

HOCHSCHULE ...

STUDIENGANG ...

Masterthesis

**Aufbau einer Plattform zur
forensischen Analyse basierend auf dem
Apache Hadoop[®] Framework**

Zur Erlangung des akademischen Grades
Master of Science

vorgelegt im Sommersemester 2018

von
Johannes Busam

Erstbetreuung: ...

Zweitbetreuung: ...

Inhaltsverzeichnis

1	Einleitung	3
1.1	Allgemeines	3
1.2	Problemstellung	3
1.3	Zielsetzung	4
1.4	Weitere Aspekte der Thesis	5
1.5	Aufbau	6
2	Vorgehen	7
2.1	Projektplanung	7
2.2	Entwicklungsumgebung	12
2.3	Testdatengenerierung	14
3	Grundlagen von Apache Hadoop®	15
3.1	Hadoop® Framework	15
3.2	Hadoop HDFS	17
3.3	Hadoop YARN	19
3.4	Apache HBASE	21
3.5	Apache Spark	23
4	Aufbau einer Analyse-Plattform	24
4.1	Allgemeines	24
4.2	Datenspeicherung im HDFS	24
4.2.1	Traditionelles Analyseverfahren	24
4.2.2	Umsetzung in der Hadoop Analyse-Plattform	25
4.3	Fachliche Probleme bei den Daten	29
4.3.1	Symbolische Links	29
4.3.2	Zugriffsrechte	29
4.4	Datenverarbeitung mit Apache Spark™	30
4.4.1	Praxisbeispiele und deren Optimierungen	31
4.5	Forensische Anforderungen	33
4.5.1	Plattform absichern	33
4.6	Visualisierung der Ergebnisse	34
5	Zusammenfassung	35
6	Ausblick	36

A	Anhang A	41
A.1	Analyse ähnlicher Projekte und Produkte	41
A.2	Lizenzierungen in dieser Arbeit	42
B	Hadoop Konfigurationen	43
B.1	Aufsetzen des aktuellen Hadoop-Frameworks	43

1 Einleitung

1.1 Allgemeines

1.2 Problemstellung

Die forensische Analyse von digitalen Beweismitteln ist in der heutigen Zeit ein wichtiger Aspekt, um in der Strafverfolgung rechtswidriges Verhalten aufzudecken oder nachzuweisen. In vielen Fällen werden informationstechnische Systeme am Tatort gefunden oder zur Tatbegehung genutzt. Einschlägig sind hierbei Angriffe auf kritische Infrastrukturen durch Computersabotage oder das Ausspähen von Daten. Aber auch Urheberrechtsverletzungen durch die Weitergabe von geschützten Medien oder Verstöße gegen das Wettbewerbsrecht werden mit Informationstechnik begangen. Je nach Dauer und Umfang der Strafhandlung werden gerade auch im Bereich der Wirtschaftskriminalität dutzende Beweismittel von informationstechnischen Systemen erhoben. Beispielsweise werden beteiligte Computer und Mobiltelefone sichergestellt. Oder es werden logische Sicherungen von Netzwerkspeichern durchgeführt.

Bei der Analyse dieser Beweismittel möchte ein forensischer Ermittler möglichst schnell einen Überblick über die sichergestellten Daten erhalten. Darauf aufbauend kann er entscheiden, welche Spuren in den Daten zum Nachweis konkreter Tathandlungen dienen und welche potentielle Beweismittel nicht weiter analysiert werden müssen.

Der kritischste Aspekt hierbei ist, in kürzester Zeit die richtigen Informationen aus allen Daten zu extrahieren. Denn gerade in der Strafverfolgung ist eine schnelle und zielgerichtete Aufarbeitung der Ermittlungsfälle erforderlich. Darüber hinaus werden während der Analyse oftmals weitere Indizien gefunden, welche wiederum zur Sicherung neuer Beweismittel führen können. Je mehr Zeit jedoch für die Analyse benötigt wird, desto höher ist die Gefahr, dass noch nicht sichergestellte Daten endgültig gelöscht werden. Beispielsweise werden Telekommunikationsverbindungsdaten nicht über längere Zeiträume gespeichert.

Zur Analyse stehen dem Forensiker etliche kommerzielle und Open Source Programme zur Auswahl. Allerdings sind im forensischen Open Source Bereich viele Programme durch die Ressourcen des Analyserechners beschränkt. Sie bieten keine Möglichkeiten rechenintensive Aufgaben performant auf mehreren Computern zu skalieren.

Aus fachlicher Sicht wäre eine Plattform sinnvoll, die anfallende Analyseaufgaben automatisiert auf allen Daten durchführt. Das System sollte die Ergebnisse unter Berücksichtigung verfügbarer Ressourcen schnellstmöglich ermitteln und dem forensischen Ermittler in einer aufbereiteten Form darstellen. Auf Basis dieser Ergebnisse könnte sich der Forensiker möglichst frühzeitig einen Überblick aller Beweismittel verschaffen, um dann bestimmte Daten

und Informationen auch in anderen spezialisierten Analysetools weiterzuverarbeiten.

1.3 Zielsetzung

Zur Lösung der Problemstellung soll in dieser Masterthesis eine Plattform zur forensischen Analyse entwickelt werden. Diese Plattform soll durch eine automatisierte Analyse und Aufbereitung forensisch relevanter Informationen dem Nutzer helfen, sich einen Überblick zu verschaffen. Der Forensiker soll dadurch effizient und zielgerichtet Datenanalysen durchführen können. Als Basis dieser Plattform soll das Apache Hadoop[®] Framework genutzt werden. Hierbei sollen Vor- und Nachteile dieser Art der Datenverarbeitung im forensischen Kontext herausgearbeitet werden.

Apache Hadoop ist ein etabliertes Open Source Framework zur verteilten Speicherung und Verarbeitung von Daten. Durch die Verwendung paralleler Algorithmen eignet sich ein Hadoop-Cluster für große Datenmengen im Terabyte-Bereich. Ein zugrunde liegendes Paradigma ist hierbei, dass die Programmausführung dort stattfindet wo auch die Daten liegen, um kostspielige Datentransporte weitgehend zu vermeiden. Aufgrund dieser Beschaffenheit könnte diese Art der Datenverarbeitung auch Geschwindigkeitsvorteile bei forensischen Analysen bieten. Das Framework selbst besteht aus mehreren Komponenten, welche spezifische Aufgaben der Datenverarbeitung übernehmen.

Die Basis bildet das verteilte Dateisystem *HDFS*¹, welches die Daten redundant auf allen Knoten des Computer-Clusters speichert. Der Zugriff auf die Daten kann über unterschiedliche Komponenten erfolgen.

So findet beispielsweise meist Apache SparkTM bei der Prozessierung und Analyse der Daten Anwendung. Auch in der Masterthesis soll Apache Spark für anfallende Analysezwecke verwendet werden. Hierbei übernimmt der Ressourcenmanager *YARN*² die Bereitstellung und Verteilung von verfügbarer Rechenleistung.

Ein Knackpunkt der Masterthesis ist die Aufbereitung der Daten für die Analyse im Hadoop-Cluster. Beispielsweise könnten die Dateien von sichergestellten Datenträgern direkt in das HDFS kopiert werden. Hierbei muss auf die Unversehrtheit der Dateiinhalte und Metadaten beim Kopieren geachtet werden.

Im Rahmen dieser Thesis soll die reine Datenanalyse vorerst auf grundlegende Operationen basieren. Unter anderem sollen beispielsweise folgende Informationen zu einzelnen Dateien ermittelt werden: Name, vollständiger Pfad, Hashsumme, Dateityp, Größe, Zeitpunkt der letzten Änderung und Erstellung.

Es soll auch eine Volltextsuche auf den Daten möglich sein. Darauf aufbauend soll der Nutzer beispielsweise gleiche Dateien und Verbindungen zwischen den einzelnen Beweismitteln erkennen können.

Optional könnte die Analyseplattform gezielt nach IP-Adressen, URLs, E-Mail-Adressen oder Positionsdaten³ suchen. Diese Operationen sollen erst implementiert werden, wenn im Rahmen der Thesis noch weitere Bearbeitungszeit vorhanden ist. Sie sind vorerst nicht

¹HDFS ist die Abkürzung für *Hadoop Distributed File System*.

²YARN ist die Abkürzung für *Yet Another Resource Negotiator*.

³Beispielsweise könnten Geopositionen oder Ortsnamen aus Dateien extrahiert werden. Diese Daten könnten dann mit ihrem geografischen Bezug auf einer Karte dargestellt werden.

Gegenstand der Thesis.

Die Resultate durchgeführter Datenanalysen sollen dem Nutzer bereitgestellt werden. Hierzu soll eine prototypische Implementierung entwickelt werden, deren grafische Oberfläche die fachlichen Aspekte der forensischen Analyse widerspiegelt. Der Forensiker soll Analyseaufgaben konfigurieren und starten können. Nach der Prozessierung soll er die Resultate der Analysen direkt einsehen können.

Allerdings ist es nicht das Ziel dieser Thesis, detaillierte Konfigurationsmöglichkeiten und unterschiedlichste Visualisierungen zu implementieren. Dies würde den Rahmen der Arbeit übersteigen. Der Fokus dieser Fachanwendung liegt bei der schlichten Anzeige der Analyseergebnisse. In diesem Kontext soll auch geprüft werden, ob existierende Programme zur Datenvisualisierung im Hadoop-Umfeld wiederverwendet werden können.

Nachfolgende Abbildung soll den groben Aufbau dieser Plattform skizzieren. Der Forensiker importiert die forensischen Rohdaten in das Hadoop-Cluster. Darauf hat er die Möglichkeit diverse Analysen auf den Daten durchzuführen. Zuletzt kann er die Ergebnisse über eine entsprechende Oberfläche einsehen und hat die Möglichkeit die Daten innerhalb oder außerhalb des Hadoop-Clusters weiterzuverarbeiten.

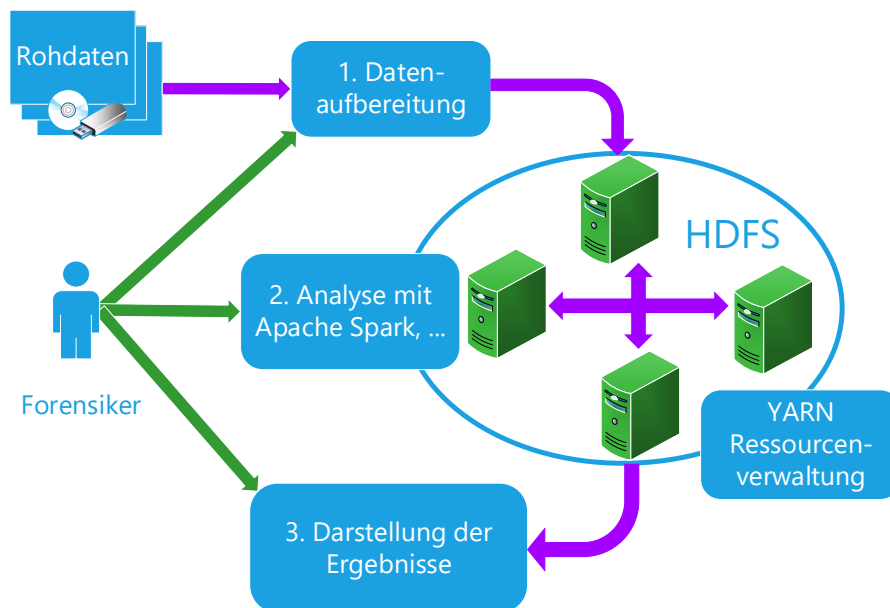


Abbildung 1.1: Datenverarbeitung im Hadoop-Umfeld

Das Ziel dieser Masterthesis ist es, dem forensischen Ermittler schnellstmöglich einen Überblick zu den einzelnen Beweismitteln und deren Zusammenhänge im Kontext einer Fallanalyse zu liefern.

1.4 Weitere Aspekte der Thesis

Bei einer realen forensischen Analyse gibt es weitere Anforderungen, die das Analysesystem erfüllen sollte. Im Rahmen der Masterthesis soll geprüft werden, ob diese Anforderungen

durch die Nutzung des Apache Hadoop Frameworks abgedeckt werden und welche technischen oder organisatorischen Regelungen getroffen werden müssen.

Das System muss gegen fremden Zugriff gesichert sein. Es muss zu jeder Zeit ersichtlich sein, welche Personen zu welchem Zweck auf das System zugreifen.

Da in vielen Fällen hochsensible personenbezogene Daten und Geschäftsgeheimnisse verarbeitet werden, müssen auch entsprechende Regelungen getroffen werden, wie nach der Analyse alle Daten restlos aus dem System gelöscht werden können.

Ein weiterer Aspekt in der Analyse ist die lückenlose Erstellung einer Beweismittelkette (Chain of Custody). Für jedes forensische Analyseergebnis müssen die Herkunft und die Verarbeitungsschritte transparent nachvollziehbar sein.

Auch die Korrektheit der Analyseergebnisse muss verifiziert werden. Hierfür sollen im Rahmen der Masterthesis entsprechende Testdaten erstellt oder beschafft werden, welche die Funktionsfähigkeit der Plattform prüfen.

Aus organisatorischer Sicht soll die Analyseplattform als Open Source Projekt bereitgestellt werden. Hierzu soll der Source-Code, die Konfiguration des Systems und die Dokumentation in einem öffentlich zugänglichen Repository verfügbar sein.⁴

1.5 Aufbau

In Kapitel 1 wird die grundlegende Problemstellung bei der forensischen Analyse beschrieben, welche diese Masterthesis lösen soll. Darauf folgt die Zielsetzung der Thesis, welche als möglicher Lösungsvorschlag zur beschriebenen Problemstellung gilt.

In Kapitel 2 folgt das allgemeine Entwicklungsvorgehen. Darin ist auch der aktuelle Projektplan enthalten, welche die Arbeitspakete definiert. Zusätzlich wird der Umgang mit Quellcode und Konfigurationsdateien als Open-Source Projekt beschrieben.

In Kapitel 3 erfolgt eine Darstellung der Apache Hadoop Plattform inklusive theoretischen Grundlagen zur Arbeitsweise des Frameworks. Des Weiteren werden darauf aufbauend Projekte Apache Spark, Apache Hive und Apache HBase und deren Einsatzbereiche erläutert.

In Kapitel 4 werden die angewendeten Tools für eine herkömmliche fachliche Analyse eines Beweismittels beschrieben. Parallel hierzu wird bei jedem herkömmlichen Programm geprüft, wie das gleiche Ergebnis mit der Analyseplattform erzielt werden kann. Diese Kapitel soll sozusagen die Überleitung von der herkömmlichen Analyse auf einem Computer hin zu Analyse im Hadoop-Cluster beschreiben. Als Ergebnis soll anschaulich dargestellt werden, welche fachliche Analyse-Schritte mithilfe eines Hadoop-Cluster sinnvoll und performant durchgeführt werden können.

