprachmodelle für verschiedene Kunden unabhängig voneinander trainiert werden können. Somit besteht die Frage, wie das Training verschiedener, unabhängiger Sprachmodelle parallelisiert und skalierbar realisiert werden kann.

Ziel dieser Arbeit ist es zunächst den aktuellen Forschungsstand darzulegen. Dazu werden verteilte Programmiermodelle betrachtet, Unterschiede verschiedener Modelle für Workflows diskutiert und eine Abgrenzung, der im Rahmen dieser Arbeit untersuchten Workflow-Engines, vollzogen. Anschließend wird in Kapitel 3 der Anwendungsfall des Sprachmodelltrainings genauer erläutert, bevor in Kapitel 4 eine Kapselung der einzelnen Stufen mittels Container-Technologien durchgeführt wird. Anschließend werden die untersuchten Workflow-Engines vorgestellt und hinsichtlich der parallelisierten Ausführung des Sprachmodelltrainings evaluiert, bevor diese in Kapitel 5 hinsichtlich der Ausführungszeit verglichen werden. Abschließend wird aus den in dieser Arbeit gewonnenen Erkenntnissen ein Vorgehensmodell entwickelt, welches bei der Integration von Workflow-Engines für bereits existierende Anwendungen, hilft. Kapitel 6 fasst die Ergebnisse der Arbeit zusammen und gibt Anregungen und Ideen für weitergehende Forschungen.

2 Stand der Forschung

Umfangreiche wissenschaftliche Anwendungen verarbeiten häufig große Datenmengen, die sich aus kleineren Datensegmenten zusammensetzen, die unabhängig voneinander bearbeitet werden können. Mittlerweile existiert eine Vielzahl verteilter Systeme, die eine parallele Ausführung von Anwendungen erlauben. Speziell zur Koordination verschiedener Anwendungen wurden Workflow-Engines entwickelt. Dabei stellt die effiziente Ausführung und damit die Ausnutzung der potentiellen Parallelität eine Herausforderung für Workflow-Engines dar. [Tay07]

In diesem Kapitel werden zunächst verteilte Programmiermodelle vorgestellt, bevor anschließend verschiedene Grundlagen von Workflows betrachtet werden. Abschließend werden verschiedene Verfahren zur Ausführung von Workflows vorgestellt.

2.1 Verteilte Programmiermodelle

Laut Fahringer und Qin adressiert die Programmierung drei grundsätzliche Probleme. Definiert wird, wie Daten dargestellt (Datenrepräsentation), verarbeitet (Berechnungen) und in welcher Reihenfolge (Steuerung) diese abgearbeitet werden. Bei der Ausführung von Anwendungen auf verteilten Systemen werden Berechnungen typischerweise auf verschiedenen Rechnersystemen ausgeführt. Für die genannten Probleme ergeben sich daher drei Aspekte, die bei verteilten Systemen berücksichtigt werden müssen:

- Datenaustausch zwischen Berechnungen
- Kapselung von verteilten Berechnungen
- Koordination von verteilten Berechnungen

[QF12, vgl. 18f]

Es existiert eine Vielzahl von Programmiermodellen für verteilte Systeme [QF12]. Zur Bewertung dieser Modelle lassen sich verschiedene Kriterien ableiten. Dazu gehört beispielsweise die Form, in der ein Programmierer die Parallelität spezifizieren muss, sowie die Frage nach der Art des Informationsaustausches und die Möglichkeit zur Synchronisation. Im Folgenden werden einige, für die Ausführung von workflowbasierten Verfahren relevante Programmiermodelle kurz erläutert.

2.1.1 Datenaustausch

Der Datenaustausch in einem verteilten System bezeichnet die Möglichkeit, Daten zwischen einzelnen Berechnungsschritten (Rechner- bzw. Knotenübergreifend) auszutauschen.

Message Passing Models Message Passing Modelle führen Prozesse in unterschiedlichen Adressräumen aus. Die Kommunikation wird durch Nachrichten zwischen zwei Knoten realisiert. Ein Beispiel für eine solche Implementierung ist MPI. [QF12]

Hybrid Models Als hybride Modelle werden Systeme bezeichnet, die eine parallele Ausführung mithilfe von Threads im selben Adressraum ermöglichen, eine knotenübergreifende Kommunikation jedoch mithilfe von Nachrichten erfolgt. Beispiele für ein hybrides Modell sind MPI und OpenMP. [QF12]

Scheduling-Paradigmen Scheduling bezeichnet den Prozess der Organisation und Steuerung von Aufgaben zur Optimierung der Auslastung in einem Berechnungsprozess. Zwei bekannte und weit verbreitete Scheduling-Paradigmen zur Ausführung von parallelen Aufgaben sind „work sharing“ und „work stealing“. Während beim Paradigma des „work sharings“ Prozessoren beim Erzeugen von neuen Aufgaben diese aktiv an andere, weniger ausgelastete Prozessoren verteilen, „klauen“ Prozessoren beim „work stealing“ Aufgaben anderer Prozessoren, sobald sie ihre eigenen Aufgaben abgearbeitet haben. Je nach Art der Aufgaben und Lastverteilung kann das „work stealing“ Paradigma Performance Vorteile gegenüber dem „work sharing“ Paradigma aufweisen. [BL99, Wan17]

Work-Queues Work-Queues implementieren das „work stealing“ Paradigma und werden bei taskbasierten Anwendungen eingesetzt, um ein dynamisches Scheduling zu ermöglichen. Dabei gibt es zwei verschiedene Ansätze. Erstens kann jeder Prozessor eine Liste von Tasks besitzen, an denen er arbeitet. Sobald diese Liste abgearbeitet ist, „klaut“ dieser Aufgaben von anderen Prozessoren. Der zweite Ansatz beinhaltet eine zentrale Verwaltung aller zu erledigenden Tasks. Wenn viele kleine Tasks abgearbeitet werden müssen, kann der zusätzliche Synchronisierungsaufwand die Performance erheblich beeinflussen. Mögliche Lösungen bieten hier beispielsweise sogenannte „Guided Self Scheduling“ Verfahren. Diese bieten die Möglichkeit, die Ausführung verschachtelter Schleifen auf Multiprozessorsystemen mit gemeinsamen Speicher zu optimieren. [PK87, Wan17]

2.1.2 Kapselung

Im Kontext von verteilten Systemen wird die Kapselung häufig zur Abstraktion von Low-Level Komplexität eingesetzt. Dem Entwickler steht zur Programmierung verteilter Anwendungen eine einfache Programmierschnittstelle (API) bereit, die beispielsweise eine einfache Kommunikation zwischen Berechnungsknoten ermöglicht, ohne dass sich der Entwickler mit der Datenübertragung auf Netzwerkebene beschäftigen muss. Fahringer und Qin geben eine Liste verschiedener Modelle an, die jedoch im Kontext der Workflow-Entwicklung keine Relevanz haben. Daher sei für weitere Informationen zu diesen Modellen auf die entsprechende Literatur verwiesen. [QF12]

2.1.3 Koordination

Koordinationsmodelle stellen Mittel und Schnittstellen zur Verfügung, um eine Reihe von heterogenen Aufgaben so zu verknüpfen, dass sie eine Anwendung bilden, die auf verteilten Systemen ausgeführt werden kann. [QF12]

Bag of Tasks Das Bag of Tasks Modell nimmt an, dass Anwendungen als Zusammenspiel unabhängiger Aufgaben gesehen werden können und diese daher parallel ausgeführt werden können. Ein Beispiel für ein solches Modell ist das Parameter Sweep, bei dem ein Programm auf mehreren Systemen mit unterschiedlichen Parametern parallel ausgeführt wird. [QF12] Dieses Verfahren eignet sich ebenfalls für den Einsatz zur Parallelisierung von unabhängigen Aufgaben in einem Programm. Beispielsweise, wenn die Datenmenge so in Teilmengen zerlegt werden kann, dass die Teilmengen von jeweils einem Knoten verarbeitet werden können.

Workflow Models stellen ein Modell dar, welches Aufgaben in einer fest definierten Reihenfolge abarbeitet, um ein wohldefiniertes Ziel zu erreichen. Man unterscheidet hier grundsätzlich zwischen datenflussbasierten Workflow-Modellen, gerichteten azyklischen Graph Modellen, Petri-Netz basierten- sowie UML Aktivitätsdiagramm basierten Workflow-Modellen. [QF12]

Taskpool Für ein optimales Scheduling von Aufgaben (Tasks) muss die Rechendauer einer jeden Aufgabe bekannt sein. Daher lassen sich statische Verfahren bei Aufgaben, deren Berechnungsdauer unbekannt ist oder stark variieren kann, nur bedingt anwenden. Eine mögliche Alternative ist der Einsatz von sogenannten Taskpools. Taskpools sind

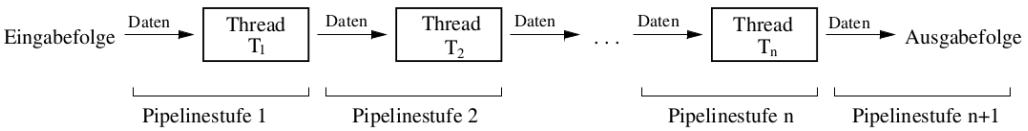


Abbildung 1: Pipeline Modell [RG07, S.306]

eine Datenstruktur, die eine Menge von Aufgaben dynamisch an verschiedene Prozessoren verteilt. [KR] Taskpools können jedoch nicht nur rechnerübergreifend eingesetzt werden, sondern auch auf Thread-Ebene, um die Leistung eines einzelnen Prozessors vollständig zu nutzen.

Pipelining Ein weiterer Ansatz zur Paralellisierung von Anwendungen stellt das Pipelining Modell dar. Dabei wird ein Strom von Daten durch eine Folge von n Threads verarbeitet $(T_1, ..., T_n)$. Jeder Thread T_i führt dabei eine bestimmte Operation bzw. Berechnung auf den Daten aus und gibt die modifizierten Daten an den nachfolgenden Thread weiter. Somit besteht zwischen allen Threads eine Ein-/Ausgabebeziehung. Während T_1 die Eingabedaten liest, gibt T_n die Ausgabefolge der Berechnung aus. Ein Thread T_i kann seine Berechnung nur dann beginnen, wenn die Ausgabedaten von Thread T_{i-1} vorliegen. Abbildung 1 visualisiert die Beziehungen zwischen den Threads. [RG07]

Dieses Verfahren eignet sich für kontrollflussorientierte Workflows. Ein Beispiel für die Anwendung des Pipelining Modells ist ein Tool zur Analyse von Tweets, welches in Echtzeit Texte auswertet. Eine Workflow-Engine, die solche Pipelines unterstützt, ist beispielsweise Apache NiFi.

2.2 Workflows

Im Folgenden werden zunächst verschiedene Konzepte der parallelen Ausführung von Anwendungen vorgestellt, bevor im Anschluss daran verschiedene Workflow-Arten erläutert werden.

2.2.1 Datenunabhängigkeit / Datenparallelität

Datenparallelität Laut Gleim und Schüle ist die „parallele Ausführung eines sequenziellen Befehlsstroms auf einer Menge gleichartiger Daten“ charakteristisch für Datenparallelität [GS12]. Ein Anwendungsfall bei wissenschaftlichen Anwendungen ist die Ausführung einer Berechnung auf einer Vielzahl von Daten. Datenparallele Operationen werden meist ausschließlich für Felder zur Verfügung gestellt und verfolgen das

SIMD Prinzip (Single instruction, multiple data). Rechner, die SIMD unterstützen, ermöglichen „die Durchführung der gleichen Operationen auf mehreren Prozessoren mit unterschiedlichen Daten“. [RG07, GS12, ALO02]

Für die Betrachtung sowie Optimierung von workflowbasierten Anwendungen ist die Datenparallelität nur bedingt anwendbar, da Felder zur Speicherung von primitiven Datentypen oder Objekten verwendet werden. Die Orchestrierung von Anwendungen durch eine Workflow-Engine beziehen sich jedoch auf ein höheres Abstraktionslevel.

Besonders im wissenschaftlichen Umfeld werden häufig Prototypen (Proof-of-concept) entwickelt, die nicht Performance optimiert sind. Die spätere Optimierung hinsichtlich der Performance ist meist mit einem hohen zeitlichen Aufwand verbunden, sodass eine Parallelisierung abstrakter Operationen sowie Aufgaben meist mit weniger Aufwand verbunden ist als eine Parallelisierung auf Zugriffsmuster sowie Feldebene. Workflow-Engines bieten die Möglichkeit einer Parallelisierung abstrakter Operationen sowie Aufgaben.

Funktionsparallelität / Taskparallelität In sequentiellen Programmen existieren oft verschiedene Programmteile, die unabhängig voneinander ausgeführt werden können. Unabhängige Programmteile können parallel zueinander ausgeführt werden. Da es sich bei den unabhängigen Programmteilen um einzelne Anweisungen, Schleifen oder auch Funktionsaufrufe handeln kann, werden diese auch häufig als Funktions- oder Taskparallelität bezeichnet. Ein Task bezeichnet dabei einen unabhängigen Programmteil. Mithilfe der Darstellung der Tasks in Form eines Taskgraphen können die Abhängigkeiten analysiert werden. Der Graph wird verwendet, um Parallelität bei der Ausführung auszunutzen sowie einen Abarbeitungsplan unter Berücksichtigung der Abhängigkeiten mithilfe eines Scheduling-Verfahrens zu berechnen. Bei paralleler Ausführung ist eine Kombination von Task- sowie Datenparallelität möglich. [RG07]

Explizite und implizite Darstellung der Parallelität Parallele Programmiermodelle lassen sich anhand der expliziten oder impliziten Darstellung der Parallelität unterscheiden. Die explizite oder implizite Darstellung bezieht sich dabei auf die Zerlegung eines Programms in Teilaufgaben und dem Vorkommen von Kommunikation und Synchronisation im zugehörigen parallelen Programm. Während eine explizite Darstellung Kenntnisse des Entwicklers über die parallele Programmierung voraussetzt, setzt eine implizite Darstellung fortgeschrittene Compiler zur Erzeugung effizienter Programme voraus. Im Folgenden werden vier verschiedene Programmiermodelle hinsichtlich der expliziten und impliziten Darstellung von Parallelität unterschieden. [RG07]

Die erste Klasse paralleler Programmiermodelle stellen Modelle dar, die **implizite Parallelität** verwenden. Sie stellen für den Entwickler die einfachste Art zur Beschleunigung von Programmen dar, da keine explizite Darstellung der Parallelität im Programm erforderlich ist. Die Parallelität wird durch den Einsatz einer funktionalen Programmiersprache oder aber durch den Einsatz von parallelisierenden Compilern erreicht. Ein parallelisierender Compiler analysiert dabei die sequentiellen imperativen Berechnungsschritte und parallelisiert diese automatisch. Zu beachten ist, dass der Kommunikations-Overhead nicht den Leistungsvorteil zunichte macht. Bei komplexeren Problemen erzielen parallelisierende Compiler daher häufig noch keine befriedigenden Ergebnisse. Bei funktionalen Programmiersprachen ist es oft schwierig, einen geeigneten Grad der Parallelisierung zu finden. [RG07]

Die zweite Klasse paralleler Programmiermodelle umfasst die Modelle, die eine **explizite Parallelität mit impliziter Zerlegung** ermöglichen. Sie erfordern eine explizite Angabe von parallelen Berechnungen, jedoch keine Aufteilung in Tasks oder eine Zuordnung zu Prozessoren. Die Darstellung von Kommunikation sowie Synchronisation ist daher nicht erforderlich. Vertreter dieser Klasse sind parallele Programmiersprachen, die sequentielle Programmiersprachen um parallele Konstrukte, z.B. Schleifen, erweitern. Beispiele für solche Sprachen sind High-Performance Fortran oder auch OpenMP. [RG07]

Die dritte Klasse paralleler Programmiermodelle stellen Modelle dar, die eine **explizite Zerlegung** von Berechnungen in Tasks erfordern. Ein Vertreter ist beispielsweise das Bulk Synchronous Parallel (BSP) Programmiermodell. [RG07]

Die vierte Klasse der parallelen Programmiermodelle benötigt neben der expliziten Zerlegung eine **explizite Zuordnung an Prozessoren**. Somit ist neben der Zerlegung in Tasks eine explizite Zuordnung an Prozessoren erforderlich. Kommunikation zwischen Berechnungen müssen dabei nicht explizit angegeben werden. „Linda“ ist eine Programmiersprache, die dieses Modell implementiert. [RG07]

Die fünfte Klasse der parallelen Programmiermodelle erfordert die Angabe für **explizite Kommunikation und Synchronisation**. Thread-Programmiermodelle sowie Message-Passing-Programmiermodelle gelten als Vertreter dieser Klasse. [RG07]

2.2.2 Datenlokalität

Die Effektivität eines verteilten Systems hängt davon ab, wie Aufgaben und Daten den zugrunde liegenden Systemressourcen zugewiesen werden [CGB+16]. Durch die immer günstiger werdenden Kosten für CPU Zyklen und die jedes Jahr größer werdenden Datensätze ist die Lage der Daten im Verhältnis zu den verfügbaren Rechenressourcen

eine wachsende Herausforderung. Das wiederholte Übertragen von großen Datenmengen stellt in vielen Anwendungen ein Bottleneck dar, besonders wenn Daten über das Wide Area Network transferiert werden müssen. Szalay et al. empfehlen daher die Verwendung von kleineren Clustern, die einen eigenen „intelligenten“ Zwischenspeicher besitzen und über ein Hochgeschwindigkeitsnetzwerk miteinander verbunden sind. Bei wiederholter Ausführung bestimmter Aufgaben ist der Scheduling Algorithmus dafür verantwortlich, die Aufgaben auf Knoten auszuführen, die bereits die benötigten Daten in ihrem Zwischenspeicher vorhalten. [SBG+18]

Die Cyprus University of Technology veröffentlichte im Jahr 2017 ein Paper, in dem ein Dateisystem (OctopusFS) vorgestellt wird, welches verschiedene Stufen der Zwischenspeicherung ermöglicht. Dabei werden HDDs, SSDs sowie RAM als Speichermedien verwendet. Je nach Anwendungsfall kann das System über Parameter konfiguriert werden, welche Daten in welchem Speichermedium vorgehalten werden sollen. Dies ermöglicht Entwicklern eine feingranulare Konfiguration, um Zugriffe auf benötigte Daten zu optimieren. Als Anwendungsbeispiel für OctopusFS nennen Kakoulli et al. die Möglichkeit, Workflows zu beschleunigen, indem Ergebnisse, die von einem Schritt in den nächsten übergeben werden, im Arbeitsspeicher abgelegt werden können, um die Datentransferdauer zu reduzieren. Weiterhin ist ein „prefetching“ von Daten in ein bestimmtes Speichermedium (HDD / SDD / RAM) möglich. Somit kann beispielsweise bei workflowbasierten Anwendungen die Wartezeit auf die benötigten Daten reduziert werden. [KH17]

2.2.3 Inkrementelle Verarbeitung

Die Methode der inkrementellen Verarbeitung ermöglicht, dass, wenn nur ein kleiner Teil der Eingabe eines Programms verändert wurde, keine vollständige Neuberechnung durchgeführt werden muss, sondern lediglich die neu hinzugekommene Eingabe verarbeitet wird. Dabei ist die Konsistenz der inkrementellen Verarbeitung gegenüber der vollständigen Verarbeitung sicherzustellen. [Mor16]

Zur Realisierung einer inkrementellen Verarbeitung von Workflows ist es notwendig, dass die Workflow-Engine entweder eine Verwaltung und Koordination der zu verarbeitenden Daten übernimmt oder das Backfill-Prinzip unterstützt wird.

Das „Backfill“-Prinzip bezeichnet die Möglichkeit, Workflows zu einem bestimmten Datum in der Vergangenheit auszuführen. Dabei wird der Workflow gestartet und erhält als Parameter das Datum des zu bearbeitenden Zeitraums. Es ist meist die Aufgabe des Entwicklers, dieses Datum in seinem Programm zu verarbeiten und dementsprechend nur die Daten aus diesem Zeitraum zu verwenden. Ebenfalls lassen sich so

weitere Scheduling-Optimierungen anwenden, da der Scheduler Aufgaben (zu einem gegebenen Datum) in beliebiger Reihenfolge abarbeiten kann, um so die Lastverteilung weiter zu optimieren. [bacb] Das „Backfill“ Verfahren eignet sich für Anwendungen, die Zeitreihen aus Datenbanken beziehen. Workflow-Engines, die das sogenannte „Backfill“-Prinzip unterstützten, sind z.B. Airflow, Digdag sowie Pachyderm. [Sta17, baca]

2.2.4 Daten- vs. kontrollflussorientierte Workflows

Workflows werden als eine Sequenz verschiedener Operationen oder Aufgaben gesehen, die zur Erledigung einer Berechnung notwendig sind. Sequenzen von Aufgaben oder Operationen lassen sich auf verschiedene Weise darstellen. Von der Darstellung über einfache Skriptsprachen oder Graph-Repräsentationen hin zu mathematischen Definitionen existiert eine Vielzahl von Darstellungsverfahren. [Tay07]

Workflows lassen sich in kontrollfluss- oder datenflussorientierte Verfahren klassifizieren. Ähnlich sind diese beiden Klassen insofern, als dass sie Interaktionen zwischen einzelnen Aktivitäten des Workflows abbilden. In kontrollflussorientierten Workflows repräsentiert eine Verbindung zwischen zwei Aktivitäten die Übergabe der Kontrolle. Eine solche Verbindung wird in graphbasierten Verfahren in der Regel durch eine gerichtete Kante repräsentiert. In datenflussorientierten Verfahren werden Abhängigkeiten zwischen Aufgaben, die eine Übergabe von Informationen / Daten darstellen, als Datenfluss in Form von gerichteten Kanten visualisiert. Der Quellknoten wird als Datenerzeuger bezeichnet, der Zielknoten als Datenkonsument. [Tay07]

Die Klassifizierung von Workflows in daten- oder kontrollflussorientierte Verfahren ist durch die Betrachtung der Verbindungen bzw. Abhängigkeiten zwischen Aufgaben bzw. Operationen in einem Workflow möglich. Falls eine Verbindung auf der Bereitstellung von Daten, wie beispielsweise einer Datei beruht, die zum Start der nachfolgenden Aufgabe bzw. Operation benötigt wird, handelt es sich um einen datengetriebenen Workflow und somit wahrscheinlich um einen datenflussorientierten Workflow. In datenflussorientierten Workflows können typischerweise alle Operatoren parallel gestartet werden. Die einzelnen Operationen begeben sich zunächst in einen Wartezustand, bis die erforderliche Datenabhängigkeit erfüllt ist. Die Repräsentation von komplexeren Kontrollstrukturen wie Schleifen sind meist nicht möglich. Anwendungsfälle sind in der Bild- sowie Signalverarbeitung zu finden. [Tay07]

Falls es sich bei den Verbindungen um zeitliche Abhängigkeiten handelt, wie beispielsweise „Aktivität A muss abgeschlossen sein, bevor Aktivität B starten darf“, handelt es sich wahrscheinlich um einen kontrollflussorientierten Workflow. Dabei liest und

schreibt die Operation vor bzw. nach Abschluss Daten in ein Dateisystem. Der Ursprung von kontrollflussorientierten Programmen ist in der Stapelverarbeitung zu finden, wo Anwendungen, wie beispielsweise Shell-Skripte (Unix-Prozesse) aneinander gereiht werden können, um die Ausführung zu koordinieren. Dabei wird die Kontrolle an den jeweiligen Unix-Prozess abgegeben. Nach Abschluss der Aufgabe wird die Kontrolle an die nachfolgende Aufgabe weitergegeben. Datenübertragungen werden über „Zwischenaufgaben“ zwischen zwei aufeinander folgenden Aufgaben realisiert. Komplexere Kontrollstrukturen wie „while, for, if“ werden typischerweise unterstützt. [Tay07]

Mischformen der beiden Klassen von Workflows sind ebenfalls möglich.

2.2.5 Business Workflows

Grundsätzlich lässt sich zwischen zwei verschiedenen Klassen bzw. Ansätzen von Workflows unterscheiden. Zum Einen existiert eine Reihe von Workflow Anwendungen die darauf abzielen, Geschäftsprozesse abbilden und ausführen zu können. Um Kommunikation zwischen verschiedenen Softwaresystemen in einem Unternehmen zu ermöglichen, orientieren sich Business Workflows stark an existierenden Kommunikationsstandards. Business Workflows werden typischerweise zur Realisierung eines Produktes oder Dienstes eines Unternehmens genutzt. Die zuverlässige Ausführung hat hierbei einen hohen Stellenwert, da mögliche Ausfälle unmittelbar für den Kunden bemerkbar sind. Die Möglichkeit der Mensch-Workflow Interaktion ist, wie auch später in Abschnitt 2.3.3 erwähnt, den Business Workflows vorbehalten. Zum Anderen existiert die Klasse der Scientific Workflows, die im folgenden Abschnitt beschrieben wird. [QCW16, SKD10, JA17]

2.2.6 Scientific Workflows

Besonders in den letzten Jahren wurde stark in Richtung wissenschaftlicher Workflows (Scientific Workflows) geforscht. Einige Forschungsansätze verfolgen das Ziel, bereits existierende Business Workflow Anwendungen dahin gehend zu erweitern, dass sie notwendige Anforderungen, wie beispielsweise die exakte Reproduzierbarkeit, unterstützen. Typischerweise sind Business Workflows weniger dynamisch als ihre wissenschaftlichen Gegenstücke. Im Gegensatz zu Business Workflows werden Scientific Workflows häufig von Wissenschaftlern erstellt, die Experten in ihrem Fachgebiet sind, jedoch nicht notwendigerweise Experten in der Softwareentwicklung. Ein weiterer Unterschied ist, dass einzelne Aufgaben in einem Scientific Workflow lange anhalten können oder es eine Vielzahl von Abhängigkeiten, beispielsweise im Bereich der Bioinformatik, geben

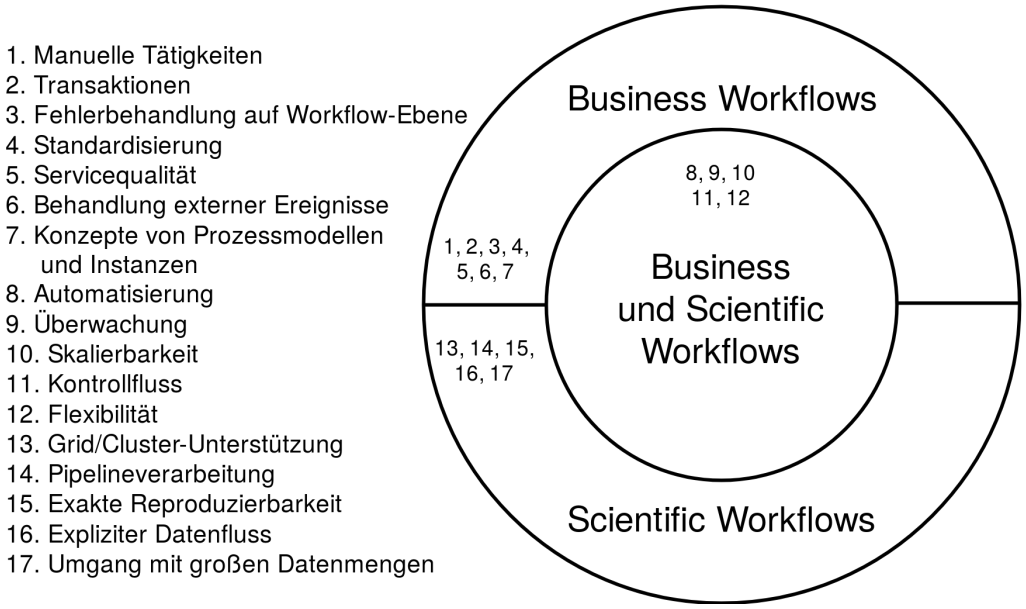


Abbildung 2: Business vs. Scientific Workflow; in Anlehnung an [SKD10, S.207]

kann. Business Workflows operieren aufgrund der Abbildung von Vorgängen flussorientiert, wohingegen Scientific Workflows zur Unterstützung der automatischen Verarbeitung von Daten meist datenorientiert definiert werden. Scientific Workflows sind darauf optimiert, große Datenmengen zu verarbeiten. Zur Beschleunigung von aufwändigen Berechnungen werden diese häufig auf Clustern oder Grid-Umgebungen ausgeführt. Im Gegensatz zu Business Workflows ist es bei Scientific Workflows wichtig, exakte Reproduzierbarkeit zu gewährleisten, explizite Datenflüsse definieren zu können sowie die Handhabung von großen Datenmengen zu ermöglichen. Der flussorientierte Ansatz von Business Workflows bietet häufig nur unzureichende Möglichkeiten, einzelne Aufgaben zu parallelisieren. Weiterhin kapseln Scientific Workflows die auszuführenden Anwendungen so weit ab, dass Wissenschaftler ausschließlich über die Schnittstelle der Workflow-Engine mit den Anwendungen kommunizieren können. Während bei Business Workflows die Interaktion mit Menschen ein fester Bestandteil ist, beschränkt sich die Interaktion bei Scientific Workflows auf die Installation, Ausführung mit Eingabedaten sowie Auswertung der Ergebnisse. Abbildung 2 visualisiert die Gemeinsamkeiten sowie Unterschiede zwischen Business und Scientific Workflows. [QCW16, SKD10, JA17, LAG+16]

2.3 Darstellung von Workflows

In diesem Abschnitt werden verschiedene Standardisierungen zur Ausführung von Workflows vorgestellt.

2.3.1 Common Workflow Language

Die Common Workflow Language (CWL) ist eine Spezifikation zur Beschreibung von Workflows. Durch die deklarative Beschreibung von Workflows sind die Anwendungen portabel und skalierbar auf einer Vielzahl von Soft- und Hardware Umgebungen. Unterstützt werden Workstations, Cluster-, Cloud- sowie High-Performance Computing (HPC) Umgebungen. CWL wurde entwickelt, um die Anforderungen datenintensiver Wissenschaften wie Bioinformatik, medizinische Bildanalyse, Astronomie, Physik und Chemie zu erfüllen. [cwl]

Die Definition bzw. Beschreibung der Workflows erfolgt in einer YAML-basierten Workflow Spezifizierungssprache. Die Unterstützung zur Ausführung von Docker-Containern ist bereits im Sprachstandard enthalten. Die CWL stellt eine Spezifikation dar, konkrete Implementierungen unterscheiden sich teilweise im Funktionsumfang. Theoretisch ist das Ausführen von Workflows im CWL Standard auf beliebigen Workflow-Engines, die den CWL Standard erfüllen, möglich. Eine Liste von Workflow-Engines, die die CWL Spezifikationen implementierten, ist auf der Projekt-Webseite[1] zu finden. [cwl]

Ein grafisches Tool zur lokalen Entwicklung von CWL Workflows ist Rabix Composer. Es bietet Unterstützung für Docker sowie die Möglichkeit, Workflows lokal zu testen. Eine verteilte Ausführung von Workflows ist nicht möglich. [rab]

Zur parallelen Ausführung existiert die Workflow-Engine Reana, welche durch die europäische Organisation für Kernforschung (CERN) entwickelt wird. [rea]

2.3.2 Workflow Description Language

Die „Workflow Description Language“ ist eine vom Broad Institute entwickelte Sprachspezifikation zur Ausführung von Workflows. Sie stellt gleichzeitig eine Alternative zur Common Workflow Language dar. Der Fokus des Projektes liegt darin, einen Standard zu schaffen, um das Schreiben sowie Lesen von Workflow-Definitionen einfach zu gestalten. Cromwell, eine Referenzimplementierung des WDL Standards, unterstützt die Ausführung von Workflows auf Systemen wie GridEngine, HTCondor, Spark sowie Googles „Pipelines API“. Seit Januar 2018 wird an einer Unterstützung der CWL in Cromwell gearbeitet. [wdl]

Ähnlich wie Rabix für die Common Workflow Language existiert ein grafischer Workflow Editor (Pipeline Builder) für die Workflow Description Language.

