

HOCHSCHULE ...

STUDIENGANG ...

---

Masterthesis

**Aufbau einer Plattform zur  
forensischen Analyse basierend auf dem  
Apache Hadoop<sup>®</sup> Framework**

---

Zur Erlangung des akademischen Grades  
Master of Science

vorgelegt im Sommersemester 2018

von  
Johannes Busam

Erstbetreuung: ...

Zweitbetreuung: ...

## Kurzfassung

Die digitale Forensik ist heutzutage ein wichtiger Teil bei der Aufklärung von Straftaten durch Ermittlungsbehörden. Nahezu in jedem Fall können digitale Beweismittel sichergestellt werden, weil informationstechnische Systeme im Alltag allgegenwärtig sind. Durch den rasanten Fortschritt von IT-Technologien, müssen auch die forensische Methoden immer wieder neu adaptiert und verbessert werden. Allein die Datenmenge, welche durch Mobiltelefone, Computer, Netzwerkspeicher und Server zusammenkommt kann durchaus im Terabyte-Bereich liegen.

Um solche Datenmengen schnell und effizient bearbeiten zu können, wird in dieser Thesis eine forensische Analyseplattform entwickelt. Sie soll anfallende Daten parallelisiert in einem Computer-Cluster speichern und aufbereiten können. Zuerst soll geprüft werden, wie diese Daten verteilt gespeichert werden können. Im nächsten Schritt sollen, aufbauend auf dem existierenden Datenbestand, neue Informationen extrahiert werden können. Hierbei sollen auch forensische Aspekte, wie beispielsweise die Erstellung einer Beweismittelkette und die Sicherheit des Computer-Clusters betrachtet werden.

Zur Datenspeicherung soll das Apache Hadoop®-Ökosystem genutzt werden. Als Ausgangslage werden mehrere Datenträgerabbilder analysiert. Hierbei zeigt sich schnell, dass diese Abbilder nicht direkt in die Analyseplattform importiert werden können, da sonst keine parallele Verarbeitung möglich wäre. In einer weiteren Variante werden die Daten auf logischer Dateiebene, Datei für Datei, in die Analyseplattform importiert. Aber auch dieser Ansatz hat Schwächen, da die originalen Dateimetadaten nicht performant gespeichert werden können.

Die finale Variante zur Datenspeicherung ist eine Mischung von zwei Datenspeicherungen. So werden die Dateimetadaten und kleine Dateien in einer spaltenorientierten Datenbank gespeichert. Wohingegen große Dateien im verteilten Dateisystem der Hadoop-Plattform abgelegt werden. Damit ist es auch möglich extrahierte Informationen aus den bestehenden Daten in der Datenbank abzuspeichern.

Bei der Datenverarbeitung wird Apache Spark genutzt, welches die Daten auf der Hadoop-Plattform parallel prozessieren kann. Es werden beispielsweise Hashsummen berechnet und Medientypen ermittelt.

Ein weiterer Aspekt in diesem Kontext ist die Implementierung einer Volltextsuche, zum performanten Zugriff der Daten. Hierzu werden die Daten im Computer-Cluster verteilt indexiert, um sie schneller durchsuchen zu können.

Letztlich ist es möglich beliebige Datenträger mit der hier entwickelten forensischen Analyseplattform zu verarbeiten. Auch ein performanter Zugriff auf die Daten ist möglich. Allerdings gibt es auch Schwächen dieser Plattform, welche zukünftig behoben werden sollen. So benötigt allein das Importieren großer Datenträger in das Analysesystem zu viel Zeit. Hier wäre es sinnvoll die Datenverarbeitung so umzustellen, dass einzelne Daten sofort prozessiert und angezeigt werden, um damit schon auf Teilergebnisse zugreifen zu können.

## **Abstract**

TODO: write abstract

## **Danksagung**

TODO: Danksagung schreiben

### **Eidesstattliche Erklärung**

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit selbstständig und ohne Verwendung anderer als der angegebenen Hilfsmittel angefertigt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten und nicht veröffentlichten Schriften entnommen sind, sind als solche kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch in keiner anderen Prüfungsbehörde vorgelegen. Alle eingereichten Versionen der Arbeit sind identisch.