Zuletzt erfolgt in Kapitel 5 eine Zusammenfassung der erarbeiteten Ergebnisse. Offene Punkte und Verbesserungen des Systems werden in Kapitel 6 diskutiert.

⁴Sicherheitskritische Informationen, wie beispielsweise Zugangsdaten, müssen unkenntlich gemacht werden.

2 Vorgehen

2.1 Projektplanung

Abbildung 2.1 zeigt die Aufteilung Masterthesis in einzelne Arbeitspakete. Das Ziel der Einarbeitungsphase ist ein grundlegendes Verständnis über die Datenverarbeitung im Hadoop-Framework zu erhalten. Zusätzlich soll eine Entwicklungsumgebung inklusive öffentlicher Versionsverwaltung eingerichtet werden. Darauf erfolgt der Aufbau eines eigenen Hadoop-Clusters und die Beschaffung von Testdaten.¹ Für die Einarbeitung und den Aufbau sind vier Wochen eingeplant (siehe Abbildung 2.2).²

Der zweite Teil behandelt die Rohdatenspeicherung im HDFS und eine Datenaufbereitung. Es soll geprüft werden, welche Struktur der Daten für eine optimale Speicherung und Verarbeitung im Hadoop-Framework erforderlich ist. Dieser Teil beansprucht abermals vier Wochen. Am Ende dieses Arbeitspaketes soll ein erster Zwischenbericht erstellt werden, welcher die bisherigen Ergebnisse enthält (siehe Abbildung 2.2).

Nach der Speicherung der Rohdaten erfolgt im dritten Arbeitspaket die Datenanalyse mit Apache Spark. Hier sollen die Daten nach anwendungsbezogenen Problemstellungen analysiert werden. Das Ergebnis ist eine Sammlung von Programmen, welche mit Apache Spark auf den Daten ausgeführt werden können. Darüber hinaus soll ermittelt werden, welche Möglichkeiten zur Ausführung dieser Spark-Anwendungen bestehen.³ Ein weiterer Aspekt der Datenanalyse beschäftigt sich mit den Möglichkeiten, wie die Ergebnisse persistiert werden können.⁴ Im Anschluss soll die Performanz der Algorithmen geprüft werden. Hier bietet sich der Vergleich zu herkömmlichen Analyseprogrammen an. Denn schließlich hat diese Thesis auch das Ziel, bei großen Datenmengen schneller Ergebnisse zu liefern als die herkömmlichen Analysewerkzeuge auf einem einzelnen Analyserechner. Für dieses Arbeitspaket sind sieben Wochen eingeplant (siehe Abbildung 2.3). Darauf folgt ein zweiter Zwischenbericht.

Im letzten Drittel der Masterthesis sollen die querschnittlichen Aspekte in der bestehenden Datenverarbeitung berücksichtigt werden. Hierbei geht es um das Absichern der Analyseplattform, die Dokumentation der Chain of Custody und das Löschen von nicht mehr verwendeten personenbezogenen Daten. Für dieses Arbeitspaket sind vier Wochen einge-

¹Auch ein Zugriff auf einen bestehendes Hadoop-Cluster ist möglich.

²Die referenzierten Gantt-Diagramme wurden mit der JavaScript-Bibliothek *dhtmlxGantt* erstellt. Der Quellcode ist unter der *GNU GPLv2*-Lizenz lizenziert. Weiter Informationen können in Kapitel A.2 im Anhang nachgelesen werden.

³Hier könnte beispielsweise das Projekt Apache Livy nützlich sein.

⁴Hier könnten die Projekte Apache Hive und Apache HBase zur Speicherung von strukturierten und unstrukturierten Daten untersucht werden.

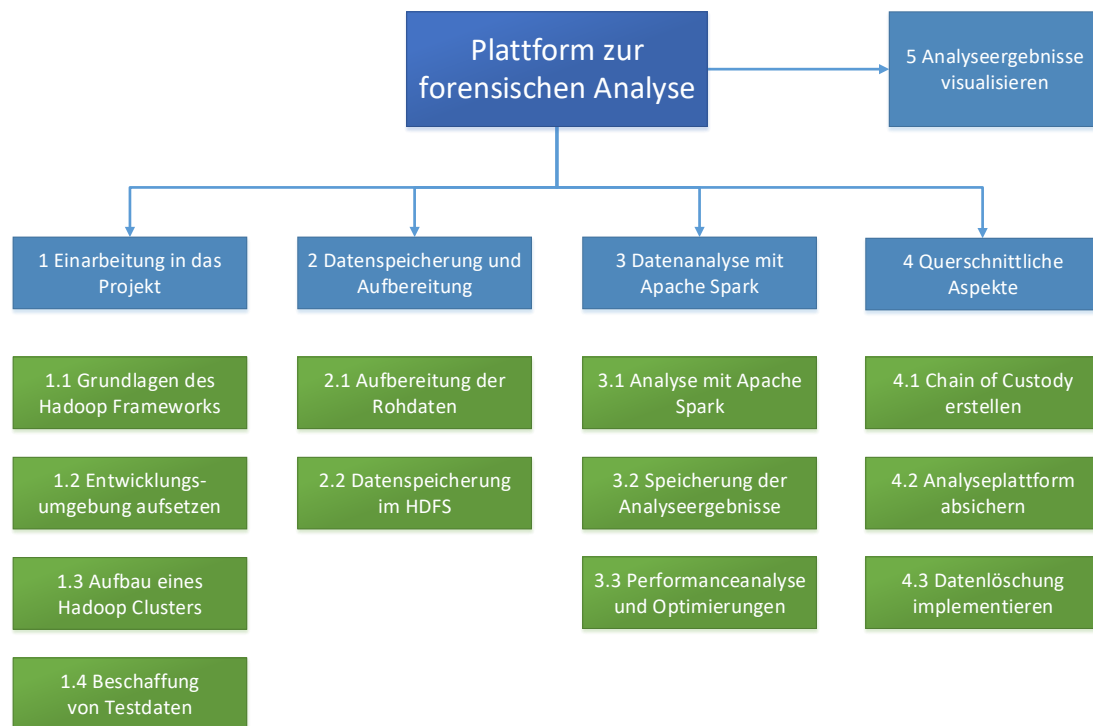


Abbildung 2.1: Arbeitspakete der Masterthesis

plant (siehe Abbildung 2.4).

Das letzte Arbeitspaket enthält eine prototypische Visualisierung der Analyseergebnisse. Hierbei soll geprüft werden, welche Möglichkeiten zur Darstellung der Ergebnisse existieren. Der Forensiker soll auf möglichst einfache Art und Weise die Ergebnisse ansehen können. Für diese Arbeit sind drei Wochen eingeplant (siehe Abbildung 2.4).

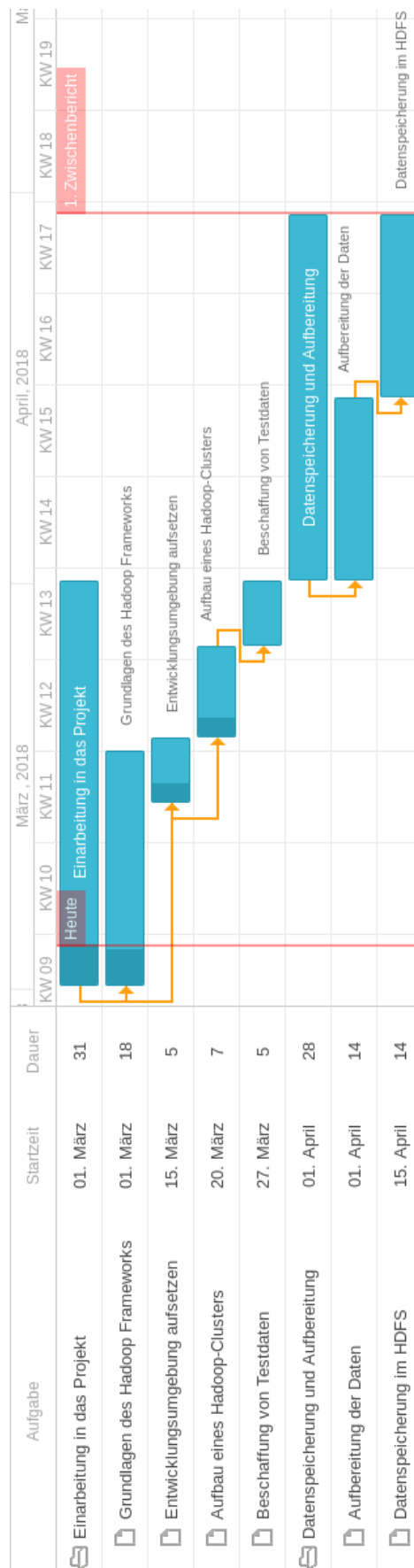


Abbildung 2.2: Projektplan Teil A - Einarbeitung und Rohdatenspeicherung (siehe Kapitel A.2)

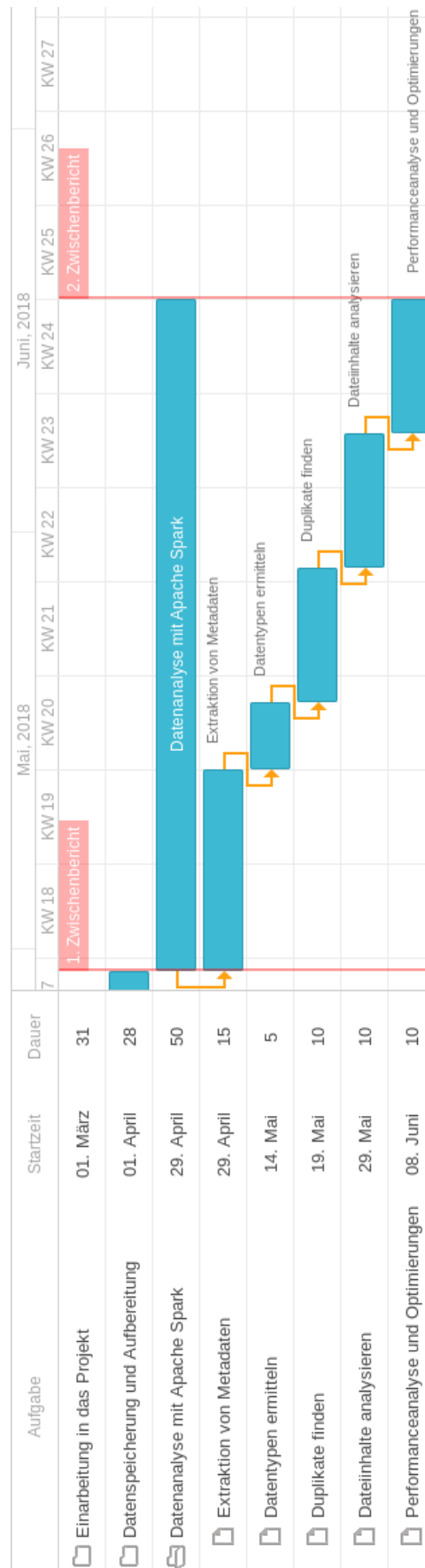


Abbildung 2.3: Projektplan Teil B - Datenanalyse (siehe Kapitel A.2)

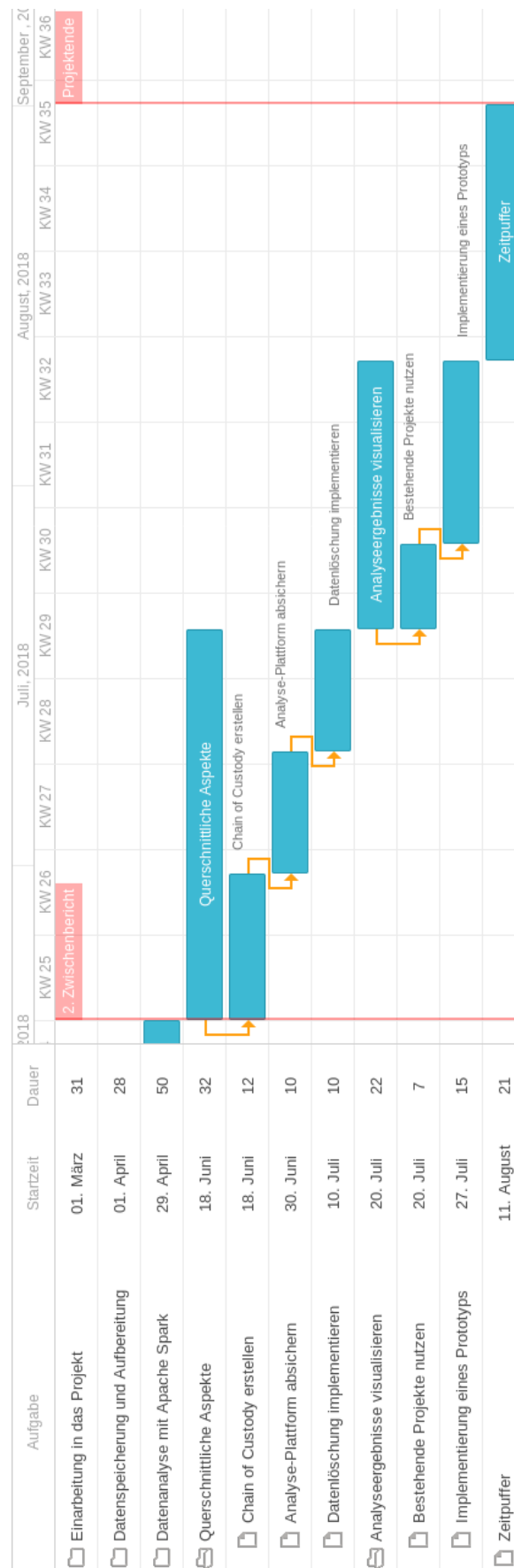


Abbildung 2.4: Projektplan Teil C - Querschnittliche Aspekte und Visualisierung (siehe Kapitel A.2)

2.2 Entwicklungsumgebung

Der Aufbau einer Test- und Entwicklungsumgebung ist ein wichtiger Bestandteil dieser Thesis. Einerseits sollen Anwendungsprogramme zur Datenverarbeitung schnell und lokal ausführbar sein. Andererseits soll die Testumgebung auf einem physikalischen Apache Hadoop Cluster basieren, um mögliche Infrastrukturprobleme identifizieren zu können und die Performanz zu testen.

Abbildung 2.5 skizziert die Komponenten der Entwicklungsumgebung. Zentraler Bestandteil ist ein Entwicklungsrechner mit der Linux-Distribution *Fedora* in der Version 27 64-bit.