[1] `https://www.commonwl.org/`

Kontrollflussperspektive	Datenperspektive	Ressourcenperspektive
BPMN	UML Klassendiagramme	Use cases
UML Aktivitätsdiagramme	ER Diagramme	Rollen-Aktivitäts-Diagramm
Petri-Netze	Objekt-Rollen-Modelle	Organigramme
Zustandsdiagramme		X.500
EPCs		

Tabelle 1: Geschäftsprozessdimensionen und deren Modellierungstechniken [RvvdAtH16]

2.3.3 Business Process Modeling Language

Business Process Modeling Language, kurz BPML, ist eine XML-Metasprache zur Beschreibung von Geschäftsprozessen. Mithilfe der Business Process Model and Notation (BPMN, deutsch Geschäftsprozessmodell und -notation) kann eine grafische Repräsentation von Geschäftsprozessen dokumentiert werden, die anschließend in BPML oder auch BPEL umgewandelt werden können [BPM11]. BPMN/BPML gehört zur Klasse der Business Workflows.

Während laut Russell et al. keine einheitliche Definition eines Geschäftsprozesses existiert, definiert Pall ihn als „die logische Organisation von Personen, Materialien, Energie, Ausrüstung und Abläufen in Arbeitstätigkeiten, die darauf abzielen, ein bestimmtes Endergebnis zu erzielen“ [RvvdAtH16, S.5]. Russell et al. leiten aus dieser Beschreibung drei Dimensionen ab. Für jede dieser Dimensionen lässt sich eine Reihe von Modellierungstechniken angeben. Tabelle 1 zeigt die Zuordnung zwischen Dimension und verwendeter Modellierungstechniken. [RvvdAtH16]

Gängige Kontrollflusspattern sind beispielsweise Sequenzen, Parallele-Split sowie Merge Operationen und Synchronisationspattern [RvvdAtH16]. Zu den Datenpattern zählen Datensichtbarkeits-, Dateninteraktions-, Datentransfer- und datenbasierte Routingpattern. Die Datensichtbarkeitspattern bestimmen, ob und wie auf Daten zugegriffen werden kann. Dateninteraktionspattern definieren die Art der Kommunikation von Daten zwischen Prozessen. Datentransferpattern definieren die konkrete Datenübertragung. Datenbasierte Routingpattern definieren die Art, wie Datenelemente den Berechnungsweg bzw. Verarbeitungsweg beeinflussen können. [RvvdAtH16] Dies kann beispielsweise über Aufgaben Vor- bzw. Nachbedingungen oder auch eventbasierte Aufgaben Trigger geschehen. [RvvdAtH16] Pattern zur Zuweisung von Ressourcen sind beispielsweise das „Push-“ oder auch „Pull-“ Pattern, bei denen aktiv bzw. nicht-aktiv Arbeit zugewiesen wird. Im zweiten Fall sind die „Ressourcen“ selbst dafür verantwortlich, Arbeitspakete abzuholen und diese zu bearbeiten. Dabei wird als Ressource ein Mensch oder eine Maschine. bezeichnet [RvvdAtH16]

Wie bereits in Abschnitt 2.2.5 erwähnt, ist die Mensch-Workflow Interaktion Business Workflows vorbehalten. Weiterhin erlaubt der Sprachstandard die Definition von organisatorischen Zusammenhängen sowie entsprechendem Rechtemanagement. Ein Beispiel dafür ist folgende Aufgabe: „Aufgabe A soll von einer Managerin der Verkaufsabteilung durchgeführt werden“. Weiterhin bietet BPMN Kommunikationsstrukturen, um ein gemeinsames Verständnis, beispielsweise zur abteilungsübergreifenden Kommunikation, zu ermöglichen. [RvvdAtH16]

Für den Einsatz in einem wissenschaftlichen Kontext sind diese Eigenschaften jedoch nur bedingt relevant, da in der Regel keine Mensch-Workflow Interaktion erforderlich ist.

2.3.3.1 Business Process Execution Language

Die „Web Services Business Process Execution Language“ (WS-BPEL), kurz BPEL ist eine Sprache zur Spezifizierung von Interaktionen mit Web Services. Die erste Spezifikation „Business Process Execution Language for Web Services“ (BPEL4WS) 1.0 wurde 2002 veröffentlicht. Über die Jahre wurden verschiedene Änderungen an der Spezifikation vorgenommen. Für Scientific Workflows eignet sich die Spezifikation nur bedingt, da BPEL einen starken Fokus auf Web Services sowie deren Kommunikation legt. Für wissenschaftliche Anwendungen, die beispielsweise aus Shell-Scripts oder Java Anwendungen bestehen sowie Anwendungen die Parallelität nutzen, eignet sich der Standard weniger. Mit der Veröffentlichung von Version 2.0 im Jahre 2007 erhielten erstmals parallele Schleifen Einzug in den Standard. Ein weiterer Nachteil des BPEL Standards ist, dass der „Datei“ Datentyp nicht unterstützt wird, welcher jedoch häufig in Scientific Workflows verwendet wird, um beispielsweise große Textdaten einzulesen. [LAG+16, Tay07]

Zur Entwicklung von Workflows existiert ein Eclipse Plugin, welches eine visuelle Entwicklung von Workflows ermöglicht. Wie bereits erwähnt, fokussiert sich der BPEL Standard stark auf die Orchestrierung von Web-Diensten. Eine Möglichkeit zur Lastverteilung von Aufgaben über mehrere Server sowie eine Möglichkeit große Datenmengen zu verarbeiten existiert nicht bzw. nur über Umwege. Zudem unterstützt BPEL keine containerisierten Anwendungen, weshalb sie für die in dieser Arbeit untersuchte Fragestellung ungeeignet ist. [Tay07, MvL]

2.4 Ausführung von Workflows

Die Aufgabe einer Workflow-Engine ist es, die Ausführung eines Workflows zu ermöglichen. Dabei ist die Art der einzelnen Aufgaben zunächst nicht näher spezifiziert. Im

Kontext von Business Workflows können dies sowohl Aufgaben sein, die eine Mensch-Workflow Interaktion benötigen, sowie Aufgaben, die keine Interaktion mit Menschen benötigen. Wie bereits in Abschnitt 2.2.6 erwähnt, beschränken sich Scientific Workflow-Engines ausschließlich auf die Ausführung ohne Interaktion mit Menschen. Scientific Workflow-Engines sind für die Datenübertragung zwischen einzelnen Aufgaben des Workflows verantwortlich. Dabei ist die Workflow-Engine für die Ressourcenverwaltung, besonders bei verteilter Ausführung von Workflows, verantwortlich. Es ist Aufgabe der Workflow-Engine einen Ausführungsplan zu erstellen. [MK17]

Eine weitere Aufgabe von Scientific Workflow-Engines ist die Sicherstellung der Reproduzierbarkeit von Experimenten. Diese wird benötigt, um Fehler sowie Experimente reproduzieren und nachvollziehen zu können.

Mittlerweile existiert eine Vielzahl verschiedener Workflow-Engines, von denen eine Auswahl in Kapitel 4 erläutert wird. Nur wenige Workflow-Engines bieten eine Möglichkeit der Datenspeicherung oder die Sicherstellung der Reproduzierbarkeit von Experimenten. Viele Systeme unterstützten die Ausführung von Workflows mittels Angabe von „bash“-Befehlen, die auf dem Computer ausgeführt werden. Ein Problem bei diesem Ansatz ist jedoch, dass Abhängigkeiten existieren oder eine bestimmte Arbeitsumgebung gefordert ist, die auf jedem Computer zu konfigurieren ist. Wenn mehrere Computer an der Berechnung eines Workflows beteiligt sind, ist sicherzustellen, dass alle beteiligten Computer identische Versionen der auszuführenden Software installiert haben. Dieser Prozess ist fehlerbehaftet und erfordert einen hohen Wartungsaufwand.

2.4.1 Containerisierung für Workflow Anwendungen

Die Software Container Technologie hat sich im Laufe der letzten Jahre als Alternative zu virtuellen Maschinen entwickelt. Container bieten im Gegensatz zu virtuellen Maschinen weniger Isolation zwischen Prozessen, dafür sind sie leichtgewichtiger und kleiner. Durch die Verwendung des Kernels des Host-Systems lassen sich Container schneller starten und beenden als virtuelle Maschinen. Die am weitesten verbreitete Containerisierungslösung ist Docker, wobei Singularity[2], uDocker[3] und Shifter[4] neue Alternativen darstellen, die im Gegensatz zu Docker die Ausführung ohne Root Rechte ermöglichen. Besonders für High-Performance Cluster oder „multi-tenant“ (mehrere Kunden) Systeme ist die Möglichkeit, Container ohne Root Rechte ausführen zu können, ein Sicherheitsvorteil. Docker-Container lassen sich über die Plattform Docker Hub[5]

[2] `https://www.sylabs.io/singularity/`
[3] `https://github.com/indigo-dc/udocker`
[4] `https://github.com/NERSC/shifter`
[5] `https://hub.docker.com`

austauschen. Ebenfalls kann zum Austausch von vertraulichen Docker-Abbildern eine unternehmensinterne „Docker Registry“ bereitgestellt werden. Bei der Ausführung einer Vielzahl von Containern wird ein Orchestrierungssystem benötigt, um Probleme während der Ausführung, Lastverteilung sowie Skalierung zu übernehmen. In den letzten Jahren ist Kubernetes[6] zu einer der bekanntesten Orchestrierungslösungen geworden, jedoch sind Docker Swarm[7] und Apache Mesos[8] ebenfalls bekannte Systeme. Der Nutzer kommuniziert ausschließlich mit der Orchestrierungslösung, die eine Verteilung von Containern auf die zur Verfügung stehende Hardware durchführt.

Häufig können Workflows in der Umgebung, in der sie erstellt wurden, erfolgreich ausgeführt werden, scheitern jedoch auf anderen Plattformen aufgrund von Unterschieden in der Ausführungsumgebung. Dabei werden Workflows häufig auf einem einzelnen Computer getestet, bevor sie auf ein Cluster zur Ausführung übertragen werden. Aufgrund der Unterschiede in der Ausführungsplattform müssen Anwendungen daher speziell auf die Zielarchitektur angepasst werden. Eine Lösungsmöglichkeit stellen containerbasierte Scheduler dar, indem sie Container zur Verteilung von Computerressourcen und zur Bereitstellung klar definierter Ausführungsumgebungen für Anwendungen einsetzen. [ZTT17]

Ein weiteres Problem stellt die Reproduzierbarkeit von Workflows dar. Workflows, die teils manuell auf einem Server installiert werden, stehen vor der Gefahr, von Versionsänderungen an der installierten Software oder auch Änderungen der zur Verfügung stehenden Daten beeinflusst zu werden. Ein Beispiel für einen äußeren Einfluss ist die Aktualisierung der Java JVM. Durch die Verwendung von Containern kann diese Gefahr reduziert werden, da die Ausführungsumgebung nicht durch äußere Einflüsse beeinflusst wird. [ZTT17, ST18]

Weiterhin bieten Container den Vorteil, dass Programme, die umfangreiche Abhängigkeiten bzw. Softwareanforderungen stellen, in einem isolierten Kontext operieren können. So enthält der Container bereits alle benötigten Softwareabhängigkeiten, ohne dabei auf Software, die auf dem Host-System installiert sein muss, angewiesen zu sein. Dies vereinfacht die Ausführung der Anwendung auf beliebigen Rechnersystemen sowie reduziert den Konfigurations- und Wartungsaufwand von Clustern. Der Vorteil der einfacheren Wartung, Ausführung sowie Reproduzierbarkeit birgt den Nachteil, dass Container als „Black-Box“ betrachtet werden können und der Zugriff zum Verständnis der Arbeitsweise des Containers erschwert wird. Besonders bei Fehlern bzw. Problemen innerhalb von Containern ist eine Analyse meist aufwändig. [SCC+18, ST18]

[6] `http://kubernetes.io`
[7] `https://docs.docker.com/engine/swarm/`
[8] `https://mesosphere.com/`

Laut Sweeney et al. führt das Zusammenfügen aller Abhängigkeiten eines Workflows in ein einzelnes Image zu einer vergleichsweise schlechten Performance, da, je nach Größe des Abbildes, viele Daten kopiert werden müssen, die für die durchzuführende Berechnung nicht benötigt werden. [ST18]

2.4.2 Datenlokalität bei der Ausführung von Workflows

Bei der verteilten Ausführung von Anwendungen stellt sich die Frage nach der Art der Datenbereitstellung vor Beginn einer Berechnung sowie der Datenspeicherung nach Abschluss einer Berechnung. Vor Beginn einer Berechnung müssen die erforderlichen Daten entweder auf dem jeweiligen Cluster-Knoten lokal vorhanden oder über ein verbundenes Speichersystem erreichbar sein. Nach Beenden einer Berechnung müssen die Daten entsprechend persistent abgespeichert werden. Der Overhead, der durch die notwendige Datenübertragung anfällt, lässt sich vermeiden, indem nicht die Daten zum Container verschoben werden, sondern vielmehr die Berechnungen zu den Daten. [SCC+18]

Das Prinzip Berechnungen zu Daten zu bringen („bring compute to the data“) entstand primär aus dem Missstand, dass auf vielen Clustersystemen Datenübertragungen teure Operationen sind, wohingegen Rechenkapazitäten vergleichsweise günstig sind. [Don16]

MapReduce stellt ein solches Verfahren dar, bei dem vergleichsweise einfache Operationen auf großen Datenmengen durchgeführt werden können. Hierbei handelt es sich häufig um Datenmengen, die zu groß für die Verarbeitung auf einem einzelnen Berechnungsknoten sind. [Don16]

Workflows werden meist in Form von einfachen gerichteten Graphen deklariert (DAG), die u.a. die Datenflüsse zwischen Berechnungsaufgaben darstellen. Bisher nutzen Workflow Management Systeme diese Information nicht. 2016 wurde dazu eine Doktorarbeit veröffentlicht, in der eine prototypische Anwendung entwickelt wurde, die Datenflüsse zwischen Berechnungen berücksichtigt und dementsprechend Container so ausführt, dass die Datenübertragung minimal ist. [Don16]

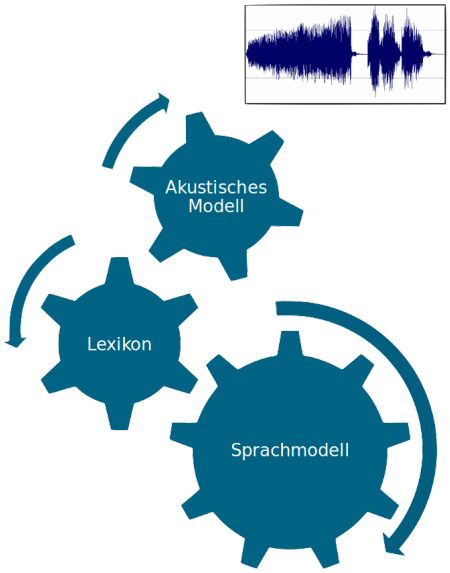


Abbildung 3: Automatische Spracherkennung

3 Sprachmodelltraining

Als Audio-Mining wird eine Technik bezeichnet, die es ermöglicht, Audioinhalte nach Wörtern oder Phrasen zu durchsuchen [Wri13]. Spracherkennungssysteme werden zur Extrahierung von Wörtern aus Audioinhalten verwendet [IAI18]. Dabei wird das Audiosignal in Text der gesprochenen Wörter umgewandelt. Dazu werden mehrere statistische Modelle sowie neuronale Netze eingesetzt, die mit Daten trainiert werden. Der Prozess zur Erkennung von Audioinhalten besteht aus drei Komponenten, die in Abbildung 3 dargestellt sind. Die erste Komponente stellt das akustische Modell dar, welches Informationen darüber enthält, wie Phoneme (altgriechisch $\varphi\omega\upsilon\acute{\eta}$, *phōnḗ*, „Laut, Ton“) einer Sprache klingen. Die zweite Komponente, das Lexikon, enthält eine Menge von Wörtern, deren Aussprache (Phonemsequenz) bekannt ist. Das Sprachmodell, welches die dritte Komponente darstellt, enthält Informationen darüber, welche Wortfolgen am wahrscheinlichsten sind. Während es für den Menschen einfach ist zu erkennen, dass die Aussage „der Vogel fliegt“ sinnvoller ist als „der Vogel flieht“, lernt der Computer dies mithilfe eines Wahrscheinlichkeitsmodells. Sowohl die Berechnung des Lexikons als auch die Berechnung des Sprachmodells werden im weiteren Verlauf unter dem Begriff des Sprachmodelltrainings zusammengefasst.

Im Folgenden wird zunächst der ursprüngliche Workflow des Sprachmodelltrainings vorgestellt, bevor anschließend die einzelnen Stufen im Detail erläutert werden. Abschließend werden verschiedene Möglichkeiten der Parallelisierung diskutiert.

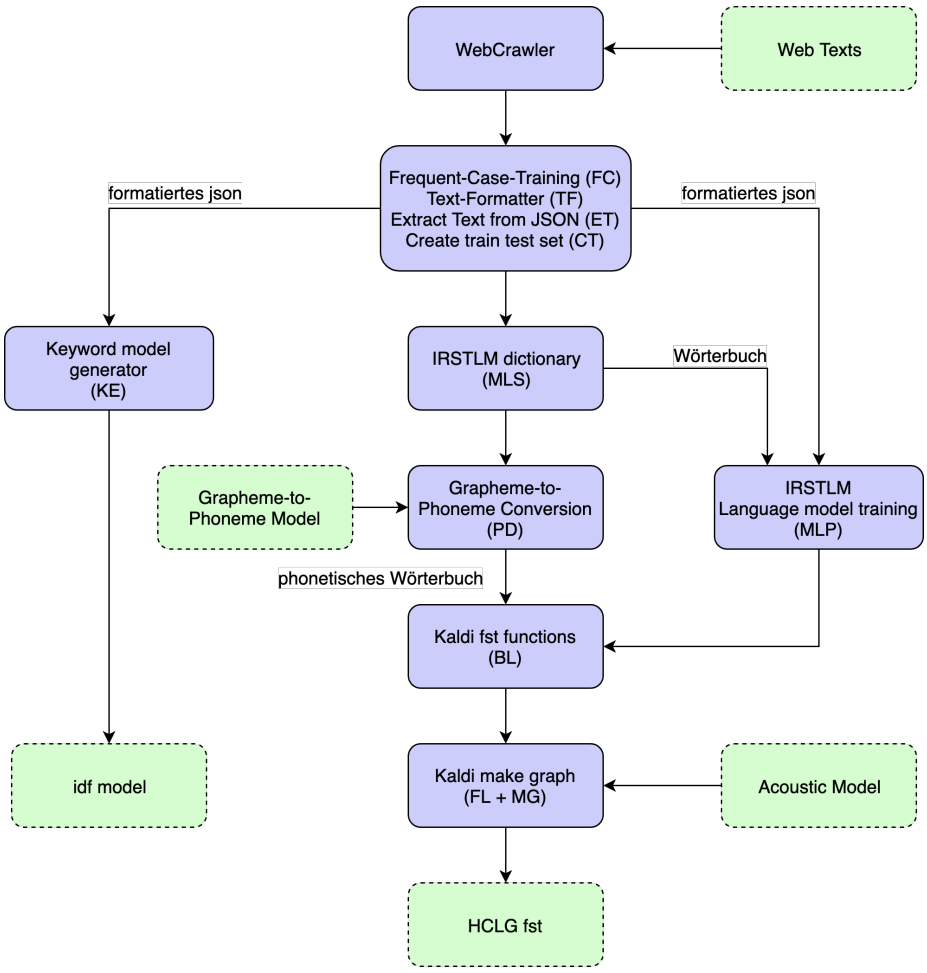


Abbildung 4: Workflow zum Training eines Sprachmodells

3.1 Workflow des Sprachmodelltrainings

Zur Berechnung des Lexikons sowie des Sprachmodells ist die koordinierte Ausführung mehrerer Programme notwendig. Abbildung 4 stellt den ursprünglichen Trainingsprozess sowie die einzelnen Verarbeitungsstufen abstrakt dar. Für das Training benötigte externe Artefakte sind grün markiert. Die erste Stufe des Trainingsprozesses „WebCrawler“ lädt Textdaten aus dem Internet herunter. Diese Stufe wird im Rahmen dieser Arbeit jedoch nicht weiter betrachtet, da zur Vergleichbarkeit von Ausführungszeiten ein fester Datensatz verwendet wird.

3.1.1 Frequent-Case-Formatter (FC)

In vielen Sprachen wird das erste Wort eines Satzes groß geschrieben. Dies kann dazu führen, dass der Computer beim Trainieren eines Sprachmodells fälschlicherweise

lernt, bestimmte Wörter auch an anderen Positionen des Satzes groß zu schreiben. Daher wird in dieser Stufe des Trainings die Vorkommenshäufigkeit einzelner Wörter in den heruntergeladenen Textdaten berechnet und als Frequent-Case-Modell gespeichert. Die heruntergeladenen Textdaten liegen für das Sprachmodelltraining im JSON Format vor. In Listing 1 ist ein Beispiel für eine Eingabedatei zu sehen. Eine mögliche Ausgabe für ein Frequent-Case-Modell ist in Listing 2 dargestellt.

```
[
  {
    "text": [ "Ein seit 27 Jahren..." ],
    "title": [ "Syrisches Ehepaar aus Borken muss..." ]
  }
]
```

Listing 1: Frequent-Case-Formatter Eingabe

```
...
abarbeitet 1503
abarbeitete 156
abarbeiteten 70
Abarbeiteten 2
Abarbeitung 295
Abarbeitungen 1
...
```

Listing 2: Frequent-Case-Modell

3.1.2 Text-Formatter (TF)

Die zuvor berechnete Vorkommenshäufigkeit von Wörtern wird in dem sogenannten Text-Formatter dazu genutzt, in den heruntergeladenen Sätzen jeweils das erste Wort eines Satzes durch die häufigste Vorkommensart eines Wortes zu ersetzen. Außerdem ist es notwendig, Zahlen durch die jeweilige ausgesprochene Version zu ersetzen. So wird die Zahl „1999“ durch die ausgeschriebene Variante „neunzehnhundertneunundneunzig“ ersetzt. Eine weitere wichtige Aufgabe des Text-Formatters ist das Tokenisieren. Das heißt, die Punktuation wird von den Wörtern getrennt. Dies verhindert, dass der Computer die Punktuation als Teil eines Wortes lernt. Beispielsweise wird aus einem Satz „Hallo, ich heiße David.“ mit entsprechenden Leerzeichen „Hallo , ich heiße David .“. Bei nicht Trennung von Wörtern und Punktuation, würde das Spracherkennungssystem das Wort „Hallo,“ als eigenständiges Wort lernen und zwischen den beiden Wörtern „Hallo,“ und „Hallo“ unterscheiden.

Der Text-Formatter nimmt sowohl die JSON Datei aus Listing 1 als auch das Frequent-Case-Modell aus Listing 2 entgegen und ersetzt Satzanfänge sowie Zahlen entsprechend. Die bestehenden Dateien im JSON Format werden dazu um zwei zusätzliche Attribute, `text_prep` und `title_prep`, erweitert. Listing 3 stellt die in Listing 1 abgebildete JSON Datei mit den neu hinzugekommenen Attributen dar.

```
[
  {
    "text": [ "Ein seit 27 Jahren..." ],
    "text_prep": [ "ein seit siebenundzwanzig Jahren..." ],
    "title": [ "Syrisches Ehepaar aus Borken muss..." ],
    "title_prep": [ "syrisches Ehepaar aus Borken muss..." ]
  }
]
```

Listing 3: Text-Formatter Ausgabe

3.1.3 Extract Text from JSON (ET)

Die durch den Text-Formatter generierten Attribute enthalten formatierten Text, der für das Training eines Sprachmodells verwendet werden kann. Die JSON Dateien enthalten jedoch viele weitere für das Training unwichtige Informationen, die durch die Stufe Extract Text from JSON entfernt werden. Dabei nimmt diese Stufe die durch den Text-Formatter formatierten JSON Dateien (vgl. Listing 3) entgegen und schreibt den Text der Attribute `text_prep` und `title_prep` zeilenweise in eine Textdatei (vgl. Listing 4).

```
ein seit siebenundzwanzig Jahren...
syrisches Ehepaar aus Borken muss...
```

Listing 4: Extract Text from JSON Ausgabe

3.1.4 Create train test set (CT)

Um die Erkennungsrate des Sprachmodells überprüfen zu können, ist es notwendig, die durch die Stufe ET erhaltenen Textdaten in Test- sowie Trainingsdaten aufzuteilen. Nach erfolgreichem Training ist es möglich, die Erkennungsrate eines Sprachmodells mithilfe der Testdaten, die das Spracherkennungsmodell zuvor noch nie gesehen hat, zu bestimmen. Für unterschiedliche Kunden ist es möglich, unterschiedliche Trainingsdaten zu verwenden, um die Erkennungsrate des Sprachmodells für bestimmte Themen zu optimieren.

3.1.5 Prepare Data Keyword IDF Estimation (KE)

Für die Berechnung von Schlüsselwörtern wird das TF-IDF (Term Frequency Inverse Document Frequency) Verfahren verwendet. IDF steht für „Inverse Document Frequency“ und beschreibt eine Methode zur Bewertung der Relevanz von Wörtern in einem Text.

Ein Wort, welches häufig vorkommt, ist nicht zwingend aussagekräftig. Beispielsweise können Artikel (der, die oder das) häufig in einem Text enthalten sein, ohne dass sie zur Bedeutung des Inhaltes beitragen. Aussagekräftige Wörter wie „Haus“ oder „Kuh“ kommen seltener vor und sind daher weniger relevant. Durch das Anwenden eines Faktors (Inverse Document Frequency Factor) wird das Gewicht von Begriffen, die häufig in einem Dokument vorkommen verringert und das Gewicht von Begriffen die selten vorkommen, erhöht. Im Kontext der Audio-Mining Anwendung ermöglicht die Extraktion der Schlüsselwörter die spätere Suche nach bestimmten Themen in Medieninhalten. [GKL18]

Eine Beschleunigung dieser Stufe wurde nicht durchgeführt, da diese Stufe in Zukunft durch eine andere Methode abgelöst wird.

3.1.6 Make Language Model (ML)

Um zu lernen, in welcher Reihenfolge bestimmte Wörter Sinn ergeben, ist es zunächst notwendig, ein Lexikon aller Wörter zu erstellen. Anschließend wird ein Wahrscheinlichkeitsmodell für Sätze berechnet, bei denen die Trainingsdaten aus Stufe CT dazu verwendet werden, die Wahrscheinlichkeit aller möglichen Sätze zu extrapolieren.

Zur Erstellung eines Sprachmodells werden die vorhandenen Textdaten zunächst in N-Gramme zerlegt. Sie sind das Ergebnis der Zerlegung eines Textes in Fragmente. Ein Fragment kann ein Zeichen, Wort oder Satz sein. Dabei werden N aufeinanderfolgende Fragmente als N-Gramm bezeichnet. Die Aussage „der Vogel fliegt“ lässt sich in folgende 2-Gramme (Wort-Fragmente) zerlegen: „der Vogel“ und „Vogel fliegt“.

Beispielsweise in einem Text über die Stadt „San Francisco“ wird die Kombination der beiden Wörter „San“ und „Francisco“ häufig vorkommen. Für den Fall von Unigrammen (1-Gram) wird sowohl „San“ und „Francisco“ eine hohe Wahrscheinlichkeit zugeordnet.

In den Trainingsdaten nicht gesehene Wörter und Phrasen (N-Gramme) können durch den Spracherkenner nicht erkannt werden. Um dies zu verhindern wird ein sogenanntes

Smoothing eingeführt, bei dem eine geringe Wahrscheinlichkeitsmasse von den gesehenen Phrasen auf ungesehene Phrasen verteilt wird. Ein Standardverfahren zur Optimierung von N-Grammen ist die Kneser-Ney Methode, die die Wahrscheinlichkeitsverteilung von N-Grammen in einem Dokument, basierend auf deren Historie, berechnet. Die Kneser-Ney Methode ist eine der am häufigsten eingesetzten Smoothing Methoden. [Sof97, Teh, Cha]

Im Kontext des Sprachmodelltrainings verwenden wir derzeit 1-5-Gramme zur Berechnung des Sprachmodells. Als Ausgabe wird eine Textdatei im ARPA Format erstellt, die die Wahrscheinlichkeit der einzelnen Wörter für N-Gramme enthält. Listing 5 zeigt einen Ausschnitt dieser Datei.

```
1-grams:
-2.88382 !
-2.94351 hallo
-6.09691 welt
...
2-grams:
-3.91009 welt !
-3.91257 hallo welt
...
3-grams:
-0.00108858 hallo welt !
```

Listing 5: Make Language Model Ausgabe (N-Gramme)

3.1.7 Phonetize Dictionary (PD)

Um die Wörter einer Sprache erkennen zu können, muss in einem phonembasierten System die Aussprache jedes Wortes, also die Umschrift in Phoneme, durchgeführt werden. Wörter sind Kombinationen verschiedener Phoneme. Dementsprechend sind Sätze Kombinationen von Wörtern, die durch Grammatik / Syntax strukturiert werden. Somit sind Phoneme in einer Sprache die Menge der bedeutungsunterscheidenden Laute.

Zur Darstellung von Phonemen existiert das International Phonetic Alphabet (IPA). Beispielsweise wird folgende Aussage „Ich esse gerne“ im IPA-Format wie folgt dargestellt: „ɪç ˈɛsə ˈgɛʁnə“.

Derzeit existieren deutsche Wörterbücher, die rund 60.000 Wörter sowie deren zugehörige Aussprache im IPA-Format enthalten. Das in Stufe `ML` erstellte Lexikon enthält jedoch mehrere Millionen Wörter. Während in den Wörterbüchern oft nur eine Wortform enthalten ist, sind in dem aus Stufe `ML` erstellten Lexikon viele verschiedene Wortformen enthalten. Um möglichst alle Wörter erkennen zu können, wird eine Extrapolation der

Aussprache für Wörter berechnet, die nicht bekannt sind. So lässt sich beispielsweise die Aussprache des Wortes „Zukunftsweisend“ durch die Aussprache der Wörter „Zukunft“ sowie „weisen“ ableiten. Nicht alle Wörter, wie zum Beispiel „Trump“ oder „Carles Puigdemont“ lassen sich korrekt extrapolieren. Sofern für diese Wörter keine korrekte Aussprache angegeben wird, werden diese durch eine deutsche Aussprache repräsentiert.

3.1.8 Build Lexicon FST (BL)

In den folgenden Abschnitten werden Finite-state transducers (FSTs) für die Modellierung aller Wahrscheinlichkeitsmodelle (akustisches Modell, Lexikon, Sprachmodell) verwendet.

Finite-state transducers sind Automaten, bei denen jeder Übergang ein Eingangslabel und ein Ausgangslabel aufweist. FSTs finden Anwendungen in Bereichen wie Spracherkennung und -synthese, maschinelle Übersetzung, optische Zeichenerkennung, Mustererkennung, Zeichenkettenverarbeitung sowie maschinelles Lernen. Häufig werden gewichtete FSTs verwendet, um ein probabilistisches Modell darzustellen (z.B. ein N-Gramm-Modell, Aussprachemodell). [opeb]

In dieser Stufe werden FSTs verwendet, um eine Überführung von Phonemsequenzen (Eingangslabel) in Wortsequenzen (Ausgangslabel) zu modellieren.

Für weitere Informationen sei auf die entsprechende Literatur verwiesen. [opea, gra, MPR02]

3.1.9 Format Language Model (FL)

Stufe `ML` speichert das Sprachmodell in einem sogenannten ARPA Format, einem Plain-Text Format, ab. Diese Stufe wandelt dieses Modell in einen Finite-state transducer (FST) um und speichert diesen binär ab. Dieser enthält die Wahrscheinlichkeiten für Wortfolgen. Dazu werden die aus Stufe `ML` berechneten N-Gramme verwendet.

3.1.10 Make Graph (MG)

Im der letzten Stufe des Sprachmodelltrainings wird aus den einzelnen FSTs, also dem akustischem Modell (`PD`), dem Lexikon (`BL`) und dem Sprachmodell (`ML`), der Graph zum Dekodieren durch Konkatenierung erzeugt.