Meersburg, den XX.XX.2018

Johannes Busam

# Inhaltsverzeichnis

<b>1 Einleitung</b>	<b>1</b>
1.1 Problemstellung . . . . .	1
1.2 Zielsetzung . . . . .	2
1.3 Aufbau . . . . .	4
<b>2 Vorgehen</b>	<b>5</b>
2.1 Projektplanung . . . . .	5
2.2 Entwicklungsumgebung . . . . .	10
2.3 Testdatengenerierung . . . . .	12
<b>3 Grundlagen von Apache Hadoop®</b>	<b>13</b>
3.1 Apache Hadoop Framework . . . . .	13
3.2 Apache Hadoop HDFS . . . . .	15
3.3 Apache Hadoop YARN . . . . .	17
3.4 Apache Spark . . . . .	19
3.5 Apache HBASE . . . . .	22
3.6 Apache ZooKeeper . . . . .	25
3.7 Apache Solr und Lucene . . . . .	26
<b>4 Datenspeicherung</b>	<b>30</b>
4.1 Allgemeiner forensischer Analyseprozess . . . . .	30
4.2 Herkömmliches Analysevorgehen . . . . .	33
4.3 Umsetzung in der Hadoop Analyse-Plattform . . . . .	36
4.4 Variante 1 - Datenträgerabbild im HDFS speichern . . . . .	36
4.5 Variante 2 - Logische Dateien im HDFS speichern . . . . .	38
4.6 Variante 3 - Speicherung in Dateicontainer . . . . .	41
4.7 Variante 4 - Speicherung mit HBASE und HDFS . . . . .	44
4.7.1 Speicherung kleiner Dateien . . . . .	44
4.7.2 Anwendungsimplementierung . . . . .	48
4.7.3 Datenmodell . . . . .	51
4.7.4 Datenspezifische Aspekte . . . . .	52
4.7.5 Zugriffs und Ausführungsrechte . . . . .	53
4.8 Fazit . . . . .	55
<b>5 Datenverarbeitung</b>	<b>56</b>
5.1 Herkömmliches Analysevorgehen . . . . .	56
5.2 Verarbeitung Apache Spark . . . . .	58
5.2.1 Praxisbeispiele und deren Optimierungen . . . . .	59

5.3	Anwendungsfälle der Datenverarbeitung . . . . .	60
5.3.1	Hashsummen ermitteln . . . . .	60
5.3.2	Duplikate erkennen . . . . .	61
5.3.3	Dateityp erkennen mit Apache Tika . . . . .	61
5.3.4	Dateien indizieren . . . . .	61
5.4	Leistungsanalyse . . . . .	62
<b>6</b>	<b>Forensische Anforderungen</b>	<b>63</b>
6.1	Beweismittelkette . . . . .	63
6.2	Plattform absichern . . . . .	63
6.2.1	Authentifizierung . . . . .	63
6.2.2	Datenverschlüsselung . . . . .	64
<b>7</b>	<b>Visualisierung der Ergebnisse</b>	<b>65</b>
<b>8</b>	<b>Diskussion der Ergebnisse</b>	<b>66</b>
<b>9</b>	<b>Zusammenfassung</b>	<b>67</b>
<b>10</b>	<b>Ausblick</b>	<b>68</b>
<b>A</b>	<b>Allgemeines</b>	<b>74</b>
A.1	Analyse ähnlicher Projekte und Produkte . . . . .	74
A.2	Lizenzierungen in dieser Arbeit . . . . .	75
<b>B</b>	<b>Datenimport</b>	<b>76</b>
B.1	Konfigurationsdateien . . . . .	76

# 1 Einleitung

## 1.1 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 Straftat werden gerade auch im Bereich der Wirtschaftskriminalität dutzende Asservate über informationstechnische Systeme erhoben. Beispielsweise werden beteiligte Computer und Mobiltelefone sichergestellt. Oder es werden logische Sicherungen von Netzwerkspeichern durchgeführt.

Bei der Analyse dieser Asservate 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 proprietäre 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 auch in anderen spezialisierten Analyseanwendungen weiterzuverarbeiten.

## 1.2 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 Forensiker helfen, sich einen Überblick zu verschaffen. Er 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 parallele Datenverarbeitung eignet sich ein Hadoop-Cluster auch zu Prozessierung von großen 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.