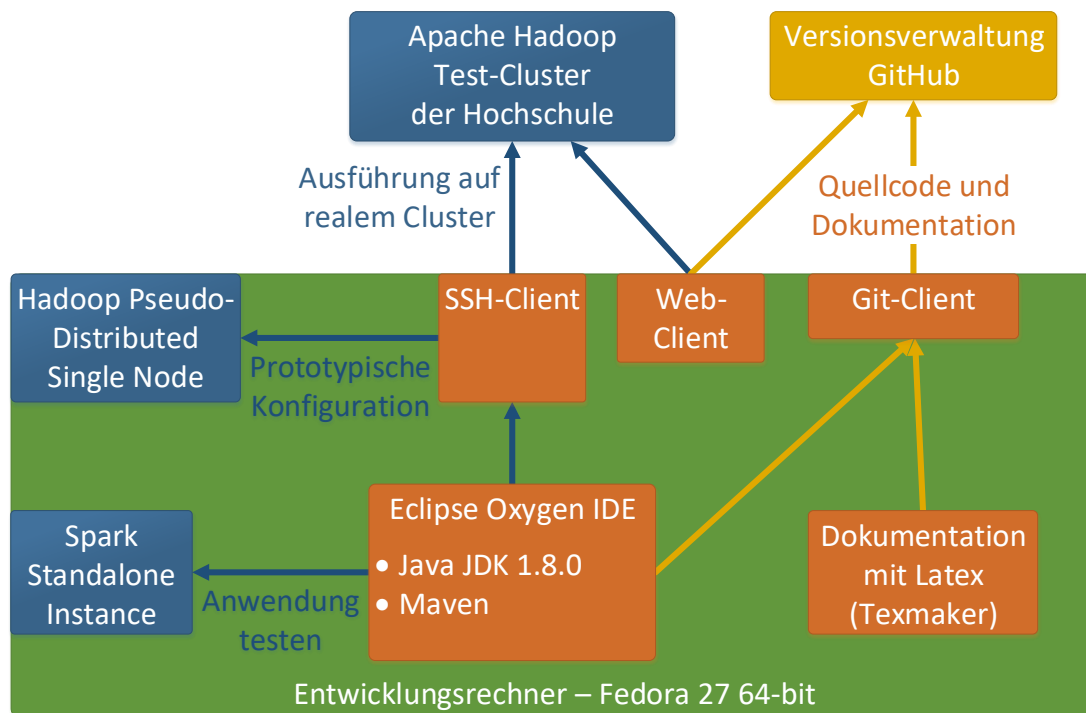


Abbildung 2.5: Komponenten der Entwicklungsumgebung

Zur Entwicklung der forensischen Analyseprogramme wird *Eclipse Oxygen* genutzt. Die Anwendungen selbst werden in Java geschrieben.⁵ Zum Bauen der ausführbaren Java-Archive (JAR-Dateien) wird *Maven* verwendet. Mit Maven können weitere Java-Bibliotheken in eigenen Programmen auf einfache Weise wiederverwendet werden.⁶

Um die gebauten Java-Programme zur Datenanalyse schnell testen zu können, kann auf dem lokalen Entwicklungsrechner eine Apache Spark Standalone Instanz gestartet werden. Diese dient ausschließlich zur simplen Ausführung von Spark-Applikationen. Hierbei arbeitet die Instanz direkt auf dem lokalen Dateisystem und nutzt kein HDFS. Darüber hinaus wird nicht YARN, sondern ein bei Apache Spark mitgelieferter Ressourcenmanager genutzt.⁷

⁵Wobei auch Python oder Scala als Programmiersprache genutzt werden kann.

⁶Diese können über ein zentrales Repository, dem sogenannten *Maven Central Repository* aus dem Internet geladen werden (siehe <https://search.maven.org/>).

⁷Siehe Kapitel 3 für eine detaillierte Erklärung von Apache Hadoop.

Analog zur Spark Standalone Instanz kann auch ein Hadoop Pseudo-Distributed Single Node auf dem lokalen Rechner gestartet werden.⁸ Mithilfe diese Single-Nodes können spezifische Konfiguration des HDFS oder des Ressourcenmanagers YARN ausprobiert werden. Letztendlich kommen diese lokale Hadoop und Spark Instanzen aber schnell an ihre Grenzen. Daher werden spezifische Konfigurationen und fertiggestellte Analyseprogramme auch auf einem realen Apache Hadoop Test-Cluster durchgeführt. Dort kann das Zusammenspiel zwischen Hadoop und Spark nachvollzogen werden. Auch entsprechende Last- und Performance-Tests sind nur auf dem Hadoop Test-Cluster sinnvoll.

Um mit dem Test-Cluster arbeiten zu können, wird ein SSH-Client benötigt. Zusätzlich gibt es auch eine Web-Oberfläche basierend auf Apache Ambari zur Konfiguration und Anzeige des aktuellen Systemzustandes.

Alle selbst erstellten Anwendungsprogramme, Konfigurationsdateien und die Dokumentation dieser Thesis sollen als Open-Source Projekte in einem öffentlichen Repository zugänglich sein. Aus fachlicher Sicht ist es gerade in der Forensik sehr wichtig dem Nutzer die Möglichkeit zu geben, den Quellcode der Analyseprogramme einsehen zu können und notfalls auf spezielle Bedürfnisse anzupassen. Darüber hinaus kann die Datenverarbeitung transparent nachvollzogen werden. Daher werden die einzelne Projekte mithilfe eines Git-Clients auf GitHub versioniert.

Nachfolgende Auflistung zeigt die Aufteilung der Projekte:

- Das Projekt *foam-thesis*⁹ enthält die schriftliche Ausarbeitung der Thesis und den Quellcode als Latex-Projekt. Als Entwicklungsumgebung wird *Texmaker* genutzt. Über den Link <https://github.com/jobusam/foam-thesis> ist der aktuelle Stand der Arbeit jederzeit einsehbar.¹⁰
- Das Projekt *foam-processing-spark* enthält den Quellcode zur Auswertung mit Apache SparkTM. Unter <https://github.com/jobusam/foam-processing-spark> befindet sich ein Maven-Projekt, welches wiederum die Java-Anwendung baut. Es werden auch entsprechende Skripte zum Starten von Spark-Anwendungen auf dem lokalen Rechner bereitgestellt.
- Das Projekt *foam-storage-hadoop* enthält alle Konfigurationsdateien zum Aufsetzen eines Hadoop-Cluster auf einem einzelnen Knoten im *Pseudo-Distributed Mode*.¹¹ Zusätzlich existieren Shell-Skripte zum Starten des Hadoop-Clusters auf einem einzelnen Knoten.¹² Das Hadoop-Cluster besteht nur aus dem HDFS und dem Ressourcenmanager YARN. Mithilfe der Skripte aus dem *foam-processing-spark* Projekt können damit Spark-Anwendungen auf einem lokalen Knoten innerhalb eines HDFS mit YARN ausgeführt werden.

⁸Hierfür muss der Entwicklungsrechner entsprechende Ressourcen bereitstellen. Es sollten mindestens eine Quad-Core-CPU, 16 GB Arbeitsspeicher und eine SSD zur Verfügung stehen, um halbwegs performant arbeiten zu können.

⁹Die Abkürzung *foam* oder auch *foAm* steht für **f**orensische **A**nalys**e**platt**f**orm

¹⁰Das kompilierte PDF-Dokument zum jeweiligen Stand wird im gleichen Projekt versioniert und ist unter dem Link <https://github.com/jobusam/foam-thesis/blob/master/main.pdf> verfügbar.

¹¹Siehe <https://github.com/jobusam/foam-storage-hadoop/tree/master/hadoop.standalone.configuration>

¹²Siehe <https://github.com/jobusam/foam-storage-hadoop/tree/master/hadoop.standalone.setup>

Derzeit ist die Lizenzierung beider Projekte noch nicht klar. Sehr wahrscheinlich wird die Thesis-Dokumentation unter der *GNU Free Documentation License (GFDL)* lizenziert, wohingegen der restliche Quellcode unter der *GNU Affero General Public License Version 3 (AGPLv3)* oder alternativ unter der Apache License 2.0 veröffentlicht werden soll. Es soll jedem möglich sein, den Quellcode einzusehen und nach belieben ändern zu können.

Aus organisatorischen Gründen, wird darauf geachtet, dass während der Ausarbeitungszeit der Thesis nur Änderung von dem Autor selbst in dem entsprechenden Repository gehostet werden.

2.3 Testdatengenerierung

TODO: Kapitel überarbeiten... Für den Aufbau einer forensischen Analyseplattform sollen entsprechende Testdaten generiert werden. Hierbei gibt es zwei unterschiedliche Fall-daten. Der erste Fall wäre ein kleines Image kleiner 10 GB. Dieses Image könnte für lokale Tests genutzt werden.

Im zweiten Fall müssen größere Images erzeugt werden, um ein passendes Szenario für das Hadoop Test-Cluster zu erzeugen. Der zweite Fall könnte 2 PCs und ein Server enthalten. Die Images der PCs sollten mindestens 100 GB groß sein, wohin gegen der Server 250 GB groß sein könnte. Ein zusätzlicher USB-Stick mit 64 GB Daten könnte auch dazu passen. Die Images dieser PCs und des Servers könnten virtualisiert erstellt werden. Folgende Applikationen könnten genutzt werden: Thunderbird, Firefox, Owncloud, Apache HTTP Server. Es soll herausgefunden werden, wann welche Daten zwischen den drei Rechnern ausgetauscht wurden.

Für lokale Tests und den Aufbau der Plattform bietet sich beispielsweise auch das Testscenario *Data Leakage Case* an, welches auf der Website *Computer Forensic Reference Data Sets* verfügbar ist.¹³

Diese Images könnten auf verdächtige Querverweise untersucht werden.

¹³Siehe https://www.cfreds.nist.gov/data_leakage_case/data-leakage-case.html, letzter Stand 28.3.2018.

3 Grundlagen von Apache Hadoop®

3.1 Hadoop® Framework

Apache Hadoop ist ein etabliertes Java-Framework zur verteilten Speicherung und Verarbeitung von Daten. Durch die parallele Ausführung von Algorithmen eignet sich ein Hadoop-Cluster für rechenaufwendige Datenanalysen. Ein primäres Paradigma ist das Konzept der *Datenlokalität*. Die auszuführenden Programme werden auf die Knoten verteilt, auf welchen auch die Daten liegen. Ressourcenintensive Datentransporte sollen weitgehend vermieden werden.[1, S. 20 ff.]

Das Framework ist für die Ausführung auf Standardhardware konzeptioniert. Es wird also keine verhältnismäßig teure Spezialhardware benötigt. Das Cluster besteht aus vielen einzelnen Knoten mit Standardhardware, welche im Verhältnis zu Spezialhardware günstiger und leicht ersetzbar ist. Der Ausfall einzelner Knoten ist die Regel und wird bei der Datenthaltung entsprechend berücksichtigt.

Das Apache Hadoop® selbst besteht aus mehreren Komponenten, welche spezifische Aufgaben übernehmen. Abbildung 3.1 stellt eine grobe Skizzierung der Komponentenlandschaft von Apache Hadoop dar.¹

Die Basis bildet das verteilte Dateisystem *Hadoop Distributed File System (HDFS)*, welches die Daten redundant auf allen Knoten des Computer-Clusters speichert. Hierbei besteht das Computer-Cluster selbst aus mehreren Knoten, auf welchen vorzugsweise ein Linux-Betriebssystem, wie beispielsweise CentOS, arbeitet.

Der Ressourcenmanager *YARN (Yet Another Resource Negotiator)* ist für die Verteilung und Bereitstellung von verfügbarer Rechenleistung verantwortlich.

Die dritte Komponente ist das *Hadoop® Map-Reduce Framework*. Hadoop Map-Reduce kann zur Datenverarbeitung genutzt werden. Hierbei werden Algorithmen parallel auf den Knoten prozessiert und die Ergebnisse im Anschluss zusammengetragen. Die einzelnen Zwischenergebnisse werden alle im HDFS abgelegt.²

Das verteilte Dateisystem HDFS und der Ressourcenmanager YARN bilden den Kern des Hadoop-Clusters. Darauf aufbauend können andere Komponenten die Daten verarbeiten

¹In der Abbildung 3.1 werden Logos der einzelnen Apache Projekte verwendet. Diese sind Handelsmarken der *Apache Source Foundation* (siehe <https://www.apache.org/>). In Kapitel A.2 im Anhang werden die Logos und deren Herkunft nochmals aufgelistet.

²Sogenannte Map-Reduce Jobs bildeten in den Anfängen von Hadoop den primären Weg, Daten verteilt zu verarbeiten. Mittlerweile wurde diese Art der Datenverarbeitung in den Hintergrund verdrängt, da andere Projekte, wie beispielsweise Apache Spark, die Daten schneller verarbeiten können oder andere Ansätze zur Verarbeitung nutzen. Dies ist beispielsweise auch der Grund, weshalb Hadoop Map-Reduce in dieser Masterthesis nicht explizit verwendet wird.

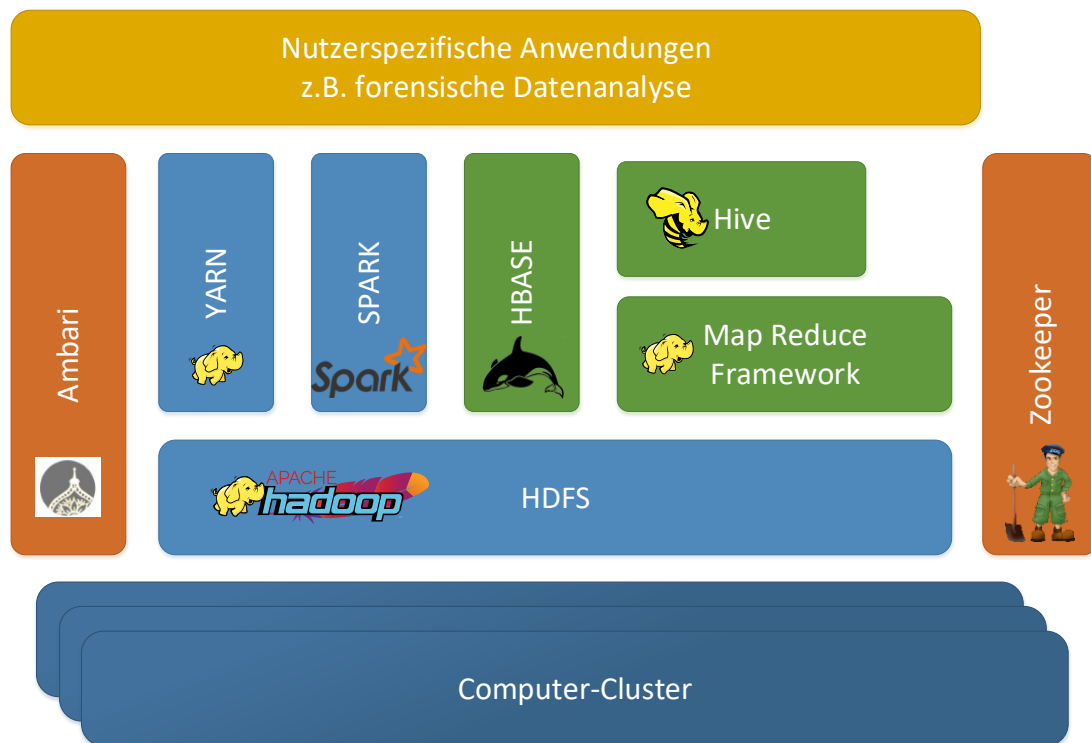


Abbildung 3.1: Apache Hadoop Ökosystem (Vgl. [1],[3]. Siehe Kapitel A.2)

oder spezielle Aufgaben durchführen.