3.2 Parallelisierung des Sprachmodelltrainings

In diesem Abschnitt wird zunächst die Ausführung des Workflows beschrieben sowie die damit verbundenen Probleme. Anschließend werden verschiedene Vorgehen zur Parallelisierung verschiedener Programme aufgezeigt, bevor abschließend für die einzelnen Stufen des Sprachmodelltrainings die im Rahmen dieser Arbeit durchgeführten Parallelisierungen erläutert werden.

3.2.1 Bisherige Ausführung

Bislang werden Sprachmodelle durch die Ausführung einer Menge von aneinandergereihten Skripten sequentiell trainiert. Zur Ausführung wird lediglich ein Thread verwendet. Die Berechnung von Sprachmodellen ist aufwändig und rechenintensiv. Wie bereits in der Einleitung erwähnt, benötigt ein Trainingsdurchlauf rund 6 Tage.

Um ein Sprachmodell für verschiedene Themengebiete trainieren zu können, ist es notwendig, die Trainingsdaten entsprechend anzupassen. Für den Workflow ist daher durch die Angabe verschiedener Parameter eine Auswahl der zu verwendenden Trainingsdaten möglich. Fehler während des Trainings, beispielsweise durch temporäre Ressourcenknappheit, müssen durch die Entwickler manuell identifiziert werden. Ein Speichern sowie Dokumentieren von Zwischenergebnissen ist derzeit manuell durch den Entwickler erforderlich.

3.3 Parallelisierung des Sprachmodelltrainings

Wie Rauber et al. schreiben, ist es zur Parallelisierung eines Programms zunächst erforderlich, dieses in kleinstmögliche Teile (Tasks) zu zerlegen. Im Anschluss werden die Abhängigkeiten der einzelnen Tasks bestimmt. Tasks stellen die kleinstmögliche Einheit der Parallelität dar, die von einer Anwendung ausgenutzt werden soll. Ein Task besteht aus einer beliebigen Folge von Berechnungen, die von einem einzelnen Prozessor ausgeführt werden. Die Granularität der Zerlegung entscheidet darüber, wie viele Tasks entstehen. Wenn eine Anwendung in zu viele Tasks zerlegt wird, ist es möglich, dass der Kommunikationsaufwand zwischen den einzelnen Tasks das Potential für eine parallele Ausführung erheblich beeinträchtigt. Zur Ausführung der Tasks existieren dynamische (vgl. Abschnitt 2.1.1) oder statische Scheduling Verfahren, die einen optimalen Ausführungsplan berechnen. [Sol15, RG07]

Für diese Arbeit sind konkrete Scheduling Verfahren nur bedingt relevant, da die Verantwortlichkeit des Scheduling bei der jeweiligen Workflow-Engine liegt.

Hingegen ist die Zerlegung des Sprachmodelltrainings in kleinstmögliche Teile durchzuführen. Dazu ist es zunächst notwendig, die einzelnen vorhandenen Skripte zu analysieren und Berechnungsschritte herauszuarbeiten. So fällt beispielsweise auf, dass in dem ursprünglichen Trainingsprozess (vgl. Abbildung 4) die Stufe Frequent-Case-Formatter sowie Text-Formatter in einer Stufe zusammengefasst sind. Für die Stufe Make Language Model besteht die Möglichkeit, die Berechnung der N-Gramme (`MLP`) parallel auszuführen. Somit ist jedoch ein zusätzlicher Berechnungsschritt zur Zusammenführung der einzelnen Teilmodelle erforderlich (Stufe `MLM`). Die Stufe `MLS` bezeichnet den Schritt zur Erstellung des Lexikons. Durch die parallele Berechnung der N-Gramme ist eine entsprechende Aufteilung der Daten vorzunehmen, die im Folgenden als Teil der Stufe (`MLS`) betrachtet wird.

Nach der Extraktion von Teilschritten des Sprachmodelltrainings als eigenständige Stufen ist es erforderlich, Abhängigkeiten zwischen den entsprechenden Stufen zu identifizieren. Dazu ist es notwendig, für jede Stufe des Trainings die Ein- bzw. Ausgaben zu definieren.

Abbildung 5 gibt einen Überblick über den gesamten Workflow des Sprachmodelltrainings. Dabei stellen die eckigen Kästen außerhalb des Datenspeichers die einzelnen Stufen des Trainings dar. Für jede Stufe ist durch die ausgehenden Kanten definiert, welche Stufe welche Daten in den Datenspeicher speichert. Ebenso definieren die eingehenden Kanten einer Stufe die benötigten Eingabedaten. Die eingehenden Kanten können als Vorbedingungen betrachtet werden, die erfüllt sein müssen, bevor eine Stufe beginnen kann. Aus der Grafik lässt sich somit ableiten, welche Teile des Trainingsprozesses parallel zueinander ausgeführt werden können und an welchen Stellen in dem Prozess eine Synchronisation stattfinden muss. Durch das direkte Verbinden von Ein- und Ausgabekanten lassen sich Abhängigkeiten zwischen den Stufen ableiten. Das in Abbildung 6 dargestellte UML Diagramm visualisiert die Abhängigkeiten der einzelnen Stufen zueinander. Aus dieser Darstellung ergeben sich implizit Teile des Sprachmodelltrainings, die parallel zueinander ausgeführt werden können. Wie dem UML Diagramm zu entnehmen ist, lassen sich nach Analyse der Abhängigkeiten sowie einer geeigneten Koordination der einzelnen Stufen des Workflows bis zu drei Aufgaben parallel ausführen.

Die Stufe des WebCrawler stellt eine Besonderheit dar, da dieser nicht direkt Teil dieses Workflows ist. Der WebCrawler ist ein eigener Dienst, der täglich fest definierte Webseiten abruft und neue Inhalte (Text) als Datei im JSON Format abspeichert. Diese Textdaten werden als Eingabe für das Training des Sprachmodells benötigt.

Eine Übersicht der Trainingsstufen sowie deren Abkürzungen ist in Tabelle 2 dargestellt.

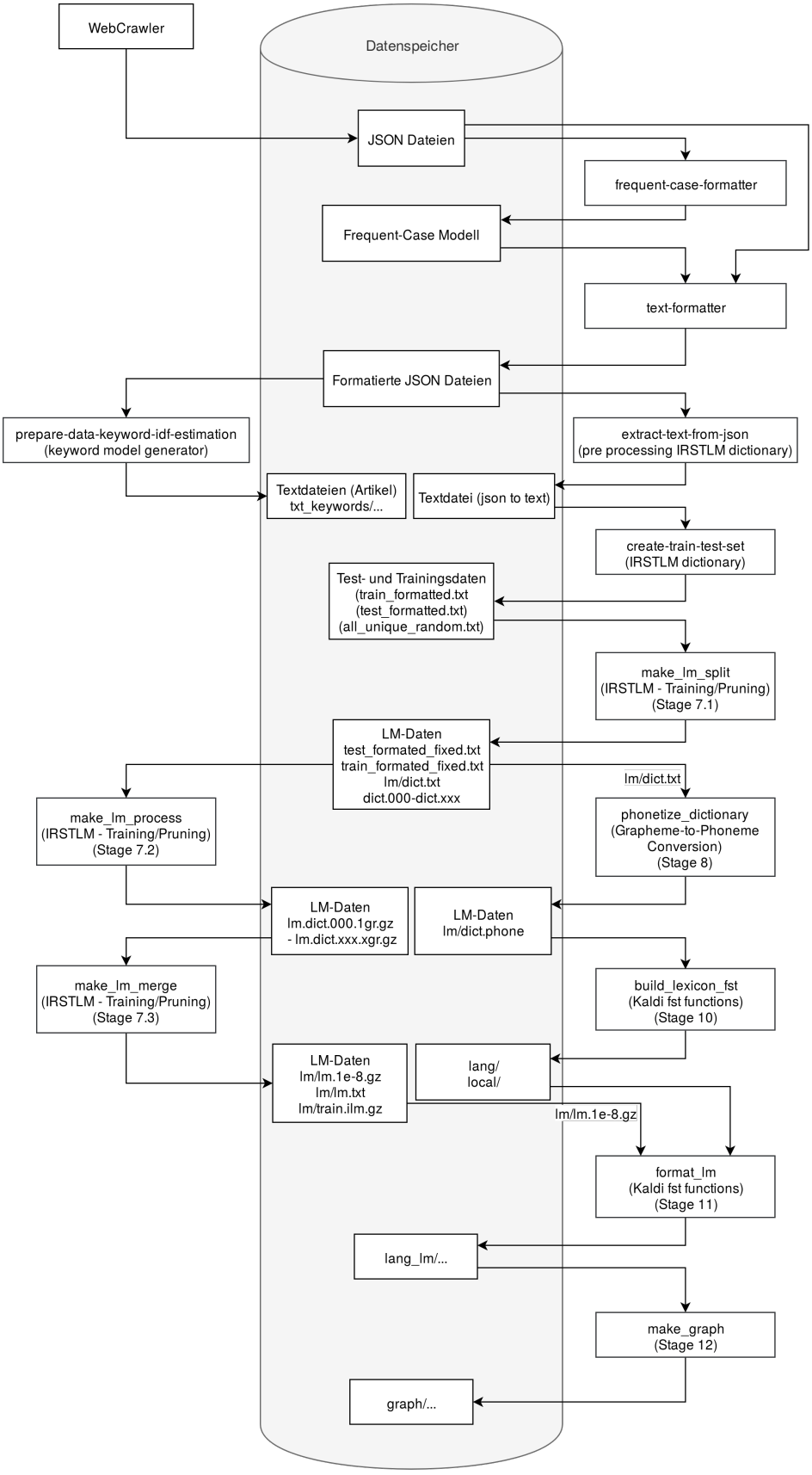


Abbildung 5: Workflow des Sprachmodelltrainings

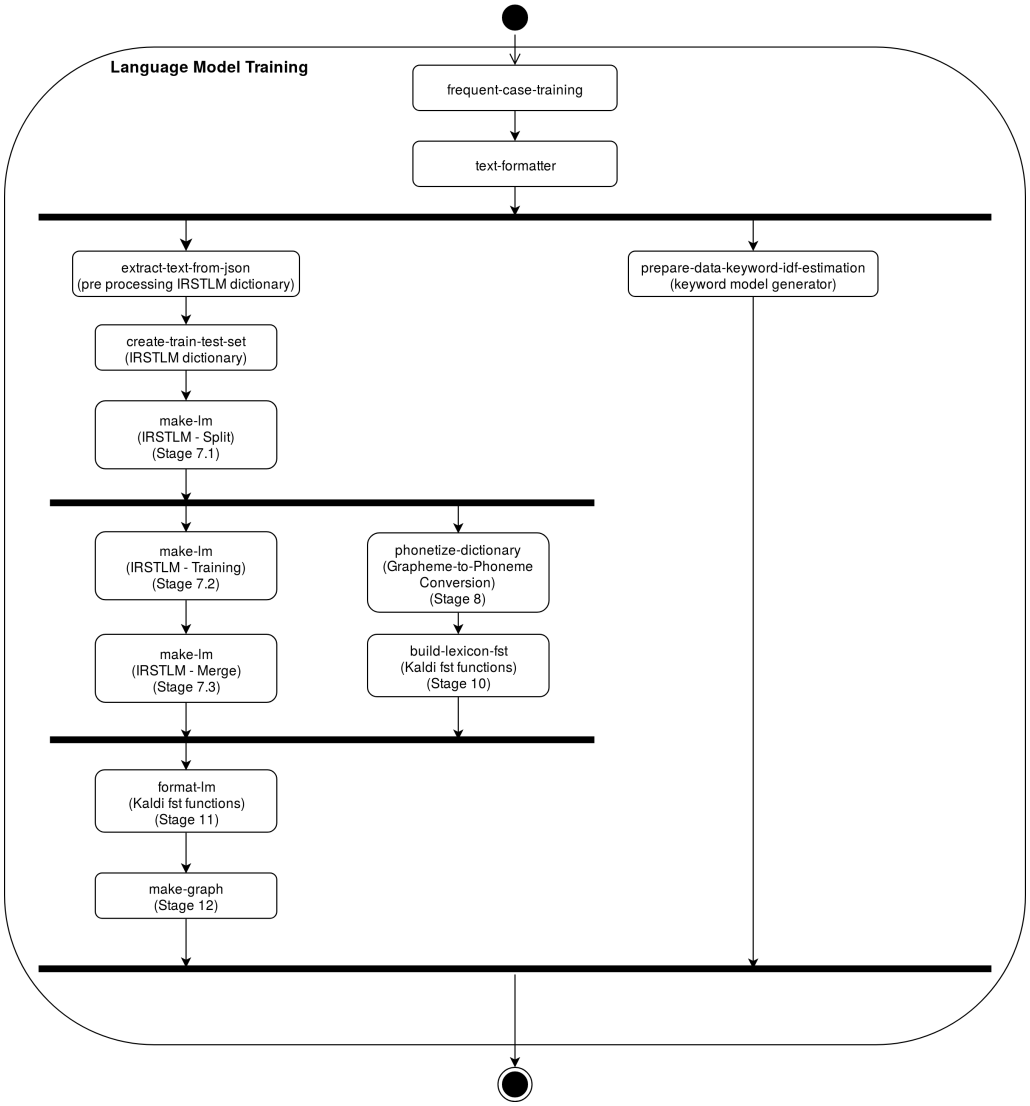


Abbildung 6: Parallelisierung des Sprachmodelltrainings (UML Aktivitätsdiagramm)

Stufe	Abkürzung
Frequent-Case-Formatter	FC
Text-Formatter	TF
Extract Text from JSON	ET
Create train test set	CT
Prepare Data Keyword IDF	KE
Make Language Model	ML
Phonetize Dictionary	PD
Build Lexicon FST	BL
Format Langage Model	FL
Make Graph	MG

Tabelle 2: Trainingsstufen

3.4 Parallelisierung einzelner Stufen

In diesem Abschnitt werden einige Stufen des Sprachmodelltrainings genauer analysiert und entsprechend parallelisiert. Um festzustellen, welche Stufen von einer Parallelisierung profitieren können, wurde das Sprachmodelltraining, wie in Abbildung 5 dargestellt, in einzelne Stufen zerlegt und anschließend sequentiell ausgeführt. Die erhaltenen Laufzeiten sind in Abbildung 7 dargestellt. Es fällt auf, dass besonders die Stufen `FC`, `TF` sowie `ML` von einer Parallelisierung profitieren können.

Aufgrund der Komplexität einzelner Stufen sowie dem Umstand, dass sich einige Algorithmen nur schwer parallelisieren lassen, wurde von einer Anpassung dieser abgesehen. Wie auch anhand der Laufzeitergebnisse in Abbildung 7 zu erkennen ist, weisen die schwer zu parallelisierenden Stufen nur eine geringe Auswirkung auf die Gesamtlaufzeit des Sprachmodelltrainings auf.

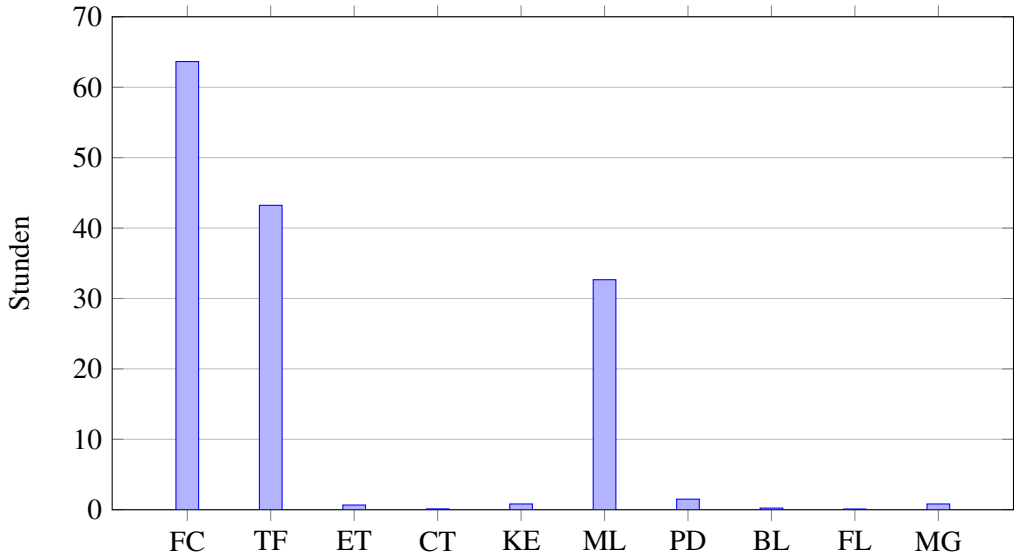


Abbildung 7: Ausführungszeiten nach Trainingsstufen (sequentiell)

3.4.1 Frequent-Case-Formatter (FC)

Die Anwendung nimmt ein Verzeichnis als Eingabeparameter an. Zur Ausführung werden alle in dem angegeben Verzeichnis enthaltenen JSON-Dateien sequentiell abgearbeitet. Dabei wird keinerlei Parallelität des Systems ausgenutzt.

Mögliche Optimierungsschritte sind die Neu- bzw. Weiterentwicklung des Frequent-Case-Formatters in einer Programmiersprache, die parallele Sprachkonstrukte enthält. Aufgrund der Beschaffenheit der Eingabedaten lässt sich Datenparallelität ausnutzen. So lässt sich für jede Textdatei ein Frequent-Case-Modell berechnen. Nach Abschluss

jeder Datei muss das erhaltene Frequent-Case-Modell mit dem „globalen“ Modell zusammengeführt werden. Dieses Problem lässt sich mithilfe des Divide-and-Conquer Paradigma lösen. Eine Implementierung dieses Paradigmas in den Algorithmus ist jedoch nur mithilfe von Architekturänderungen möglich. Daher wurde das ursprüngliche Programm zur Berechnung des Frequent-Case-Modells in einem neuen Programm gekapselt, welches das Divide-and-Conquer Paradigma implementiert. Dieses Programm erkennt, welche Dateien bereits verarbeitet wurden, und berechnet bei wiederholter Ausführung lediglich die Dateien, die neu hinzugekommen sind. Zu jeder Eingabedatei wird das entsprechende Frequent-Case-Modell gespeichert. Diese werden anschließend zu einem globalen Frequent-Case-Modell zusammengefügt. Der neue, inkrementelle Verarbeitungsfluss zur Berechnung des Frequent-Case-Modells ist in Abbildung 8 dargestellt. Die einzelnen Eingabedateien lassen sich parallel bearbeiten. Lediglich der Schritt zur Zusammenführung der Frequent-Case-Modelle wird sequentiell ausgeführt. Der Zeitaufwand dieses Schrittes ist zu vernachlässigen, da dieser für das Zusammenführen aller Teilmodelle rund 10 Minuten benötigt.

Zur Parallelisierung sowie inkrementellen Verarbeitung wurde im Rahmen dieser Arbeit ein Hilfsprogramm in Python entwickelt. Die Parallelisierung wurde mithilfe des sogenannten „Bag of Tasks“ Paradigma durchgeführt. Aufgrund der Unabhängigkeit zwischen den einzelnen Textdateien lassen sich die einzelnen Dateien als Aufgaben (Tasks) definieren. Diese werden anschließend in einer beliebigen Reihenfolge von den zur Verfügung stehenden Ressourcen abgearbeitet. In Python lässt sich dieses Paradigma mithilfe eines Prozess- oder Thread-Pool realisieren. Listing 6 zeigt eine gekürzte Version des Programms zur parallelen Ausführung des Frequent-Case-Formatter. Aufgrund der Tatsache, dass täglich maximal eine Datei pro Crawler zum Datenbestand hinzukommt, werden nach der einmaligen Verarbeitung aller Daten die Frequent-Case-Modelle nur für die neu hinzugekommenen Dateien berechnet.

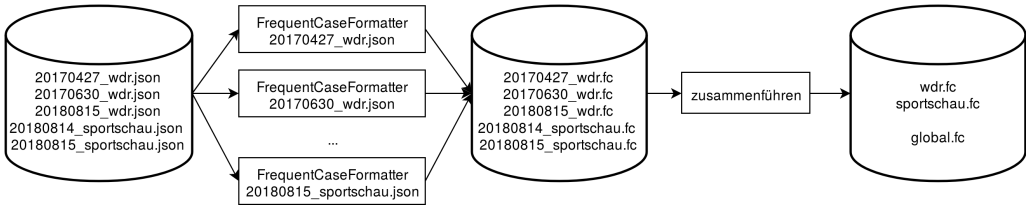


Abbildung 8: Frequent-Case-Formatter Verarbeitungsfluss

```
import multiprocessing
from concurrent.futures import ProcessPoolExecutor

cpu_count = multiprocessing.cpu_count()
pool = ProcessPoolExecutor(cpu_count)

def call_frequent_case_training(json_file):
    """Run FrequentCaseTrain on single json file"""
    # ...

for f in os.listdir("crawls/unprocessed/"):
    pool.submit(call_frequent_case_training, f)

pool.shutdown()
```

Listing 6: Frequent-Case-Formatter - Prozess-Pool Python

Durch das Starten des angepassten Frequent-Case-Formatter auf einem Computer werden alle zur Verfügung stehenden Prozessorkerne für die Berechnung genutzt. Eine verteilte Berechnung über mehrere Knoten wurde vernachlässigt, da nach dem einmaligen Training des Datensets täglich nur eine neue Datei pro Crawler hinzukommt.

3.4.2 Text-Formatter (TF)

Wie auch der Frequent-Case-Formatter arbeitet der Text-Formatter dateiorientiert. Eine inkrementelle Verarbeitung wird durch einen Vergleich des Eingabe- sowie Ausgabeverzeichnisses des Text-Formatter erreicht. Mithilfe des Vergleichs der Dateinamen können Dateien identifiziert werden, die bislang nicht verarbeitet wurden. In dieser Stufe kommt, wie auch bereits in der vorherigen Stufe zur Beschleunigung der Berechnung des Frequent-Case-Modells mithilfe eines Prozess-Pools, ein in Python entwickeltes Hilfsprogramm zum Einsatz.

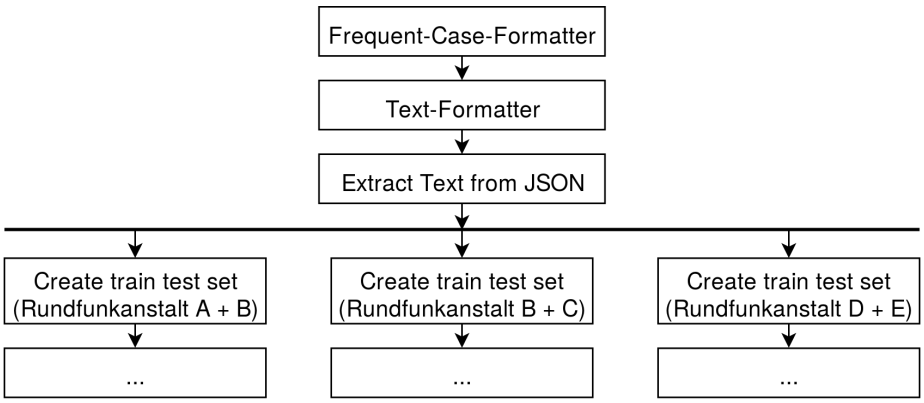


Abbildung 9: Paralleles Training verschiedener Sprachmodelle

3.4.3 Make Language Model (ML)

Die Berechnung der N-Gramme wird bislang sequentiell durchgeführt. Die für die Berechnung eingesetzte Bibliothek IRSTLM (IRST Language Modeling Toolkit) unterstützt die parallele Ausführung auf Grid Systemen, wie beispielsweise der Sun Grid Engine. Die Implementierung dieser Funktion ist auch auf andere Systeme übertragbar. Der Prozess setzt sich, wie folgt, zusammen. Zunächst wird aus den Eingabedaten ein Wörterbuch erstellt, welches anschließend in gleichgroße Teile zerlegt wird. Anschließend wird eine parallele Ausführung zur Berechnung der Wahrscheinlichkeiten durchgeführt, bevor diese Teilwahrscheinlichkeiten zu einem Modell zusammengeführt werden. Die parallele Ausführung lässt sich ebenfalls durch die Verwendung des „Bag of Tasks“ Paradigma realisieren. [Ber10, GKL18]

3.5 Weitere Möglichkeiten der Parallelisierung

Wie bereits erwähnt ist es Ziel dieser Anwendung, aus den zur Verfügung stehenden Trainingsdaten ein Sprachmodell zu trainieren. Dabei kann für ein Sprachmodell deklariert werden, welche Trainingsdaten bzw. welche Webseiteninhalte in dem jeweiligen Trainigsdurchlauf berücksichtigt werden sollen. Dies ist beispielsweise dann erforderlich, wenn Sprachmodelle für unterschiedliche Regionen mit unterschiedlichen Akzenten trainiert werden. Daraus entstehen zwei mögliche Arten der Parallelisierung. Zum einen lassen sich Sprachmodelltrainings parallel ausführen, zum anderen können die einzelnen Stufen des Sprachmodelltrainings parallelisiert werden.

Ersteres führt zu einer kürzeren Ausführungsdauer beim Training mehrerer Sprachmodelle. Hingegen führt zweiteres zur Beschleunigung der jeweiligen Trainingsdurchläufe. Die erwähnten Möglichkeiten zur Parallelisierung sind keineswegs exklusiv und werden daher kombiniert.

Bei der Ausführung mehrerer paralleler Sprachmodelltrainings ist zu beachten, dass parallel laufende Prozesse nicht dieselben Daten verarbeiten. Angenommen es sollen zwei Sprachmodelle trainiert werden (Rundfunkanstalt A + Rundfunkanstalt B) sowie (Rundfunkanstalt B + Rundfunkanstalt C). Beide Sprachmodelle enthalten die Textdaten der Rundfunkanstalt B. Wenn nun in beiden Sprachmodelltrainings der Frequent-Case-Formatter beginnt, die noch nicht verarbeiteten Daten zu berechnen, werden in beiden Anwendungen die gleichen Daten doppelt verarbeitet. Erst ab Stufe „Create train test set“ können die beiden Anwendungen parallel ausgeführt werden. Daher wird bei der Umsetzung des Workflows darauf geachtet, dass das Training verschiedener Sprachmodelle erst ab der Stufe „Create train test set“ parallel zueinander ausgeführt wird. Abbildung 9 visualisiert die parallele Ausführung für die oben genannten Modelle.

Komponente	Beschreibung
Prozessor	AMD Opteron™ Prozessor 6234 (12 Kerne / 24 Threads)
Arbeitsspeicher	128GB
Festplattenspeicher	320GB HDD
Betriebssystem	Ubuntu Server 16.04 LTS

Tabelle 3: Server Informationen

4 Umsetzung des parallelisierten Sprachmodelltrainings

Um zu gewährleisten, dass das Training von Sprachmodellen reproduzierbar sowie skalierbar ist, werden im Rahmen dieser Arbeit containerbasierte Workflow-Engines untersucht. Zunächst ist zur Ausführung von Anwendungen mithilfe von Docker, eine Containerisierung der bestehenden Anwendung bzw. der einzelnen Stufen der Anwendung durchzuführen. Nach einer Beschreibung des Prozesses zur Containerisierung, der damit verbundenen Probleme sowie einer Evaluation des eingesetzten Datenspeichers wird die Einrichtung eines Clusters mithilfe von Rancher beschrieben. Abschließend werden verschiedene Workflow-Engines vorgestellt, die ebenfalls hinsichtlich der Eignung zum Training eines Sprachmodells evaluiert werden.

Für die im Rahmen dieser Arbeit durchgeführten Laufzeitmessungen wurden drei Server mit identischer Ausstattung des Fraunhofer Instituts genutzt. Die Spezifikationen der einzelnen Server des Clusters sind in Tabelle 3 dargestellt.

4.1 Containerisierung

In diesem Abschnitt werden nach einer Beschreibung derzeitiger Probleme des Sprachmodelltrainingsprozesses, verschiedene Lösungsansätze evaluiert. Abschließend werden, unter Berücksichtigung der Integration in Docker, verschiedene Möglichkeiten der Datenspeicherung untersucht.

4.1.1 Problem

Ein Problem des bisherigen Trainingsprozesses stellt die Einrichtung sowie Installation der benötigten Anwendungen dar. Neben den Anwendungen selbst werden Abhängigkeiten sowie entsprechend konfigurierte Umgebungsvariablen vorausgesetzt. Dies schränkt die mögliche Ausführungsplattform ausschließlich auf Linux basierter Systeme

ein. Einige der benötigten Abhängigkeiten sind nicht als ausführbare Programme verfügbar, sondern müssen für die jeweilige Plattform manuell kompiliert werden. Aufgrund von fehlenden Abhängigkeiten in Ubuntu 18.04 ist das Audio-Mining Projekt derzeit ausschließlich mit Ubuntu 16.04 kompatibel. Weiterhin ist eine fest definierte Verzeichnisstruktur erforderlich, um eine Zusammenarbeit der verschiedenen Komponenten zu ermöglichen.

Zur Reduktion des Installationsaufwandes sowie zur Vermeidung der beschriebenen Probleme wurde ein Docker-Container erstellt, der alle notwendigen Softwarekomponenten enthält. Allerdings ist das Abbild dieses Docker-Containers rund 15GB groß. Das Bauen des Containers benötigt, aufgrund der notwendigen Kompilierung der verschiedenen Anwendungen, mehrere Stunden. Aufgrund der vorhandenen Infrastruktur benötigt der Transfer des Docker-Abbildes mehrere Stunden. Aus diesem Grund eignen sich Abbilder, die alle Abhängigkeiten enthalten, für lokale Tests, jedoch aufgrund der Größe des Abbildes nicht zur produktiven Ausführung auf mehreren Servern (vgl. Abschnitt 2.4.1).

Wie bereits in Abschnitt 2.4.1 erläutert, werden Anwendungen durch den Einsatz von Docker-Abbildern reproduzierbar. Durch die Bereitstellung einer vordefinierten Umgebung ist garantiert, dass die Programme plattformübergreifend funktionieren. Durch die Definition des Trainingsprozesses als Workflow lässt sich für jede Stufe des Trainings ein eigener Docker-Container erstellen. Diese Container enthalten ausschließlich die zur Ausführung dieser Stufe benötigten Daten, um die Übertragung der Abbilder aus der Fraunhofer Docker Registry zu beschleunigen. Eine der angewendeten Optimierungstechniken ist das sogenannte „Multi-Stage Docker Build“, welches in Abschnitt 4.1.2 erläutert wird.

Stufe	Abkürzung	Docker-Abbild
Frequent-Case-Formatter	FC	frequent-case-formatter
Text-Formatter	TF	text-formatter
Extract Text from JSON	ET	extract-text-from-json
Create train test set	CT	create-train-test-set
Prepare Data Keyword IDF Est.	KE	prepare-data-keyword-idf-estimation
Make Language Model	ML	kaldi-decoding-graph
Phonetize Dictionary	PD	kaldi-decoding-graph
Build Lexicon Fst	BL	kaldi-decoding-graph
Format Langage Model	FL	kaldi-decoding-graph
Make Graph	MG	kaldi-decoding-graph

Tabelle 4: Trainingsstufen sowie deren zugehörige Docker-Abbilder

4.1.2 Multi-Stage Docker Build

Mithilfe des sogenannten Docker Multi-Stage Builds lassen sich verschiedene Docker-Abbilder aufeinander aufbauen. Daher wurde zur Reduktion der Abbildgröße für jede Stufe des Sprachmodelltrainings ein eigenes Abbild erstellt. Während die Basis der jeweiligen Abbilder identisch ist, enthält jedes Abbild die für die entsprechende Stufe notwendigen Anwendungen. In Listing 7 ist das `Dockerfile` zur Erstellung des „Text-Formatter“ Containers abgebildet. [Ell, Inc18]

```
1  FROM fraunhofer.de/iais.nm/speech/training/speech-environment:0.3.0 as builder
2  FROM fraunhofer.de/iais.nm/speech/training/base-image-python:0.3.2
3
4  COPY --from=builder /opt/speech_environment/ifinder/bin/TextFormatter ./ifinder/
       bin/TextFormatter
5  ENV PATH="/opt/speech_environment/ifinder/bin/:${PATH}"
6
7  COPY prepare_data.sh prepare_data.sh
8  COPY text_formatter.py text_formatter.py
9
10 ENTRYPOINT [ "bash", \
11              "/opt/speech_environment/prepare_data.sh", \
12              "crawled_data", \
13              "frequent_case_model", \
14              "add_txt", \
15              "1" ]
```

Listing 7: Dockerfile des Text-Formatter Containers

In der produktiven Version des Abbildes können so Anwendungen, die zum Bauen der benötigten ausführbaren Dateien erforderlich sind, wie beispielsweise Compiler, fehlen. Abbildung 10 zeigt die Abhängigkeiten der verschiedenen Docker-Abbilder. Das Basisabbild enthält alle notwendigen Anwendungen zum Bauen der Anwendung wie make, cmake und Python. Die in der Abbildung angegebene Größe bezieht sich jeweils auf den benötigten Speicher des jeweiligen Abbildlayer. So ergibt sich beispielsweise für das Abbild des `text-formatter` eine Gesamtgröße von $140MB + 360MB + 3MB = 503MB$. Wenn bereits das „BaseImagePython“ auf einem Knoten des Cluster gespeichert ist, ist bei der Anforderung des Abbilds „Text-Formatter“ lediglich die Differenz zur jeweils noch nicht gespeicherten Abbildlayergröße zu übertragen.