Ein wichtiger Aspekt der Masterthesis ist die Aufbereitung der Daten für die Analyse im Hadoop-Cluster. Hierbei sollen mehrere Möglichkeiten analysiert werden, wie diese Aufbereitung und Speicherung der Daten im Hadoop-Cluster gelingen kann. Dabei muss auch auf die Unversehrtheit der Dateiinhalte und Metadaten bei der Aufbereitung geachtet werden. Darauf aufbauend soll eine fachliche Verwaltungsstruktur entwickelt werden, die es auch erlaubt mehrere Asservate von beliebigen Ermittlungen parallel zu verarbeiten. Dadurch können auch Zusammenhänge zwischen unterschiedlichen Asservaten identifiziert werden.

Im Rahmen der Thesis soll die Datenanalyse vorerst grundlegende Funktionen unterstützen. So sollen die Metadaten, wie beispielsweise Name, Dateipfad, Hashsumme, Dateityp, Größe und Zeitstempel ermittelt werden und zu weiteren Analysen zur Verfügung stehen. 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, Web-Adressen, E-Mail-Adressen oder Positionsdaten<sup>1</sup> suchen.

Die Resultate durchgeföhrter Datenanalysen sollen dem Nutzer bereitgestellt werden. Hierzu wird eine grafische Oberfläche benötigt, welche die fachlichen Aspekte der forensischen Analyseplattform widerspiegelt. Im Rahmen der Thesis sollen Möglichkeiten analysiert werden, wie eine grafische Oberfläche aussehen könnte. In diesem Kontext soll auch geprüft werden, ob existierende Programme zur Datenvisualisierung im Hadoop-Umfeld wiederverwendet werden könnten. Die Implementierung einer grafischen Oberfläche ist im Rahmen dieser Thesis jedoch nicht angedacht.

Abbildung 1.1 skizziert das angestrebte Analysevorgehen mit dieser Analyseplattform. Der Forensiker soll digitale Beweismittel in das Hadoop-Cluster importieren können. Nachfolgend hat er die Möglichkeit diverse Analysen auf den Daten durchzuführen. Die Ergebnisse könnten später über eine entsprechende Oberfläche visualisiert werden.

---

<sup>1</sup>Beispielsweise könnten Geopositionen oder Ortsnamen aus Dateien extrahiert werden. Diese Daten könnten dann mit ihrem geografischen Bezug auf einer Karte dargestellt werden.

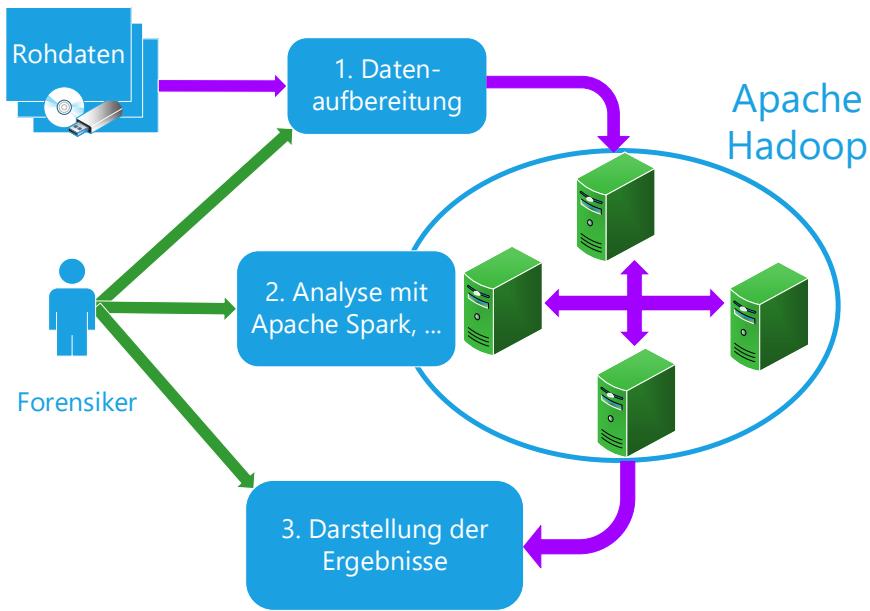


Abbildung 1.1: Analysevorgehen

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.

Bei einer realen forensischen Analyse gibt es weitere Anforderungen, die das Analysesystem erfüllen sollte. 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.

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

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.

Im Rahmen der Masterthesis soll diese Aspekte zumindest theoretisch und wenn möglich auch praktisch analysiert werden.

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 einer öffentlich zugänglichen Versionsverwaltung verfügbar sein.