So wird beispielsweise in dieser Thesis *Apache SparkTM* bei der Prozessierung und Analyse der Daten genutzt. Der Vorteil von Apache Spark ist eine performante Datenverarbeitung, da einerseits die Daten verteilt verarbeitet werden und andererseits Zwischenergebnisse und temporäre Daten im Arbeitsspeicher der einzelnen Rechenknoten gehalten werden.³ Desweiteren bietet *Apache HiveTM* eine Möglichkeit Dateien im HDFS mithilfe einer SQL ähnlichen Syntax⁴ abzufragen. Hierbei nutzt die Komponente wiederum das Map-Reduce Framework von Hadoop. Apache Hive ist jedoch keine reine Datenbank, sondern arbeitet auf den Dateien im HDFS.

Apache HBASE[®] hingegen ist eine spaltenorientierte Key-Value Datenbank. Sie wurde eigens für Apache Hadoop implementiert, um große Datenmengen performant zu speichern.

Das Hadoop-Ökosystem als Ganzes muss auch konfiguriert und überwacht werden. Um die Verfügbarkeit einzelner Instanzen zu gewährleisten und gegebenenfalls redundante Verarbeitungswege anzubieten, wird *Apache ZookeeperTM* genutzt. Mit Zookeeper ist es auch möglich Konfigurationen und Änderungen im Cluster zu verteilen. Zum eigentlichen konfigurieren und überwachen des Hadoop-Clusters wird *Apache AmbariTM* genutzt.

Zusätzlich existieren weitere Projekte im Hadoop-Ökosystem, welche für die forensische Analyseplattform von Verwendung sein können. Hierzu gehören:

³Durch das In-Memory Computing ist Apache Spark deutlich schneller als das bereits vorgestellte Hadoop Map-Reduce.

⁴Dem sogenannten HiveQL.

- *Apache Livy* zur Ausführung von Apache Spark Anwendung über eine REST-Schnittstelle.⁵
- *Apache NiFi* ermöglicht das Aufbereiten von Daten und organisiert Datenimporte.
- *Apache UIMATM* zur Analyse von unstrukturierten Daten, wie beispielsweise Texte und Mediadateien. **TODO: Kann UIMA überhaupt in Hadoop eingesetzt werden.**
- *Apache Accumulo[®]* als Alternative zu Apache HBASE?

Prinzipiell sind viele Komponenten unabhängig voneinander. So kann ein HDFS ausschließlich zur Datenhaltung aufgebaut werden, ohne eine Komponente zur Datenverarbeitung verwenden zu müssen. Umgekehrt lassen sich Komponenten zur Datenverarbeitung, wie Apache Spark, auch ohne das HDFS und YARN nutzen und könnten damit auch in andere Umgebungen integriert werden. Die einzelnen Komponenten entfalten jedoch gerade durch die Kombination miteinander ihre Potential zur performanten Datenanalyse.

Es gibt einige Unternehmen, die sich speziell darauf spezialisiert haben dieses Apache Hadoop Ökosystem und weiter noch nicht erwähnte Komponenten zu einzelnen Analyseplattformen zusammenzufassen. Sie bieten hierfür entsprechender kostenpflichtiger Support, wobei diese Plattformen im reinen Betriebe kostenfrei sind. So wird im Praxisteil der Masterthesis beispielsweise die *Hortonworks Data Platform (HDP)* des Unternehmens *Hortonworks* genutzt.

3.2 Hadoop HDFS

Das Hadoop Distributed Filesystem (HDFS) ist ein verteiltes Dateisystem, welches die Grundlage zu Speicherung von Daten im Hadoop-Ökosystem bietet. Nachfolgende Zwecke soll es erfüllen.

Es soll ausfallsicher sein. In der Standardkonfiguration wird jede Datei dreifach auf unterschiedlichen physikalischen Knoten gespeichert. Damit kann selbst bei einem Ausfall von zwei Knoten immer noch auf die Datei zugegriffen werden. Darüber hinaus verteilt das HDFS die Dateien automatisch und regeneriert sich selbst nach Ausfällen von Knoten. In großen Computer-Clustern mit mehreren hundert Knoten ist ein Ausfall eines Knoten kein Sonderfall sondern die Regel. Daher muss sich das HDFS selbst heilen können, um auch ohne manuelle Administration weiter verfügbar zu sein.

Das HDFS (und auch Hadoop im allgemeinen) soll horizontal skalierbar sein. Wird mehr Speicher benötigt, sollen einfach noch Knoten hinzugefügt werden können.

Das HDFS ist auf hohen Datendurchsatz und die Speicherung großer Datenmengen ausgelegt. So können einzelne Dateien mehrere Gigabyte bis hin zu Terrabyte groß sein und es können mehrere Millionen Dateien im HDFS gespeichert werden. Die Optimierung auf einen möglichst hohen Datendurchsatz geht mit einer schlechteren Reaktionszeit im Vergleich zu herkömmlichen Dateisystemen einher.

Das Prinzip *Write-once-Read-many* wird im HDFS implementiert. Wenn Daten einmal geschrieben wurden, dann werden sie normalerweise nicht mehr geändert. Dies ermöglicht ein einfacheres Kohärenzmodell. Dies fördert den Lesedurchsatz indem die Unterstützung der Modifikation von Daten stark eingeschränkt wird. Ein wahlfreies Schreiben in eine existierende Datei wird beispielsweise nicht unterstützt. Änderungen an Daten, welche von

⁵ *Representational State Transfer (REST)* bezeichnet ein Programmierparadigma in verteilten Systemen. Hierbei werden Ressourcen über HTTP angefordert, gespeichert und verarbeitet.

Algorithmen vorgenommen werden, resultieren in neuen Datensätzen. Darüber hinaus gilt das Prinzip der Datenlokalität. Algorithmen werden dort ausgeführt, wo die Daten liegen, um das Verschieben von Daten über das Netzwerk zu vermeiden.[6]

Der Aufbau eines HDFS bildet eine Master-Slave Architektur aus *NameNodes* und *DateNodes*. Der NameNode ist einmalig im verteilten System vorhanden und enthält alle Meta-informationen zu den Dateien. Eine Datei selbst wird in ein oder mehrere Blöcke aufgeteilt und auf mehreren DateNodes gespeichert. Der NameNode organisiert diese Speicherung und bestimmt, wo welche Daten persistiert werden. Über den NameNode selbst fließen aber keine Rohdaten von Dateiinhalten. Auf Dateisystemebene ist das HDFS wie gängige Dateisysteme hierarchisch organisiert. Jede Datei wird über einen absoluten Pfad eindeutig bestimmt und erhält entsprechende Metadaten, wie Dateirechte und Zeitstempel.

Abbildung 3.2 verdeutlicht die Struktur im HDFS. Angenommen es soll die Datei `/home/foo.txt` gespeichert werden. Dies kann mit dem Terminalprogramm `hdfs` durchgeführt. Das Programm selbst ist hier der HDFS-Client und hat Zugang zum Hadoop-Cluster. Der HDFS-Client speichert zuerst die Metadaten der Datei auf dem Name Node. Der Name Node bekommt die Größe der Datei auch mit und entscheidet dann, in wie viele Blöcke sie unterteilt werden soll. Darauf hin ermittelt für jeden einzelnen Block, auf welchen Date Nodes dieser Block gespeichert werden soll. Diese Blockaufteilung und die Zuordnung zu den Date Nodes werden an den HDFS-Client zurückgeschickt. Dieser übermittelt die Blöcke an einen der Data Nodes. Sobald der erste Data Node einen Block hat, sorgt er dafür die Blöcke an die anderen Data Nodes weiterzuleiten. Die Data Nodes selbst stehen auch in Kontakt zum Name Node und reporten ihren Zustand und die momentan gespeicherten Blöcke. Der Name Node bekommt darüber auch mit, wenn ein Data Node ausfällt. Ein Block hat in der Standardkonfiguration 128 MB. Er kann aber auch bis zu 512 MB Größe konfiguriert werden. Dies wirft die Frage auf, ob das HDFS gerade für sehr kleine Dateien, wie sie bei der Analyse von Datenträgern auch vorkommen, nicht zu viel Speicher verschwendet.

Hierbei wird der gleiche Block immer in unterschiedlichen Data Nodes angelegt. Es ist nicht erlaubt den gleichen Block mehrmals im gleichen Data Node zu replizieren. Daher ist die Anzahl der Replikationen auch kleiner gleich der Anzahl Data Nodes im System. Wichtig hierbei ist auch, dass im Produktivsystem auf jedem physikalischen Knoten auch nur ein DataNode oder ein NameNode läuft. Denn würden beispielsweise mehrere DateNodes auf dem gleichen physikalischen Knoten laufen, so wäre bei einem Ausfall nicht mehr garantiert, dass die Dateiinhalte auch noch auf mindestens zwei anderen Knoten laufen. Denn der Replikationsmechanismus im HDFS kann nicht erkennen, ob jeder Knoten physikalisch unabhängig arbeitet. Allerdings hat Hadoop eine sogenannte *Rack-Awareness*. So ist es möglich zu bestimmen, welche physikalischen Server in einem gemeinsamen Rack laufen. Abhängig davon, versucht das HDFS die Daten teilweise im selben Rack redundant zu speichern aber auch einige Replikationen außerhalb des Racks anzulegen. So kann auch der Ausfall eines Racks im Notfall kompensiert werden.

In Testumgebungen ist aber schön möglich sogar NameNode und mehrere DateNodes auf einem Knoten laufen zu lassen. Allerdings greifen die Mechanismen für eine Toleranz gegenüber Hardwareausfällen dann nicht mehr.

Wie oben ersichtlich, ist der Name Node die Schlüsselstelle im HDFS-Cluster. Dieser bildet einen *Single Point of Failure*. Denn bei einem Ausfall wäre das HDFS nicht mehr einsatzbereit. Es existiert ein sogenannter *Secondary Name Node*. Dieser erhält die Me-

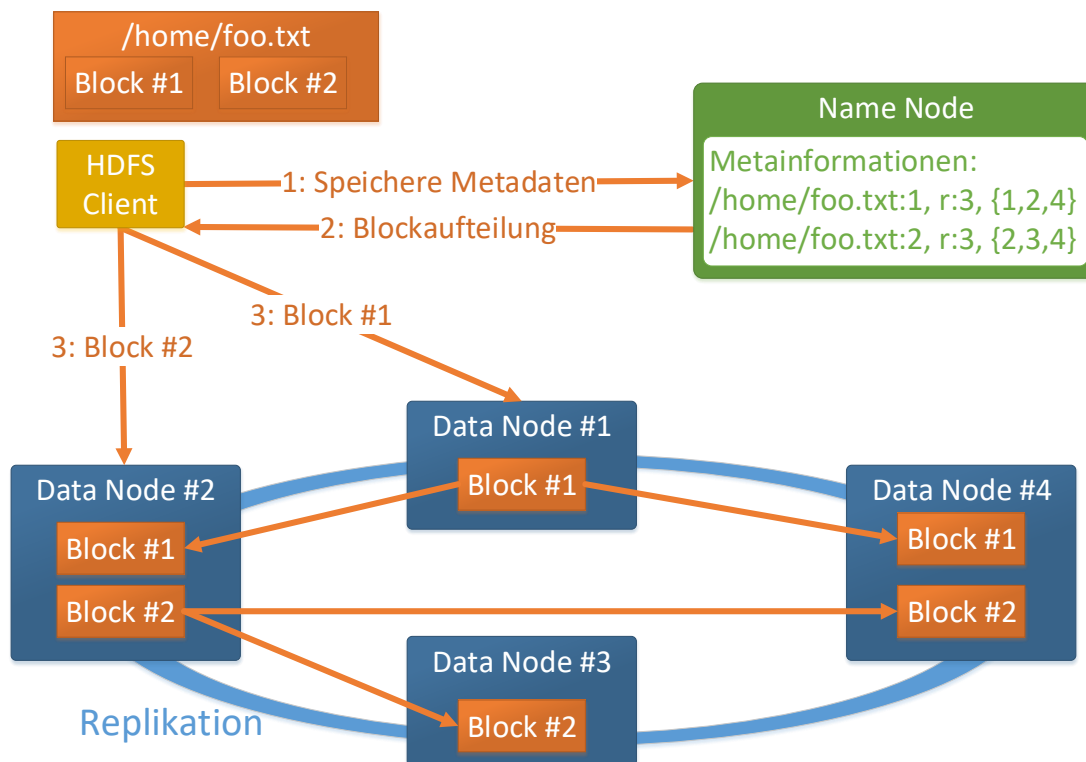


Abbildung 3.2: HDFS - Datenspeicherung im Verbund (Vgl. [6],[3])

tainformationen und erstellt daraus regelmäßig Checkpoints. Der Name Node hält die Metainformationen im Arbeitsspeicher. Es existiert aber auch eine Datei *FsImage* und ein *EditLog*, welche persistent auf der Festplatte gespeichert sind. Das *FsImage* selbst beschreibt einen Zustand der Dateisystemmetainformationen zu einem gewissen Zeitpunkt. Im *EditLog* befinden sich alle Änderungen seit dem letzten Checkpoint bis zum aktuellen Zeitpunkt. Der Secondary Name Node erstellt aus dem *FsImage* und dem *EditLog* regelmäßig neue Checkpoints, die dann der produktive Name Node bei einem möglichen Neustart wiederverwenden kann. Der *Secondary Name Node* unterstützt also den (First) Name Node, er kann ihn aber nicht ersetzen.

Daher ist es möglich auch einen sogenannten *Standby Name Node* zu konfigurieren. Dieser kann einspringen, sobald der erste Name Node ausgefallen ist. Allerdings muss dieser extra konfiguriert werden. Dafür kann aber dann der Secondary Name Node deaktiviert werden.[3, S. 88] **TODO: prüfen ob das wirklich stimmt!**

Das HDFS selbst kann über mehrere Wege genutzt werden. Es gibt eine Kommandozeilenschnittstelle, die sogenannte *FS Shell*. Es ist möglich über eine Java oder C++ - Schnittstelle Datenzugriff zu erhalten. Oder das Dateisystem kann über eine REST-Schnittstelle via HTTP(S) genutzt werden. Auch das Mounten als *Network File System (NFS)* ist möglich?

3.3 Hadoop YARN

YARN ist ein Ressourcenmanager, welcher die verfügbaren Ressourcen innerhalb des Hadoop Clusters organisiert und die Ausführungsreihenfolge von Jobs plant und überwacht.