4.1.3 Datenspeicherung

Aufgrund der ursprünglichen verzeichnisorientierten sowie sequentiellen Bearbeitung des Trainingsprozesses wurde auf das abteilungsinterne RAID-System mithilfe des NFS

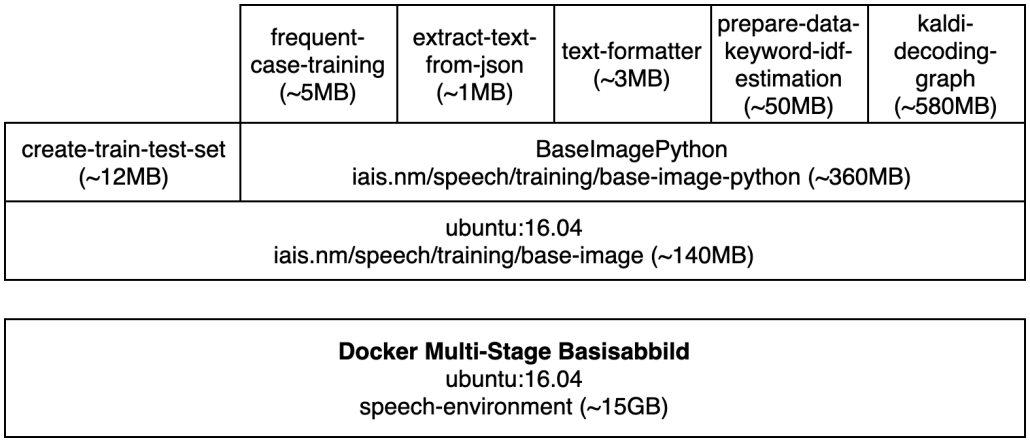


Abbildung 10: Multi-Stage Docker Builds Abhängigkeiten

(Network File System) Protokolls zugegriffen. Bei der Parallelisierung der Anwendung konnten verschiedene Probleme mit dieser Art der Datenspeicherung identifiziert werden. Zum einen ist, aufgrund der zentralen Speicherung, keine Datenlokalität möglich. Weiter befindet sich das Cluster an einem physikalisch anderen Ort als das RAID-System, weshalb die Bandbreite zwischen Server und RAID-System ein Bottleneck darstellt. Zur Verwendung des NFS Protokolls müssen sich beide Geräte im selben Netzwerk befinden. Somit ist es nur bedingt möglich, ein externes Cluster, beispielsweise die Google Kubernetes Engine oder Amazon AWS, für Berechnungen miteinzubeziehen.

Im Rahmen dieser Arbeit wurden zwei Arten zur Speicherung von Daten verglichen. Folgende Kriterien gilt es zu beachten:

- Speichern von JSON-, Text- sowie Binärdateien
- Beliebig große Dateien (< 1TB)
- Löschen von „alten“ Daten (optionales Kriterium)

MongoDB Aufgrund der vorliegenden Daten im JSON Format wurde zunächst der Einsatz von MongoDB analysiert. MongoDB ist eine dokumentenorientierte Datenbank, die Daten im JSON Format speichert. Die maximale Dateigröße für ein JSON Dokument liegt bei 16MB. Für größere Daten stellt MongoDB das sogenannte „GridFS“ zur Verfügung. Daten in dem GridFS sind lediglich als „Datei“ erreichbar und nicht über Eigenschaften des JSON Dokument. Die Dateigrößen der zu analysierenden Dokumente reichen von wenigen Kilobytes bis 100MB. Ebenfalls setzt der Einsatz von MongoDB grundlegende Änderungen an der bisherigen Anwendung voraus, da die bisherige Art der Speicherung dateibasiert ist und keine Möglichkeit zum Beziehen von Daten aus einer Datenbank vorgesehen ist.

S3 Speicher Moderne Workflow-Engines unterstützen häufig nur über Umwege die Einbindung von NFS Laufwerken. Hingegen wird der Objektspeicher „Amazon S3“ von vielen Workflow-Engines unterstützt. Falls eine Speicherung der zu verarbeitenden Daten auf Amazon Servern nicht möglich / gewünscht ist, können alternativ Open-Source Implementierungen eines S3 basierten Speichers herangezogen werden. Eine „On-Premises“ Speicher-Lösung ist zu bevorzugen, sofern die Berechnungen ebenfalls „On-Premises“ durchgeführt werden. Wie bereits in Abschnitt 2.2.2 erwähnt, ist eine hohe Bandbreite entscheidend für die effiziente Ausführung, da die Übertragungsgeschwindigkeiten des Netzwerks häufig einen Bottleneck darstellen. Open-Source Alternativen für Amazon S3 Speicher sind beispielsweise Minio[9] oder Ceph[10].

Ceph ist, wie auch NFS, ein verteiltes Dateisystem. Ceph bietet drei verschiedene Arten von Speicher an: Swift / S3 kompatiblen Objektspeicher (RADOS Gateway), virtuelle Blockgeräte (RADOS Block Devices) und ein verteiltes Dateisystem (CephFS). Objekte werden verteilt und redundant gespeichert, so dass bei einem Ausfall einer Komponente das System automatisch „geheilt“ werden kann. Aufgrund der Tatsache, dass die Installation sowie Einrichtung im Vergleich zu Minio aufwändiger ist und im Rahmen dieser Arbeit lediglich eine Teilmenge der von Ceph bereitgestellten Funktionen genutzt wird, wird Ceph nicht weiter evaluiert.

Im Vergleich zu Ceph handelt es sich bei Minio um einen reinen Objektspeicher. Minio eignet sich für die Datenspeicherung von unstrukturierten Daten wie beispielsweise Text-Dateien, Log-Dateien oder Container-Abbildern. Die maximale Größe für ein Objekt (Datei) liegt bei 5TB. Minio ist als Docker-Abbild verfügbar, wodurch die Installation eines einzelnen, nicht-verteilten Datenspeichers mithilfe eines Befehls erfolgen kann. Ein weiterer Vorteil der Verwendung von Minio ist, dass der Speicherort des S3 Speichers über den Docker-Run Befehl gesteuert werden kann. So lässt sich beispielsweise bei ausreichend großem Arbeitsspeicher ein S3 Speicher anlegen, dessen Daten im Arbeitsspeicher abgelegt werden, um die Übertragungsgeschwindigkeit weiter zu erhöhen. Zu beachten ist, dass die Bandbreite zwischen Knoten im Netzwerk entsprechend groß sein muss. Entsprechende Befehle zum Starten der jeweiligen Minio Container sind in Listing 8 dargestellt.

Durch die Replikation der Daten an allen Knoten im Cluster kann eine Datenlokalität erreicht werden. Aufgrund von begrenztem Speicherplatz im Cluster wird dieser Ansatz jedoch nicht weiter verfolgt.

Aufgrund der Unterstützung durch eine Vielzahl von Workflow-Engines wird eine „On-Premises“ Installation von Minio als zentraler Datenspeicher für alle im Rahmen dieser

[9] `https://www.minio.io/`
[10] `http://docs.ceph.com/docs/mimic/radosgw/s3/`

Arbeit durchgeführten Laufzeitmessungen verwendet.

```
# using temporary storage
docker run \
    -e "MINIO_ACCESS_KEY=minio" -e "MINIO_SECRET_KEY=minio123" \
    minio/minio server /data

# using hdd/sdd (mount host directory)
docker run \
   -e "MINIO_ACCESS_KEY=minio" -e "MINIO_SECRET_KEY=minio123" \
   -v ~/development/minio/:/data/ \
   minio/minio server /data

# using tmpfs (90gb limit)
docker run \
    -e "MINIO_ACCESS_KEY=minio" -e "MINIO_SECRET_KEY=minio123" \
    --tmpfs /data/:rw,noexec,nosuid,size=94371840k \
    minio/minio server /data
```

Listing 8: Minio Installationsbefehle

Docker Integration Für Minio existiert eine Kommandozeilen Anwendung, über die ein Kopieren von Daten möglich ist. Dies erfordert jedoch die Einbindung von zusätzlicher Logik in den jeweiligen Docker-Containern, um den Kopiervorgang vor bzw. nach einer Berechnung durchzuführen. Listing 9 zeigt die zusätzlich benötigten Anweisungen zum Kopieren der Eingabedaten bzw. Ergebnisse.

Alternativ bietet Docker über den Einsatz eines Plugins die Möglichkeit, S3 Speicher in Docker-Container als Verzeichnis einzubinden. Im Gegensatz zu dem vorherigen Ansatz ist somit kein zusätzlicher Code in den Containern erforderlich. Außerdem können Berechnungen bereits starten, ohne zunächst auf den Abschluss des Kopiervorgangs warten zu müssen. Ein Nachteil dieser Methode ist, dass das Plugin in Docker zu installieren ist. Je nach Ausführungsplattform (On-Premises / Cloud / etc.) könnte dies, beispielsweise aufgrund von fehlenden Berechtigungen, nicht möglich sein.

```
mc cp --recursive minio/audio-mining/crawls/unprocessed/ /crawls/
FrequentCaseFormatter ...
mc cp --recursive /crawls/ minio/audio-mining/crawls/processed/
```

Listing 9: Minio Kopiervorgang

Datenherkunft (engl. „Data Provenance“) bezeichnet die Möglichkeit, den Weg eines Ergebnisses über eine Vielzahl von Berechnungen hinweg bis zu dem ursprünglichen

Eingabedatum zurückverfolgen zu können. Dazu gehören alle durchgeführten Berechnungen sowie Zwischenergebnisse. [sii, pac]

Einige Workflow-Engines bieten durch die Verwendung eines Versionsverwaltungssystems die Möglichkeit, für jedes Datum einer Berechnung die Herkunft zu bestimmen. Somit kann für jedes Ergebnis nachvollzogen werden, mit welchem Algorithmus aus welchen Daten das jeweilige Ergebnis berechnet wurde.

4.2 Cluster Verwaltung

Zur Gewährleistung der Reproduzierbarkeit werden containerbasierte Workflow-Engines eingesetzt (vgl. Einleitung Kapitel 4). Da viele der untersuchten Workflow-Engines die Ausführung mittels Kubernetes unterstützen, wird im Rahmen dieser Arbeit Kubernetes als Ausführungsplattform verwendet.

Eine Projektanforderung ist, dass die Berechnung des Sprachmodelltrainings innerhalb des Fraunhofer Instituts durchgeführt wird. Somit ist die Nutzung von Managed Kubernetes wie „Google Kubernetes Engine“ oder „Amazon Elastic Container Service for Kubernetes“[11] nicht möglich. Für die im Rahmen dieser Arbeit durchzuführenden Experimente wurde das in der Einleitung zu Kapitel 4 vorgestellte Cluster verwendet. Zur Installation sowie Verwaltung eines On-Premises Kubernetes Cluster wurden verschiedene Managementplattformen untersucht.[12] Das Betriebssystem DC/OS (Distributed Cloud Operating System) wurde nicht, entgegen der ursprünglich Planung, untersucht. Grund dafür ist, dass eine Neuinstallation des Betriebssystems seitens des Fraunhofer Instituts nicht möglich ist.

Eine Plattform, die keine Neuinstallation des Betriebssystems erfordert, stellt Rancher dar. Rancher ist ein Open-Source Anwendung zur Verwaltung sowie Orchestrierung von Kubernetes Clustern. Zum 01. Mai 2018 wurde Rancher in Version 2.0 der Kubernetes Management Plattform veröffentlicht. Beginnend mit Version 2.0 ist es möglich, lokale Cluster sowie entfernte Kubernetes Cluster zu verwalten. Eine Installation eines Kubernetes On-Premises Clusters ist möglich. Dazu wird die eigens entwickelte Rancher Kubernertes Engine (RKE) verwendet.

Neben der Nutzerverwaltung ist es möglich, Projekte anzulegen sowie Kubernetes Namespaces zu verwalten. Ein Alerting-Modul kann konfiguriert werden, um Entwickler oder Cluster-Administratoren über vordefinierte Zustände im Cluster zu informieren bzw. zu warnen. Wenn beispielsweise ein Lastlimit überschritten wird oder der Festplattenspeicher knapp wird, lässt sich eine vordefinierte Warnmeldung versenden. Dies

[11] `https://kubernetes.io/docs/setup/pick-right-solution/`
[12] https://blog.kublr.com/choosing-the-right-containerization-and-cluster-management-tool-fdfcec5700df

ist im Kontext der Audio-Mining Anwendung hilfreich, da so automatisiert Probleme des Clusters an entsprechende Stelle weitergeleitet werden können, ohne dass der Status des Clusters manuell überprüft werden muss.

Derzeit befindet sich eine Möglichkeit des Monitoring von Clustern in der Entwicklung.[13] Dabei geschieht das Monitoring vollautomatisch, berücksichtigt Benutzerrechte und ermöglicht eine Ansicht auf Projekt sowie Anwendungsebene. Aus den so gewonnenen Metriken lassen sich Benachrichtigungsregeln ableiten. Die Visualisierung erfolgt mithilfe der Open-Source Plattform Grafana. Die Möglichkeit, die Auslastung des Clusters zu überwachen, ist besonders hilfreich, um mögliche Probleme oder Abstürze des Sprachmodelltrainings, beispielsweise durch den Überlauf von Arbeitsspeicher, erkennen zu können.

Im Rahmen dieser Arbeit wurden verschiedene Skripts entwickelt, die eine vollautomatische Installation eines Kubernetes Clusters mit einer beliebigen Anzahl von Nodes und einer Monitoring Lösung auf Basis von Docker Swarm ermöglichen. Die Skripts sowie eine ausführliche Anleitung sind auf GitHub[14] verfügbar.

4.3 Workflow-Engines

Dieses Kapitel analysiert zunächst die existierenden Workflow-Engines. Mithilfe verschiedener Kriterien erfolgt eine Auswahl relevanter Workflow-Engines für das Sprachmodelltraining.

Im Folgenden werden die im Rahmen dieser Arbeit untersuchten Workflow-Engines hinsichtlich des Sprachmodelltrainings verglichen. Für jede Workflow-Engine werden Erkenntnisse beschrieben, die für bzw. gegen die Nutzung der jeweiligen Workflow-Engine im Rahmen des Sprachmodellstrainings sprechen. Eine Gliederung lässt sich wie folgt aufstellen:

- Installation und Einrichtung
- Architektur
- Notwendige Anpassung der Anwendung
- Ausführung
- Oberfläche
- Fazit

[13] `https://rancher.com/blog/2018/2018-10-18-monitoring-kubernetes/`
[14] `https://github.com/David-Development/cluster-setup`

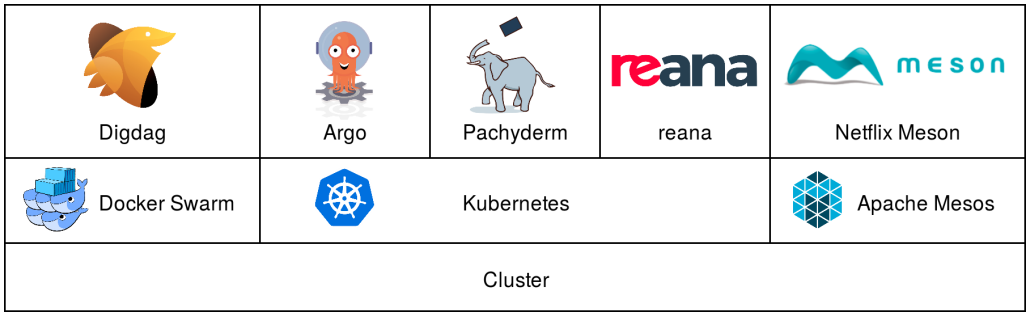


Abbildung 11: Untersuchte Workflow-Engines

In Abbildung 11 sind die untersuchten Workflow-Engines sowie deren zugehörige Ausführungsplattform abgebildet. Obwohl die Workflow-Engine „Meson“ nicht getestet wurde, ist sie aus Gründen der Vollständigkeit in der Abbildung enthalten.

4.3.1 Übersicht

Auf der Webseite der Common Workflow Language findet sich eine Liste bereits existierender Workflow-Engines.[15] Diese Liste wird laufend aktualisiert und enthält derzeit über 230 verschiedene Systeme. Dies verdeutlicht die Notwendigkeit von Workflow-Engines. Bei genauerer Analyse fällt auf, dass ein Großteil dieser Workflow-Engines eigene Sprachstandards mitbringt und die Systeme untereinander nicht kompatibel sind. Viele Systeme wurden für spezielle Anwendungsfälle entwickelt und sind nur schwer auf andere Probleme übertragbar. Weiter fällt auf, dass viele Workflow-Engines unvollständig und mangelhaft dokumentiert sind, so dass eine Abschätzung des Funktionsumfanges bzw. der Funktionen der jeweiligen Projekte schwierig ist. Dies erschwert die Möglichkeit, Projekte zu testen. Daher wurde eine Analyse einer Vielzahl dieser GitHub Projekte durchgeführt, um die Weiterentwicklungsaktivität der jeweiligen Workflow-Engine zu messen. Als Kriterium wurde die Anzahl der wöchentlichen Commits über die letzten Monate betrachtet, um somit einen Trend identifizieren zu können. Durch eine Analyse der GitHub Seiten der entsprechenden Projekte wurde anhand der Commit-Historie festgestellt, dass nur ein Bruchteil der Systeme aktiv weiter entwickelt werden.

Im August 2016 untersuchten Kowalik et al. verschiedene Workflow-Engines für eine Batch-Verarbeitungsanwendung des „Large Synoptic Survey Telescope“ in Chile. Zum damaligen Zeitpunkt beinhaltete die o.g. Liste aller Workflow-Engines rund 70 Systeme. Ein Problem stellt die Anpassung bzw. Konzeption der Workflow-Engines auf spezielle Anwendungsfelder bzw. Ökosysteme dar. Bereits 2016 konnte kein klarer „Gewinner“ identifiziert werden, so dass die acht vielversprechendsten Workflow-Engines untersucht

[15] `https://s.apache.org/existing-workflow-systems`

wurden. Untersucht wurden: Apache Airflow, CloudSlang, Makeflow, PanDA, Pegasus, Pinball, RADICAL-Pilot sowie Swift. [KCDK]

Die GitHub Seite „awesome-workflow-engines“[16] enthält eine Liste verschiedener aktueller Workflow-Frameworks bzw. Systeme, die größtenteils durch namhafte Unternehmen wie Spotify, Google oder Adobe unterstützt oder verwendet werden. Neben einer Reihe von Workflow-Engines mit eigenem Sprachstandard, wie beispielsweise Airflow, Argo oder Digdag, gibt es zum aktuellen Zeitpunkt verschiedene Workflow-Engines wie jBPM, Activiti, Flowable und Imixs-Workflow, die den aktuellen Business Process Model and Notation (BPMN) Standard u.a. in Version 2 implementieren.

Um geeignete Workflow-Engines für das Training eines Sprachmodells zu identifizieren, wurden zunächst einige Kriterien definiert, die eine Workflow-Engine unterstützen muss. Aufgrund der Vielzahl von existierenden Workflow-Engines sowie der mangelhaften Dokumentation wurden im Rahmen dieser Untersuchung nur Workflow-Engines betrachtet, die über einen zwölfmonatigen Zeitraum mehr als fünf Commits pro Woche aufweisen konnten. Eine anhaltende Commit-Aktivität ist erforderlich, um bei Problemen sowie Fehlern in der Workflow-Engine, Unterstützung zu erhalten. Bei Projekten mit geringer Aktivität häufen sich die Fehlermeldungen in den entsprechenden GitHub Seiten bereits, ohne beantwortet zu werden. Durch diese erste Filterung kann eine Vorauswahl an Projekten, wie in Abbildung 12 dargestellt, getroffen werden. Es ist zu beachten, dass Workflow-Engines wie Cromwell[17], Apache NiFi[18] sowie cctools[19] aus Gründen der Lesbarkeit aus dem Diagramm entfernt wurden, da sie eine vergleichbare Commit-Historie wie Pegasus[20] aufweisen. Trotz der hohen Anzahl an Commits pro Woche wird die Workflow-Engine toil nicht untersucht, da lediglich die Ausführung auf Amazon AWS, Microsoft Azure und Google Compute Engine möglich ist. Aufgrund der Anforderung, das Sprachmodelltraining innerhalb des Fraunhofer Campus durchzuführen, kommt keine dieser Cloud-Plattformen in Frage.

4.3.2 Kriterien

Die im Rahmen dieser Arbeit definierten Kriterien beruhen auf verschiedenen Kriterien, die bereits in anderen Studien sowie Forschungsprojekten zum Vergleich von Workflow-Engines herangezogen wurden.

[16] `https://github.com/meirwah/awesome-workflow-engines`
[17] `https://github.com/broadinstitute/cromwell`
[18] `https://nifi.apache.org/`
[19] `https://github.com/cooperative-computing-lab/cctools`
[20] `https://pegasus.isi.edu/`

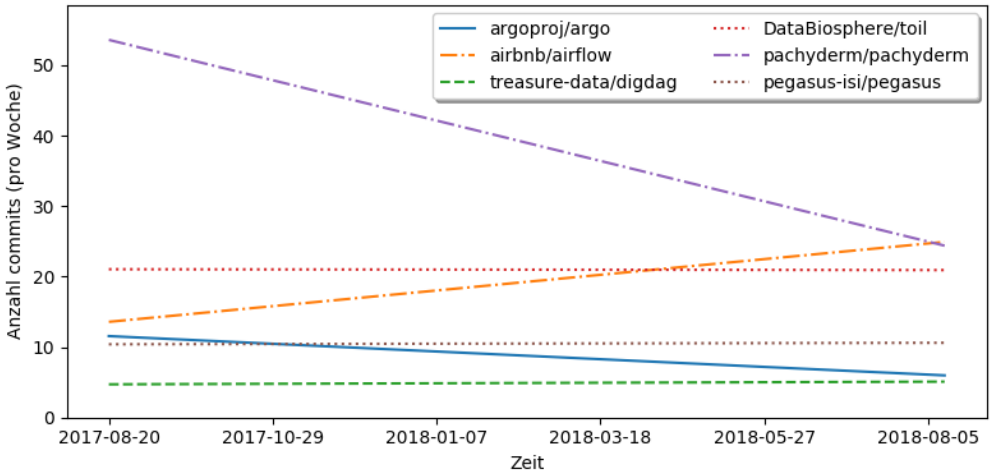


Abbildung 12: Commit Aktivität von Workflow-Engine GitHub Repositories

Beispielsweise definieren Kowalik et al. folgende Kriterien zur Bewertung von Workflow-Engines. So beziehen sich technische Merkmale, die berücksichtigt werden müssen, auf die dem Nutzer entgegengebrachte Flexibilität (GUI / Code / Skript), die Interoperabilität, die Skalierbarkeit, die Performance sowie die Möglichkeit des Monitoring von Workflows über eine grafische Oberfläche. Ein weiteres Kriterium stellt der entgegengebrachte Support dar. Hier wird beispielsweise bewertet, ob Entwickler auf Fragen antworten und wie groß die Community hinter den Open-Source Projekten ist. Als letztes Kriterium wird die User-Experience genannt, die u.a. die Vollständigkeit der Dokumentationen, den Aufwand zur Einrichtung eines Workflow-Engines sowie die Lernkurve, die zum Erstellen von Workflows benötigt wird, bewertet. [KCDK]

Taylor et al. definieren zunächst drei generische Designziele für Workflow-Engines. Zunächst sollen Standards verwendet werden, um die Wiederverwendbarkeit von Workflows zu erhöhen. Außerdem ist eine Integration bzw. Bereitstellung von Monitoring Schnittstellen erforderlich, um die Ausführung der Workflows über einen Web Service zu überwachen. Als spezifische Anforderungen für wissenschaftliche Anwendungen wird die Integration mit bestehenden Anwendungen genannt („legacy code“). Des Weiteren wird eine „experimental flexibility“ gefordert. Dies bedeutet, dass wissenschaftliche Workflow-Sprachen die inkrementelle Entwicklung von Workflows, wie beispielsweise das Entfernen von Aufgaben aus laufenden Workflows, unterstützen sollten. Eine wichtige Eigenschaft von wissenschaftlichen Anwendungen ist die Reproduzierbarkeit sowie die Möglichkeit die Herkunft von Daten für einen bestimmten Workflow zu überprüfen. Die Komposition von bestehenden Workflows in neue Workflows ist neben der Unterstützung von lang laufenden Berechnungen sowie der Unterstützung zur Ausführung einer großen Anzahl von Workflows, als Kriterium zu erwähnen. [Tay07]

Neben diesen allgemeinen Kriterien lassen sich weitere Kriterien definieren, die besonders im Kontext des Sprachmodelltrainings relevant sind. Die Berechnung von Sprachmodellen ist aufwändig und rechenintensiv. So muss zunächst eine hinreichend große Menge an Textdaten vorliegen, um ein Sprachmodell trainieren zu können. Das eigentliche Training lässt sich in verschiedene Stufen unterteilen. So sind zunächst mehrere Vorverarbeitungsschritte notwendig, bevor das eigentliche Training des Sprachmodells, welches sich ebenfalls aus mehreren Stufen zusammensetzt, beginnen kann. Diese Stufen lassen sich als Workflow auffassen. Eine wichtige Eigenschaft des Sprachmodelltrainings ist, dass die Vorverarbeitungsschritte inkrementell ausgeführt werden können. Dies bedeutet, dass Textdaten, die bereits einmal verarbeitet wurden, nicht bei jedem Training neu berechnet werden müssen.

Die Einrichtung, Überwachung und Wartung der verschiedenen Sprachmodelltrainings ist zeitaufwändig und komplex. Für jedes Sprachmodell muss ein eigener Workflow mit speziellen Parametern definiert werden. Bislang werden diese Workflows durch eine Reihe von Skripten definiert. Fehler während des Trainings müssen durch den Entwickler entsprechend manuell identifiziert werden. Ein Speichern sowie Dokumentieren von Zwischenergebnissen ist derzeit manuell durch den Entwickler erforderlich. Ebenfalls ist bislang keine inkrementelle Verarbeitung von Daten implementiert. Durch die Definition der Workflows mithilfe von Skripten sowie dem manuellen Aufwand, der zur Ausführung der einzelnen Trainingsprozesse erforderlich ist, ist eine Skalierung der Anwendung nicht wart- sowie durchführbar.

Ein Problem stellt weiterhin der Umstand dar, dass die Programme in unterschiedlichen Programmiersprachen (Python / C++ / bash) entwickelt wurden. Aufgrund der Diversität der Software, die zur Ausführung benötigt wird, ist eine Einrichtung bzw. Installation auf mehreren Systemen aufwändig. Wie bereits in Abschnitt 2.4.1 erwähnt, kann die Reproduzierbarkeit von Workflows lediglich durch den Einsatz von Containern garantiert werden. Aus diesem Grund werden im Kontext dieser Arbeit ausschließlich Workflow-Engines betrachtet, die Unterstützung für die Ausführung von Docker basierten Containern mitbringen.

Tabelle 5 zeigt einen Auszug der im Rahmen dieser Arbeit durchgeführten Analyse verschiedener Workflow-Engines. Workflow-Engines lassen sich in datenflussorientierte (DF) und kontrollflussorientierte Verfahren (KF) unterscheiden. Eine Unterstützung beider Verfahren ist möglich. Es fällt auf, dass alle untersuchten Workflow-Engines die Ausführung von gerichteten Graphen unterstützen. Die Art der Darstellung unterscheidet sich je nach Workflow-Engine. Ebenfalls wird eine deklarative Beschreibung von Workflows unterstützt.

Eine Tabelle mit weiteren Kriterien ist auf der eingereichten CD hinterlegt. Es ist zu

beachten, dass die vollständige Tabelle in englischer Sprache verfasst wurde, um interessierten, internationalen Lesern die Möglichkeit zu bieten, diese als Entscheidungshilfe einzusetzen.

	Argo	Digdag	Pachyderm	Reana	Meson
Art	DF & KF	KF	KF	DF & KF	DF & KF
Plattform	Kubernetes	Docker (Swarm)	Kubernetes	Kubernetes	Apache Mesos
Webinterface	DAG und Gantt	Gantt	DAG	-	DAG und Gantt
Code / DSL	YAML	Python oder YAML	JSON	CWL	JSON
Dokumentation	nur Beispiele	✓	✓	✓	✓
Zeitbasierte Ausf.	-	✓	✓	✓	✓
Backfill	-	✓	✓	-	✓
Gerichteter Graph	✓	Partiell	✓	✓	✓
Clusterfähig	✓	✓	✓	✓	✓
Container (Docker)	✓	✓	✓	✓	✓
Speicher	S3-Speicher	-	S3-Speicher	-	-
Datenaustausch	✓	-	✓	✓	Metadaten
Datenherkunft	✓	-	✓	-	-
Zwischenergebnisse	✓	-	✓	✓	✓
Benachrichtigungen	Partiell	✓	Partiell	Partiell	✓

Tabelle 5: Vergleich von Workflow-Engines

4.3.3 Argo

Einleitung Argo[21] ist eine Open-Source Workflow-Engine für Kubernetes. Argo bietet zur Koordination von Aufgaben zwei verschiedene Darstellungsweisen. Zum einen können Aufgaben als Sequenz („steps“) definiert werden, die entweder sequentiell oder parallel ausgeführt werden. Zum anderen lässt sich ein gerichteter Graph („dag“) definieren, über den Abhängigkeiten zwischen Aufgaben beschrieben werden. Eine Mischform von steps sowie dags ist ebenfalls möglich. Die Analyse, welche Aufgaben (steps) parallel ausgeführt werden können, wird von Argo übernommen. Argo unterstützt komplexere Kontrollstrukturen wie Schleifen und Bedingungen sowie die Parametrisierung von Workflows. Über sogenannte „Exit Hooks“ können nach Abschluss des Trainings Benachrichtigungen, beispielsweise per E-Mail, versendet werden.

[21] `https://argoproj.github.io/`

Neben einer Benutzeroberfläche bietet Argo ein Kommandozeilenprogramm, welches das Starten sowie Stoppen von Workflows ermöglicht. Des Weiteren lässt sich der Status sowie die Ausführungshistorie von Workflows abfragen. Durch das Starten eines Workflows wird von Argo, je nach hinterlegter Definition, für jede Stufe ein oder mehrere Container parallel gestartet, die nach Abschluss der Berechnung wieder beendet werden. [Mac]

Der Einsatz von Argo eignet sich laut der Fachzeitschrift „Heise Developer“ immer dann, wenn eine einfache Möglichkeit gefragt ist, Workflows zu implementieren. Aufgrund der Architektur sowie des Funktionsumfanges eignet sich Argo für „sporadisch ausgeführte Workflows, da sich Worker Pods für den Workflow hoch- und wieder herunterfahren lassen“. [Mac]

Installation und Einrichtung Zur Installation von Argo wird ein existierendes Kubernetes Cluster vorausgesetzt. Bei der Verwendung von Rancher als Cluster-Manager sollte für Argo ein neues Projekt sowie ein neuer Namensraum (engl. „Namespace“) angelegt werden, um Kollisionen mit anderen Projekten im Cluster zu vermeiden. Anschließend lässt sich Argo in den Namensraum installieren. Um Rancher die entsprechenden Argo Dienste zuzuweisen, lassen sich Namensräume zu Projekten zuordnen. Sobald Argo gestartet ist, sind zwei Deployments in Rancher aktiv. Zum einen die „argo-ui“ und zum anderen der „workflow-controller“, der zur Koordination der Workflows verantwortlich ist.