### 1.3 Aufbau

In Kapitel 1 wird das Eingangsproblem und die Ziele dieser Masterthesis beschrieben.

In Kapitel 2 folgt das allgemeine Entwicklungsvorgehen. Darin ist der aktuelle Projektplan enthalten, welcher die Arbeitspakete definiert. Zusätzlich wird die genutzte Entwicklungsumgebung skizziert.

In Kapitel 3 erfolgt eine Darstellung der Apache Hadoop Plattform inklusive theoretischen Grundlagen zur Arbeitsweise des Frameworks. Des Weiteren werden darauf aufbauende Projekte, wie beispielsweise Apache Spark und Apache HBase erklärt.

In Kapitel 4 werden unterschiedliche Varianten zur Datenspeicherung und Aufbereitung analysiert. Innerhalb dieses Kapitels wird eine Möglichkeit entwickelt, wie die Daten eines Asservats im Hadoop-Cluster gespeichert werden können, um sie später parallelisiert verarbeiten zu können. Zu Beginn wird das herkömmliche Analysevorgehen in Verbindung mit der Analyseanwendung *Autopsy* beschrieben, um fachlich relevante Aspekte bei der Analyse herauszuarbeiten.

Die eigentliche Datenverarbeitung wird Kapitel 5 beschrieben. Hier wird ein Ansatz vorgestellt, wie die Daten parallel verarbeitet werden können. Anhand dieses Ansatzes werden Hashsummen der Daten berechnet und die Medientypen der Dateien ermittelt. Zum Schluss wird eine Möglichkeit vorgestellt, wie die Informationen für eine Volltextsuche aufbereitet werden können.

In Kapitel 7 werden querschnittliche Aspekte zur Datensicherheit und zur Beweismittelkette von forensischen Analysen skizziert.

Die Visualisierung der Informationen ist ein interessanter Aspekt der forensischen Analyseplattform, welcher in Kapitel 6 beschrieben wird. Dort werden einige Möglichkeiten erläutert, wie eine Visualisierung aussehen könnte.

In Kapitel 8 werden die gewonnenen Ergebnisse dieser Masterthesis kritisch hinterfragt. Hierbei wird geprüft, ob die Erwartungen an eine performante Datenaufbereitung und Analyse erfüllt werden. Es wird auch aufgezeigt, ob das Apache Hadoop Framework überhaupt die Anforderungen einer forensischen Analyseplattform erfüllen kann.

Zuletzt erfolgt in Kapitel 9 eine Zusammenfassung der erarbeiteten Ergebnisse. Offene Punkte und Verbesserungen des Systems werden in Kapitel 10 diskutiert.

# 2 Vorgehen

## 2.1 Projektplanung

Zur Realisierung einer forensischen Analyseplattform wurde ein Projektplan erstellt, welcher die einzelnen Aufgaben im Rahmen der Masterthesis enthält. Abbildung 2.1 zeigt die Aufteilung in diese 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. Danach erfolgt der Aufbau eines eigenen Hadoop-Clusters und die Beschaffung von Testdaten.<sup>1</sup> Für die Einarbeitung und den Aufbau sind vier Wochen eingeplant (siehe Abbildung 2.2).<sup>2</sup>

Der zweite Teil behandelt die Datenaufbereitung und Speicherung im Hadoop-Cluster. Es soll geprüft werden, welche Struktur der Daten für eine optimale Speicherung und Verarbeitung im Hadoop-Framework erforderlich ist. Für diesen Teil sind vier Wochen Bearbeitungszeit geplant.

Am Ende des Arbeitspaketes soll ein erster Zwischenbericht erstellt werden, welcher die bisherigen Ergebnisse enthält (sieh 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. Ein weiterer Aspekt der Datenanalyse beschäftigt sich mit den Möglichkeiten, wie die Ergebnisse persistiert werden können.<sup>3</sup> 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 querschnittliche Aspekte in die bestehende Analyseplattform integriert werden. Hierbei geht es um das Absichern der Plattform, die Dokumentation der Beweismittelkette und um das sichere Löschen von Asservaten. Für dieses Arbeitspaket sind vier Wochen eingeplant (siehe Abbildung 2.4).

---

<sup>1</sup>Hierbei wird ein bestehendes Hadoop-Cluster genutzt und um zusätzliche Softwarepakete ergänzt.

<sup>2</sup>Die 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.