Es gibt einen *Resource Manager*, welcher nur die Ressourcen verwaltet. Auf jedem Knoten, welcher auch Datenverarbeitungen durchführt, ist ein *Node Manager* aktiv. Zuletzt gibt es noch einen *Application Manager* für jeden einzelnen Job, der ausgeführt werden soll. Der Application Manager kontrolliert die Ausführung des Jobs. Abbildung 3.3 zeigt die Komponenten von YARN im Cluster.

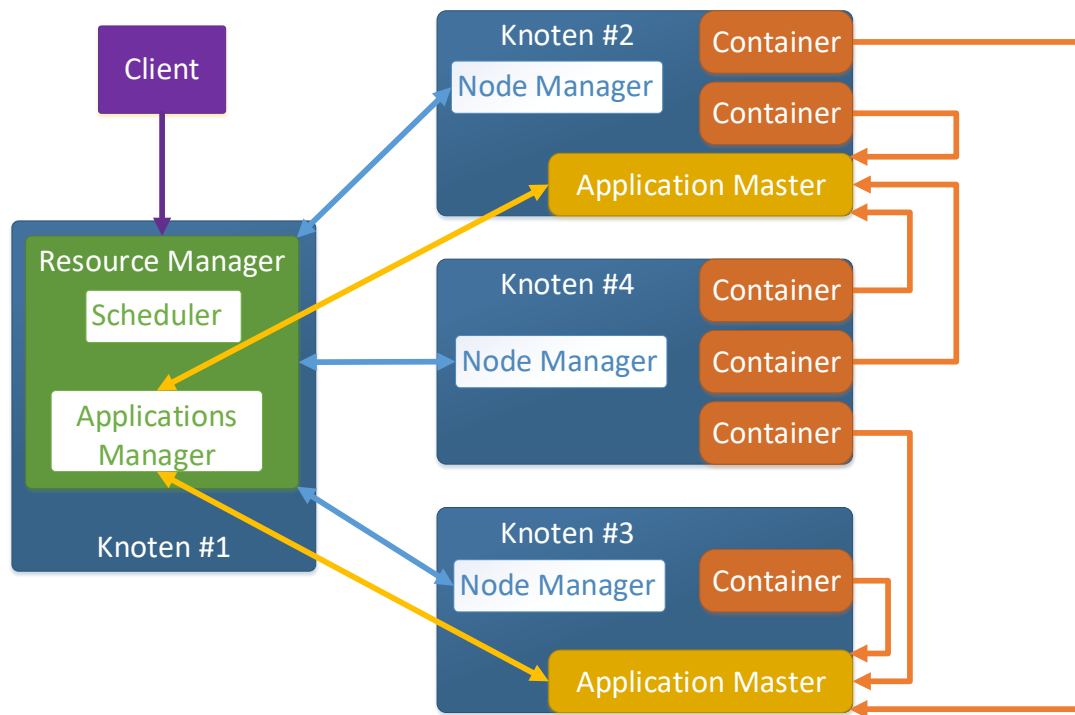


Abbildung 3.3: Ressourcenverteilung mit YARN (Vgl. [5],[3])

Ein Job oder eine Anwendung besteht aus mehreren Tasks. Diese Tasks können parallel in mehreren sogenannten Container ausgeführt werden. Ein Container ist eine abstrakte parallele Verarbeitungseinheit, welche bestimmte CPU- und Speicher-Ressourcen enthält. Es können mehrere dieser Container auf einem Knoten innerhalb des Clusters ausgeführt werden. Beispielsweise werden bei einem Knoten mit einer Quad-Core CPU und Hyperthreading (mit insgesamt 8 ausführbaren Threads) bis zu 8 Container erstellt. Bei 32 GB Arbeitsspeicher könnten dann jedem Container 4 GB zugeteilt werden.⁶ Derzeit werden für den Container die Anzahl der CPU-Cores (Ausführbare CPU-Threads) und die Größe des nutzbaren Arbeitsspeichers definiert.[3, S. 48 ff.]

Wenn nun eine Anwendung über YARN im Cluster ausgeführt werden, soll dann sendet anfragenden Client eine Anfrage an den Resource Manager. Für jeden auszuführenden Job erstellt der Resource Manager den ersten Container. In diesem Container wird dann der Application Manager gestartet, welcher sich dann im weiteren Verlauf um die Ausführung des Jobs kümmert. Der Resource Manager selbst kennt die Anwendung nicht, noch

⁶In der Praxis ist es meistens weniger, da entsprechende Ressourcen für das darunter liegende Betriebssystem und YARN selbst reserviert werden.

weiß er wie diese ausgeführt werden. Er ist nur dafür zuständig Ressourcen zu verteilen. Der Application Master hingegen ist sehr spezifisch. Wird zum Beispiel eine Apache Spark Anwendung mit YARN ausgeführt, so ist der Application Master der sogenannte *Spark App Master*. Nachdem nun der Application Master mit im ersten erzeugten Container gestartet wurde, kann dieser wiederum neue Ressourcen beim Resource Manager anfordern. An dieser Stelle zeigt sich der Vorteil von YARN in Kombination mit dem HDFS. Denn bei der Anforderung von Ressourcen gibt der Application Manager an, wieviele Container (inklusive Arbeitsspeicher und CPU) er benötigt. Zusätzlich übermittelt er die Dateiblöcke, welche er aus dem HDFS braucht und an welchen Knoten er welche Container gerne starten würde. So würde der Application Manager 1 auf dem Knoten 2 (siehe Abbildung 3.3) einen Container auf dem Knoten 2 und zwei Container auf dem Knoten 4 mit beispielsweise 1 GB Arbeitsspeicher und 1 Core anfordern. Denn der Application Master weiß, dass dort die benötigten Datenblöcke im HDFS gespeichert sind. Hierbei ist es wichtig zu verstehen, dass die Knoten aus Abbildung 3.3 den Data Nodes aus Abbildung 3.2 entsprechen.⁷

Der Application Master erhält dann, die Zustimmung von Resource Manager, nachdem Scheduler die geforderten Ressourcen entsprechend eingeteilt hat. Darauf fordert der Application Manager den Node Manager auf den jeweiligen Knoten auf, entsprechende Container zu erstellen.

Die einzelnen Node Manager stehen in Kontakt zum Resource Manager und reporten ihm, den aktuellen Status des Knoten und dessen Auslastung.

Nach der Ausführung der einzelnen Tasks innerhalb den Container und dem Abschluss des Jobs, schickt der Application Manager die Ergebnisse direkt zurück zum Client.⁸ Danach meldet er sich beim Resource Manager ab. Zuletzt kümmert sich der Resource Manager dann über das Freigeben von allokierten Ressourcen.

Ähnlich wie beim Prozessscheduling in ein herkömmlichen Betriebssystem, gibt es auch für YARN unterschiedlicher Algorithmen, die festlegen, in welcher Reihenfolge und Zeitdauer die einzelnen Jobs ausgeführt werden. Bekannte Scheduler sind der *Fair Scheduler* und der *Capacity Scheduler*. Abhängig von der genutzten Plattform/Distribution einzelner Hersteller ist für YARN ein anderer Scheduler konfiguriert. In etlichen Fällen wird der Capacity Scheduler als Standard konfiguriert, da dieser versucht alle Knoten möglichst effizient auszusteuern um den höchstmöglichen Datendurchsatz durch erreichen. Der Fair-Scheduler prüft hingegen, dass jedem Job die gleichen Ressourcen zugeteilt werden, um möglichst alle Jobs parallel bedienen zu können. **TODO: Scheduling prüfen!**

In großen Clustern wird die Prozessierung in mehrere Sub-Cluster mit eigenen Resource Managern aufgeteilt. Diese Struktur wird in der Literatur als *Federated YARN* beschrieben und soll die Skalierbarkeit von YARN in großen Clustern ermöglichen.

3.4 Apache HBASE

Apache HBASE[®] ist eine spaltenorientierte *NoSQL*-Datenbank. Sie entstand auf den Grundlagen der *BigTable*-Datenbank von Google und wurde für das Speichern von Daten im Hadoop-Umfeld entwickelt.⁹

⁷Wobei ein physikalischer Knoten, auf welchem ein Data Node läuft nicht zwingend auch für die Datenverarbeitung mit YARN verwendet werden muss. Beziehend auf das Paradigma der Datenlokalität ist dies aber der Normalfall, dass ein Knoten, welcher Daten persistiert, auch Daten verarbeiten wird.

⁸Hierbei werden die fachlichen Ergebnisse, meistens als Datei im HDFS gespeichert.

⁹Der Name *HBASE* basiert auf der Kombination von *Hadoop* und *Database*.

Der Begriff *NoSQL*-Datenbank steht hierbei für *Not only SQL* und beschreibt letztlich Datenbanken, welche Daten vorwiegend nicht in herkömmlichen relationalen Datenbankschemata speichern. Größtenteils sind diese Datenbanken schemafrei und können horizontal skaliert werden. Diese Bedingungen sind optimal zur Speicherung großer unstrukturierter Datenmengen.

Anhand des sogenannten *CAP-Theorems* können diese Datenbanken kategorisiert werden. Das CAP-Theorem besteht aus den Eigenschaften Konsistenz, Verfügbarkeit und Partitionstoleranz und besagt, dass maximal zwei dieser drei Eigenschaften von einer Datenbank garantiert werden können. Konsistenz beschreibt hier die Garantie, dass alle Knoten im verteilten System den gleichen Datenstand haben. Die Verfügbarkeit bezieht sich auf die dauerhafte Erreichbarkeit der Daten. Wohingegen die Partitionstoleranz die Funktionsfähigkeit selbst bei Ausfall einzelner Knoten im Datenbank-Verbundsystem garantiert. Apache HBASE garantiert hierbei die Eigenschaften der Partitionstoleranz und der Konsistenz. Dies führt dazu, dass die Verfügbarkeit der Daten weniger stark ausgeprägt ist. [1, S. 189 ff.]

Während herkömmliche relationale Datenbanken die Daten zeilenweise speichern, werden in spaltenorientierten Datenbanken die Daten spaltenweise gespeichert. Hierbei werden die Daten der einzelnen Spalten gruppiert in sogenannten *Column Families* abgespeichert. Der Vorteil dieser Speicherart hängt stark von deren fachlichen Nutzung ab. Wenn alle Daten einer Spalte abgefragt werden, dann können diese Daten effizienter gelesen werden da sie zusammen persistiert wurden. Bei einer relationalen Datenbank hingegen wird bei solchen Anfragen die ganze Zeile mit allen Feldern gelesen, obwohl nur ein kleiner Teil dieser Daten wirklich benötigt wird.

Apache HBASE basiert eben auf dieser Spaltenorientierung. Die Datenbank wird im Rahmen dieser Thesis für das Speichern beliebiger Dateiinhalte und Dateimetadaten verwendet. Die Datenstruktur wird in Abbildung XXX dargestellt. TODO: Abbildung erstellen

Anhand der Abbildung werden einige Eigenschaften von HBASE sichtbar. So existieren zwei *Column Families* mit den Namen *metadata* und *content*. Die Column Family *metadata* enthält wiederum die Spalten *relativeFilePath*, *fileType*, *fileType* oder auch *lastModified*. Die Werte werden alle als Binär-Inhalt in die Tabelle geschrieben. Eine Zeile hat dann jeweils einen eindeutigen Spaltenschlüssel, wie zum Beispiel *row0*. Über diesen Schlüssel können die spezifischen Inhalt der einzelnen Spalten erfragt werden. Interessant hierbei ist, dass die Spaltenwerte optional sind und nicht für jede Zeile existieren müssen. Aufgrund der Gruppierung und Speicherung in Column Families benötigen leere Spaltenwerte auch keinen Speicherplatz. Ein einzelne Zeile beschreibt letztlich den Wert einer Spalte für eine konkrete Zeile. Hierbei wird zu jeder Zeile auch ein Zeitstempel gespeichert. Mithilfe diesen Zeitstempels können auch ältere Werte einer Zeile ausgelesen werden. So wird bei einer Modifikation einer konkreten Zeile der Wert inklusive eines neuen Zeitstempels geschrieben. Es ist jedoch immer noch möglich, ältere Zustände der Zeile zu lesen.

Die einzelnen Zellen einer Zeile (definiert durch den eindeutigen Zeilenschlüssel) werden intern als verkettete Liste gespeichert. Darüber hinaus werden die Zeilen selbst auch als verkettete Liste abgelegt. Aufgrund dieser Struktur ist die Analogie zur einer mehrstufigen *Map*, wie sie aus den meisten Programmiersprachen bekannt ist, einfacher zu verstehen, als die Darstellung mit einer klassischen Tabelle.

Die Skalierbarkeit und die Partitionstoleranz wurden bei der Entwicklung von HBASE be-

rücksichtigt. Es baut auf dem Hadoop HDFS auf und speichert darin die Daten. Analog zu HDFS, YARN oder Spark existiert auch hierbei eine Master-Slave Architektur über alle Knoten hinweg. Die einzelnen Knoten werden bei HBASE *Region Server* genannt und speichern jeweils unterschiedliche Teil der in HBASE angelegten Tabellen. Nachfolgende Abbildung zeigt die physikalische Aufteilung. TODO: Abbildung erstellen.

Prinzipiell ist es möglich in einem Cluster nicht auf allen Knoten einen Region-Server auszuführen. Um das Prinzip der Datenlokalität bestmöglich zu unterstützen, macht es jedoch gerade beim Verarbeiten der Daten aus HBASE am ehesten Sinn die Region-Server zumindest auf den Knoten laufen zu lassen, wo beispielsweise die Application Container von YARN beziehungsweise die Worker von Apache Spark laufen. Somit können die Daten direkt auf dem Knoten verarbeitet werden, wo sie auch gespeichert sind und müssen nicht über das Netzwerk an andere Knoten gesendet werden.¹⁰

TODO: Erstellung eines adäquaten Zeilenschlüssels und dessen Auswirkungen (numerische Sortierung, Hot Spot, ...)

3.5 Apache Spark

¹⁰Sie auch Kapitel 3.3 und Kapitel 3.5.

4 Aufbau einer Analyse-Plattform

4.1 Allgemeines

Im Praxisteil dieser Arbeit soll eine Analyse-Plattform auf Basis von Hadoop aufgebaut werden. Um die fachlichen Anforderungen an das Analyse-System herauszuarbeiten, soll das herkömmliche Analyseverfahren betrachtet werden.

Die Ausgangslage liefert beispielsweise ein Testszenario, welches ein Datenträgerabbild eines virtualisierten Linux-PCs enthält. Das Abbild selbst ist 10 GB groß.

Dieses Abbild soll als Referenz für die einzelnen Analyse-Schritte gelten. Hierbei sollen in jedem der nachfolgenden Unterkapitel die fachlichen Verarbeitungsschritte auf herkömmlichen Wege mit der hier erstellten Analyse-Plattform verglichen werden.