Standardmäßig ist die Benutzeroberfläche von Argo (argo-ui) nicht außerhalb des Clusters erreichbar. Derzeit existieren drei Möglichkeiten, die Benutzeroberfläche über die IP-Adresse eines Knotens im Cluster zu erreichen. Zum einen bietet Kubernetes die Möglichkeit, eine Port-Weiterleitung einzurichten `kubectl port-forward`. Für diese ist es erforderlich, dass dauerhaft ein zusätzliches Programm im Hintergrund ausgeführt wird. Des Weiteren werden derzeit ausschließlich TCP Verbindungen[22] unterstützt. Während erster Tests mit Argo erwies sich diese Methode als nicht zuverlässig, da häufig Verbindungsprobleme auftraten. Ebenfalls stellte sich die Verwendung eines `kubectl proxy` als nicht hinreichend zuverlässig heraus. Eine dritte Möglichkeit stellt die Verwendung eines „LoadBalancer“ dar. Diese Option ist nur auf unterstützten Clustern, wie beispielsweise Google Cloud Platform, AWS sowie Azure, verfügbar und nicht für „On-Premises“ Kubernetes Cluster Installationen[23].

Alternativ lässt sich ein `nodePort` konfigurieren. Ein Nachteil der Verwendung eines `nodePort` ist, dass die zugewiesene Port-Nummer zufällig generiert wird und somit ein

[22] `https://github.com/kubernetes/kubernetes/issues/47862`
[23] `https://metallb.universe.tf/`

Nachschlagen der Adresse über die Kubernetes Oberfläche oder Kommandozeile notwendig ist. In Listing 10 ist eine vereinfachte Version des Installationsskriptes abgebildet. Um den Zugriff über einen fest definierten Port zu ermöglichen, wird im Rahmen dieser Arbeit ein sogenannter `hostPort` verwendet. Aufgrund von möglichen Adresskonflikten mit anderen Programmen auf dem auszuführenden System ist dieser Ansatz lediglich für Testzwecke geeignet.

```
# Create argo project
rancher projects create argo

# Create argo namespace
rancher namespaces create argo

# Install Argo Services
rancher kubectl apply -n argo -f \
  https://raw.githubusercontent.com/argoproj/argo/v2.2.1/manifests/
    install.yaml

# Associate argo namespace to project
ARGO_PROJ_ID=$(rancher projects | grep "argo" | awk '{print $1;}')
rancher namespace associate "argo" "${ARGO_PROJ_ID}"

# Expose hostPort
rancher kubectl patch deployment argo-ui -n argo --type='json' \
  -p='[{"op":"add","path":"/spec/template/spec/containers/0/ports",
     "value": [{"hostPort": 8001, "containerPort": 8001}]}]'
```

Listing 10: Argo Installation

Architektur Argo ist als sogenannte Kubernetes „Custom Resource Definition“ implementiert. Dies erlaubt die Definition von Workflows im bekannten Kubernetes YAML Format. Per Definition wird jede Stufe eines Workflows durch genau einen Container repräsentiert. Mehrstufige Workflows werden wahlweise durch eine Sequenz oder durch einen Abhängigkeitsgraphen (DAG) verschiedener Aufgaben repräsentiert.

Parallelität auf Aufgabenebene lässt sich durch die Verwendung von Schleifen realisieren. Argo führt die einzelnen Teilaufgaben, sofern ausreichend Rechenkapazitäten zur Verfügung stehen, parallel aus. Die Parallelität von Aufgaben lässt sich durch die Angabe verschiedener Ressourcen-Anforderungen an eine Aufgabe, wie beispielsweise dem benötigten Festplattenspeicher, die Anzahl der benötigten CPU-Kerne oder auch den benötigten Arbeitsspeicher, beeinflussen. Schleifen werden im Rahmen dieser Arbeit zur Ausführung der Stufe `MLP` verwendet.

Über „Exit-Hooks“ lassen sich bestimmte Aktionen nach der Ausführung eines Workflows definieren. Im Bezug auf die Audio-Mining Anwendung wurde der Exit-Hook dazu genutzt, eine E-Mail mit dem Ergebnis des Trainings zu versenden. Ebenfalls wäre es an dieser Stelle möglich, das neue Sprachmodell zu dem entsprechenden Kunden zu senden und nach einem erfolgreichen Trainingsdurchlauf die Zwischenergebnisse zu bereinigen, um Festplattenspeicher freizugeben.

Um Docker-Abbilder herunterzuladen, die in einer privaten Docker Registry gespeichert sind, kann mithilfe des Attributs `imagePullSecrets` ein Geheimnis (Secret) angegeben werden, in dem die entsprechenden Zugangsdaten gespeichert sind. Ein solches Secret beinhaltet das Token, welches für die Anmeldung an der Registry verwendet wird.[24]

Weiterhin lassen sich globale Attribute definieren, die von allen Stufen des Trainings verwendet werden können.

Lisiting 11 stellt einen Auszug der Deklaration des Workflows zum Sprachmodelltraining dar. Zu erwähnen ist, dass die vollständige Deklaration des Workflows rund 780 Zeilen umfasst.

[24] `https://kubernetes.io/docs/tasks/configure-pod-container/pull-image-private-registry/`

```
apiVersion: argoproj.io/v1alpha1
kind: Workflow
metadata:
  generateName: speech-training-
spec:
  entrypoint: pre-processing
  onExit: exit-handler
  imagePullSecrets:
    - name: fraunhofer-docker-registry
  arguments:
    parameters:
    - name: language-models
      value: '["wdr sportschau", "wdr br"]'
    - name: base-dir
      value: data/

  volumes:
  - name: ramfs
    emptyDir:
      medium: Memory
      sizeLimit: "70Gi"

  templates:
  - name: pre-processing
    dag:
      tasks:
      - name: frequent-case-training
        template: frequent-case-training

      - name: text-formatter
        dependencies: [frequent-case-training]
        template: text-formatter

  - name: frequent-case-training
    retryStrategy:
      limit: 2
    container:
      image: xxx.fraunhofer.de/frequent-case-training:0.2.0
      env:
      - name: BASE_DIR
        value: "{{workflow.parameters.base-dir}}"
      volumeMounts:
      - name: ramfs
        mountPath: /opt/speech_environment/data/
```

Listing 11: Auszug des Workflows zum Sprachmodelltraining in Argo

Notwendige Anpassung der Anwendung Zur Anbindung von Argo an einen S3 Speicher ist es notwendig, die Verbindungsinformationen der Konfigurationsdatei des „workflow-controller“ Container hinzuzufügen. Da Argo keine inkrementelle Datenverarbeitung unterstützt, ist es notwendig eine Hilfsanwendung zu entwickeln, die eine inkrementelle Verarbeitung ermöglicht. Weitere Informationen zu der entwickelten Hilfsanwendung sind Abschnitt 3.4.1 zu entnehmen.

Um Daten zwischen zwei Stufen zu transferieren, wird zunächst ein Eingabe- bzw. Ausgabe-Artefakt für jedes Template eines Containers definiert. In der Deklaration des Workflows wird für jede Stufe angegeben, welche vorhandenen Artefakte als Eingabe verwendet werden. Argo generiert aus der Deklaration von Ein- bzw. Ausgabeartefakten automatisch einen Ausführungsbaum.

Die Speicherung der Artefakte erfolgt als zip-Archiv. Die Komprimierung wird dabei durch `gzip` auf einem einzelnen Thread durchgeführt. Bei größeren Datenmengen führt dies zu erheblichen Performance Einbußen.[25]

Ausführung Die Ausführung eines Workflows ist mithilfe einer Kommandozeilenschnittstelle möglich. Hingegen wird die Ausführung eines Workflows anhand eines vordefinierten Zeitplans nicht unterstützt. Mit einem Hilfsprogramm, wie beispielsweise Cron, lässt sich dieses Problem vermeiden. Die Konfiguration eines solchen Cron-Jobs erfordert jedoch entsprechende Berechtigungen auf dem Cluster. Bei Managed Kubernetes Installationen kann dies ein Problem darstellen, da dort keine Cron-Jobs verfügbar sind.

Zu erwähnen ist, das Workflows nur vollständig ausgeführt werden können und somit immer der gesamte Workflow gestartet werden muss, um eine spezielle Stufe des Workflows testen zu können. Dies verlangsamt besonders bei lange laufenden Aufgaben die Fehlersuche bzw. den Testprozess von Workflows. Ein Workflow kann nach einem Absturz oder Fehler nicht an dieser Stelle fortgesetzt werden, sondern muss vollständig neu gestartet werden.

Oberfläche Über die grafische Benutzeroberfläche lässt sich der aktuelle Ausführungsstatus von Workflows überwachen. Workflows lassen sich in Form eines Gantt-Diagramms oder eines gerichteten Graphen darstellen. Für jede Stufe des Trainings können die entstandenen Artefakte heruntergeladen sowie die entsprechenden Docker Log-Dateien angesehen werden. Das Editieren sowie Ausführen von Workflows über die grafische Benutzeroberfläche ist nicht möglich. Der im Rahmen dieser Arbeit entwickelte

[25] `https://github.com/argoproj/argo/issues/1122`

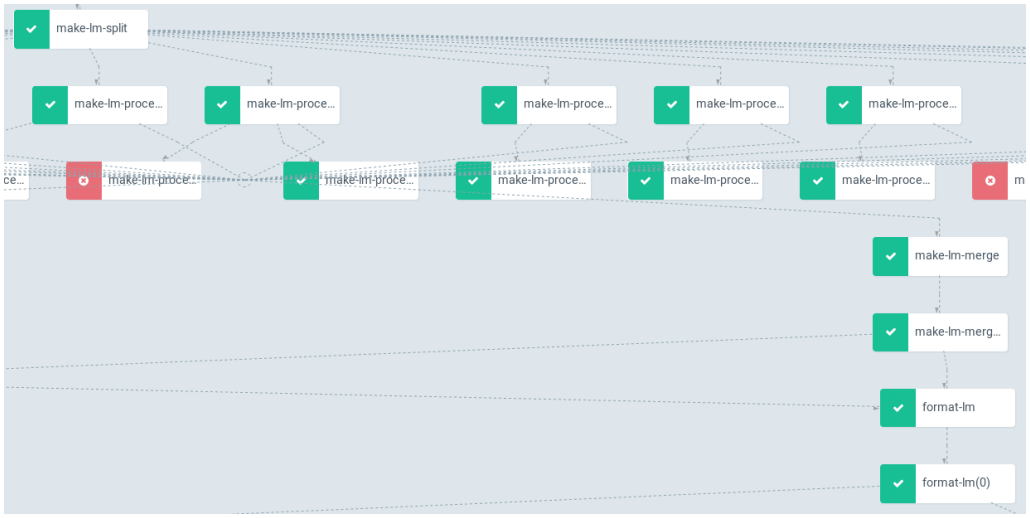


Abbildung 13: Argo Probleme bei Darstellung von Parallelität

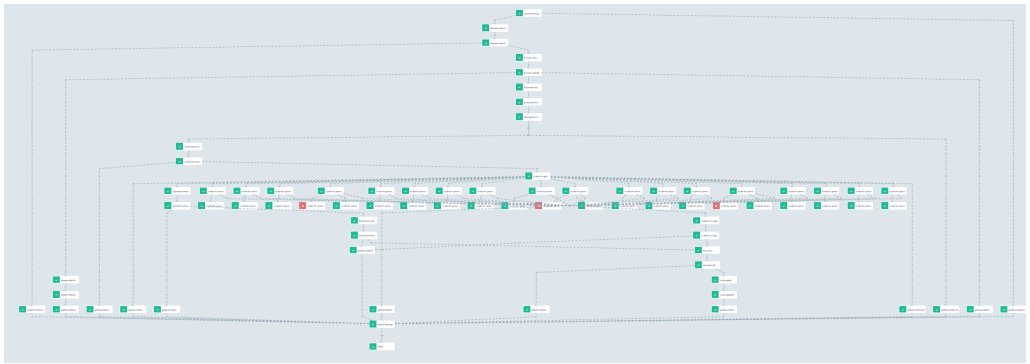


Abbildung 14: Argo Workflow zum Sprachmodelltraining

Workflow ist bereits zu umfangreich, um in akzeptabler Weise in der Graphansicht von Argo dargestellt zu werden. Ein Problem stellt hierbei die Darstellung von Parallelität dar. Kanten zwischen einzelnen Knoten sind nicht erkennbar und verlaufen, wie in Abbildung 13 zu sehen, übereinander. Eine „Übersicht“ des gesamten Workflows als Graph ist nur bedingt möglich (vgl. Abbildung 14), da ein Vergrößern des Graphen aufgrund des Aufbaus der zugrunde liegenden Webanwendung nicht möglich ist.

Fazit Zusammenfassend lässt sich sagen, dass die Konfigurationsdatei unübersichtlich sowie schwer wartbar ist. Eine Aufteilung der Konfiguration ist nicht möglich. Ein Editieren von Parametern des Workflows ist ausschließlich über den Einsatz der Kommandozeilenschnittstelle möglich. Neben der fehlenden Möglichkeit Workflows zeitgesteuert auszuführen, ist die Darstellung des Workflows zum Training eines Sprachmodells bedingt hilfreich.

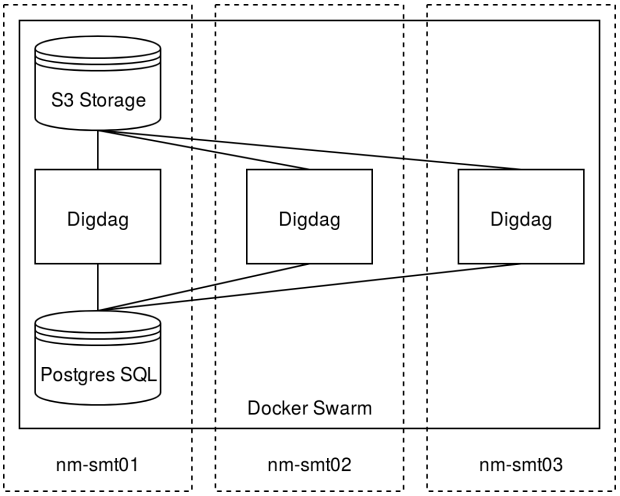


Abbildung 15: Digdag Orchestrierung mithilfe von Docker Swarm

4.3.4 Digdag

Einleitung Digdag[26] unterstützt die Definition von Workflows in einer speziell entwickelten Markup-Sprache oder in Python. Anders als bei Argo lassen sich hier Aufgaben nur durch die explizite Angabe von parallelen Stufen ausführen. Die Angabe von Abhängigkeiten zwischen Aufgaben ist nicht vorgesehen. Alle Aufgaben werden in einem gemeinsamen Arbeitsverzeichnis ausgeführt, über das Daten zwischen einzelnen Stufen des Trainings ausgetauscht werden können. Digdag ist eine ausführbare Java-Anwendung, die lediglich ein installiertes Java-Runtime-Environment voraussetzt. Eine Skalierung ist über den Einsatz von Docker Swarm möglich. Ebenfalls sollte eine Skalierung über Kubernetes möglich sein, diese wurde jedoch nicht im Rahmen dieser Arbeit untersucht.

Installation und Einrichtung Im Gegensatz zu Argo beschränkt sich die Installation der Digdag Workflow-Engine auf den Download einer ausführbaren Datei. Eine Skalierung der Workflow-Engine ist nicht offiziell möglich. Allerdings lässt sich durch die Orchestrierung einer einzigen Datenbank sowie mehreren Instanzen von Digdag Skalierbarkeit erreichen. Eine Orchestrierung ist durch die Verwendung eines Docker Swarm Clusters möglich.

Die im Rahmen dieser Arbeit verwendete Architektur ist in Abbildung 15 dargestellt. `nm-smt01`, `nm-smt02` sowie `nm-smt03` stellen die verwendeten Server des Clusters dar.

[26] `https://www.digdag.io/`

Architektur Während der Ausführung von Workflows wird das aktuelle Arbeitsverzeichnis durch Digdag verändert, um Kollisionen bei der parallelen Ausführung zu vermeiden. Dies spiegelt sich bei der Nutzung von Docker-Containern wieder.[27] Während der Ausführung wird das Arbeitsverzeichnis zu `/tmp/[random-identifier]/` geändert.

Definiert werden Workflows als Sequenz von Aufgaben. Aufgaben (Tasks) lassen sich zur einfacheren Darstellung gruppieren. So lässt sich beispielsweise eine Unterteilung in `pre-processing` sowie `language-model-training` durchführen. Parallelität lässt sich nur explizit durch die Angabe des `_parallel` Schlüsselwortes erreichen. Somit erfolgt die Angabe der parallel auszuführenden Aufgaben explizit.

Für die Definitionen von Workflows wird die Historie gespeichert. So lässt sich zurückverfolgen, wann welche Änderung an einer Workflowdefinition vollzogen wurde. Die Versionierung ist mit den entsprechenden Workflow-Durchläufen verknüpft. Digdag bietet keine Möglichkeit zur Datenspeicherung, so dass eine implizite Versionierung der Daten nicht möglich ist. Ebenfalls ist der Anwender für den gesamten Prozess der Datenverwaltung, Versionierung sowie Speicherung von Artefakten selbst verantwortlich.

In einem Fehlerfall lässt sich definieren, ob ein Workflow wiederholt wird oder eine entsprechende Benachrichtigung per E-Mail versendet wird. Neben der Angabe von zeitlichen Einschränkungen für Workflows lassen sich Zeitpläne, für die zeitgesteuerte Ausführung von Workflows, definieren. So ist es beispielsweise möglich, einen Workflow täglich zu einer bestimmten Uhrzeit zu starten.

Notwendige Anpassung der Anwendung Digdag unterstützt bislang keine Lade- sowie Speicheroperationen von S3 Speichern. Somit ist es erforderlich, dass die einzelnen Container des Sprachmodelltrainings die notwendigen Daten aus einem S3 Speicher laden, Berechnungen durchführen sowie die Ergebnisse abspeichern. Dabei ist zu beachten, dass keine existierenden Ergebnisse überschrieben werden dürfen. Digdag generiert bei jeder Ausführung eines Workflows eine eindeutige Zahl, über die sich ein Durchlauf identifizieren lässt. Diese Zahl lässt sich als Versionierungskriterium verwenden.

Wie bereits im vorherigen Abschnitt erwähnt, ändert Digdag das aktuelle Arbeitsverzeichnis innerhalb der Container. Die bislang verwendeten Anwendungen zum Sprachmodelltraining sind bislang nicht in der Lage, mit einer solchen Änderung umzugehen. Daher ist es erforderlich, die Docker-Container manuell mittels `docker run` Befehl zu starten. Mithilfe von Variablen ist es möglich, wiederkehrende Befehle einmalig zu definieren. Dennoch wird die Darstellung des Workflows zum Sprachmodelltraining unübersichtlich.

[27] `https://github.com/treasure-data/digdag/issues/854`

Ausführung Die Ausführung sowie Erstellung von Workflows ist sowohl über die Kommandozeilenschnittstelle sowie das webbasierte Interface möglich. Die wiederkehrende Ausführung von Workflows nach einem festen Zeitplan ist über die Angabe des `schedule` Attributs möglich. Ziel des Trainings eines neuen Sprachmodells ist es, jede 24 Stunden ein neues Modell veröffentlichen zu können. Je nach Auslastung des Clusters ist es denkbar, dass ein Training in Ausnahmefällen mehr als 24 Stunden benötigt. Um zu vermeiden, dass Digdag automatisch das nächste Training nach 24 Stunden startet, lässt sich mithilfe des Attributs `skip_on_overtime: true` die nächste geplante Ausführung überspringen, sofern die vorherige Ausführung nicht abgeschlossen ist.

Aufgrund der fehlenden Unterstützung von S3 Speichern ist es notwendig, Daten aus S3 Speichern in die Container zu kopieren oder ein entsprechendes Docker Plugin zur Einbindung von S3 Speichern zu installieren. Die Installation des Plugins ist auf dem Host-System durchzuführen und ist somit nur bedingt für den Einsatz in managed Docker Swarm Clustern geeignet. Beide Ansätze haben Vor- und Nachteile. Wie auch bei Argo benötigt ersterer je nach Datenmenge sowie Netzwerkgeschwindigkeit einige Zeit, bevor Berechnungen beginnen können. Während der Berechnungen wird das Netzwerk nicht zur Datenübertragung genutzt. Durch das Einbinden eines S3 Speichers in die Container können Berechnungen sofort gestartet werden. Ein Vorteil der Einbindung des S3 Speichers ist, dass auf den Berechnungsknoten kein zusätzlicher Speicherplatz benötigt wird. Jedoch wird die Lese- und Schreibgeschwindigkeit stark durch die Latenz sowie Bandbreite des Netzwerks beeinflusst.

Wie auch bei Argo ist zu erwähnen, dass Workflows nur vollständig ausgeführt werden können und im Fehlerfall der gesamte Workflow neu gestartet werden muss.

Oberfläche Digdag bietet im Gegensatz zu Argo keine graphbasierte Darstellung der Workflows. Der aktuelle Ausführungsstatus sowie die Workflow Definitionen lassen sich abrufen. Die Darstellung einzelner Durchläufe von Workflows ist in Form eines Gantt-Diagramms möglich. Workflows können sich aus mehreren Teil-Workflows zusammensetzen. In dem in Listing 12 dargestellten Workflow sind die Teilworkflows `preprocessing` sowie `lmtraining` in externen Dateien ausgelagert.

Über die Weboberfläche lassen sich Log-Daten der Container anzeigen. In Experimenten hat sich die Darstellung der Log-Dateien als nicht zuverlässig herausgestellt, da bei manchen Containern die Log-Daten, trotz erfolgreicher Ausführung, vollständig fehlten.

```
timezone: Europe/Berlin

schedule:
  daily>: 01:00:00

_export:
  crawler: [wdr spiegel welt-2018 br zeit]
  HOST: "127.0.0.1"

+preprocessing:
  _retry: 2
  call>: preprocessing.dig

+lmtraining:
  call>: lm_training.dig

_error:
  mail>:
    data: "Report␣at:␣http://${HOST}:65432/attempts/${session_id}/"
  subject: "Audio-Mining␣training␣report - Error␣occured"
  to: [ email@beispiel.de ]
```

Listing 12: Digdag Workflow Definition

Fazit Digdag ist standardmäßig nicht für die verteilte Ausführung ausgelegt. Jedoch können durch den Einsatz von Docker Swarm mehrere Instanzen der Digdag Workflow-Engine gestartet werden, um eine Lastverteilung zu erreichen.

Durch die fehlende Möglichkeit der Datenspeicherung sowie die fehlende Möglichkeit, Daten zwischen einzelnen Stufen des Trainings zu transferieren, eignet sich Digdag nur bedingt für den Anwendungsfall des Sprachmodelltrainings. Die Datenverwaltung sowie Versionierung ist Aufgabe des Entwicklers. Besonderer Fokus der Digdag Workflow-Engine liegt auf der zeitgesteuerten Ausführung von Workflows.

4.3.5 Common Workflow Language - Rabix Composer & Reana

Einleitung Wie bereits in Abschnitt 2.3.1 beschrieben, stellt Rabix eine Ansammlung von Open-Source Anwendungen zum Erstellen sowie Ausführen von wissenschaftlichen Workflows dar. Rabix Composer ist ein grafischer Editor zum Erstellen von Workflows mithilfe der Common Workflow Language. Das Ausführen von Workflows ist dabei lokal möglich. Zur Ausführung der einzelnen Stufen eines Workflows werden Docker-Container verwendet. In Abbildung 16 ist die grafische Oberfläche des Rabix Composers abgebildet.

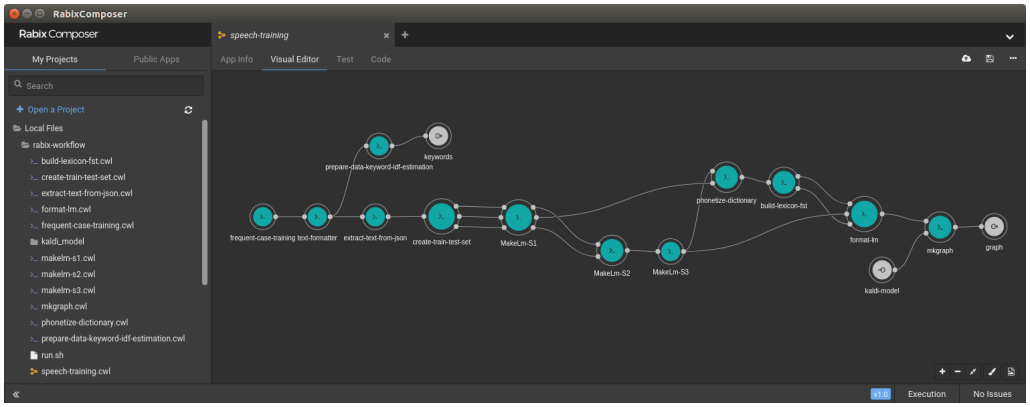


Abbildung 16: Rabix Composer

Reana ist eine wiederverwendbare Plattform zur Ausführung von reproduzierbaren wissenschaftlichen Workflows. Sie unterstützt Entwickler bzw. Forscher dabei, ihre Daten sowie Algorithmen so zu strukturieren, dass die Ausführung der Workflows in entfernten Rechenzentren durchgeführt werden kann. Dabei können die Anweisungen bei der lokalen Ausführung mithilfe von Docker sowie der Ausführung auf einem entfernten Cluster mithilfe von Kubernetes wiederverwendet werden. Die Unterstützung für Kubernetes Cluster wurde im April 2018 veröffentlicht. Das Projekt wird als Open-Source Projekt entwickelt und ist auf GitHub[28] zu finden. Entwickelt wird die Software primär durch ein Entwickler Team der „Europäischen Organisation für Kernforschung“ (CERN)[29].

Zur grafischen Erstellung der Workflows wurde die Anwendung Rabix Composer verwendet. Unterstützt wird die Ausführung von Docker-Containern auf dem lokalen Computer. Zur Ausführung in einem Cluster lässt sich die Workflow-Engine Reana verwenden, welche als Anwendung in einem Kubernetes Cluster installiert wird. Aufgrund von Problemen[30] während der Installation von Reana wurde diese Workflow-Engine nicht weiter untersucht. Ebenso existiert derzeit keine grafische Benutzeroberfläche zur Verwaltung sowie Überwachung von Workflows. Nach Rücksprache mit den Entwicklern der Workflow-Engine Reana ist die Entwicklung einer Benutzeroberfläche für Februar 2019 geplant.[31]

Während der lokalen Ausführung mittels Rabix Composer wird der Fortschritt des Workflows durch eine Hervorhebung der derzeit aktiven Stufen visualisiert. Die parallele Ausführung einer Stufe lässt sich nicht visualisieren. Die Benutzeroberfläche bietet außerdem die Möglichkeit, die Ausführungszeit der einzelnen Stufen zu analysieren. Diese Ergebnisse sind jedoch nur bedingt repräsentativ, da eine Ausführung lediglich auf

[28] `https://github.com/reanahub`
[29] `http://www.reana.io/`
[30] `https://github.com/reanahub/reana-cluster/issues/117`
[31] `https://github.com/reanahub/reana-ui/issues/21#issuecomment-435793587`

der lokalen Hardware ausgeführt werden kann und besonders im Kontext dieser Arbeit ein vollständiges Training aufgrund der Ressourcenanforderungen des Sprachmodelltrainings nur auf dem vorgesehenen Cluster möglich ist.

Fazit Mithilfe des Rabix Composers lassen sich Workflows lokal entwickeln sowie testen. Zur skalierten Ausführung lassen sich diese Workflow Definitionen auf andere CWL-fähige Workflow-Engines, wie beispielsweise Reana, übertragen. Durch Installationsprobleme wurden keine Laufzeitmessungen mit Reana durchgeführt. Aufgrund der Inkompatibilität der zur Verfügung stehenden Berechnungsinfrastruktur wurden keine weiteren Untersuchungen durchgeführt. CWL-fähige Workflow-Engines wie Arvados, Toil und cwl-tes unterstützen die Ausführung auf Plattformen wie AWS oder GCP. [cwl] Eine Ausführung des Workflows außerhalb des Campus wurde seitens des Fraunhofer Instituts ausgeschlossen.

4.3.6 Pachyderm

Einleitung Pachyderm ermöglicht eine vollständig reproduzierbare Ausführung von Workflows durch die Verwaltung von Eingabedaten sowie den zur Berechnung eingesetzten Docker-Abbildern. Workflows werden in Form von Pipelines definiert, die genau einer Stufe eines Workflows entsprechen. Ein Workflow besteht aus mehreren Pipelines, deren Reihenfolge sich implizit durch die Angabe von Ein- bzw. Ausgaben ergibt. Um Änderungen an den Daten festhalten zu können, verwendet Pachyderm ein versioniertes Dateisystem, mit dem Git ähnliche Commits beim Schreiben von Daten möglich sind. Dies ermöglicht, sofern die Daten und Algorithmen dies unterstützen, ein iteratives sowie inkrementelles Bearbeiten von Daten. Sobald eine Änderung in einem Datenspeicher (Repository) erkannt wird, wird eine Pipeline mit den neuen bzw. geänderten Daten gestartet. Über reguläre Ausdrücke lässt sich die Granularität der Datenverteilung konfigurieren. Im Gegensatz zu Argo nutzt Pachyderm langlebige Worker Pods, die während der gesamten Ausführung einer Pipeline aktiv sind. Somit existiert an dieser Stelle ein Vorteil gegenüber Argo, da Pachyderm Container wiederverwendet und somit das Erzeugen bzw. Zerstören von Worker Pods entfällt. Pachyderm eignet sich daher besonders für Workflows, die sehr häufig bzw. kontinuierlich ausgeführt werden. Workflows lassen sich durch die Angabe von Ein- bzw. Ausgaben implizit erstellen. Durch die Versionierungsmöglichkeiten lässt sich exakt nachvollziehen, welche Daten durch welchen Container bzw. mit welchen Eingabedaten entstanden sind. [Mac]

Im Vergleich zu den anderen im Rahmen dieser Arbeit untersuchten Workflow-Engines, ist Pachyderm ein teilweise kommerzielles Produkt. Die eigentliche Workflow-Engine sowie das Kommandozeilen-Programm sind als Open-Source Anwendung verfügbar.

Funktionen wie die grafische Weboberfläche, Zugriffsrechtemanagement, Statistiken und Dateibrowser sind Enterprise Kunden vorbehalten. [32] Die Funktionen lassen sich jedoch kostenlos innerhalb eines zweiwöchigen Testzeitraums evaluieren.

Installation und Einrichtung Pachyderm ist eine Workflow-Engine, die auf einem Kubernetes Cluster ausgeführt wird. Zur Trennung von Workloads ist es sinnvoll, für Pachyderm ein eigenes Projekt sowie einen eigenen Namensraum anzulegen. Pachyderm bietet über das mitgelieferte Kommandozeilen-Programm die Möglichkeit, eine Konfigurationsdatei für Kubernetes generieren zu lassen. Nach dem Starten von Pachyderm sind drei Deployments aktiv, „dash“ zur Bereitstellung der Weboberfläche, „etcd“ sowie „patchd“ zur Orchestrierung der Workflow-Ausführung.