<sup>3</sup>Dafür soll Apache HBase zur Speicherung von strukturierten und unstrukturierten Daten untersucht werden.

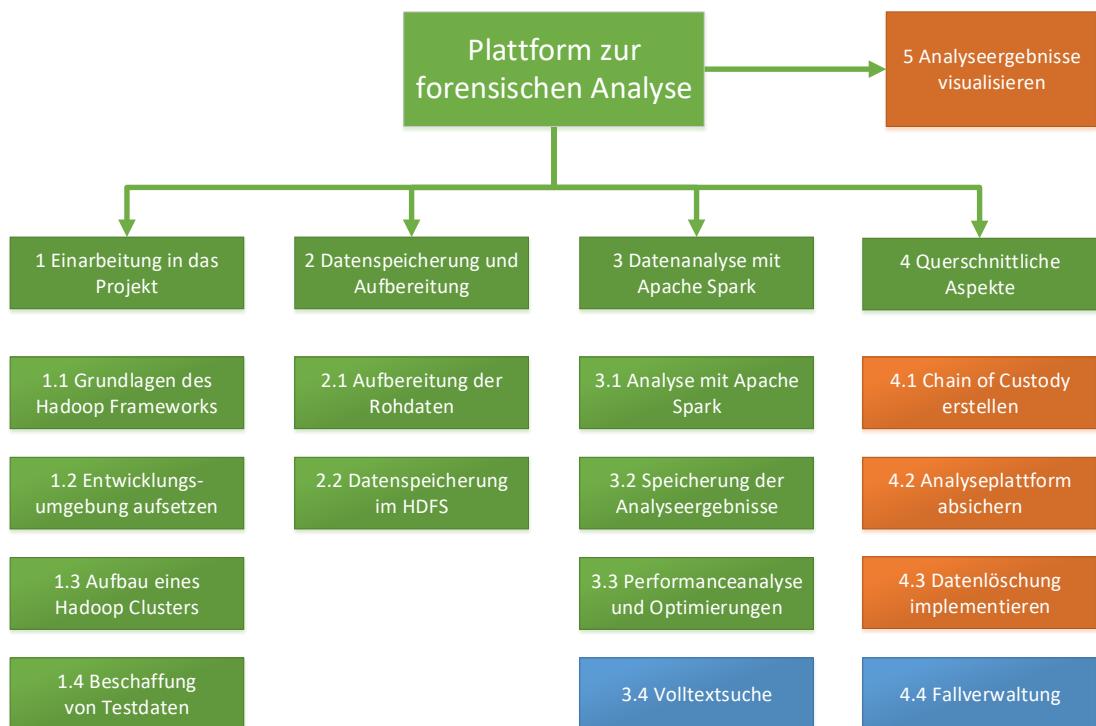


Abbildung 2.1: Arbeitspakete der Masterthesis

Das letzte Arbeitspaket enthält eine prototypische Visualisierung der Analyseergebnisse. Hierbei soll geprüft werden, welche Möglichkeiten zur Darstellung der Ergebnisse existieren. Für diese Arbeit sind drei Wochen eingeplant (siehe Abbildung 2.4).

## Projektverlauf

Während dem Projektverlauf wurde die Planung teilweise angepasst. Es wurden einige Aspekte aus der Planung entfernt (orange hinterlegt in Abbildung 2.1). So wurden die Visualisierung der Ergebnisse, die Erstellung der Beweismittelkette, das Absichern der Analyseplattform und die forensisch korrekte Datenlöschung nicht implementiert sondern nur theoretisch erläutert. Der Hauptgrund dafür war eine intensive Analyse und Entwicklung, wie die Daten im Hadoop-Cluster gespeichert werden können. Hier wurden mehrere Varianten getestet und die ursprünglich angedachte Bearbeitungszeit verlängerte sich.

Andererseits sind auch neue Arbeitspakete hinzugekommen (blau hinterlegt in Abbildung 2.1). So wurde bei der Datenanalyse mit Apache Spark sichtbar, dass die Informationen und Analyseergebnisse performant durchsuchbar sein müssen. Daher wurde untersucht, wie eine Volltextsuche aller gespeicherten Daten im Hadoop-Cluster realisiert werden könnte. Ein anderer Aspekt ist die Implementierung einer Fallverwaltung. Denn damit können nun mehrere Asservate in das System importiert werden, um Zusammenhänge identifizieren zu können.