Ein gängiges Analyseverfahren besteht aus der Speicherung des Abbildes auf dem lokalen Analyse-Rechner und dem Analysieren des Beweismittels mit Betriebssystemprogrammen unter Linux oder mithilfe des Open-Source Analysetools *Autopsy*¹ unter Windows. Im kommerziellen Bereich existieren noch deutlich mehr Analyse-Tools mit größerem Funktionsumfang.

4.2 Datenspeicherung im HDFS

4.2.1 Traditionelles Analyseverfahren

In einem traditionellen Analysefall wäre als Beweismittel ein Datenträgerabbild des verdächtigen Computers erstellt worden. Mithilfe klassischer Opensource-Tools wie beispielsweise *dd* kann damit zum Beispiel ein Image im RAW-Format erstellt werden. Andere Tools, wie beispielsweise *FTK-Imager* können auch Datenträgerabbilder in speziellen Container-Formaten erstellen und lesen. Beispielsweise gibt es das *EnCase Physical*-Format mit der Dateierweiterung *e01*, oder das *Advanced Forensic Format* mit der Erweiterung *.aff*.²

Einige Tools nutzen spezielle Container-Formate. Teilweise wird auch unterschieden, ob es sich um ein vollständiges Disk-Image handelt oder um ein logisches Dateiarchiv. Bei dem vollständigen Disk-Image werden auch nicht allokierte Speicherbereiche innerhalb des Dateisystems, der Partition oder des Datenträgers gesichert. Hier können sich potentiell versteckte und gelöschte Dateifragmente befinden. Mit diversen Tools kann ein sogenanntes File-Carving durchgeführt werden. Hierbei wird nur anhand der Rohdaten im Abbild versucht, Daten zu extrahieren. Dies ist gerade auch dann sinnvoll, wenn nach gelöschten

¹Siehe <https://www.sleuthkit.org/autopsy/>. Hierbei wird die Version 4.6.0 in der 64-bit Variante unter Windows 10 Pro genutzt.

²Siehe <https://support.accessdata.com/hc/en-us/articles/222778608-What-Image-Formats-Do-AccessData-Products-Support>. Stand: 4.4.2018.

Dateien gesucht wird oder die Dateisystemmetadaten irreparabel beschädigt wurden. Ein logisches Dateiarhiv hingegen enthält wirklich nur die Dateien auf einer logischen Ebene und keine unallokierten Speicherbereiche. Von Vorteil hierbei ist eine geringere Speichergröße. Allerdings tritt durch die logische Sicherung ein potentieller Informationsverlust auf, da unallokierte Speicherbereiche nicht berücksichtigt werden, die aber dennoch potentiell auswertbare Informationen liefern könnten. Vertreter logischer Dateiarhive sind die allseits bekannten Archiv-Formate wie beispielsweise ZIP oder TAR.

Letztlich werden die Beweismittel in unterschiedlichsten Formaten auf dem lokalen Analyse-Rechner gespeichert. Darauf aufbauend können die Daten in spezifische Formate konvertiert werden. Dies hängt aber meistens davon ab, wie sie weiter verarbeitet werden sollen und welche Werkzeuge zu dieser Verarbeitung genutzt werden.

Im konkreten Testszenario ist das Datenträger-Abbild eines Linux-Rechners im RAW-Format auf dem lokalen Analyse-Rechner gespeichert. Das Abbild selbst kann ein oder mehrere Partitionen enthalten. Innerhalb der Partition werden Daten mithilfe unterschiedlicher Dateisysteme strukturiert gespeichert. Diese Dateisysteme können unter Windows mit dem Werkzeug *X-Mount* oder unter Linux direkt mit dem Befehl *mount* schreibgeschützt gemountet werden. Darauf wird das Dateisystem vom Betriebssystem interpretiert und als logisches Volume auf dem Analyse-Rechner bereitgestellt. Nun können die Dateien mit beliebigen Werkzeugen analysiert werden.

In der Praxis hat das einfache schreibgeschützte Mounten den Vorteil, dass der Analyst relativ schnell im Dateisystem beliebige Dateien finden und dessen Inhalt mit diversen Tools anzeigen kann. Gerade für eine schnelle Vorprüfung ist dies sinnvoll. Im nachfolgenden Kapitel sollen nun Möglichkeiten zur Speicherung und Aufbereitung des Datenträgerabbildes mithilfe der forensischen Analyse-Plattform untersucht werden.

4.2.2 Umsetzung in der Hadoop Analyse-Plattform

Die Datenspeicherung im HDFS ist wohl der wichtigste Aspekt in dieser Thesis. Hierbei geht es nicht nur darum, wie das HDFS die Daten verwaltet sondern vielmehr um die Art und Weise, wie Daten forensisch korrekt gespeichert werden können.

Der Standardfall wäre hierbei ein Rohdatenträgerabbild (RAW-Image), welches das Beweismittel darstellt. Hierbei liegen die Dateien als fragmentierte Blöcke in einer spezifischen Datenstruktur vor, welche das Dateisystem des Abbildes beschreiben. Je nachdem, ob ein Datenträgerabbild direkt von einer Partition eines Datenträgers oder vom ganzen Datenträger erstellt wurde, sind in dem Abbild unter Umständen auch mehrere Partitionen samt Partitionstabelle enthalten. Von diesen Partitionen kann wiederum jede einzelne Partition ein eigenes Dateisystem, wie beispielsweise FAT, ext4, oder NTFS enthalten. Dieses Dateisystem enthält dann die eigentlichen Dateien, welche logisch zusammengesetzt werden müssen. Viele Linux-Distributionen bieten hier bereits eine weitreichende Unterstützung zum Lesen und Schreiben dieser Dateisysteme. Hierzu können die Dateisysteme einzelner Partitionen des Datenträgerabbildes *gemountet* werden.

Zur Speicherung der Daten im HDFS gibt es mehrere Möglichkeiten, deren Vor- und Nachteile nachfolgend dargestellt werden.

Variante 1 - Datenträgerabbild im HDFS speichern

Die naheliegende Variante zur Speicherung der Beweismittel, wäre die Datenträgerabbilder direkt im HDFS abzuspeichern. Allein die Größe der Abbilder wäre nicht problematisch. Um eine entsprechende Aufteilung kümmert sich das HDFS.

Allerdings hat die Lösung den entscheidenden Nachteil, bei der Weiterverarbeitung der Daten. Auf Betriebssystemebene lassen sich solche Datenträger einfach mounten. Zumal die meisten Betriebssysteme auch ein großen Teil der unterschiedlichen Dateisysteme auf einem Datenträger erkennen können. Aber innerhalb des Hadoop-Frameworks findet sich keine Unterstützung zum Lesen von beliebigen Dateisystemen. Denn normalerweise nutzen JAVA-Applikationen eine definierte Schnittstelle auf Basis logischer Dateien, die wiederum vom Betriebssystem bereitgestellt werden. Um die logischen Dateien aus dem Datenträgerabbild extrahieren zu können, müsste für jedes einzelnes Dateisystem eine eigene Implementierung geschrieben werden.

Darüber hinaus wäre das Extrahieren der Dateien aus einem Dateisystem auf einem Datenträgerabbild auch nicht wirklich performant. Angenommen das Auslesen würde mit Apache Spark durchgeführt werden. Aufgrund der eingangs beschriebenen Datenlokalität (siehe Kapitel 3) wären auf jedem Knoten einzelne Blöcke von beispielsweise 128 MB Größe vorhanden. Um dann im ext4 Dateisystem eine Datei lesen zu können, sollten zumindest die Dateisystemmetadaten aus dem 1. Block in jedem Knoten verfügbar sein. Darüber hinaus können die Fragmente einer einzelnen Datei verstreut auf innerhalb des Dateisystems liegen. Je nach Fragmentierungsgrad des Dateisystems, müsste dann auf einem Data-Node in vielen Fällen weitere Blöcke anderer Knoten nachgeladen werden, da dort spezifische Fragmente der einzelnen Datei liegen könnten. Diese würde das Prinzip der Datenlokalität aushebeln. TODO: abbildung von ext4 dateisystem inklusive datei, welche überall verteilt ist. + letzter Absatz überarbeiten!

Variante 2 - Logische Dateien im HDFS speichern

Die zweite Variante wäre das Beweismittel auf dem lokalen Analyserechner zu mounten und alle Dateien auf logischer Ebene direkt in das HDFS zu importieren. Damit wäre die gesamte Dateisystemstruktur aus dem Datenträgerabbild im HDFS abgelegt. Allerdings wäre File Carving beziehungsweise das Auffinden von gelöschten Dateien nicht mehr im Hadoop-Framework möglich, sofern nicht auch der das Datenträgerabbild selbst nochmals in das HDFS importiert wird.

Interessanter an dieser Variante ist aber das Verhalten des HDFS bezüglich der Metadaten und der unterschiedlichen Dateigröße. Bei dem Importierten müsste darauf geachtet werden, dass alle Metadaten des lokalen Dateisystems im Datenträgerabbild unverändert in das HDFS kopiert werden. Welche Metadaten bereits im HDFS mit angelegt werden, zeigt Abbildung 4.1 anhand eines Ausschnitts aus der Web-Repräsentation eines HDFS.

Daraus ist ersichtlich, dass jede Datei entsprechende Dateirechte hat und einem Nutzer und einer Gruppe zugeordnet ist. Zusätzlich wird die Größe (nicht bei Verzeichnissen) und der Zeitstempel der letzten Änderung gespeichert. Die Anzahl der Replikationen und die Blockgröße sind spezifisch für das HDFS. Jede Datei kann auf einer unterschiedlichen Anzahl von Knoten repliziert sein. Die Standardkonfiguration definiert 3 Replikationen im realen Cluster, wobei Verzeichnisse nur logisch auf dem Name Node gehalten werden und damit auf keinem DataNode explizit repliziert werden. Auch die Blockgröße ist in der Stan-

Browse Directory

/data

Go!

Show

25

entries

Search:

<div><input type="checkbox"/></div>	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name	<div><input type="checkbox"/></div>
<div><input type="checkbox"/></div>	drwxr-xr-x	johannes	supergroup	0 B	Apr 04 06:38	0	0 B	DiagrammeUndRessourcen	<div><input type="checkbox"/></div>
<div><input type="checkbox"/></div>	-rw-r--r--	johannes	supergroup	12.9 MB	Apr 04 06:38	1	128 MB	M105 Studienbriefe.pdf	<div><input type="checkbox"/></div>
<div><input type="checkbox"/></div>	-rw-r--r--	johannes	supergroup	2.06 MB	Apr 04 06:38	1	128 MB	M107-windows10-05-31.pdf	<div><input type="checkbox"/></div>

Abbildung 4.1: HDFS - Dateieigenschaften

dardkonfiguration auf 128 MB festgelegt. Wie im Grundlagenkapitel erwähnt, werden die Dateien in mehreren Blöcken zu maximal 128 MB (konfigurationsabhängig) gespeichert und auch repliziert.

Bezogen auf klassische Dateisysteme entspricht dies auch einem Block im ext4-Dateisystem oder einem Cluster im NTFS-Dateisystem (TODO: Quellenangabe!!! S.22 in M111). Es ist letztendlich die kleinste allozierbare Dateneinheit im Dateisystem. Dies bedeutet allerdings nicht, dass für jede Datei im HDFS auf den jeweiligen Data Nodes immer mindestens 128 MB Speicher belegt werden. Denn die reale Speicherbelegung auf dem Data Node beschränkt sich auch auf die reale Größe der Datei im lokalen Dateisystem des Data Nodes.

Mit dem Befehl aus Listing 4.1 können Dateien von einem lokalen Verzeichnis in das HDFS importiert werden.

```
1 hdfs dfs -put test.pdf /test.pdf
```

Listing 4.1: Befehl zum Speichern einer Datei im HDFS

Hierbei werden die ursprünglichen Metadaten der Datei nicht übernommen. So beschreibt der oben erwähnte Modifikationszeitstempel den Zeitpunkt, zu dem die Datei im HDFS das letzte Mal geändert wurde. Dies entspricht initial dem Import-Zeitpunkt. Auch werden Nutzer und Gruppenrechte nicht übernommen. Prinzipiell wäre es aber möglich die Metadaten aus dem lokalen Dateisystem mit in das HDFS zu übernehmen.³ Ratsam ist dies jedoch nicht, da gerade auch der Nutzer, die Gruppe und die dazugehörigen Zugriffsrechte in einem produktiven HDFS-Cluster verwendet werden, um Zugriffsbeschränkungen einzelner Nutzer und einzelner Programme umzusetzen.

In der Theorie können noch weitere Metadaten gespeichert werden. Hierbei können erweiterte Dateiattribute beliebige Informationen über die Datei speichern. Mit nachfolgenden Befehlen kann beispielsweise der originale Zeitstempel der Erstellung einer Datei hinzugefügt und ausgelesen werden. Hierbei kann der Name des Attributs und dessen Inhalt frei gewählt werden.

```
1 # Create custom file attribute
```

³Hierzu kann dem `put`-Befehl aus Listing 4.1 der Parameter `-p` mit übergeben werden.

```

2 | hdfs dfs -setfattr -n user.ntfs.creationtime -v "2018-04-07T11
   | :14:42,798583789+02:00" /test.pdf
3 |
4 | # Read custom file attribute
5 | hdfs dfs -getfattr -d -n user.ntfs.creationtime /test.pdf

```

Listing 4.2: Befehl zum Hinzufügen und Auslesen von Metadaten

Mit dem obigen Befehl wäre es also prinzipiell möglich, Metadaten des ursprünglichen Dateisystems zu übernehmen. Allerdings müssen diese Metadaten zur Weiterverarbeitung zuerst für jede Datei im HDFS eingetragen werden.

Zusätzlich müssen beim Prozessieren der Daten mit Apache Spark die Metadaten auslesbar sein. Dieses Auslesen ist umständlich aber möglich.⁴

Letztlich ist es wohl aber auch ein Performance-Problem, denn diese Metadaten werden auf dem Namenode gespeichert.⁵ Somit müsste jeder Datanode diese Informationen wiederum explizit am Namenode anfragen.

Das Fazit der Variante 2 lautet daher, dass die Dateimetadaten des originalen Datenträgerabbildes höchstens als erweiterte Attribute im HDFS abgelegt werden könnten. Beim Importieren müssten diese Metadaten bei jeder einzelnen Datei explizit nachgetragen werden. Pro Datei wäre dies nochmals ein eigener HDFS-Aufruf. Nicht zuletzt werden diese erweiterten Metadaten physikalisch im Namenode gespeichert. Dadurch benötigt der Namenode mehr Speicher. Viel schwerwiegender jedoch ist, dass die erweiterten Metadaten beim Prozessieren mit Spark angefordert werden müssten und somit der Namenode viel zu stark beansprucht werden würde.

Aus diesen Gründen ist die Variante 2 nicht akzeptabel.

TODO: weiterer Aspekt - performantes Prozessieren von vielen kleinen Dateien / bzw. Dateien unterschiedlicher Größe!!!

Variante 3 - ZIP-File bzw. Hadoop Archive?