Architektur Pachyderm verwendet drei zentrale Komponenten: Repositories, Pipelines sowie Jobs. Repositories werden zur versionierten Speicherung von Daten genutzt. Als Pipelines werden einzelne Verarbeitungsschritte bezeichnet. Workflows lassen sich durch die Komposition verschiedener Pipelines implizit abbilden. Eine Pipeline nimmt Eingaben beispielsweise in Form einer Datei entgegen, führt eine Berechnung auf diesen aus und speichert das Ergebnis in Form einer Datei. Mithilfe regulärer Ausdrücke wird die Aufteilung der Daten zur parallelen sowie inkrementellen Verarbeitung definiert. So ist es möglich, eine inkrementelle Verarbeitung auf Verzeichnis- oder Dateiebene zu ermöglichen. Gleichzeitig führt eine Aufteilung auf Dateiebene zu einer höheren Parallelität, da Pachyderm Taskparallelität für die angegebene Aufteilung voraussetzt. Zur Ausführung werden sequentiell, entsprechend der Aufteilung nach, die einzelnen Teile (Chunks) verarbeitet. Über den Parameter `parallelism_spec` lässt sich die Parallelität als konstanter Wert oder prozentual zu der Cluster Größe definieren. Die zu bearbeitenden Eingabedaten sind für den jeweiligen Container über das Verzeichnis `/pfs/{input-repository-name}/` erreichbar. Ergebnisse werden in dem Verzeichnis `/pfs/out/` abgelegt. Schreibkonflikte werden durch das „Anhängen“ der Dateien gelöst. Wenn beispielsweise zwei parallel ausgeführte Aufgaben in eine Datei `ausgabe.txt` schreiben, enthält diese Datei nach der Ausführung der Pipeline beide Ausgaben untereinander. Diese Art der Schreibkonfliktlösung ist bei der Entwicklung von Anwendungen zu berücksichtigen. Mithilfe dieser Art der Schreibkonfliktlösung ist es im Rahmen des Sprachmodelltrainings möglich, eine Versionierung sowie inkrementelle Verarbeitung auf Dateiebene zu ermöglichen, ohne dass diese durch den Entwickler explizit implementiert oder angegeben werden muss.

[32] `https://www.pachyderm.io/enterprise.html`

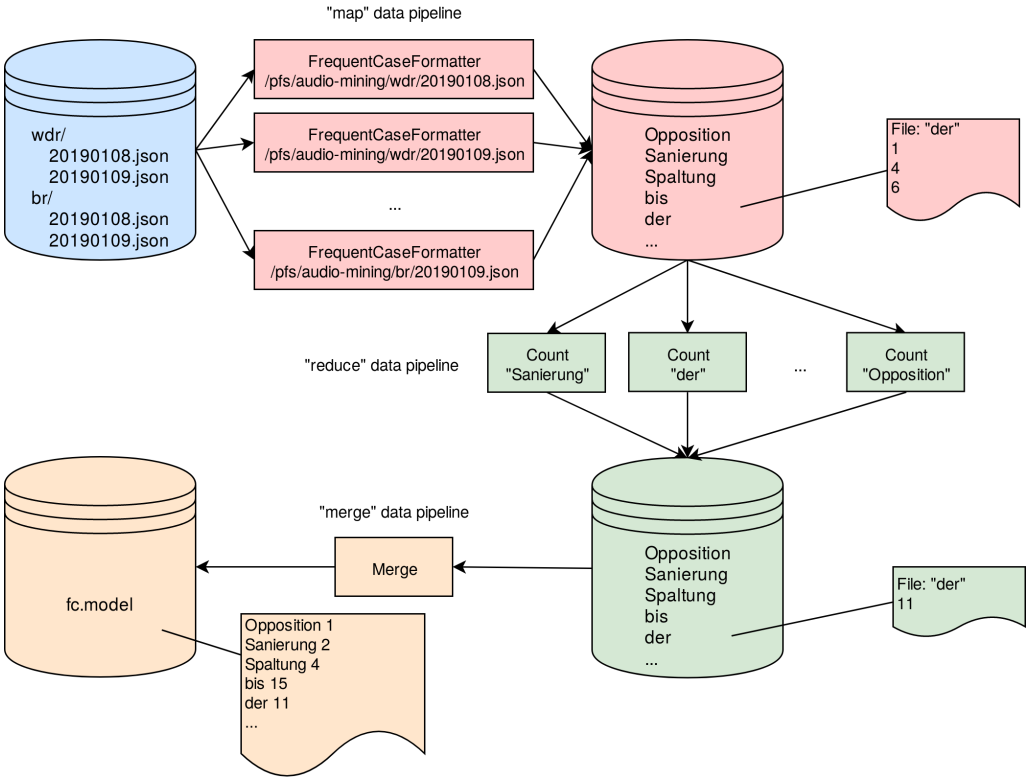


Abbildung 17: Pachyderm Frequent-Case-Formatter (nicht praktikabel)

Notwendige Anpassung der Anwendung Um die Vorteile dieser Art der Datenspeicherung zu nutzen, wurde exemplarisch die erste Stufe des Trainings entsprechend angepasst. Jedes Dokument (JSON-Datei) wird somit genau von einem Container bearbeitet. Für jedes Wort in einem Dokument wird eine eigene Datei angelegt, deren Dateiname dem Wort entspricht. Inhalt dieser Datei ist die Anzahl der Vorkommen des jeweiligen Wortes in dem zu verarbeitenden Dokument. Nach erfolgreicher Ausführung der Pipeline enthält jedes Dokument mehrere Zeilen mit jeweils der Anzahl der Vorkommen eines Wortes (untereinander). Diese Summe lässt sich mithilfe einer Reduktions-Pipeline zusammenfassen, so dass jedes Dokument die Summe aller Vorkommen enthält. Diese Dateiliste lässt sich anschließend mithilfe einer weiteren Pipeline als Dokument konvertieren, in dem in jeder Zeile ein Wort sowie mit Leerzeichen getrennt die Anzahl der Vorkommen festgehalten ist. Diese modifizierte Art des Frequent-Case-Formatter ist in Abbildung 17 dargestellt. Experimente zeigen, dass dieses Verfahren selbst bei kleinsten Datenmengen (<100MB) bereits zu einer mehrtägigen Berechnungsdauer führt. Ursache dafür ist die Anzahl der Dateien, die während des Verfahrens generiert werden. Eine weitere Möglichkeit ist die Speicherung aller Wörter in einer einzelnen Datei (vgl. Listing 2). Durch die Art der Datenspeicherung werden alle Dokumente konkateniert. Da in dieser konkatenierten Liste Wörter mehrfach vorkommen, ist es notwendig, diese zu normalisieren. Abbildung 18 stellt eine praktikable Implementierung des Schrittes

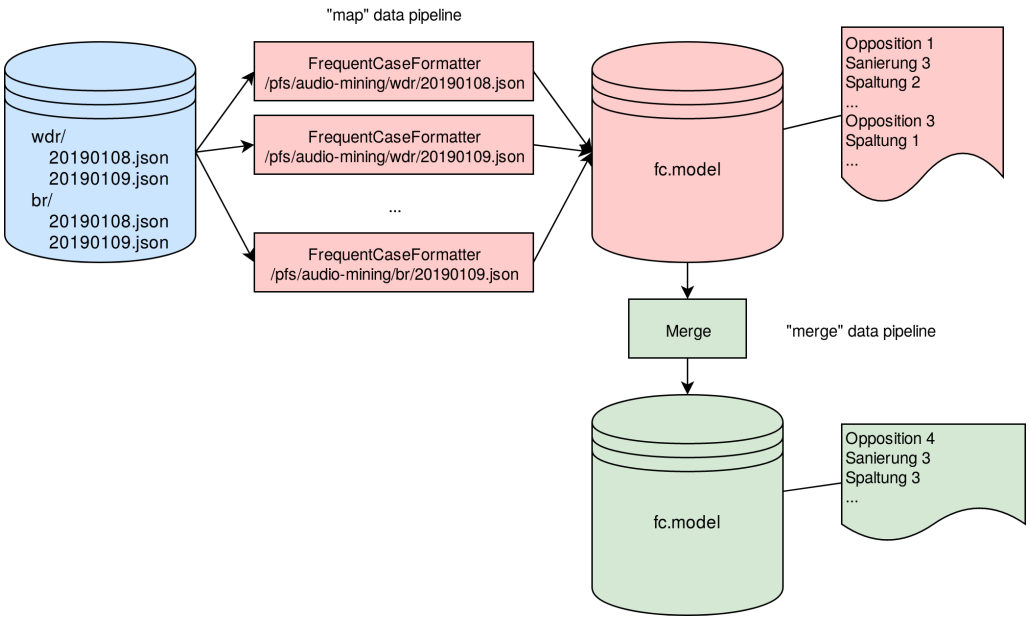


Abbildung 18: Pachyderm Frequent-Case-Formatter (praktikabel)

zur inkrementellen Berechnung dar. Durch die dateiorientierte Aufteilung ist lediglich eine Implementierung des Schrittes „reduce“ notwendig. Die Anwendung des Frequent-Case-Formatter bedarf keinerlei Änderungen.

Ausführung Listing 13 zeigt die Pipeline Definition zur Berechnung des Frequent-Case-Modells. Mithilfe des `image` Attributs lässt sich das zu verwendende Docker-Abbild angeben. Das `cmd`-Attribut enthält den in dem Container auszuführenden Shell-Befehl. Über das `input` Attribut lässt sich die entsprechende Granularität der Datenaufteilung festlegen. Diese hat Einfluss auf die Möglichkeit der inkrementellen sowie parallelen Verarbeitung. Die Aufteilung lässt sich mithilfe eines regulären Ausdrucks angeben. So bedeutet ein `/`, dass alle Dateien in dem Repository als Eingabe zu verwenden sind. In diesem Fall existiert keine Möglichkeit zur inkrementellen oder parallelen Berechnung. Durch die Verwendung eines regulären Ausdrucks wie `/*`, wird für jede Datei oder jeden Ordner in dem Repository ein Container gestartet. Dies ermöglicht eine inkrementelle sowie parallele Verarbeitung. Ebenfalls ist es möglich, durch die Angabe des Ausdrucks `/*/*` eine inkrementelle und parallele Verarbeitung für Dateien in Unterordnern zu erreichen. Dies ist besonders im Kontext des Sprachmodelltrainings hilfreich, da in dem Hauptverzeichnis lediglich Ordner der jeweiligen Crawler enthalten sind.

Mithilfe des Attributs `egress` lassen sich Artefakte aus einem Workflow exportieren und beispielsweise in einem S3 Speicher ablegen.

Sofern ein Container aufgrund eines Fehlers für ein einzelnes Datum fehlschlägt, wird

nach der Behebung des Fehlers, beispielsweise durch Ändern des Docker-Abbilds, lediglich dieses Datum neu berechnet. Auf Wunsch, unter anderem nach einer Änderung des Algorithmus, lässt sich eine Pipeline vollständig neu berechnen. Dies geschieht über die Angabe des `--reprocess` Arguments während eines Update-Prozesses einer Pipeline.

Durch die Information über die bereits prozessierten Daten werden automatisch ausschließlich neu hinzugekommene Daten verarbeitet. Im Bezug auf das Sprachmodelltraining ermöglicht diese Eigenschaft ein implizites inkrementelles Training.

Pipelines starten automatisch, sobald ein Commit in einem Eingabe Repository ausgeführt wird. Somit ist ein manuelles Starten von Workflows bzw. Pipelines nicht notwendig. Durch die Verwendung eines speziellen Datenspeicher Repository ist eine zeitgesteuerte Ausführung von Pipelines möglich. Dazu wird mithilfe eines Cron-Jobs eine Datei mit dem jeweiligen Datum der Ausführung in ein automatisch generiertes Repository eingefügt.

Ein Vorteil der Darstellung von Workflows als einzelne Pipelines ist die Möglichkeit, Workflows schrittweise zu entwickeln. So lässt sich im Fehlerfall einer Stufe auf den bereits vorhandenen Ergebnissen der vorherigen Stufen aufbauen. Dies ermöglicht im Vergleich zu Digdag und Argo eine schnellere sowie inkrementelle Entwicklung von Workflows.

```
{
    "pipeline": {
        "name": "frequent-case-training"
    },
    "transform": {
        "cmd": ["tfc.sh","/pfs/audio-mining","wdr,br","/pfs/out"],
        "image": "registry.fraunhofer.de/training/pachyderm:0.2.2",
        "image_pull_secrets": [ "fraunhofer-docker-registry" ],
        "working_dir": "/opt/speech_environment/lm_scripts/"
    },
    "input": {
        "atom": {
            "repo": "audio-mining",
            "glob": "/*/*"
        }
    },
    "parallelism_spec": {
        "constant": 4
    }
}
```

Listing 13: Pachyderm Workflow Definition

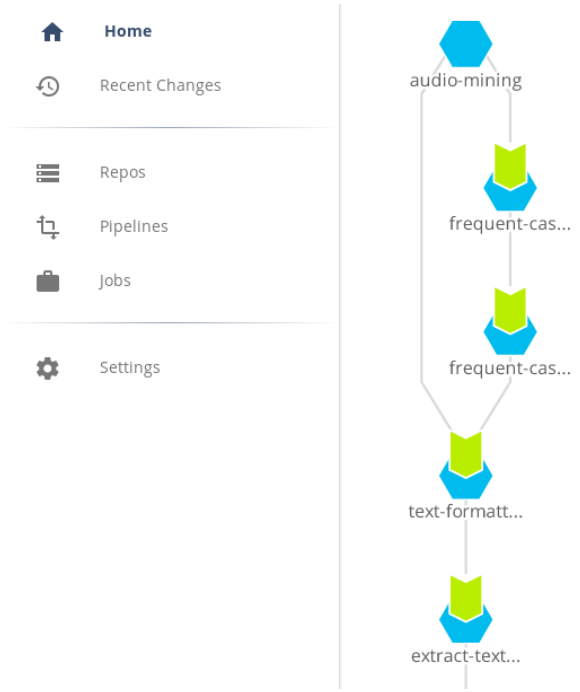


Abbildung 19: Pachyderm (Pachdash)

Oberfläche Die Ausführung sowie Erstellung von Workflows ist sowohl über die Kommandozeilenschnittstelle als auch über die Weboberfläche möglich. Wie bereits erwähnt, handelt es sich bei der Weboberfläche um ein kostenpflichtiges Zusatzprodukt. Die Weboberfläche bietet die Möglichkeit, mithilfe diverser Formulare Workflows bzw. Pipelines zu erstellen. Über die Kommandozeilenschnittstelle werden Workflows im JSON-Format definiert.

Die Benutzeroberfläche bietet eine Graphansicht aller vorhandenen Pipelines sowie deren Beziehungen zueinander. Außerdem können alle aktiven Jobs, deren Status und Eingaben nachvollzogen werden. Im Rahmen dieser Arbeit konnte festgestellt werden, dass bei größeren Datenmengen (> 1 Mio. Dateien) die Darstellung der Benutzeroberfläche erheblich langsamer wird. Eine genaue Analyse der Ursache war aufgrund von zeitlichen Einschränkungen im Rahmen dieser Arbeit nicht möglich. Eine erste Vermutung legt nahe, dass die Geschwindigkeitseinbußen auf die eingesetzten HDDs zurückzuführen sind. Ebenfalls lässt sich über die Weboberfläche die Historie von Daten anzeigen und abrufen. Somit ist ein Nachvollziehen der Datenherkunft, neben den verwendeten Algorithmen, für jedes Datum möglich. Abbildung 19 zeigt die Graphrepräsentation eines Teils des Sprachmodelltrainings.

Fazit Ein Vorteil von Pachyderm gegenüber anderen Workflow-Engines wie Argo und Digdag ist, dass eine Datenspeicherung, Parallelisierung und inkrementelle Verarbeitung

nicht manuell durch den Entwickler implementiert werden muss, sondern dies implizit, durch die Angabe des Datenverteilungsschemas, geschieht. Dies ermöglicht den nahezu unveränderten Einsatz der bisherigen Anwendung zum Sprachmodelltraining.

Während der Implementierung des Workflows mithilfe von Pachyderm stellt sich die Fehlersuche häufig als schwierig heraus, da Fehlermeldungen teilweise nicht in der grafischen Weboberfläche angezeigt werden und nur über die Verwendung einer Vielzahl von Kommandozeilenbefehlen identifiziert werden können.

4.3.7 Meson

Meson wurde von Netflix zur Ausführung sowie ressourceneffizienten Ausführung von Machine-Learning Workflows entwickelt. Im Gegensatz zu den anderen im Rahmen dieser Arbeit vorgestellten Workflow-Engines wird Meson auf dem Open-Source Cluster Manager Mesos[33] ausgeführt. Netflix setzt Meson zur Berechnung diverser Pipelines, wie beispielsweise zur Berechnung von Modellen für Film-Vorschläge, ein.

Ein Ziel Mesons ist es, die Geschwindigkeit, Zuverlässigkeit und Reproduzierbarkeit von Workflows garantieren zu können, ohne Vorschriften der zu verwendenden Programmiersprache zu definieren. Meson bietet keine Möglichkeit zur Datenspeicherung. Daten und Ergebnisse sind daher durch die Entwickler manuell in die bzw. aus den Workflows zu transferieren.

Derzeit existieren nur wenige Workflow-Engines, die Mesos als Cluster-Manager einsetzen. Einige der bekanntesten Workflow-Engines wie Chronos[34] wird bereits seit rund einem Jahr nicht mehr weiterentwickelt. Somit stellt Meson eine neue Alternative dar. Bislang befindet sich Meson noch in der Entwicklungsphase. Eine Veröffentlichung ist laut Davis Shepherd, einer der Entwickler von Meson, geplant. Einen Zeitplan, wann mit der Veröffentlichung zu rechnen ist, gibt es derzeit nicht. Um die Möglichkeiten sowie Funktionen von Meson mit anderen Workflow-Engines vergleichen zu können, führte Shepherd eine entsprechende Bewertung anhand der Kriterien in Tabelle 5 durch.

4.4 Skalierungsmethoden von Workflows

Zur Skalierung von Workflows müssen zunächst die problematischen Stellen eines Workflows hinsichtlich der Ressourcenauslastung identifiziert werden. Dazu ist eine Überwachung der Workflows während der Ausführung notwendig. In diesem Kapitel wird zunächst eine Möglichkeit beschrieben, verteilte Anwendungen zu überwachen, um Bott-

[33] `http://mesos.apache.org/`
[34] `https://mesos.github.io/chronos/`

lenecks zu identifizieren. Anschließend werden die vertikale und horizontale Skalierbarkeit sowie die Lastskalierbarkeit untersucht.

4.4.1 Profiling von verteilten Anwendungen

Zur Beschleunigung von existierenden Workflows ist es zunächst erforderlich, die Teile eines Workflows zu identifizieren, die die längste Ausführungsdauer haben. Im Kontext des Sprachmodelltrainings variiert die sequentielle Laufzeit einzelner Stufen stark. Während einige Stufen des Trainings mit einem Datensatz von rund 24GB 60 Stunden (Frequent-Case-Formatter) benötigen, sind andere Stufen des Trainings innerhalb von 5 Minuten (Format-LM) beendet. Weitere Faktoren stellen die Häufigkeit der Ausführung einer Stufe dar sowie die Veränderung der Ausführungszeit bei einer inkrementellen Verarbeitung. Wie bereits in Abschnitt 3.4.1 erwähnt, wurde durch die inkrementelle Verarbeitung des Frequent-Case-Formatter erreicht, dass lediglich neue Daten verarbeitet werden müssen und somit die Ausführungszeit bei inkrementeller Verarbeitung, je nach Datenmenge, auf wenige Minuten reduziert werden kann.

Ein häufiges Problem bei Zeitmessungen von verteilten Anwendungen liegt darin, dass viele verschiedene Anwendungen Ressourcen eines Clusters teilen und somit die Geschwindigkeit der einzelnen Berechnungen beeinflussen können.

Ein weiteres Problem stellt die Analyse der Laufzeit von containerisierten Anwendungen dar. Bei einer lokalen Docker sowie Docker Swarm Installation lässt sich der aktuelle Ressourcenverbrauch eines Containers mithilfe des Befehls `docker stats` ermitteln.

Ein Nachteil dieser Systeme ist, dass eine Analyse lediglich auf Container-Ebene möglich ist. Mithilfe der Anwendung Grafana[35], InfluxDB[36] sowie telegraf[37] lässt sich eine Monitoring Anwendung erstellen, die eine Überwachung mehrerer Container ermöglicht. Dies ist besonders dann wichtig, wenn durch die parallele Ausführung einer Stufe mehrere Container zeitgleich ausgeführt werden. Mithilfe der Möglichkeit der Gesamtübersicht über alle Systemressourcen wie CPU-, Arbeitsspeicher-, Festplattenspeicher- und Netzwerkauslastung lassen sich potentielle Bottlenecks einer Anwendung identifizieren. Probleme können beispielsweise redundante Datenübertragungen, unzureichende Prozessorauslastung und der Überlauf von Arbeits- sowie Festplattenspeicher darstellen. Abbildung 20 zeigt das entwickelte Dashboard für einen Trainingsdurchlauf des Sprachmodelltrainings in Argo. Mithilfe dieser Ansicht können Stufen des Workflows identifiziert werden, die von einer parallelen Ausführung profitieren können. Der Abbildung ist zu entnehmen, dass die zur Verfügung stehenden Ressourcen nicht vollständig

[35] `https://grafana.com/`
[36] `https://github.com/influxdata/influxdb`
[37] `https://github.com/influxdata/telegraf`

Abbildung 20: Grafana Dashboard (Sprachmodelltraining mittels Argo)

ausgenutzt werden. Lediglich die erste sowie zweite Stufe des Workflows (Frequent-Case-Formatter und Text-Formatter) nutzen die zur Verfügung stehende Rechenleistung eines Knotens nahezu vollständig aus. Die roten Markierungen stellen den Wechsel zwischen den jeweiligen Stufen des Workflows dar. Ein Nachteil der Monitoring-Lösung ist, dass die Systemressourcen analysiert werden. Durch die parallele Ausführung von anderen Anwendungen auf dem Cluster können somit die Ergebnisse für eine konkrete Anwendung verfälscht werden. Eine Lösung auf Projektebene für Kubernetes Cluster mithilfe der Cluster-Orchestrierungslösung Rancher wird derzeit entwickelt (vgl. Abschnitt 4.2).

4.4.2 Vertikale Skalierung vs. horizontale Skalierung

Zur Skalierung der Leistung eines Clusters existieren zwei Ansätze, die vertikale und die horizontale Skalierung. Während die horizontale Skalierung mehr Leistung durch Hinzunahme weiterer Berechnungsknoten erreicht, wird eine vertikale Skalierung durch die Ausstattung besserer Hardware, wie beispielsweise eine leistungsstärkere CPU oder mehr Arbeitsspeicher, erreicht. Die vertikale Skalierung ist die einfachere Form der Skalierung, da durch sie i.d.R. keine Anpassungen an der auszuführenden Anwendung vorzunehmen sind. Allerdings ist die vertikale Skalierung durch die Kosteneffektivität, physikalische Einschränkungen und die Verfügbarkeit spezieller Hardware eingeschränkt. Um die Vorteile einer horizontalen Skalierung nutzen zu können, müssen die auszuführenden Anwendungen eine verteilte Ausführung unterstützen. Die horizontale Skalierung ist durch den Einsatz von Standardhardware oft wirtschaftlicher, als hohe Kosten für spezialisierte Hardware aufzuwenden. [LSL+14, Erb12]

Im Rahmen dieser Arbeit hat sich gezeigt, dass eine vertikale Skalierung für viele Stufen des Trainings effektiver ist als eine horizontale Skalierung. Dies liegt vor allem an den verwendeten Anwendungen, die teils nur schwer oder nicht parallelisierbar sind. Somit profitieren diese Anwendungen von wenigen leistungsstarken CPU-Kernen. Das im Rahmen dieser Arbeit verwendete Cluster enthält viele CPU-Kerne, die jedoch im einzelnen wenig Leistung bieten. So ist das ursprüngliche Training auf einem modernen Intel Core i7 Prozessor (Intel Core i7-6700) rund 40% schneller als auf dem im Cluster verbauten AMD Prozessor (AMD Opteron Prozessor 6234). Um eine horizontale Skalierung zu evaluieren, wurden die Laufzeitmessungen dennoch auf dem in Abschnitt 4.2 beschriebenen Cluster durchgeführt.

Allgemein lässt sich sagen, dass für Workflows, deren Stufen Parallelität nutzen können, insbesondere bei der parallelen Verarbeitung von Dateien, sich eine horizontale Skalierung empfiehlt. Falls viele Stufen des Workflows nicht parallelisierbar sind, eignet sich eine vertikale Skalierung. Hierzu ist eine Analyse der Anwendung mit den in Abschnitt 4.4.1 beschriebenen Methoden zur Identifizierung von Performance Problemen durchzuführen. Workflow-Engines, wie beispielsweise Pachyderm, unterstützen durch die Angabe eines „NodeSelector“ die Möglichkeit, eine bestimmte Aufgabe eines Workflows auf einer speziellen Hardware auszuführen. Dies kann beispielsweise hilfreich sein, wenn zur Berechnung einer Aufgabe eine Grafikkarte benötigt wird, diese jedoch nicht auf allen Knoten des Clusters bereitsteht.

4.4.3 Lastskalierbarkeit

Einige Anbieter von managed Kubernetes Lösungen wie „Google Kubernetes Engine“[38] bieten die Möglichkeit, die Clustergröße automatisch entsprechend der Arbeitslast anzupassen. Dies ermöglicht eine horizontale Skalierung der Anwendung, die bei einer Nutzung eines On-Premises Clusters nicht möglich ist. Somit lassen sich im Fall von Ressourcenknappheit neue Ressourcen allokieren. Eine Abrechnung erfolgt in der Regel auf Basis der Nutzungsdauer der Knoten. Es ist zu evaluieren, ob der Einsatz eines managed Kubernetes Clusters für den Anwendungsfall des Sprachmodelltrainings, besonders unter Betrachtung der finanziellen Aspekte, Vorteile gegenüber einer On-Premises Lösung bietet.

Je nach Anwendungsfall ist zu untersuchen, ob diese Art der Skalierung hilfreich ist, da, wie bereits erwähnt, nicht jede Anwendung von einer horizontalen Parallelisierung profitiert. Besonders im Kontext des Sprachmodelltrainings wäre es möglich, entsprechend viele Rechenkapazitäten für parallele Stufen, wie den Frequent-Case-Formatter und den Text-Formatter, zu allokieren und diese nach Beenden dieser Stufen freizugeben.

Pachyderm bietet weitere Funktionen zur Optimierung einer automatischen Lastskalierung[39], die sich besonders für die Ausführung von rechenintensiven und langanhaltenden Aufgaben eignen.

Kubernetes als Ausführungsplattform Existierende Workflow-Engines lassen sich auf einer Vielzahl von Plattformen ausführen (vgl. [cwl]). Durch die Einschränkung, dass lediglich Workflow-Engines mit Unterstützung für Container untersucht werden, wurden im Rahmen dieser Arbeit primär Docker Swarm sowie Kubernetes basierte Systeme evaluiert. Ein Vorteil, Kubernetes als Ausführungsplattform zu verwenden, liegt in der Möglichkeit, dynamisch Rechenleistung, unter Berücksichtigung der Auslastung des Clusters, anzufordern (vgl. Abschnitt 4.4.3).

[38] `https://cloud.google.com/kubernetes-engine/docs/concepts/cluster-autoscaler`
[39] `http://docs.pachyderm.io/en/latest/managing_pachyderm/autoscaling.html`

5 Evaluation

Zunächst wird eine Evaluation hinsichtlich der Laufzeit vorgenommen, bevor die Ergebnisse dieser Arbeit abstrahiert und Anwendungsfelder der untersuchten Workflow-Engines beschrieben werden. Abschließend wird ein Vorgehensmodell beschrieben, das bei der Umsetzung eines Scientific Workflows mithilfe der im Rahmen dieser Arbeit vorgestellten Anwendungen unterstützt.

5.1 Performance Vergleich

Im folgenden Abschnitt wird zunächst der theoretisch mögliche Speedup berechnet, bevor die Ergebnisse der durchgeführten Laufzeittests verglichen werden.

Um die Reproduzierbarkeit sowie Vergleichbarkeit der ausgeführten Tests zu gewährleisten, wird ein festes, vorgegebenes Datenset verwendet. Dieses beinhaltet 3378 Dateien mit einer Gesamtgröße von 23,9GB. Alle Laufzeitmessungen wurden auf dem vorgesehenen Cluster durchgeführt. Alle Messungen wurden drei mal ausgeführt. Die im Rahmen dieser Arbeit angegebenen Laufzeiten spiegeln den Mittelwert der gemessenen Laufzeiten wider. Eine häufigere Durchführung der Experimente ist aufgrund der benötigten Trainingszeit im Kontext dieser Arbeit nicht möglich.

Das Cluster wurde exklusiv für die im Rahmen dieser Arbeit durchgeführten Tests reserviert, um Abweichungen der Laufzeit sowie Einflüsse durch andere Anwendungen ausschließen zu können. Während der Ausführung der Tests wurde festgestellt, dass das auf dem System installierte Antivirenprogramm eine nicht unerhebliche zusätzliche CPU-Auslastung verursacht. Je nach Stufe des Sprachmodelltrainings erreicht die zusätzliche durch das Antivirenprogramm generierte Auslastung rund 50% der zur Verfügung stehenden Rechenleistung. Diese Beeinflussung der zur Verfügung stehenden Rechenleistung spiegelt sich in den gemessenen Laufzeiten der Anwendungen wider, die teils starke Schwankungen aufweisen.

5.1.1 Theoretischer Speedup

Der Speedup wird verwendet, um den Einfluss auf die Performance nach einer Ressourcenverbesserung, wie beispielsweise dem Einsatz eines leistungsstärkeren Prozessors oder der Parallelisierung einer Anwendung, zu messen. Der Speedup S wird als Faktor definiert, der den relativen Leistungsunterschied zwischen zwei Systemen misst, die das gleiche Problem lösen. [Kra, RW17]

Im Rahmen dieser Arbeit wird die Kenngröße des Speedups verwendet, um die Laufzeit des ursprünglichen Programms mit den Laufzeiten der untersuchten Workflow-Engines zu vergleichen. Aus dieser lässt sich die Kenngröße der parallelen Effizienz (E) berechnen. Beide Größen sind neben der Problemgröße, die im Rahmen dieser Arbeit konstant ist, abhängig von der Anzahl der verwendeten Prozessoren. Die parallele Beschleunigung wird durch den Quotienten der Rechenzeit für einen Prozessor und der Rechenzeit für P Prozessoren berechnet. Der Speedup wird folgendermaßen definiert: $S(n,P) = \frac{T(n,1)}{T(n,P)}$. Die Effizienz eines Programms ist als Verhältnis zwischen Speedup und Prozessoranzahl definiert: $E(n,P) = \frac{S(n,P)}{P}$. Dabei ist zu beachten, dass P nicht zwingend die Anzahl der Prozessoren in einem verteilten System widerspiegeln muss, sondern ebenfalls als Anzahl der zur Verfügung stehenden Kerne eines Prozessors betrachtet werden kann. [Kra, RW17]

Im Allgemeinen lässt sich die Gesamtlaufzeit für sequentielle Workflows durch die Summe aller Teillaufzeiten bestimmen. Die Beschleunigung eines Workflows lässt sich entweder durch die parallele Verarbeitung von Daten (Datenparallelität) oder durch die parallele Ausführung unabhängiger Aufgaben (intrinsische Workflow-Parallelität - auch bekannt als „embarrassingly parallel“) realisieren. Eine Kombination beider Verfahren ist ebenfalls möglich. Wie bereits erwähnt, lassen sich Workflows als gerichtete Graphen visualisieren. Dabei stellen die Kanten bzw. Pfade Abhängigkeiten zwischen Aufgaben dar. Ein kritischer Pfad ist der längste Pfad hinsichtlich der Ausführungszeit zwischen dem Start- und End-Knoten. [Tay07]

Exemplarisch lässt sich ein Workflow mit drei Aufgaben ($s1, s2, s3$) definieren. Dabei gilt, dass $s1$ sowie $s2$ unabhängig voneinander sind und $s3$ erst beginnen kann, wenn sowohl $s1$ als auch $s2$ beendet sind. Abbildung 21 visualisiert die Abhängigkeiten des Beispiel-Workflows, die datenparallele Ausführung und die Kombination der datenparallelen Ausführung sowie die intrinsische Workflow-Parallelität. Sowohl die datenparalle als auch die intrinische Workflow-Parallelität wurden bereits in Abschnitt 2.2.1 als Datenparallelität und Taskparallelität vorgestellt.