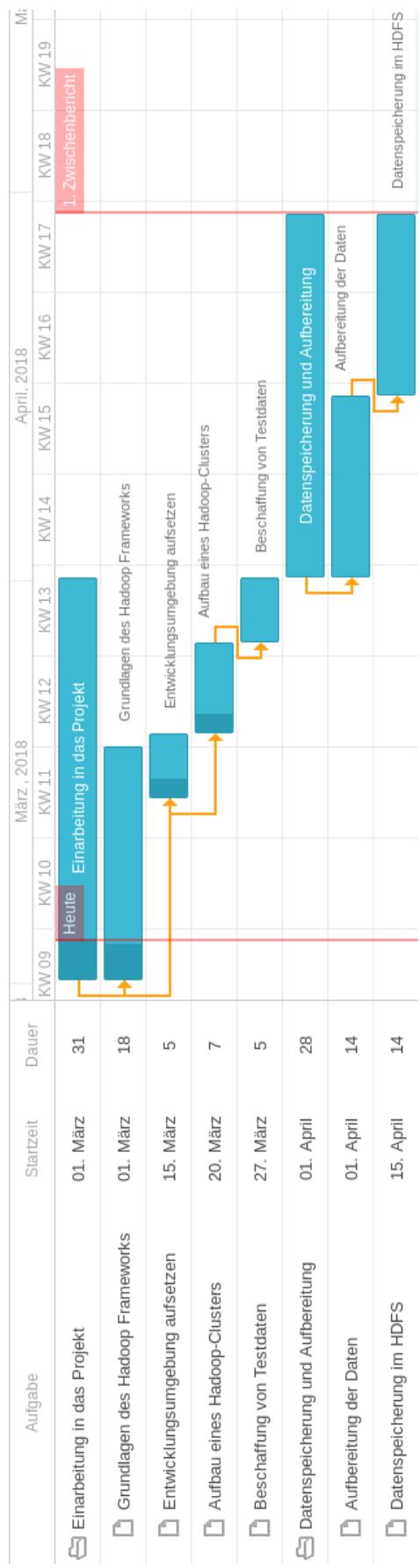


Abbildung 2.2: Projektplan Teil A - Einarbeitung und Rohdatenspeicherung (siehe Kapitel A.2)

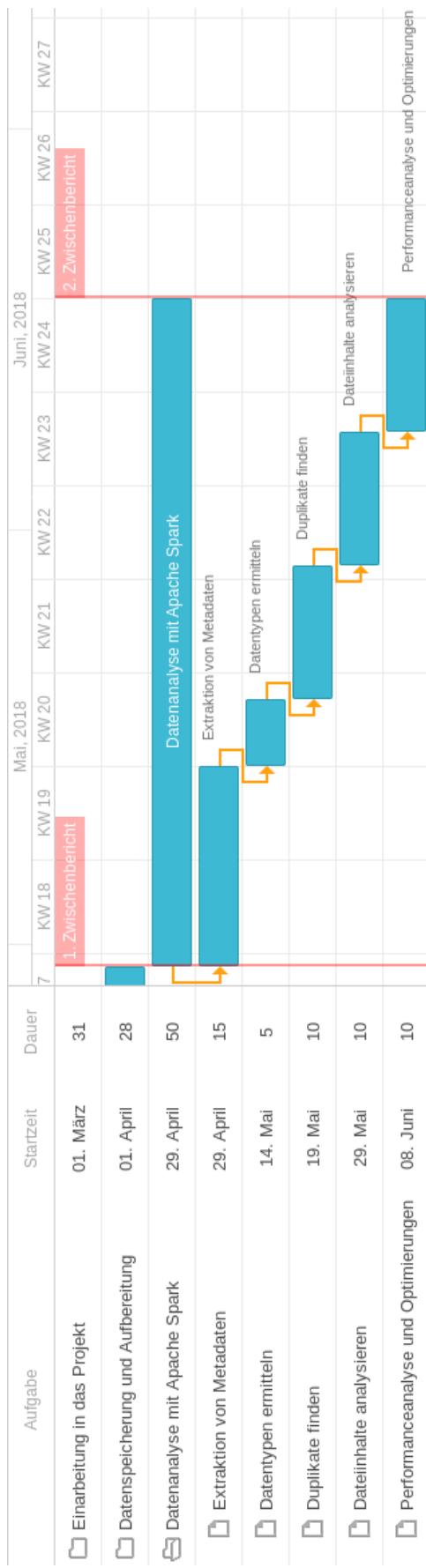


Abbildung 2.3: Projektplan Teil B - Datenanalyse (siehe Kapitel A.2)



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 28 64-bit.