Variante 4 - Costum Forensic Format

Metadaten, Blockaufteilung, logische Sicherung
nachteil, carving muss vorher passieren...

Variante 5 - Speicherung mit HBASE und HDFS

Diese Variante beschreibt einen Ansatz, bei welchem die Metadaten und sehr kleine Dateien direkt in HBASE persistiert werden. Dateien größer 10 MB werden direkt im HDFS abgelegt und im jeweiligen Dateieintrag in HBASE verlinkt. Auch hier muss beim Import der Daten eine eigene Anwendung die Informationen aus dem Datenträgerabbild auslesen und entsprechend ihrer Struktur in HBASE und HDFS speichern. Das GitHub-Projekt *foam-data-import* enthält diese Anwendung (siehe <https://github.com/jobusam/foam-data-import>).

⁴Siehe Metadaten-Extraktion im Projekt *foam-processing-spark* unter <https://github.com/jobusam/foam-processing-spark>.

⁵Siehe <https://de.hortonworks.com/blog/hdfs-metadata-directories-explained/>, Stand 6.4.2018.

4.3 Fachliche Probleme bei den Daten

4.3.1 Symbolische Links

Es gibt einige Hürden beim Importieren der Datenträger in das HDFS. Wie bereits erwähnt werden die Daten auf logischer Dateiebene in das Hadoop-System importiert. Hierbei müssen spezielle Dateitypen berücksichtigt werden. Ein Beispiel ist die Verarbeitung von symbolischen Links, welche gerade unter Linux-basierten Betriebssystemen beziehungsweise in der EXT-Dateisystemfamilie auftreten können. Denn wenn symbolische Links in einem Dateisystem gespeichert werden und letzteres im Analysesystem gemountet wird, so können diese Links auch auf Dateien außerhalb des Dateisystems verweisen. Denn letztlich interpretiert das Betriebssystem diese symbolischen Links. Bei der forensischen Analyse könnte diese aber zu schwerwiegenden Fehlern der Analyseergebnisse führen, wenn beispielsweise Inhalte des Analyserechners verarbeitet werden, welche ursprünglich nicht auf dem Asservat vorhanden waren. Daher muss beim Import auf geprüft werden, ob die Datei einem symbolischen Link entspricht. Ist dies der Fall darf, der symbolische Link nicht interpretiert werden.

4.3.2 Zugriffsrechte

Ein weiterer Aspekt ist die Beschränkung der Dateizugriffe auf Basis der vorgegebenen Zugriffsrechte. Angenommen es wird ein Datenträgerabbild auf einem Analysesystem gemountet. Dann kann dies im Normalfall nur mit Root-Rechten durchgeführt werden. Der forensische Analyse benötigt also zumindest auf seinem Analyse-Rechner privilegierte Ausführungsrechte.

Beim Import von Dateien auf eben diesem gemounteten Dateisystem des Datenträgerabbildes sind jedoch die Dateizugriffsrechte weitaus interessanter. Denn das Analyse-Betriebssystem berücksichtigt diese Zugriffsrechte. Während diese Problematik beispielsweise bei NTFS-Dateisystemen eine untergeordnete Rolle spielt, so werden hingegen bei EXT-Dateisystemen die Unix-Dateirechte gespeichert und auch auf dem Analysesystem interpretiert. Daher kann der Nutzer und dessen ausgeführte Programme, welche die Daten aus dem Dateisystem auslesen, nicht in allen Fällen auf alle Dateien zugreifen.

Die einfachste Möglichkeit um die Problematik der Zugriffsrechte zu umgehen, wäre das Ausführen der Datenimport-Applikation mit Root-Rechten. Andererseits sollte die Datenimport-Applikation aber nicht mit Root-Rechten ausgestattet werden, da dies zu Sicherheitslücken und unvorhergesehenen Rechteauserweiterungen im Fehlerfall führen könnte. Darüber hinaus kann bei einem Fehlverhalten der Anwendung das Analysesystem beschädigt werden. Letztlich braucht die Anwendung zum Datenimport nur die Berechtigungen zum Lesen von Dateien mit Root-Rechten. Dies würde dem *Need-to-know-Prinzip* genügen.

Eigentlich müsste beim Mounten des Dateisystems dem Betriebssystem mitgeteilt werden können, dass die Dateirechte des gemounteten Dateisystems ignoriert werden sollen. Diese Option ist aber nicht möglich⁶.

Eine weitere Alternative wäre die Möglichkeit mit Access Control Lists (ACL) zu arbeiten und dem nichtprivilegierten Nutzer Rechte zum Lesen der Dateien zu geben. Oder umgekehrt alle Dateien dem nichtprivilegierten Nutzer zuzordnen, welcher wiederum den

⁶Zumindest konnte keine funktionierende Variante gefunden werden (siehe Man-Page des Mount-Befehls).

Datenimport startet. Hierzu müsste die Datenträgerkopie schreibend gemountet werden, damit die Rechte jeder Datei angepasst werden können. Dies würde wiederum dazu führen, dass das Datenträgerabbild als sichergestelltes Asservat geändert werden würde. Daher ist diese Lösung auch nicht geeignet.

Eine weitere Alternative ist die Nutzung von Posix Capabilities⁷. Dies scheint unter CentOS/Fedora wohl die beste Variante zu sein. Zum Lesenden Zugriff auf Dateien müsste die Posix Capability `CAP_DAC_READ_SEARCH` gesetzt werden.

Mit nachfolgenden Kommando könnte diese Capability dem Analyseprogramm gesetzt werden. Damit könnte dann auch ein nicht-privilegierte Nutzer lesenden Zugriff auf privilegierte Dateien erhalten.

```
1 setcap CAP_DAC_READ_SEARCH /bin/forensic_import
```

Listing 4.3: Befehl zum Setzen von Posix Capabilities

Allerdings funktioniert diese Art hauptsächlich bei Binärprogrammen aber nicht bei Shell-Skripten oder Java-Anwendungen. Ein ähnliche Alternative zu Posix Capabilities ist das Setzen des SUID-Bits als Unix-Dateirecht für die Programmdatei. Aber auch gilt wieder, dass dies für Binärprogramme funktioniert aber nicht für interpretierte Skripte oder Java-Anwendungen, welche wiederum in der Java Virtual Machine laufen.

Die Nutzung von SELINUX empfiehlt sich hier auch nicht. SELINUX ist eine Erweiterung des Linux-Kernels um detaillierte Zugriffskontrollen zu setzen. Diese Erweiterung ermöglicht eine detaillierte Definition, welche Anwendung oder Nutzer auf einzelne Dateien zugreifen dürfen. Allerdings werden auch unter SELINUX immer zuerst die Unix-Dateirechte und zusätzlich die in SELINUX definierten Zugriffskontrollen geprüft. Daher kann auch mit SELINUX das Problem nicht behoben werden.

Zuletzt gibt es noch eine Variante, welche die Problematik mit den Dateirechten lösen kann. Mit dem Projekt *bindfs*⁸ können unter Linux Dateisystemverzeichnisse neu gemountet werden und ihre Zugriffsrechte verändert werden. Der nachfolgende Befehl mountet das existierende Verzeichnis mit den enthaltenden Dateien in einem neuen Verzeichnis und setzt beispielsweise bei jeder Datei die aktuelle ID des Nutzers als Datei-Owner und Group.

```
1 sudo bindfs -u $(id -u) -g $(id -g) src_dir/ target_dir/
```

Listing 4.4: Nutzung von Bindfs zum Ändern von Dateirechten

Der Befehl selbst benötigt Root-Rechte. Jedoch kann der Nutzer danach alle Dateien des Zielverzeichnisses lesen. Der einzige Nachteil an dieser Lösung ist, dass der Besitzer und die Gruppe jeder einzelnen Datei nun der Nutzer des Analysesystems ist.

Dieser Nachteil muss zukünftig behoben werden, damit die forensische Analyseplattform auch die Besitzer und Gruppen einer Datei korrekt auswerten kann.

4.4 Datenverarbeitung mit Apache Spark™

Der physikalische Aufbau wurde bereits im Grundlagenkapitel zu Apache Spark behandelt (siehe Kapitel 3.5). In diesem Kapitel sollen primär die Algorithmen und die Verarbeitung

⁷Siehe Manpages mit folgendem Befehl: `man 7 capabilities`.

⁸Siehe <https://bindfs.org/>.

der Daten aus logischer Sicht betrachtet.

Bei Apache Spark gibt es seit der Version 2.0.0 diverse APIs, wie Daten geladen werden können. Es besteht die Möglichkeit Daten mithilfe von Resilient Distributed Datasets (RDDs) zu laden und zu verarbeiten. Aufbauend auf diesen RDDs können, die Daten gemappt, gefilter oder aggregiert werden. Diese Möglichkeit gibt es schon immer in Apache Spark. Seit der Version 2.0.0 gibt es nun auch DataFrames und DataSets (TODO: gibt es beides erst seit v2.0?). Diese Datenstrukturen beschreiben eher eine Schnittstelle aus Sicht von Tabellen. Die Implementierung dieser Typen baut wieder auf den RDDs auf. Doch welche Strukturen eignen sich für die Anwendungsfälle in dieser Thesis?

DataFrames und DataSets sind optimiert für strukturierte und semi-strukturierte Daten. Diese Daten lassen sich beispielsweise in Tabellenstrukturen einlesen und verarbeiten. Es gibt High-Level Operationen auf diesen Tabellen, welche dem klassischen SQL Syntax sehr nahe kommen?! Apache Spark selbst kann bei der Nutzung von DataFrames und DataSets viele Optimierungen bei der Ausführung und Verarbeitung durchführen. Andererseits sind diese Strukturen ungeeignet bei unstrukturierten Daten, wie beispielsweise Multimediadateien und eben auch beliebigen Dateien.[4, S. 66 ff.]

Wie beim Datenimport schon beschrieben, sind die Metadaten der analysierten Datenträger strukturiert beziehungsweise semi-strukturiert in HBASE abgespeichert. Prinzipiell wäre es also möglich, auch mit DataSets und DataFrames auf diese Daten zuzugreifen. Letztlich kommt es auch auf die Anbindung zwischen Apache Spark und Apache HBASE an. Hierbei gibt es primär zwei unterschiedliche Connectoren⁹. Der *Hortonworks SHC* Connector ermöglicht die Interaktion mit Daten in HBASE und nutzt dafür die DataFrame/DataSet Datenstrukturen.¹⁰ Also Pendant auf Basis von RDDs existiert ein weiterer *hbase-spark* Connector. Letzterer wird im Rahmen dieser Thesis genutzt, um Daten von HBASE zu lesen und zu schreiben.¹¹

4.4.1 Praxisbeispiele und deren Optimierungen

Gerade bei der Verarbeitung großer Datenmengen und unter Berücksichtigung des Prinzips der Datenlokalität existieren einige Fallstricke und Hürden bei der Implementierung der Datenverarbeitung. Im Hadoop-Umfeld und bei der Entwicklung im Spark-Context geht es nicht nur um die Art und Weise, wie die Algorithmen auf die Daten angewendet werden, sondern in erster Linie auch immer darum **wo** die einzelnen Programmteile ausgeführt werden. Der Entwickler sollte immer wissen, in welchem Verarbeitungskontext er sich befindet. Zu dieser Problematik werden in diesem Kapitel einige Beispiele herausgegriffen, welche während der Bearbeitung dieser Thesis aufgetreten sind.

⁹Bei Apache Spark sind Connectoren eine Art von Java-Bibliotheken, welche es ermöglichen im Apache Spark Ausführungskontext auf andere Systeme, wie beispielsweise Datenbanken oder Dateisysteme, zuzugreifen.

¹⁰Siehe <https://github.com/hortonworks-spark/shc>, Stand: 15.6.2018.

¹¹Siehe <https://github.com/apache/hbase/tree/master/hbase-spark>, Stand: 15.6.2018 und deren Nutzung im Projekt *foam-processing-spark* unter <https://github.com/jobusam/foam-processing-spark>, Stand: 16.5.2018.

Weniger ist mehr TODO

Ungenutzte Daten so früh wie möglich aus der Verarbeitung rausnehmen. Siehe Problematik beim HBASE-Spark Connector. Entweder ich mache einen Full-Table Scan und fordere alle Daten an, um sie später im Spark-Executor auszuführen, oder ich versuche schon beim Zugriff der Daten in den Region-Server mit ColumnFamilies und Filter-Operationen nur die Daten anzufordern, welche auch wirklich benötigt werden.

Caching - Performanz vs. Ressourcen TODO

Hashing-Problem. Ist es geschickter Daten zu Cachen anstatt sie zweifach anzufordern? Funktioniert Caching überhaupt mit nicht serialisierbaren Daten?

Faulheit ist der Schlüssel zum Erfolg TODO

Lazy-Loading und Ausführung bei RDDs

Teile und Herrsche TODO

Balancing and Repartitionieren. Aufteilung der Last zu gleichen Teilen! Gerade beim Ausprobieren und Testen ist es einfach, die Resultate eines RDDs nach der Datenverarbeitung über eine Konsole auszugeben. Doch hierbei muss genau überlegt werden, wie diese Resultate ausgegeben werden (siehe Listing 4.5). In der ersten Variante wird auf dem RDD die Methode `collect()` aufgerufen und die daraus erhaltene Liste von Objekten wird über ein Logger-Objekt in das Log-File dieser Ausführung geschrieben.

Auf der ersten Blick ist aber nicht ersichtlich, was diese Methode wirklich bewirkt. Wie bereits in Kapitel 3.5 (TODO: check reference) beschrieben, wird bei der Ausführung einer Spark-Anwendung ein sogenannter Spark-Driver gestartet. Dieser wiederum fordert eine gewisse Anzahl von Exekutoren an, die die eigentlich Datenverarbeitung übernehmen (Master-Slave-Prinzip). Hierbei laufen die Executors auf einzelnen Knoten innerhalb des Clusters. In dem Moment, in welchem die `collect()`-Methode auf einem RDD ausgeführt wird, werden die Daten des RDDs *eingesammelt*. Dies bedeutet, dass die Daten des RDDs, welche vorher verteilt auf allen Executors im Arbeitsspeicher geladen wurden, nun jetzt an den Spark-Driver geschickt werden. Dieser sammelt sozusagen die Ergebnisse der Executors ein. Diese Mechanismus ist an sich nicht problematisch und funktioniert auch gerade beim Testen mit kleinen Datenmengen. Bei großen RDDs hingegen, werden auch wieder alle Daten an der Driver geschickt und in den meisten Fällen wird dies den begrenzten Arbeitsspeicher des Drivers überfordern. Die Applikation wird mit einer `OutOfMemoryException` beendet!.