Besonders bei datenintensiven Aufgaben, die viele Daten von der Festplatte lesen und schreiben, kann eine Kombination von berechnungs- und datenintensiven Aufgaben zu einer Beschleunigung des Systems beitragen, da, während das System auf Daten des Datenspeichers wartet, Teile der berechnungsintensiven Aufgabe ausgeführt werden können. Im Kontext des Sprachmodelltrainings konnte dieses Phänomen mehrfach festgestellt werden. Während der Datenübertragung (Netzwerk) sind Prozessoren nicht ausgelastet, so dass diese zur Berechnung rechenintensiver Aufgaben genutzt werden können. Ein Beispiel dafür ist die Ausführung des „Prepare Data Keyword IDF Estimation“ Containers parallel zu anderen Stufen, da dieser lediglich einen Prozessorkern benötigt und

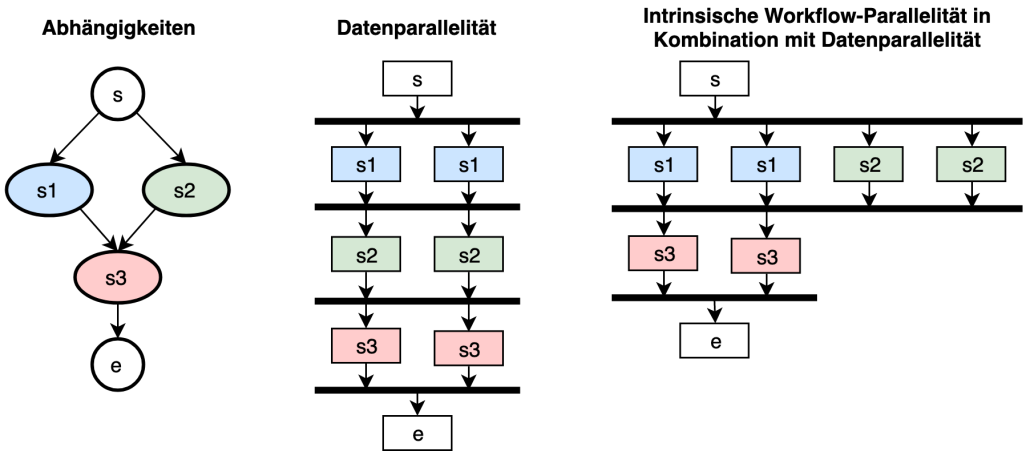


Abbildung 21: Aufgaben und Stufenparallelität

kontinuierlich Datenübertragungen durchführt.

Aufgrund der ursprünglichen Ausführung des Sprachmodelltrainings auf einem CPU-Kern lässt sich zur Berechnung des optimalen Speedup die Gesamtlaufzeit durch die Anzahl der zur Verfügung stehenden CPU-Kerne dividieren. Formell lässt sich dies beschreiben durch: $1/N$, wobei N der Anzahl der CPU-Kerne im Cluster entspricht. Diese Definition nimmt an, dass durch die Parallelisierung kein zusätzlicher Aufwand, wie beispielsweise durch Kommunikation und Synchronisation, entsteht. Daher wird der optimale Speedup in der Praxis nur in Ausnahmefällen erreicht. Zu beachten ist, dass nicht alle Stufen des Trainings parallelisiert werden können und dies in der Berechnung des theoretisch erreichbaren Speedups zu berücksichtigen ist. [RW17]

Sequentielle Ausführung ohne explizite Parallelität Die Laufzeit für die sequentielle Ausführung lässt sich formell definieren. Formel 1 beschreibt die Laufzeit des ursprünglichen sequentiellen Workflows. Dazu wird die Summe über alle Datensätze und Programme gebildet. n_w bezeichnet die Stufen des Workflows, n_D die Anzahl der Datensets. Der Index der Stufen wird als i bezeichnet. Somit gilt: $i \in [0, n_w - 1]$. j bezeichnet den Index des Datensets, entsprechend gilt hier: $j \in [0, n_D - 1]$. Die Anzahl n_{ML} repräsentiert die Anzahl der in Stufe `ML` zu verarbeitenden Datensätze (Chunks). Diese Anzahl kann manuell angepasst werden. Im Rahmen dieser Arbeit wurde die Anzahl n_{ML} aus Gründen der Vergleichbarkeit auf 18 festgelegt. In Tabelle 6 sind neben den Abkürzungen der einzelnen Stufen die gemessenen, sequentiellen Laufzeiten aufgelistet.

$$T(n,1) = \sum_{i<n_w} \sum_{j<n_D} T_{i,j} \tag{1}$$

Stufe	Abkürzung	Laufzeit (in Stunden)
Frequent-Case-Formatter	FC	63.65
Text-Formatter	TF	43.24
Extract text from JSON	ET	0.67
Create train test set	CT	0.12
Prepare Data Keyword IDF	KE	0.82
Make Language Model Split	MLS	0.47
Make Language Model Process	MLP	31.68
Make Language Model Merge	MLM	0.41
Phonetize Dictionary	PD	1.5
Build Lexicon FST	BL	0.23
Format Language Model	FL	0.06
Make Graph	MG	0.82
Gesamtlaufzeit		143,8

Tabelle 6: Abkürzungen der Trainingsstufen und deren sequentiellen Laufzeiten

Datenparallele Ausführung Im Rahmen dieser Arbeit wurde ein Cluster mit insgesamt 72 CPU-Kernen verwendet. Im Folgenden wird daher für die Anzahl der Prozessoren $P = 72$ gesetzt. Gleichung 2 beschreibt die datenparallele Ausführungsdauer. Die Stufen `FC`, `TF` sowie `ET` sind parallelisierbar. Außerdem ist Stufe `MLP` ebenfalls in Abhängigkeit der gewählten Chunk-Anzahl parallelisierbar. Alle weiteren Stufen des Trainingsprozesses lassen sich, wie bereits in Abschnitt 3 erwähnt, nicht oder nur schwer parallelisieren.

$$\begin{aligned} T(n,P) = &\frac{\sum_{j<n_D} T_{FC,j}}{P} + \frac{\sum_{j<n_D} T_{TF,j}}{P} + \frac{\sum_{j<n_D} T_{ET,j}}{P} + T_{CT} + T_{KE} \\ &+ T_{MLS} + \frac{T_{MLP}}{n_{ML}} + T_{MLM} + T_{PD} + T_{BL} + T_{FL} + T_{MG} \end{aligned} \tag{2}$$

Datenparallele Ausführung und intrinsische Workflow-Parallelität Gleichung 3 beschreibt die datenparallele Ausführungsdauer unter Berücksichtigung der intrinsischen Parallelität. Dies bedeutet, dass unabhängige Aufgaben parallel zueinander ausgeführt werden können. Die parallele Ausführung mit intrinsischer Parallelität ist in Abschnitt 3.3 beschrieben sowie in Form eines Aktivitätsdiagramms in Abbildung 6 dargestellt.

$$
\begin{aligned}
T(n,P)_V &= \frac{\sum_{j<n_D} T_{FC,j}}{P} + \frac{\sum_{j<n_D} T_{TF,j}}{P} + \max\Bigg\{KE, \\
&\qquad \frac{\sum_{j<n_D} T_{ET,j}}{P} + CT + MLS + \max\left\{\frac{MLP}{n_{ML}} + MLM, PD + BL\right\} + FL + MG\Bigg\} \\
T(n,P)_I &= \max\left\{KE, CT + MLS + \max\left\{\frac{MLP}{n_{ML}} + MLM, PD + BL\right\} + FL + MG\right\}
\end{aligned}
\tag{3}
$$

Aus der in Gleichung 3 abgebildeten Formel lässt sich die zu erwartende Ausführungszeit eines vollständigen und inkrementellen Trainings berechnen. In Gleichung 4 werden die im Rahmen dieser Arbeit gemessenen Laufzeiten in die in Gleichung 3 dargestellten Formeln eingesetzt. So liegt die theoretisch zu erwartende Dauer eines vollständigen Sprachmodelltrainings bei etwa 7:45 Stunden. Für ein inkrementelles Training lässt sich der Anteil der Stufen `FC`, `TF` sowie `ET` vernachlässigen. Somit ergibt sich für ein inkrementelles Training eine theoretisch zu erwartende Laufzeit von rund 3:40 Stunden.

$$
\begin{aligned}
T(n,P)_V &= \frac{3819\text{ Min}}{72} + \frac{2594\text{ Min}}{72} + \max\Bigg\{49\text{ Min}, \frac{40\text{ Min}}{72} + 7\text{ Min} + 28\text{ Min} \\
&\quad + \max\left\{\frac{1901\text{ Min}}{18} + 25\text{ Min}, 306\text{ Min} + 14\text{ Min}\right\} + 3\text{ Min} + 49\text{ Min}\Bigg\} \\
&= 53\text{ Min} + 36\text{ Min} + \max\left\{49\text{ Min}, 1\text{ Min} + \max\left\{130\text{ Min}, 320\text{ Min}\right\} + 52\text{ Min}\right\} \\
&= 89\text{ Min} + \max\left\{49\text{ Min}, 1\text{ Min} + 320\text{ Min} + 52\text{ Min}\right\} \\
&= 89\text{ Min} + \max\left\{49\text{ Min}, 372\text{ Min}\right\} \\
&= 461\text{ Minuten} \\
&= 7,68\text{ Stunden} \\
T(n,P)_I &= \max\Bigg\{49\text{ Min}, 7\text{ Min} + 28\text{ Min} + \max\left\{\frac{1901\text{ Min}}{18} + 25\text{ Min}, 90\text{ Min} + 14\text{ Min}\right\} \\
&\quad + 3\text{ Min} + 49\text{ Min}\Bigg\} \\
&= \max\left\{49\text{ Min}, 35\text{ Min} + \max\left\{130\text{ Min}, 104\text{ Min}\right\} + 52\text{ Min}\right\} \\
&= \max\left\{49\text{ Min}, 35\text{ Min} + 130\text{ Min} + 52\text{ Min}\right\} \\
&= \max\left\{49\text{ Min}, 217\text{ Min}\right\} \\
&= 217\text{ Minuten} \\
&= 3,61\text{ Stunden}
\end{aligned}
\tag{4}
$$

Der theoretische Speedup für ein vollständiges Training mit 72 CPU-Kernen beträgt 19,18 (vgl. Gleichung 5). Die Effizienz hingegen liegt bei 0,27. Dies bedeutet, dass die Auslastung der zur Verfügung stehenden Hardware lediglich 27% beträgt.

$$\begin{aligned} S(n,P)_V &= \frac{T(n,1)_V}{T(n,P)_V} = \frac{8843 \text{ Min}}{T(n,P)_V} = \frac{8843 \text{ Min}}{461 \text{ Min}} = 19,18 \\ E(n,P)_V &= \frac{S(n,P)_V}{P} = \frac{19,18}{72} = 0,27 \end{aligned} \tag{5}$$

Für ein inkrementelles Training lässt sich der theoretische Speedup nur schätzen, da ein inkrementelles Training in der ursprünglichen Anwendung nicht möglich ist. Pro Tag werden weniger als 10MB neue Textdaten erwartet, deren Verarbeitung rund eine Minute in Anspruch nimmt. Bei der Berechnung der zu erwartenden Laufzeit für ein inkrementelles Training wird daher die Dauer der Stufen `FC`, `TF` sowie `ET` von der Gesamtausführungsdauer abgezogen. So ergibt sich folgender zu erwartender Speedup für die inkrementelle Ausführung:

$$\begin{aligned} T(n,1)_I &= T(n,1)_V - T_{FC} - T_{TF} - T_{ET} \\ &= 8843 \text{ Min} - 3819 \text{ Min} - 2594 \text{ Min} - 40 \text{ Min} = 2390 \text{ Min} \\ S(n,P)_I &= \frac{T(n,1)_I}{T(n,P)_I} = \frac{2390 \text{ Min}}{T(n,P)_I} = \frac{2390 \text{ Min}}{217 \text{ Min}} = 11,01 \\ E(n,P)_I &= \frac{S(n,P)_I}{P} = \frac{11,01}{72} = 0,15 \end{aligned} \tag{6}$$

Gleichung 6 zeigt, dass durch den Wegfall der gut zu parallelisierenden Stufen (`FC`, `TF`) der zu erwartende Speedup sowie die Effizienz des Systems bei einer inkrementellen Ausführung drastisch sinkt. Durch die verbleibenden, nur schwer parallelisierbaren Stufen des Workflows sinkt die zu erwartende Effizienz des Clusters bei einer inkrementellen Verarbeitung auf 15%. Wie bereits in Abschnitt 4.4.2 erwähnt, ist aufgrund der schwer parallelisierbaren Aufgaben des Sprachmodelltrainings eine vertikale Skalierung effektiver als eine horizontale Skalierung.

5.1.2 Performance Vergleich

In diesem Kapitel werden die gemessenen Laufzeiten evaluiert. Abweichungen von der erwarteten Laufzeit werden dabei für jede Plattform analysiert.

Vorab lässt sich sagen, dass ein linearer Speedup nicht zu erwarten ist, da auf dem Cluster ein Antivirensystem läuft, welches, wie bereits erwähnt, in Abhängigkeit des Datenverkehrs und der verwendeten Programme, eine nicht unerhebliche CPU-Last verursacht. Ein vollständiges Deaktivieren des Antivirensystems ist aufgrund von Sicherheitsbestimmungen seitens des Fraunhofer Instituts nicht möglich.

In Abbildung 22 sind die Ausführungszeiten, geordnet nach den jeweiligen Stufen des Sprachmodelltrainings, dargestellt. Es fällt auf, dass besonders die Stufen `FC`, `TF` und `ML` von einer Parallelisierung profitiert haben. Bei genauerer Betrachtung ist zu erkennen, dass die Ausführung „nur“ um den Faktor 15-20 beschleunigt werden konnte, obwohl 72 CPU-Kerne zur Berechnung bereitstehen. Ein Grund für das nicht Erreichen des erwarteten Speedups liegt darin, dass sowohl die Stufe `FC` wie auch `TF` nur partiell parallelisiert wurden. Wie in Abschnitt 3.4.1 und 3.4.2 beschrieben, wurde die Parallelisierung nur hinsichtlich eines einzelnen Servers durchgeführt. Durch die Verwendung von 24 CPU-Kernen für die Stufen `FC` sowie `TF` beträgt der optimale Speedup dieser Stufen bei der Ausführung von Digdag sowie Argo lediglich 24, obwohl 72 CPU-Kerne zur Verfügung stehen. Ein weiterer Faktor, der die Laufzeit beeinflusst, ist die Notwendigkeit der Datenübertragung. Anders als bei dem ursprünglichen Training müssen Daten über eine Netzwerkverbindung in den entsprechenden Docker-Container geladen werden. Nach Abschluss der Berechnung müssen die Ergebnisse über das Netzwerk abgespeichert werden. Ebenfalls ist die zusätzliche CPU-Last des Antivirensystems nicht zu vernachlässigen.

Im Gegensatz zu Argo und Digdag ist aufgrund der Architektur von Pachyderm eine Nutzung aller zur Verfügung stehenden Ressourcen in Stufe `FC` wie auch `TF` möglich. Zur Ausführung des Trainings mittels Pachyderm wurde sowohl für die Stufe `FC` als auch `TF` eine konstante Parallelität von 50 verwendet. Dies bedeutet, dass 50 Dateien parallel bearbeitet werden können. Während Argo und Digdag lediglich eine Parallelität von 24 verwenden, unterscheidet sich die Laufzeit nur um wenige Minuten. Eine Parallelität von 50 ist bereits ausreichend, um eine nahezu 100%ige Auslastung des Clusters zu erreichen. Dies ist durch weitere Prozesse zu erklären, die zur Ausführung des Systems benötigt werden. Hierzu zählt beispielsweise Kubernetes, Grafana, InfluxDB sowie das Antivirensystem. Eine höhere Parallelität führte zu einer längeren Ausführungszeit. Aufgrund des zeitintensiven Trainings sowie der im Rahmen dieser Arbeit zur Verfügung stehenden Zeit werden an dieser Stelle keine weiteren Messungen hinsichtlich des optimalen Parallelisierungsfaktors durchgeführt.

Besonders auffällig sind Schwankungen der Stufe `KE` (vgl. Abbildung 22), die durch Netzwerklatenzen sowie den aktivierten On-Access-Scan des installierten Antivirensystems zu erklären sind. Während Argo und Digdag diese Stufe sequentiell ausführen, ist

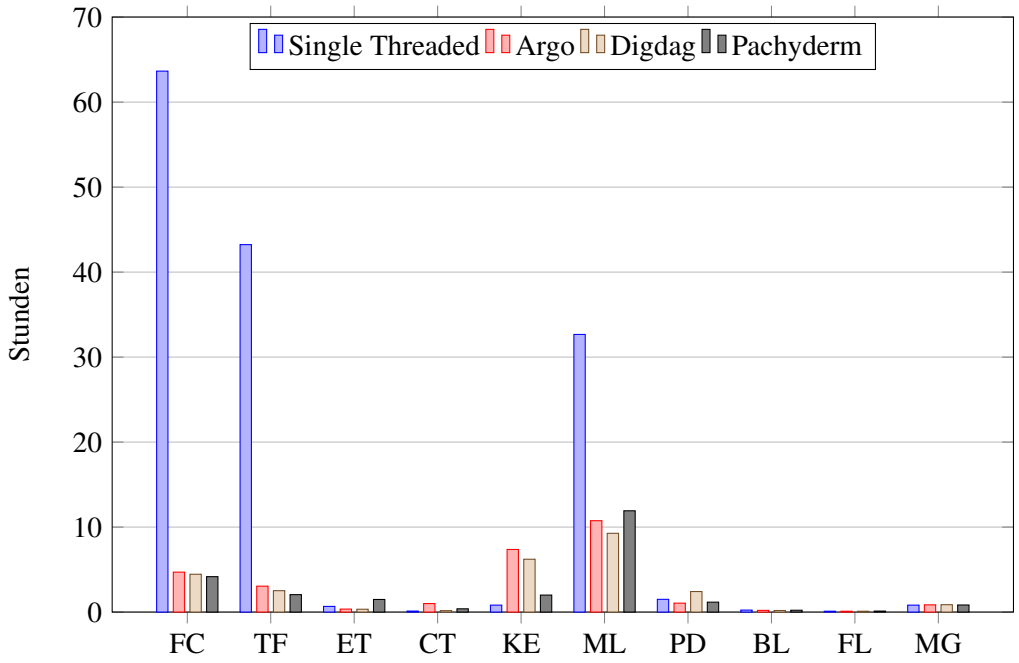


Abbildung 22: Ausführungszeiten nach Trainingsstufen

Pachyderm durch den datenorientierten Ansatz in der Lage, Datenparallelität implizit auszunutzen. Dennoch benötigt Pachyderm mehr als doppelt so lange wie die ursprüngliche, nicht containerisierte Anwendung. Dies ist durch Netzwerkübertragungen sowie den zusätzlichen Aufwand, der zum Starten und Stoppen der Container notwendig ist, zu erklären. Weitere Tests zur genauen Analyse des Overheads durch Container sind in einer weiterführenden Arbeit durchzuführen.

Im Gegensatz zu Argo und Digdag ist es in Pachyderm nicht möglich, ein temporäres Dateisystem während der Berechnung zu verwenden. Aufgrund der im Cluster eingesetzten Festplatten (HDDs) wurde während der Ausführung mit Pachyderm eine nahezu 100%ige Auslastung dieser festgestellt. Dies ist auf die hohe Parallelität der ausgeführten Workflows zurückzuführen. Für die Zukunft gilt es zu evaluieren, ob die Trainingszeit durch den Einsatz von SSDs reduziert werden kann.

Im Vergleich zu Argo und Digdag benötigt Pachyderm mehr als doppelt so viel Zeit zur Extraktion der zu verarbeitenden Textdaten aus den JSON-Dokumenten (Stufe ET). Da diese Aufgabe sehr speicherintensiv ist, ist auch für diese Aufgabe zu untersuchen, ob sich der Einsatz eines SSD Speichers positiv auf die Laufzeit auswirkt.

Die Übertragung von Artefakten zwischen einzelnen Stufen eines Workflows realisiert Argo durch das Kopieren komprimierter Archive. In Abhängigkeit der Artefaktgröße kann der Komprimier- bzw. Dekomprimierprozess mehrere Minuten in Anspruch nehmen. Argo benötigt für die Stufe CT durch die Komprimierung von Artefakten in Argo

	Single Thread	Argo	Digdag	Pachyderm
Frequent-Case-Formatter (FC)	63.65	4.7	4.45	4.16
Text-Formatter (TF)	43.23	3.05	2.52	2.05
Extract text from JSON (ET)	0.67	0.35	0.33	1.48
Create train test set (CT)	0.12	1.0	0.17	0.05
Prepare Data Keyword IDF (KE)	0.82	7.37	6.22	2.0
Make Lang. Model Split (MLS)	0.47	1.31	0.58	0.56
Make Lang. Model Process (MLP)	31.68	7.5	7.05	9.14
Make Lang. Model Merge (MLM)	0.42	1.93	1.59	1.78
Phonetize-Dictionary (PD)	1.5	1.05	2.42	1.16
Build-Lexicon-FST (BL)	0.23	0.2	0.18	0.21
Format-LM (FL)	0.06	0.1	0.1	0.11
Make-Graph (MG)	0.82	0.85	0.87	0.83
Gesamtlaufzeit (intrinsisch und datenparallel)		19.5	17.83	23
Speedup (vollständig)		7.55	8.26	6.47

Tabelle 7: Laufzeiten (in Stunden)

mehr als 50 Minuten länger als in Digdag (vgl. Tabelle 7). Dies ist primär dem eingesetzten Komprimierungsprogramm geschuldet.[40]

In den durchgeführten Experimenten erzielte Digdag eine vergleichbare Ausführungszeit wie Argo. Besonders bei Stufen des Trainings, in denen wenige besonders große Dateien generiert werden, ergibt sich aufgrund der manuellen Datenspeicherung ohne zip-Komprimierung ein Geschwindigkeitsvorteil gegenüber Argo (vgl. `CT`, `MLS`). Gleichzeitig ist bei Stufen, die viele kleine Dateien ausgeben, ein Nachteil aufgrund der Datenübertragung festzustellen.

Die längere Ausführungszeit der Stufen `KE` und `CT` (vgl. Tabelle 7) sind durch den notwendigen Datentransfer zu erklären. Besonders bei der Stufe `KE` führt dies zu einer Verlängerung der Ausführungszeit um mehr als den Faktor 5. Aufgrund der eingebauten Datenverarbeitung ist Pachyderm erheblich schneller als die eigens entwickelte Datenverwaltung für Argo und Digdag. Von weiteren Versuchen, die Ausführung mittels Argo und Digdag zu beschleunigen, wurde abgesehen, da diese Stufe, wie bereits in Abschnitt 3.1.5 erwähnt, in Zukunft durch eine andere Methode abgelöst wird.

Aufgrund der Parallelisierung der Stufe `MLP` wurden die Stufen `MLS` sowie `MLM` neu hinzugefügt und sind in dem ursprünglichen Workflow nur teilweise enthalten. Durch den zusätzlichen Berechnungsaufwand verlängert sich die Ausführungsdauer der Stufen `MLS` und `MLM`. Der Geschwindigkeitsvorteil der Stufe `MLP` kompensiert jedoch diesen Mehraufwand. In Abbildung 22 ist aus Gründen der Übersichtlichkeit lediglich die Stufe `ML`

[40] `https://github.com/argoproj/argo/issues/1122`

enthalten. Der angegebene Wert setzt sich als Summe der Laufzeiten `MLS`, `MLP` und `MLM` zusammen.

Zu beachten ist, dass bei der parallelen Ausführung eine gleichmäßige Auslastung des Systems vorliegt. So konnte als Performance Problem identifiziert werden, dass die Chunks der Stufe `MLP` keine gleiche Ausführungsdauer aufweisen. Dies liegt daran, dass Wörter, die in dem zuvor erstellten Lexikon weiter oben stehen, häufiger vorkommen und somit zu einer insgesamt höheren Ausführungsdauer führen.[41] Dieser Effekt lässt sich durch die Erhöhung der Anzahl zu erstellender Chunks beheben. Dies führt jedoch in Argo zu einer unbrauchbaren Repräsentation des Workflows (vgl. Abbildung 14). Pachyderm hingegen stellt die Aufteilung von einzelnen Pipelines in Chunks nicht explizit dar, wodurch eine Skalierung ohne Auswirkungen auf die Darstellung erfolgen kann.

Auf die gemessenen Laufzeiten der Stufen `PD`, `BL`, `FL` und `MG` wird nicht näher eingegangen, da diese nur geringe Laufzeitunterschiede aufweisen und auf die Gesamtlaufzeit einen geringen Einfluss haben.

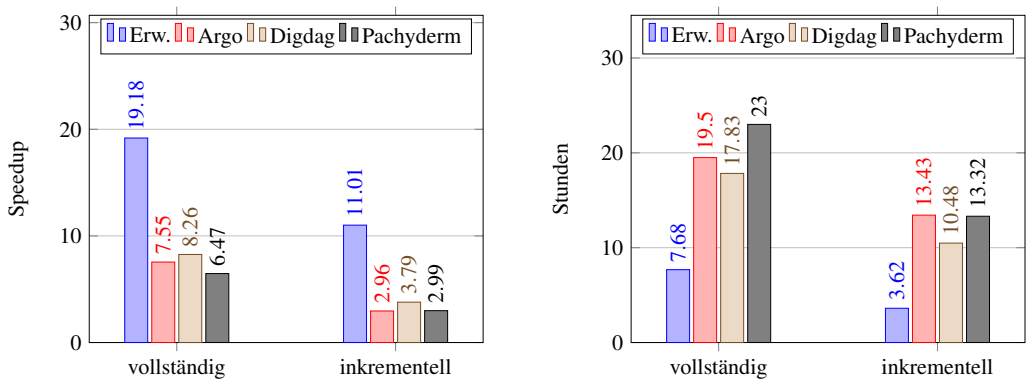
In Abbildung 23a ist der Speedup der einzelnen Workflow-Engines sowohl für ein vollständiges als auch für ein inkrementelles Training im Vergleich zu dem zuvor berechneten, zu erwartenden Speedup dargestellt. Es fällt auf, dass der erwartete Speedup, wie bereits zuvor ausgeführt, nicht erreicht wurde. Besonders bei der inkrementellen Ausführung ist eine große Abweichung zu der erwarteten Laufzeit festzustellen. Diese ist primär auf die zuvor beschriebenen Probleme der Parallelisierbarkeit der Stufe `MLP` zurückzuführen.

Abbildung 23b stellt die gemessenen Laufzeiten dar. Wie auch bei den Ergebnissen des Speedups fällt auf, dass die Ausführungszeit deutlich über der erwarteten Laufzeit liegt. Wie bereits erwähnt, ist dies auf eine Vielzahl von Faktoren zurückzuführen, wie Parallelisierbarkeitsprobleme der Stufe `MLP`, das eingesetzte Antivirensystem, zusätzliche Datenübertragung, langsame Festplattengeschwindigkeit sowie Overhead durch die Ausführung von Containern.

Zusammenfassung Das Ziel, ein Sprachmodell pro Tag trainieren zu können, wurde erreicht. Es fällt jedoch auf, dass die Laufzeit um mehr als das Doppelte von den erwarteten Laufzeiten sowohl bei den vollständigen als auch bei den inkrementellen Trainingsdurchläufen abweicht.

Die durchgeführten Geschwindigkeitsmessungen zeigen, dass ein linearer Speedup bei einer verteilten Ausführung im Kontext des Sprachmodelltrainings nicht zu erwarten

[41] `http://lium3.univ-lemans.fr/mtmarathon2010/lectures/04-irstlm.pdf`



(a) Speedup (72 CPU-Kerne) (b) Laufzeit (72 CPU-Kerne)

Abbildung 23: Speedup und Laufzeit

ist. Neben der zusätzlich benötigten Datenübertragung zwischen den einzelnen Knoten ist der hohe sequentielle und nicht parallelisierbare Anteil des Trainingsprozesses ausschlaggebend bzw. ein limitierender Faktor hinsichtlich der Beschleunigung der Anwendung. Des Weiteren ist zu beachten, dass externe Anwendungen, wie beispielsweise ein Antivirensystem, die Laufzeit durch eine konkurrierende Verwendung der zur Verfügung stehenden Hardware erheblich beeinflussen kann. Zur Durchführung von reproduzierbaren Laufzeitmessungen ist die exklusive Nutzung der zur Verfügung stehenden Ressourcen zu gewährleisten.

Je nach Anwendungsfall ist zu untersuchen, ob eine horizontale Skalierung, beispielsweise durch Datenparallelität, möglich ist oder ob eine vertikale Skalierung von Vorteil ist.

Weiterhin ist festzuhalten, dass bislang keine der im Rahmen dieser Arbeit untersuchten Workflow-Engines eine datenlokale Ausführung von Workflows unterstützt. Eine datenlokale Ausführung könnte die Workflows merklich beschleunigen, da derzeit, wie auch anhand der Ergebnisse zu sehen ist, Zeit durch die Datenübertragung verloren geht.

5.2 Anwendungsfelder & Vorgehensmodell

In diesem Kapitel werden verschiedene Anforderungen definiert die eine Anwendung erfüllen muss, um von der Ausführung durch eine Workflow-Engine profitieren zu können. Der Fokus liegt dabei auf Scientific Workflow-Engines. Anschließend wird ein Vorgehensmodell entwickelt, dass den Prozess der Anpassung einer bestehenden Anwendung zur Ausführung auf einer Scientific Workflow-Engine beschreibt. Aufgrund der Vielzahl an verfügbaren Workflow-Engines sowie der Möglichkeit, Workflows re-

produzierbar zu entwickeln, beschränkt sich das Vorgehensmodell auf containerbasierte Workflow-Engines.

5.2.1 Voraussetzungen

Zur Ausführung sowie Beschleunigung von bestehenden Anwendungen mithilfe von Scientific Workflow-Engines lassen sich verschiedene Kriterien definieren, die eine Anwendung erfüllen muss. Die Vielzahl von Workflow-Engines ist ein Indiz für die vielen Anwendungsbereiche. Es existiert keine Workflow-Engine, die für alle Anwendungsfälle geeignet ist. Ein entscheidendes Merkmal wissenschaftlicher Workflows ist die fehlende Interaktion zwischen Mensch und Workflow. So werden Workflows manuell oder automatisch mit einer bestimmten Eingabe bzw. Parametern gestartet. Während der Ausführung eines Workflows findet keine Mensch-Workflow Interaktion statt.

Eine wirkliche Gewährleistung der Reproduzierbarkeit sowohl auf Datenebene als auch auf Ebene der Anwendung lässt sich durch Workflow-Engines wie Argo und Pachyderm erreichen, die zur Ausführung Docker-Container verwenden und eine versionierte Art der Datenspeicherung bieten.

5.2.2 Vorgehensmodell

Ein Ausschnitt der im Rahmen dieser Arbeit untersuchten Workflow-Engines sowie Kriterien sind in Tabelle 5 dargestellt.

Im Rahmen dieser Arbeit wurde ein Vorgehensmodell entwickelt, welches die Wahl einer geeigneten Workflow-Engine erleichtert und bei der damit verbundenen Anwendungsumstrukturierung hilft. Im Folgenden ist eine Auflistung der notwendigen Schritte beschrieben, die im Anschluss detailliert erläutert werden.

Vorgehen

1. Unterstützung von Reproduzierbarkeit
2. Analyse sowie Zerlegung der Anwendung in Teilschritte
3. Eingesetzte Programmiersprachen
4. Datenspeicherung
5. Datentransfer zwischen Stufen
6. Geschwindigkeitanforderungen / Echtzeitanforderungen
7. Ausführungsdauer der Schritte
8. Imperative oder deklarative Beschreibung von Workflows

- Wie werden Abhängigkeiten zwischen Containern beschrieben?