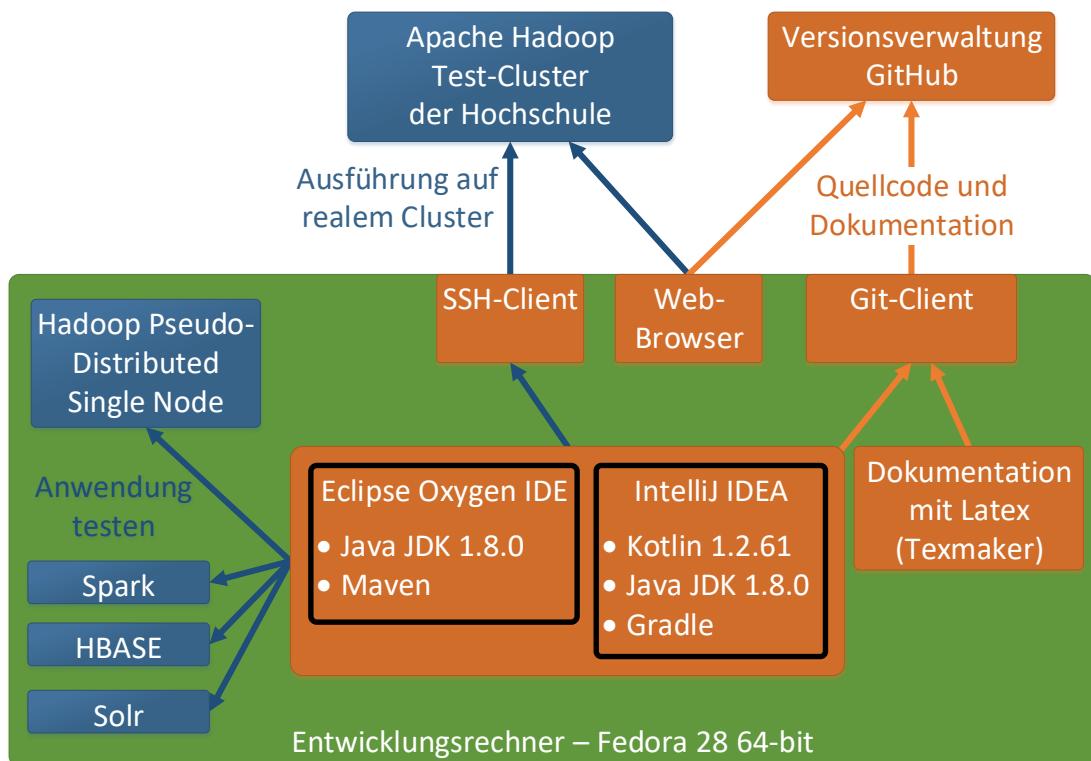


Abbildung 2.5: Komponenten der Entwicklungsumgebung

Zur Entwicklung der forensischen Analyseprogramme wird *Eclipse Oxygen* genutzt. Die Anwendungen selbst werden in Java geschrieben.<sup>4</sup> Zum Bauen der ausführbaren Java Archive (JAR) wird *Maven* verwendet. Mit Maven können weitere Java-Bibliotheken in eigenen Programmen auf einfache Weise wiederverwendet werden.<sup>5</sup>

Zusätzlich befindet sich die Entwicklungsumgebung *IntelliJ IDEA* in der kostenlosen Community Variante auf dem Entwicklungsrechner. Mit der IntelliJ IDEA wird die Datenimport-Anwendung in *Kotlin* entwickelt. Kotlin ist eine statisch typisierte Programmiersprache zur Anwendungsentwicklung auf verschiedenen Plattformen.<sup>6</sup> Sie ist interoperabel mit Java. Die Anwendungen können in der *Java Virtual Machine* (JVM) ausgeführt werden.

<sup>4</sup>Wobei auch Python oder Scala als Programmiersprache genutzt werden kann.

<sup>5</sup>Diese können über ein zentrales Repository, dem sogenannten *Maven Central Repository* aus dem Internet geladen werden (siehe Link <https://search.maven.org/>. Letzter Zugriff 26.8.2018).