Daher ist es sinnvoll auch schon beim Testen mit kleinen RDDs vorzugsweise die `take()`-Methode zu nutzen. Diese tut das gleiche wie, die `collect()`-Methode mit dem Zusatz, dass sie nur eine bestimmte Anzahl von Einträgen sammelt. Dadurch wird selbst bei größeren RDDs der Speicher nicht ausgehen.

```
1
2 HbaseReader hbr = new HbaseReader(jsc, hbaseConfigFile);
3 JavaRDD<Metadata> forensicMetadata = hbr.getForensicMetadata();
4
5 # use collect() method
6 forensicMetadata.collect().stream()
7   .forEach(m -> LOGGER.info("Entry = .", m));
```

```

8
9 # use take(int amount) method
10 forensicMetadata.take(10).stream()
11 .forEach(m -> LOGGER.info("Entry = .", m));

```

Listing 4.5: Spark Java RDD collect()-Methode

4.5 Forensische Anforderungen

4.5.1 Plattform absichern

Ursprünglich spielt das Thema der Datensicherheit bei Apache Hadoop keine Rolle und gewann erst nach und nach an Relevanz. Anfänglich wurde immer angenommen, dass das Hadoop-Clusters aus vertrauenswürdigen Maschinen besteht, welche von vertrauenswürdigen Nutzern in abgesicherten Umgebungen verwendet wird¹². Mittlerweile hat sich der Bedarf nach Sicherheit deutlich erhöht, da oftmals riesige vertrauliche Datensätze verarbeitet werden, welche bei Angriffen sehr schnell abfließen könnten.

Todo: Das Absichern des Hadoop-Clusters bezieht sich primär auf die Nutzung von Kerberos zur Authentifizierung. Es gibt etliche weitere Projekte, wie beispielsweise Apache Ranger, Apache Atlas und Apache Knox. Sie alle adressieren einen bestimmten Aspekt zur Verbesserung der Systemsicherheit. Allerdings werde ich mich hauptsächlich auf den Einsatz von Kerberos beschränken und prüfen, welche Vorteile diese Lösung bietet und welche Probleme dabei auftauchen können. Darüber hinaus ist es meines Wissens auch möglich, die Daten auf logischer Ebene zu verschlüsseln (im verteilten Dateisystem HDFS). Dies würde einen unbefugten physischen Zugriff erschweren. Diesen Punkt werde ich für die Thesis als optionales Arbeitspaket im Hinterkopf behalten. Wahrscheinlich werde ich mit den anderen Themen aber schon genügend Arbeit haben.

Authentifizierung

Standardmäßig wird Hadoop in Kombination mit Kerberos verwendet, um einen allgemeinen Zugriffsschutz zu ermöglichen.[2]

Eine Alternative könnte hier auch Cloudera Sentry, Apache Ranger, Apache Atlas oder Apache Knox¹³ sein.

Datenverschlüsselung

Prinzipiell lässt sich die Datenverschlüsselung in die Szenarien *Persistenzverschlüsselung* und *Transportverschlüsselung* unterteilen. Das HDFS bietet eine Verschlüsselung an, wobei die Komplexität bei Key-Management liegt. Denn schließlich kann ein Hadoop-Cluster mehrere hundert Knoten mit jeweils mehreren Datenträger enthalten. Sie alle müssten eigene Verschlüsselungsschlüssel nutzen. Die Verschlüsselung selbst kann direkt auf Betriebssystemebene beispielsweise auf LUKS aufbauen, oder sie findet auf logischer Ebene im HDFS statt.[2]

¹²Vgl. <https://www.infoq.com/articles/HadoopSecurityModel>.

¹³<https://knox.apache.org/>

Darüber hinaus ist die Transportverschlüsselung auch möglich. So müssen die einzelnen Services wie Webzugriffe mit TLS verschlüsselt werden.¹⁴

Letztlich stellt sich die Frage, welche Angriffe den mit Datenverschlüsselung vermieden werden sollen.

4.6 Visualisierung der Ergebnisse

Ziel dieser forensischen Analyseplattform ist es, dem Nutzer einen Überblick bei der Datensichtung zu geben. Hierbei ist es essentiell entsprechende Visualisierungen zu verwenden. Welche Ziele sollen erreicht werden?

- Für jede Datei sollen Name, Pfad, Größe, Hashsumme, Dateityp, Owner und Group, Zugriffsrechte und die Zeitstempel der Erstellung und letzter Speicherung angezeigt werden.
- Nach all diesen Parametern kann auch gesucht werden.
- Auffinden von Duplikaten anhand der Hashsummen
- Indizierung für schnelle textbasierte Inhaltssuche?
- Zeitleiste? (wohl eher optional)
- Wordcloud, geographische Visualisierung, Flare-Chart, Tree-Map, Calendar-Chart als Timeline?
- Webframeworks wie <https://d3js.org/>¹⁵
- Neo4j
- Open Source Community Variante Helical Insight
- Apache Superset für Visualisierung (siehe Ambari Cluster Services)
- Apache Grafana?

¹⁴Weiter Infos unter: <https://www.infoq.com/articles/HadoopSecurityModel> und <https://community.hortonworks.com/articles/102957/hadoop-security-concepts.html>.

¹⁵Siehe auch <https://bl.ocks.org/mbostock/4063550> oder <https://bl.ocks.org/mbostock/5944371> oder <https://bl.ocks.org/mbostock/1046712> oder <https://bl.ocks.org/mbostock/4063269>. Letzteres wäre charakteristisch für foAm. oder <http://xliberation.com/googlecharts/d3concept.html>

5 Zusammenfassung

6 Ausblick

Im praktischen Teil

Literatur

- [1] Jonas Freiknecht. *Big Data in der Praxis*. 1. Auflage. Carl Hanser Verlag, 2014.
- [2] Joey Echeverria Ben Spivey. *Hadoop Security: Protecting Your Big Data Platform*. 1. Auflage. O'Reilly Media, Inc, 2015.
- [3] Sam R. Alapati. *Expert Hadoop Administration*. 1. Auflage. Addison Wesley, 2016.
- [4] Krishna Sankar. *Fast Data Processing with Spark 2*. 3. Auflage. Packt Publishing, 2016.
- [5] o. V. *Apache Hadoop YARN*. Version 3.0.0. Apache Software Foundation. 8. Dez. 2017. URL: <https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html> (besucht am 19.03.2018).
- [6] o. V. *HDFS Architecture*. Version 3.0.0. Apache Software Foundation. 8. Dez. 2017. URL: <https://hadoop.apache.org/docs/r3.0.0/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html> (besucht am 17.03.2018).

Abbildungsverzeichnis

1.1	Datenverarbeitung im Hadoop-Umfeld	5
2.1	Arbeitspakete der Masterthesis	8
2.2	Projektplan Teil A - Einarbeitung und Rohdatenspeicherung (siehe Kapitel A.2)	9
2.3	Projektplan Teil B - Datenanalyse (siehe Kapitel A.2)	10
2.4	Projektplan Teil C - Querschnittliche Aspekte und Visualisierung (siehe Kapitel A.2)	11
2.5	Komponenten der Entwicklungsumgebung	12
3.1	Apache Hadoop Ökosystem (Vgl. [1],[3]. Siehe Kapitel A.2)	16
3.2	HDFS - Datenspeicherung im Verbund (Vgl. [6],[3])	19
3.3	Ressourcenverteilung mit YARN (Vgl. [5],[3])	20
4.1	HDFS - Dateieigenschaften	27

Tabellenverzeichnis

Listings

4.1	Befehl zum Speichern einer Datei im HDFS	27
4.2	Befehl zum Hinzufügen und Auslesen von Metadaten	27
4.3	Befehl zum Setzen von Posix Capabilities	30
4.4	Nutzung von Bindfs zum Ändern von Dateirechten	30
4.5	Spark Java RDD collect()-Methode	32
B.1	Konfiguration des Hadoop-Frameworks	43

A Anhang A

A.1 Analyse ähnlicher Projekte und Produkte

Im Bereich der IT-Sicherheit und Incident Response existiert für Unternehmensinfrastrukturen das Apache Projekt *Metron*, welches auf dem Hadoop Framework aufbaut.¹

Ziel dieses Projektes ist es Sicherheitsvorfälle zu finden und zu analysieren. Hierbei kann Apache Metron auch mit Telemetriedaten umgehen.²

Eine entsprechende Abgrenzung zu diesem Projekt besteht aufgrund der unterschiedlichen Projektziele. Diese Thesis bezieht sich auf die forensische Analyse von Beweismitteln und informationstechnischen Systemen. Es ist nicht das Ziel Sicherheitsvorfälle in unternehmenskritischen Infrastrukturen zu analysieren.

Das Open-Source Framework *Turbinia* ist ein weiteres Projekt, welches ähnliche Ziele verfolgt.³ Der Grundgedanke ist die Automatisierung und Skalierung forensischer Analysen in Computer-Clustern. Prinzipiell hat dieses Projekt das gleiche Ziel, wie diese Masterthesis. Aufwendige Analysen sollen parallelisiert verarbeitet werden, um sie schneller zu verarbeiten. Das Projekt ist aktiv⁴. Allerdings ist es jedoch in einer frühen Alpha-Phase und daher noch nicht ausgereift. Dieses Projekt basiert auch auf einer Master-Client Architektur. Es bietet aber keine Nutzung auf Basis eines verteilten Dateisystems an. Es muss dafür gesorgt werden, dass jeder Knoten auf alle verfügbaren Daten (Beweismittel) zugreifen kann. Im Rahmen dieser Thesis hingegen, wird durch die Nutzung von Apache Hadoop, eine verteilte Speicherung von Daten unterstützt. Darüber hinaus werden entwickelte Algorithmen dort ausgeführt, wo die Daten liegen und nicht umgekehrt.

Ein klassisches Analyse-Werkzeug in der Forensik ist *Autopsy*. Es basiert auf *The Sleuth Kit* und ist kostenlos.⁵ Mit dem Werkzeug können Hashsummen berechnet oder auch Multimediadateien analysiert werden. Autopsy ist ein Single-Node Analyseprogramm und läuft vorzugsweise auf einem eigenen Analyserechner pro Nutzer.

Es gibt auch die Möglichkeit das Programm kollaborativ zu verwenden. Dabei gibt es einen zentralen Netzwerkspeicher, welcher alle Beweismittel enthält. Es ist möglich mit mehreren Nutzern parallel am gleichen Fall zu arbeiten und Analyseergebnisse in Echtzeit zu teilen. Diese Art der verteilten Analyse zeigt Ähnlichkeiten zu dieser Thesis auf.

Allerdings geht es bei diesem kollaborativen Ansatz vielmehr darum, an einem großen Fall mit mehreren Nutzern zu arbeiten und Ergebnisse einfacher zusammenzutragen. Einzelne

¹Siehe <https://metron.apache.org/> (Stand: 5.3.2018).

²Siehe <https://www.heise.de/developer/meldung/Cybersecurity-Apache-Metron-wird-Top-Level-Projekt-3695901.html> (Stand: 5.3.2018)

³Siehe <https://github.com/google/turbinia> (Stand 5.3.2018).

⁴Dies ist daran erkennbar, dass der letzte Commit in das Github-Repository am 26.01.2018 erfolgte.

⁵Siehe <https://www.sleuthkit.org/autopsy/> (Stand 5.3.2018).

Analysen finden aber immer nur auf einem konkreten Analyserechner statt. Ein parallele Verarbeitung durch eine horizontale Skalierung wird durch die Anzahl parallel arbeitender Nutzer geschaffen. Jedoch kann das System nicht automatisiert einzelne Analysen auf allen verfügbaren Knoten verarbeiten, wie es in dieser Thesis geplant ist.

Autopsy selbst bietet keine Möglichkeiten forensische Analysen im Cluster durchzuführen. Allerdings gibt es eine Variante des *The Sleuth Kits*, welche das gleiche Ziel verfolgt, wie in dieser Thesis. Hierbei wird die Funktionalität des *The Sleuth Kits* in einen Apache Hadoop Cluster übertragen (siehe https://www.sleuthkit.org/tsk_hadoop/index.php). Das Projekt selbst nutzt eben Apache Hadoop und auch Apache HBASE zur Speicherung von Datenträgern im Cluster. Zur Prozessierung der Daten wird allerdings nicht Apache Spark genutzt, sondern das Hadoop interne Map-Reduce Verfahren. Darüber hinaus wurden seit 2012 keine Änderungen an dem Open-Source Projekt gemacht (siehe Source-Code Repository auf GitHub unter https://github.com/sleuthkit/hadoop_framework). Es ist nicht bekannt aus welchen Gründen die Datenverarbeitung im Cluster eingestellt wurde.

A.2 Lizenzierungen in dieser Arbeit

- Die dargestellten Gantt-Diagramme (siehe Abbildungen 2.2, 2.3, 2.4) wurden mit der JavaScript-Bibliothek *dhtmlxGantt* erstellt. Das Projekt selbst ist unter <https://github.com/DHTMLX/gantt> zu finden. Der Quellcode ist unter der *GNU GPLv2*-Lizenz lizenziert. Die aktuelle Bibliothek kann unter <https://dhtmlx.com/docs/products/dhtmlxGantt/download.shtml> heruntergeladen werden. Stand: 21.3.2018.

Nachfolgend werden die Logos aufgelistet, welche in Abbildung 3.1 dargestellt werden. Die Logos der Projekte und die Projektnamen sind Handelsmarken der Apache Source Foundation (siehe <https://www.apache.org/>). Sie dürfen in Publikationen genutzt werden.⁶

- Apache AmbariTM Logo von <https://ambari.apache.org/>, Stand 21.3.2018.
- Apache Hadoop[®] Logo von <https://hadoop.apache.org/>, Stand 21.3.2018.
- Apache SparkTM Logo von <https://spark.apache.org/>, Stand 21.3.2018.
- Apache HBASE[®] Logo von <https://hbase.apache.org/>, Stand 21.3.2018.
- Apache HiveTM Logo von <https://hive.apache.org/>, Stand 21.3.2018.
- Apache ZookeeperTM Logo von <https://zookeeper.apache.org/>, Stand 21.3.2018.

⁶Siehe auch <https://www.apache.org/foundation/marks/>, Stand: 21.3.2018.

B Hadoop Konfigurationen

B.1 Aufsetzen des aktuellen Hadoop-Frameworks

Listing B.1 zeigt die Schritte zum Konfigurieren des Hadoop-Frameworks

```
1  #Following code works for Fedora 27
2
3  #Create a new Hadoop user
4  sudo groupadd hadoop
5  sudo adduser -g hadoop hduser
6
7  #Change account
8  #get root
9  sudo -i
10 #change to hduser without setting any password for hduser
11 sudo -l hduser
12
13 #Create a ssh access without password!
14
15 ssh-keygen -t rsa -P '' -f ~/.ssh/id_rsa
16 cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
17 chmod 0600 ~/.ssh/authorized_keys
18
19 #Store initial fingerprint
20 ssh localhost
21
22
23 #Unzip latest hadoop version (3.0.0)
24
25 #Change owner to hduser
26 sudo chown hduser:hadoop -R hadoop-3.0.0
27
28 #Configure
29
30
31 #Start HDFS
32 ./sbin/start-dfs.sh
```

Listing B.1: Konfiguration des Hadoop-Frameworks