9. Horizontale Skalierung
    - Durch Aufteilen der Daten
    - Durch parallele Algorithmen
10. Entwickeln / Testen / Optimieren
    - Implementierung des Workflows mithilfe einer Workflow-Engine
    - Identifizierung von Bottlenecks (vgl. Abschnitt 4.4.1)
    - Parallelisierung (implizit oder explizit)

Reproduzierbarkeit Um die Reproduzierbarkeit von datenbasierten Anwendungen zu ermöglichen, sind folgende vier Fragen zu beantworten. Eine detaillierte Beantwortung dieser Fragen hilft bei der Zerlegung der Anwendung in logisch unabhängige und somit potentiell parallelisierbare Komponenten.

1. Welche Daten werden verarbeitet?
    - Wo befinden sich die Eingabedaten?
    - Dateien / Parameter / Datenbankabfragen
2. Welches Programm wird zur Berechnung verwendet?
    - Eigenes Programm / Analyse Framework / etc.
3. Laufzeitumgebung
    - Betriebssystem / benötigte Software sowie Pakete / CPU- und Speicheranforderungen
4. Wie werden Stufen ausgeführt?
    - Shell-Kommandos / lokale oder entfernte Berechnungen / etc.

Wie bereits erwähnt, bieten nicht alle untersuchten Workflow-Engines die Möglichkeit, Daten zu speichern. Eine vollständige Reproduzierbarkeit bieten lediglich Argo und Pachyderm. Bei dem Einsatz von Workflow-Engines wie Digdag, Reana und Meson ist der Entwickler eines Workflows für eine entsprechende Versionierung und Reproduzierbarkeit hinsichtlich der verwendeten Daten verantwortlich. Sowohl Digdag als auch Meson wurden u.a. für die Ausführung von Workflows entwickelt, die Daten aus einer Datenbank abrufen. Mithilfe des Backfill-Prinzips lässt sich sowohl in Digdag als auch in Meson die Reproduzierbarkeit von Workflows sicherstellen.

Analyse / Zerlegung der Anwendung: Um Vorteile einer Workflow-Engine, wie die parallele sowie inkrementelle Verarbeitung, zu nutzen, ist eine Zerlegung der Anwendung in einzelne Verarbeitungsschritte erforderlich. Die Zerlegung hat, wie in Abschnitt 3.3 beschrieben, Einfluss auf die Parallelisierbarkeit einer Anwendung.

Eingesetzte Programmiersprachen: Ein weiteres Kriterium stellen die verwendeten Programmiersprachen dar. Viele Workflow-Engines unterstützen die direkte Ausführung von Programmiersprachen. So eignet sich beispielsweise die Workflow-Engine Airflow zur Orchestrierung von Python-basierten Workflows. Eine Programmiersprachen unabhängige Lösung stellen containerbasierte Workflow-Engines dar. Diese ermöglichen neben der Programmiersprachen unabhängigen Definition von Workflows die Ausführung von Teilschritten eines Workflows, die in verschiedenen Programmiersprachen entwickelt wurden.

Datenspeicherung Möglichkeiten, Daten sowie Artefakte abzuspeichern, unterscheiden sich zwischen allen untersuchten Workflow-Engines. Während einige Systeme, wie Digdag und Meson, keine Datenverwaltung unterstützen, bieten Argo und Pachyderm eine Möglichkeit, Daten versioniert zu speichern. Dabei sind Ausgabedaten in Pachyderm in einem speziellen Verzeichnis abzulegen, während Argo die Angabe eines Pfades ermöglicht, aus dem nach Abschluss der Berechnung Daten gesichert werden.

Datentransfer zwischen Stufen Neben der Versionierung sowie Datenspeicherung stellt die Unterstützung der Datenübertragung zwischen einzelnen Stufen eines Workflows ein weiteres Kriterium bei der Wahl der Workflow-Engine dar. Für die reine Koordination der Ausführung von Aufgaben eignen sich Workflow-Engines wie Digdag oder Argo. Für datenorientierte Workflows eignet sich beispielsweise Pachyderm.

Geschwindigkeitsanforderungen / Echtzeitanforderungen Analysen hinsichtlich der Eignung der Workflow-Engines für High-Performance Anwendungen wurden im Rahmen dieser Arbeit nicht durchgeführt. Di Tommaso et al. untersuchten bereits 2015 die Laufzeitauswirkung von Docker-Containern bei einer Anwendung der Bioinformatik. Die durchgeführten Experimente zeigen, dass Docker-Container nur einen geringfügigen Einfluss auf langlaufende Berechnungen haben. Hingegen konnte bei Berechnungen, die durchschnittlich rund 45 Sekunden benötigen, eine Verlangsamung um den Faktor 1.6 festgestellt werden. [DTPC+15]

Es ist jedoch anzumerken, dass bei containerorientierten Workflow-Engines Unterschiede in der Ausführungszeit festzustellen sind. Beispielsweise erstellt Argo für jede Stufe

eines Workflows einen eigenen Pod. Wie bereits in Abschnitt 4.3.6 erwähnt, verwendet Pachyderm Pods mehrfach. Somit entfällt die im Vergleich zu Argo benötigte Zeit zum Erstellen sowie Löschen von Pods. Dennoch bleibt die Eignung zur Ausführung von High-Performance Anwendungen mithilfe von containerisierten Workflow-Engines zu untersuchen. Wie bereits von Di Tommaso et al. gezeigt, ist der zusätzliche Zeitaufwand zum Starten und Stoppen von Containern bei der Ausführung von vielen kurzlebigen Aufgaben problematisch. Der genaue Overhead, der durch das Verwalten von Containern entsteht, ist in einer weiterführenden Arbeit zu untersuchen.

Wie auch im Rahmen dieser Arbeit in den Stufen „Frequent-Case-Formatter“ sowie „Text-Formatter“ ist je nach Anwendungsfall zu untersuchen, ob eine Parallelisierung mithilfe eines Taskpools innerhalb eines Containers Geschwindigkeitsvorteile bietet und die Workflow-Engine lediglich zu Koordinationszwecken verwendet wird.

Durch die Verwendung von Kubernetes kann der Startvorgang einzelner Schritte eines Workflows mehrere Sekunden beanspruchen. Ebenfalls ist ein Priorisieren von Workflows i.d.R. nicht möglich, weshalb Workflows bei einer hohen Auslastung des Clusters unter Umständen auf freie Ressourcen warten müssen. Für Anwendungen die eine Echtzeit nahe Ausführung von Aufgaben benötigen, ist zu untersuchen, welche Workflow-Engines dies unterstützen.

Ausführungsdauer der Schritte Je nach Anwendungsfall kann die Ausführungsdauer einzelner Schritte stark variieren. Während im Kontext des Sprachmodelltrainings langlaufende Schritte ausgeführt werden, existieren Anwendungsfälle, wie beispielsweise in der Bioinformatik, in denen einzelne Schritte mit verschiedenen Parametern ausgeführt werden müssen. Je nach Anwendungsfall ist die Performance der Workflow-Engines in Abhängigkeit der Ausführungsdauer einzelner Schritte zu untersuchen.

Imperative oder deklarative Beschreibung von Workflows Workflows unterscheiden sich in der Art, wie Abhängigkeiten definiert werden. Je nach Anwendungsfall ist zu evaluieren, ob ein imperativer oder deklarativer Stil zu bevorzugen ist. Bei der deklarativen Angabe von Abhängigkeiten übernimmt die Workflow-Engine die Analyse, welche Aufgaben parallel zueinander ausgeführt werden können. Dies ist besonders im wissenschaftlichen Umfeld wichtig, wo Wissenschaftler Spezialisten auf ihrem Fachgebiet sind, jedoch nicht unbedingt Experten im Bereich der parallelen Programmierung.

Möglichkeiten der horizontalen Skalierung Je nach Anwendungsfall lassen sich Anwendungen durch die parallele Verarbeitung von Daten beschleunigen. Optimalerweise lässt sich ein linearer Speedup erreichen. Jedoch ist eine horizontale Skalierung

nicht für jede Anwendung praktikabel, weshalb eine Beschleunigung nur bedingt möglich ist.

Für datenparallele Anwendungen eignet sich die Workflow-Engine Pachyderm, da, wie auch im Rahmen dieser Arbeit ausgeführt (vgl. Abschnitt 4.3.6 und 5.1.2), eine horizontale Skalierung ohne explizite Angabe der Parallelität erfolgt. Dies ermöglicht, wie anhand einzelner Stufen des Sprachmodelltrainings gezeigt, eine vollständige Ausnutzung der zur Verfügung stehenden Rechenleistung.

Identifizierung von Bottlenecks Mithilfe von Monitoring Lösungen zur Überwachung der Ressourcenauslastung lassen sich unzureichend optimierte Stufen eines Workflows identifizieren, deren Beschleunigung zu einer kürzeren Ausführungsdauer des Workflows führt. Weitere Informationen hierzu sind in Abschnitt 4.4.1 zu finden.

Implizite vs. explizite Parallelität Während Entwickler bei der Verwendung expliziter Parallelität mehr Entscheidungsspielraum bzw. Einfluss auf die Ausführung einer Anwendung haben, ist die implizite Parallelität von Workflows hilfreich, um eine Verbesserung hinsichtlich der Laufzeit zu erreichen, ohne Kenntnisse über parallele Konstrukte zu besitzen oder den Gesamtfluss des Workflows zu verstehen.

5.2.3 Zusammenfassung

Das in diesem Kapitel vorgestellte Vorgehensmodell bietet eine Hilfestellung bei der Wahl einer geeigneten Workflow-Engine für eine neue oder bereits existierende Anwendung. Für Scientific Workflows sind hierbei grundlegende Anforderungen zu erfüllen. Sofern eine Mensch-Workflow Interaktion notwendig ist, sind Business Workflow-Engines zu evaluieren. Containerbasierte Scientific Workflow-Engines bieten die Möglichkeit, reproduzierbare Workflows zu erstellen. Das entwickelte Vorgehensmodell beschäftigt sich mit Fragen hinsichtlich der Zerlegung einer Anwendung in Teilschritte, der Analyse von Bottlenecks sowie der Aufteilung von Daten zur Parallelisierung. Mithilfe dieses Vorgehensmodells sowie dem in Abschnitt 4.3.2 vorgestellten Vergleich verschiedener Workflow-Engines wird die Wahl einer passenden Workflow-Engine erleichtert.

6 Zusammenfassung

Ziel dieser Arbeit war es, verschiedene Workflow-Engines hinsichtlich der Eignung des Training eines Sprachmodells am Fraunhofer Institut zu evaluieren. Dazu wurden zunächst Workflow-Engines ausgewählt, für die das Sprachmodelltraining entsprechend implementiert wurde. Aufgrund diverser, in Kapitel 4 beschriebener Schwierigkeiten, waren teils umfangreiche Änderungen an der jeweiligen auszuführenden Anwendung des Sprachmodelltrainings erforderlich. Zum Vergleich der Workflow-Engines wurden verschiedene Laufzeitmessungen durchgeführt. Des Weiteren wurde mit den im Rahmen dieser Arbeit gewonnenen Erkenntnissen ein Vorgehensmodell entwickelt. Im Folgenden werden sowohl die Workflow-Engines als auch die erhaltenen Ergebnisse kritisch begutachtet und Verbesserungsvorschläge sowie Ideen für weitere Forschungen vorgestellt.

6.1 Ergebnisse

Es wurde gezeigt, dass bislang keine „perfekte“ Workflow-Engine für wissenschaftliche Anwendungen existiert. Aufgrund der unterschiedlichen Anwendungsfälle ist eine allgemeine Lösung nur schwer realisierbar. Viele Workflow-Engines sind auf bestimmte Anwendungsfälle spezialisiert. Mithilfe der Common Workflow Language wurde ein erster Versuch unternommen, Workflows zu standardisieren und somit eine berechnungsumgebungsunabhängige Darstellung zur Ausführung auf Workflow-Engines zu ermöglichen. Bislang unterstützen jedoch erst wenige Workflow-Engines die Ausführung dieser. Lediglich Reana unterstützt die Ausführung von Workflows mittels der Common Workflow Language sowie Docker-Containern.

Um die Reproduzierbarkeit von wissenschaftlichen Ergebnissen garantieren zu können, ist neben der containerbasierten Ausführung eine Versionierung der Daten zu gewährleisten. Bislang unterstützt lediglich Pachyderm die vollständige Reproduzierbarkeit von Workflows.

Die Parallelisierung von Workflows stellt viele Workflow-Engines vor Herausforderungen. So unterstützt lediglich Pachyderm eine datenparallele Ausführung von Workflows. Für Workflow-Engines wie Argo und Digdag sind, wie in Abschnitt 4.3 beschrieben, entsprechende Anpassungen notwendig. So sind beispielsweise die Datenaufteilung und Zuweisung der Daten an die entsprechenden Container durch den Entwickler zu übernehmen. Nach entsprechender Anpassung konnten alle getesteten Workflow-Engines das Ziel, ein Sprachmodell pro Tag zu trainieren, erreichen. Die Laufzeitunterschiede sind gering und primär auf Umgebungsfaktoren, wie Antivirensysteme oder Festplattenauslastung, zurückzuführen.

Eine Ausführung von Workflows unter Berücksichtigung der Datenlokalität wird derzeit von keiner der untersuchten Workflow-Engines unterstützt.

Es wurde gezeigt, dass im Kontext des Sprachmodelltrainings kein Vorteil durch eine Implementierung des Backfill Prinzips identifiziert werden konnte. Das Backfill Prinzip eignet sich besonders für Anwendungsfälle, die beispielsweise täglich Berechnungen auf den jeweiligen Daten des Tages ausführen müssen. Da im Kontext des Sprachmodelltrainings lediglich das aktuellste Sprachmodell relevant ist, wird diese Funktionalität nicht benötigt.

Bei den durchgeführten Laufzeitmessungen konnte festgestellt werden, dass leistungsstärkere Prozessoren, besonders Prozessoren mit einer besseren Single-Thread Performance, einen erheblichen Leistungsvorteil bei der Ausführung des Sprachmodelltrainings erzielen konnten. Dies liegt vor allem an dem hohen sequentiellen Anteil der Anwendung.

Für jeden Anwendungsfall ist zu entscheiden, inwieweit die jeweilige Workflow-Engine Aufgaben automatisieren soll.

Es fällt auf, dass Bemühungen zur Standardisierung von Scientific Workflows noch nicht ausreichend sind, um Workflow-Engines zu vereinheitlichen bzw. eine Interoperabilität dieser zu ermöglichen. Viele Systeme sind aufwändig einzurichten und nur schwierig zu testen / analysieren. Ein lokales grafisches Entwickeln von Workflows und ein anschließendes Ausführen auf einem Cluster sind wünschenswert. Die dazu durchgeführten Tests mit Rabix Composer sowie Reana konnten aufgrund von Installationsproblemen nicht durchgeführt werden. [Tay07]

Durch die im Rahmen dieser Arbeit angefertigten Skripts sowie die containerisierte Ausführung mithilfe der beschriebenen Workflow-Engines sind die Ergebnisse dieser Arbeit reproduzierbar und können somit als Grundlage für weitere Untersuchungen genutzt werden. Themen bzw. Fragestellungen, die untersucht werden können, sind in Abschnitt 6.2 beschrieben.

Das entwickelte Vorgehensmodell bietet eine Hilfestellung bei der Auswahl einer geeigneten Workflow-Engine sowie den durch die Implementierung verbundenen Änderungen einer Anwendung.

6.2 Ausblick

Die Workflow-Engine Meson konnte, da diese derzeit noch nicht veröffentlicht ist, nicht getestet werden. Davis Shepherd, ein Netflix Entwickler, führte daher eine Bewertung der Workflow-Engine mithilfe der in Abschnitt 4.3.2 vorgestellten Kriterien durch. Bis

auf die Ausnahme der versionierten Datenspeicherung unterstützt die Workflow-Engine alle für das Sprachmodelltraining benötigten Funktionen. Somit ist eine Untersuchung der Workflow-Engine nach Veröffentlichung zu empfehlen.

Eine weitere Workflow-Engine, für die derzeit an einer Unterstützung der Common Workflow Language und der Ausführung von Docker-Containern gearbeitet wird, ist Taverna[42]. Alternativ ist die Nutzung der CWL-fähigen Workflow-Engine toil[43] zu untersuchen. Zu beachten ist, dass diese lediglich die Ausführung in Amazon AWS, Microsoft Azure sowie in der Google Compute Engine unterstützt. Eine lokale Installation mithilfe von Kubernetes oder Docker Swarm ist nicht möglich.

Wie bereits in Abschnitt 5.1.2 erwähnt, ist zu untersuchen, welcher Parallelisierungsfaktor für das Training mit Pachyderm geeignet ist. Entscheidend ist, wie die Granularität der Datenaufteilung gewählt wird. So konnten im Rahmen dieser Arbeit enorme Performance Einbußen bei der Ausgabe mehrerer Millionen Dateien festgestellt werden.

Eine genauere Untersuchung der Auswirkungen des Antivirensystems können ebenfalls neue Erkenntnisse liefern. Durch eine Messung mit vollständig abgeschaltetem Antivirensystem kann der Einfluss des Antivirensystems auf die Trainingsdauer bei einer parallelen sowie verteilten Ausführung untersucht werden.

Weiter kann untersucht werden, ob ein verteilter Minio Speicher ebenfalls Performance Vorteile im Vergleich zu einer alleinstehenden Installation liefert. Hierbei könnte vor allem eine Replikation der Daten die Ausführung beschleunigen, da dies zu einer künstlichen Datenlokalität führt. Bei ausreichend großem Arbeitsspeicher bietet es sich ebenfalls an, den Minio Speicher in einem temporären Dateisystem zu verwenden. Dies ist jedoch nur für kleine Datenmengen realisierbar. Bei diesem Ansatz ist beispielsweise durch eine Replikation der Daten, der Verlust dieser, beispielsweise bei Stromausfall, zu verhindern.

Um die Eignung der untersuchten Workflow-Engines für High-Performance Anwendungen bewerten zu können, ist es notwendig, weitere Experimente durchzuführen, in denen untersucht wird, welcher Overhead durch das Starten, Stoppen sowie Verwalten von Containern entsteht. Klop untersucht den Overhead der Container-Ausführung in seiner Arbeit. Dabei beziehen sich die Ergebnisse auf die direkte Ausführung von Containern in Docker Swarm und Kubernetes. Wichtiger ist jedoch, wie schnell Workflow-Engines bestimmte Aufgaben abarbeiten können. Pachyderm erstellt Pods zu Beginn einer Pipeline und führt diese so lange aus, wie die Pipeline existiert. Sobald neue Daten ankommen, ist der entsprechende Container bereits gestartet und kann sofort mit den entsprechenden Berechnungen beginnen. In Argo ist dies nicht möglich. Interessant wäre ein Vergleich

[42] `https://issues.apache.org/jira/browse/TAVERNA-900`
[43] `http://toil.ucsc-cgl.org/`

der Ausführungszeiten der beiden Workflow-Engines bei Ausführung vieler, kurzer Aufgaben. [Klo18]

Aufgrund der hohen Anzahl der zur Verfügung stehenden Workflow-Engines, ist eine genaue Analyse einzelner Systeme im Rahmen dieser Arbeit nicht möglich. Das Ziel einer weiterführenden Arbeit könnte daraus bestehen, die bereits partiell ausgefüllte Tabelle zum Vergleich von Workflow-Engines weiterzuführen, um neben der reinen Auflistung verschiedener Workflow-Engines diese anhand verschiedener Kriterien beurteilen zu können, um somit die Eignung für einen gegebenen Anwendungsfall abschätzen zu können.

Literatur

[ALO02] Götz Alefeld, Ingrid Lenhardt und Holger Obermaier: *Parallele numerische Verfahren*. Springer, 2002, ISBN 978-3-642-56350-8.

[baca] *Digdag - Command reference*. `https://docs.digdag.io/command_reference.html#backfill`, besucht: 2018-12-07.

[bacb] *Optimizing Scheduling Behavior - Backfill, Node Sets, and Preemption*. `http://docs.adaptivecomputing.com/mwm/7-1-3/help.htm#topics/optimization/backfill.html`, besucht: 2018-12-07.

[Ber10] Nicola Bertoldi: *IRSTLM Toolkit*, September 2010. `http://lium3.univ-lemans.fr/mtmarathon2010/lectures/04-irstlm.pdf`.

[BL99] Robert D. Blumofe und Charles E. Leiserson: *Scheduling Multithreaded Computations by Work Stealing*. J. ACM, 46(5):720–748, #sep# 1999, ISSN 0004-5411. `http://doi.acm.org/10.1145/324133.324234`.

[BPM11] *Business Process Model And Notation Specification Version 2.0*, Januar 2011. `https://www.omg.org/spec/BPMN/2.0/PDF`.

[CGB+16] Fan Chung, Ronald Graham, Ranjita Bhagwan, Stefan Savage und Geoffrey M. Voelker: *Maximizing Data Locality in Distributed Systems*. 2016. `https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/jcss.pdf`.

[Cha] Eugene Charniak: *Language modeling and probability*. `http://cs.brown.edu/courses/cs146/assets/files/langmod.pdf`.

[cwl] *Common Workflow Language*. `https://www.commonwl.org/`, besucht: 2018-12-07.

[Don16] Patrick Joseph Donnelly: *DATA LOCALITY TECHNIQUES IN AN ACTIVE CLUSTER FILE SYSTEM DESIGNED FOR SCIENTIFIC WORKFLOWS*. Dissertation, University of Notre Dame, 2016. `http://ccl.cse.nd.edu/research/papers/pdonnelly-thesis.pdf`.

[DTPC+15] Paolo Di Tommaso, Emilio Palumbo, Maria Chatzou, Pablo Prieto, Michael L. Heuer und Cedric Notredame: *The impact of Docker containers on the performance of genomic pipelines*. PeerJ, 3:e1273, #sep# 2015, ISSN 2167-8359. `https://doi.org/10.7717/peerj.1273`.

[Ell] Alex Ellis: *Builder pattern vs. Multi-stage builds in Docker*. `https://blog.alexellis.io/mutli-stage-docker-builds/`, besucht: 2018-11-30.

[Erb12] Benjamin Erb: *Concurrent Programming for Scalable Web Architectures*. Diploma Thesis, Institute of Distributed Systems, Ulm University, April 2012. `http://www.benjamin-erb.de/thesis`.

[GKL18] Michael Gref, Joachim Köhler und Almut Leh: *Improved Transcription and Indexing of Oral History Interviews for Digital Humanities Research*. In: *LREC*, 2018. http://www.lrec-conf.org/proceedings/lrec2018/pdf/137.pdf.

[gra] *Decoding graph construction in Kaldi: A visual walkthrough*. http://vpanayotov.blogspot.com/2012/06/kaldi-decoding-graph-construction.html, besucht: 2019-01-17.

[GS12] Urs Gleim und Tobias Schüle: *Multicore-Software: Grundlagen, Architektur und Implementierung in C/C , Java und C#*. Dpunkt, 2012.

[IAI18] Fraunhofer IAIS: *Audio Mining*, 2018. https://www.iais.fraunhofer.de/de/geschaeftsfelder/content-technologies-and-services/uebersicht/AudioMining.html, besucht: 2018-07-18.

[Inc18] Docker Inc.: *Use multi-stage builds*, 2018. https://docs.docker.com/develop/develop-images/multistage-build/, besucht: 2018-11-30.

[JA17] S. M. Jaybhaye und V. Z. Attar: *A review on scientific workflow scheduling in cloud computing*. In: *2017 2nd International Conference on Communication and Electronics Systems (ICCES)*, Seiten 218–223, Oct 2017.

[KCDK] Mikolaj Kowalik, Hsin Fang Chiang, Greg Daues und Rob Kooper: *DMTN-025: A survey of workflow management systems*. https://dmtn-025.lsst.io/.

[KH17] Elena Kakoulli und Herodotos Herodotou: *OctopusFS: A Distributed File System with Tiered Storage Management*. Seiten 65–78, Mai 2017.

[Klo18] Isaac Klop: *Containerized Workflow Scheduling*. Technischer Bericht, University of Amsterdam, 2018. https://homepages.staff.os3.nl/~delaat/rp//2017-2018/p71/report.pdf.

[KR] Matthias Korch und Thomas Rauber: *A Comparison of Task Pools for Dynamic Load Balancing of Irregular Algorithms*. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.90.8205&rep=rep1&type=pdf.

[Kra] Dr. Klaus Dieter Krannich: *Grundlagen des parallelen Programmierens - Effizienz*. https://www.math.tu-cottbus.de/~kd/parallel/vorl/vorl/node20.html, besucht: 2019-02-12.

[LAG+16] Chee Sun Liew, Malcolm P. Atkinson, Michelle Galea, Tan Fong Ang, Paul Martin und Jano I. Van Hemert: *Scientific Workflows: Moving Across Paradigms*. ACM Comput. Surv., 49(4):66:1–66:39, Dezember 2016, ISSN 0360-0300. http://doi.acm.org/10.1145/3012429.

[LSL+14] C. Liu, M. Shie, Y. Lee, Y. Lin und K. Lai: *Vertical/Horizontal Resource Scaling Mechanism for Federated Clouds*. In: *2014 International Conference on Information Science Applications (ICISA)*, Seiten 1–4, May 2014.

[Mac] *Machine Learning im Kubernetes-Cluster*. `https://www.heise.de/developer/artikel/Machine-Learning-im-Kubernetes-Cluster-4226233.html?seite=3`, besucht: 2018-12-27.

[MK17] N. Mohanapriya und G. Kousalya: *Execution of workflow applications on cloud middleware*. In: *2017 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS)*, Seiten 1–6, March 2017.

[Mor16] Akimasa Morihata: *Incremental Computing with Abstract Data Structures*. In: Oleg Kiselyov und Andy King (Herausgeber): *Functional and Logic Programming*, Seiten 215–231, Cham, 2016. Springer International Publishing, ISBN 978-3-319-29604-3.

[MPR02] Mehryar Mohri, Fernando Pereira und Michael Riley: *Weighted Finite-state Transducers in Speech Recognition*. Computer Speech Language, 2002. `https://cs.nyu.edu/~mohri/pub/csl01.pdf`.

[MTT18] F. Marozzo, D. Talia und P. Trunfio: *A Workflow Management System for Scalable Data Mining on Clouds*. IEEE Transactions on Services Computing, 11(3):480–492, May 2018, ISSN 1939-1374.

[MvL] Simon Moser und Tammo van Lessen: *Developing, Deploying and Running a Hello World BPEL Process with the Eclipse BPEL Designer and Apache ODE*. `http://people.apache.org/~vanto/HelloWorld-BPELDesignerAndODE.pdf`.

[NEKH+18] Jon Ander Novella, Payam Emami Khoonsari, Stephanie Herman, Daniel Whitenack, Marco Capuccini, Joachim Burman, Kim Kultima und Ola Spjuth: *Container-based bioinformatics with Pachyderm*. bioRxiv, 2018. `https://www.biorxiv.org/content/early/2018/04/20/299032`.

[opea] *OpenFst Examples*. `http://www.openfst.org/twiki/bin/view/FST/FstExamples`, besucht: 2019-01-17.

[opeb] *OpenFst Library*. `http://www.openfst.org/twiki/bin/view/FST/WebHome`, besucht: 2019-01-17.

[pac] *The Data Science Bill of Rights*. `https://www.pachyderm.io/dsbor.html`, besucht: 2019-01-02.

[PK87] C. D. Polychronopoulos und D. J. Kuck: *Guided Self-Scheduling: A Practical Scheduling Scheme for Parallel Supercomputers*. IEEE Transactions on Computers, C-36(12):1425–1439, Dec 1987, ISSN 0018-9340.

[QCW16] R. Qasha, J. Cała und P. Watson: *A framework for scientific workflow reproducibility in the cloud*. In: *2016 IEEE 12th International Conference on e-Science (e-Science)*, Seiten 81–90, Oct 2016.

[QF12] Jun Qin und Thomas Fahringer: *Scientific Workflows Programming, Optimization, and Synthesis with ASKALON and AWDL*. Springer Berlin Heidelberg, 2012.

[rab] *Rabix - Reproducible Analysis for Bioinformatics*. `http://rabix.io/`, besucht: 2018-12-07.

[rea] *reana - Reproducible research data analysis platform*. `http://www.reanahub.io/`, besucht: 2018-12-07.

[RG07] Thomas Rauber und Rünger Gudula: *Parallele Programmierung*. Springer Vieweg, 2007, ISBN 978-3-540-46549-2.

[RvvdAtH16] Nick Russell, Wil M.P. van van der Aalst und Arthur H. M. ter Hofstede: *Workflow Patterns: The Definitive Guide*. The MIT Press, 2016, ISBN 0262029820, 9780262029827.

[RW17] Thomas Richter und Thomas Wick: *Einführung in die Numerische Mathematik*. Springer Berlin Heidelberg, 2017. `https://doi.org/10.1007/978-3-662-54178-4`.

[SBG+18] Alex Szalay, Julian Bunn, Jim Gray, Ian Foster und Ioan Raicu: *The Importance of Data Locality in Distributed Computing Applications*, 2018. `http://pcbunn.cithep.caltech.edu/Workflows-NSF-Szalay.htm`, besucht: 2018-12-07.

[SCC+18] Ola Spjuth, Marco Capuccini, Matteo Carone, Anders Larsson, Wesley Schaal, Jon Novella, Paolo Di Tommaso, Cedric Notredame, Pablo Moreno, Payam E Khoonsari, Stephanie Herman, Kim Kultima und Samuel Lampa: *Approaches for containerized scientific workflows in cloud environments with applications in life science*. August 2018. `https://peerj.com/preprints/27141.pdf`.

[sii] *Data Provenance*. `http://siis.cse.psu.edu/provenance.html`, besucht: 2019-01-04.

[SKD10] M. Sonntag, D. Karastoyanova und E. Deelman: *Bridging the Gap between Business and Scientific Workflows: Humans in the Loop of Scientific Workflows*. In: *2010 IEEE Sixth International Conference on e-Science*, Seiten 206–213, Dec 2010.

[Sof97] EAGLES SWLG SoftEdition: *Absolute discounting and backing-off*, May 1997. `http://wwwhomes.uni-bielefeld.de/gibbon/Handbooks/gibbon_handbook_1997/node217.html`.

[Sol15] Yan Solihin: *Fundamentals of Parallel Architecture*. Chapman and Hall/CRC, 2015. https://www.crcpress.com/Fundamentals-of-Parallel-Multicore-Architecture/Solihin/p/book/9781482211184.

[ST18] Kyle M. D. Sweeney und Douglas Thain: *Efficient Integration of Containers into Scientific Workflows*. In: *Proceedings of the 9th Workshop on Scientific Cloud Computing*, ScienceCloud'18, Seiten 7:1–7:6, New York, NY, USA, 2018. ACM, ISBN 978-1-4503-5863-7. http://doi.acm.org/10.1145/3217880.3217887.

[Sta17] Dustin Stansbury: *Understanding Apache Airflow's key concepts*, 2017. https://medium.com/@dustinstansbury/understanding-apache-airflows-key-concepts-a96efed52b1a, besucht: 2018-12-07.

[Tay07] Ian J. Taylor: *Workflows for e-Science: scientific workflows for grids*. Springer-Verlag, 2007, ISBN 978-1-84628-757-2.

[Teh] Yee Whye Teh: *A Bayesian Interpretation of Interpolated Kneser-Ney NUS School of Computing Technical Report TRA2/06*. http://www.stats.ox.ac.uk/~teh/research/compling/hpylm.pdf.

[Wan17] Chun Kun Wang: *Selection of Parallel Runtime Systems for Tasking Models*. 2017 International Conference on Computational Science and Computational Intelligence, 2017.

[wdl] *The formal WDL language specification*. https://software.broadinstitute.org/wdl/documentation/spec#introduction, besucht: 2018-12-07.

[Wri13] Howard Wright: *Phonetic audio mining, audio searching, speech analytics*, 2013. http://www.hakwright.co.uk/audio-mining.html, besucht: 2018-07-18.

[ZTT17] Chao Zheng, Ben Tovar und Douglas Thain: *Deploying High Throughput Scientific Workflows on Container Schedulers with Makeflow and Mesos*. 2017. http://ccl.cse.nd.edu/research/papers/makeflow-mesos-ccgrid17.pdf.