<sup>6</sup>Siehe Link <https://kotlinlang.org/>. Letzter Zugriff: 24.8.2018.

Gegenüber Java bietet sie diverse Sprachkonstrukte zur Optimierung des Programmcodes an. Darüber hinaus können alle Bibliotheken aus dem Java-Umfeld auch in Kotlin genutzt werden. Zum Bauen der Kotlin-Anwendungen wird *Gradle* genutzt, welches analog zu Maven Abhängigkeiten zu Drittbibliotheken und deren Versionen verwaltet.<sup>7</sup>

Um die gebauten Java- und Kotlin-Programme schnell zu testen, können alle notwendigen Komponenten auch lokal auf dem Entwicklungsrechner gestartet werden. Hierzu gehört ein Hadoop-Knoten im sogenannten *Pseudo-Distributed* Modus, eine lokale Spark-Instanz, eine HBASE-Instanz und eine Solr-Instanz zur Datenindexierung.<sup>8</sup>

Mithilfe dieser Komponenten können auch spezifische Konfigurationen getestet werden.<sup>9</sup> Letztendlich kommen die lokalen Instanzen schnell an ihre Grenzen, gerade wenn größere Datenmengen analysiert werden sollen. Daher werden spezifische Konfigurationen und fertiggestellte Analyseprogramme auch auf einem realen Apache Hadoop-Cluster durchgeführt. Dort kann das Zusammenspiel zwischen den Komponenten nachvollzogen werden. Auch entsprechende Last-Tests sind nur auf dem Hadoop Test-Cluster möglich. Um mit dem Test-Cluster arbeiten zu können, wird ein SSH-Client benötigt. Zusätzliche 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*<sup>10</sup> 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.<sup>11</sup>
- Das Projekt *foam-data-import* enthält den Quellcode zum Importieren von Asservaten in das Hadoop-Cluster. Unter <https://github.com/jobusam/foam-data-import> befindet sich die Kotlin-Anwendung, welche wiederum mit Gradle gebaut werden kann.
- Das Projekt *foam-processing-spark* enthält den Quellcode zur Auswertung mit Apache Spark<sup>TM</sup>. 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.

---

<sup>7</sup>Siehe auch Kapitel 4.7.2 für weitere Informationen zur Datenimportanwendung.

<sup>8</sup>Siehe Kapitel 3 für eine detaillierte Erklärung der Komponenten.

<sup>9</sup>Hierfür muss der Entwicklungsrechner entsprechende Ressourcen bereitstellen. Es sollte mindestens eine Quad-Core-CPU, 16 GB Arbeitsspeicher und eine SSD zur Verfügung stehen, um performant arbeiten zu können.

<sup>10</sup>Die Abkürzung *foam* oder auch *foAm* steht für **forensische Analyseplattform**

<sup>11</sup>Das 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.

- Das Projekt *foam-storage-hadoop* enthält alle Konfigurationsdateien zum Aufsetzen eines Hadoop-Clusters auf einem einzelnen Knoten im *Pseudo-Distributed Mode*.<sup>12</sup> Zusätzlich existieren Shell-Skripte zum Starten des Hadoop-Clusters auf einem einzelnen Knoten.<sup>13</sup> Mithilfe der Skripte aus dem *foam-processing-spark* Projekt können damit Spark-Anwendungen ausgeführt werden.

Derzeit ist die Lizenzierung der Projekte noch unklar. 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.

### 2.3 Testdatengenerierung

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

---

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

<sup>13</sup>Siehe Link <https://github.com/jobusam/foam-storage-hadoop/tree/master/hadoop.standalone.setup>

# 3 Grundlagen von Apache Hadoop<sup>®</sup>

## 3.1 Apache Hadoop Framework

Apache Hadoop<sup>®</sup> 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.[4, S. 20 ff.]

Das Framework ist für die Ausführung auf Standardhardware konzipiert. 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 Datenhaltung entsprechend berücksichtigt.

Hadoop selbst besteht aus mehreren Komponenten, welche spezifische Aufgaben übernehmen. Abbildung 3.1 stellt eine grobe Skizzierung der Komponentenlandschaft der Plattform dar.<sup>1</sup>

Die Basis bildet das *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, läuft.

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.<sup>2</sup>

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

---

<sup>1</sup>In 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.

<sup>2</sup>Sogenannte 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 genutzt wird.

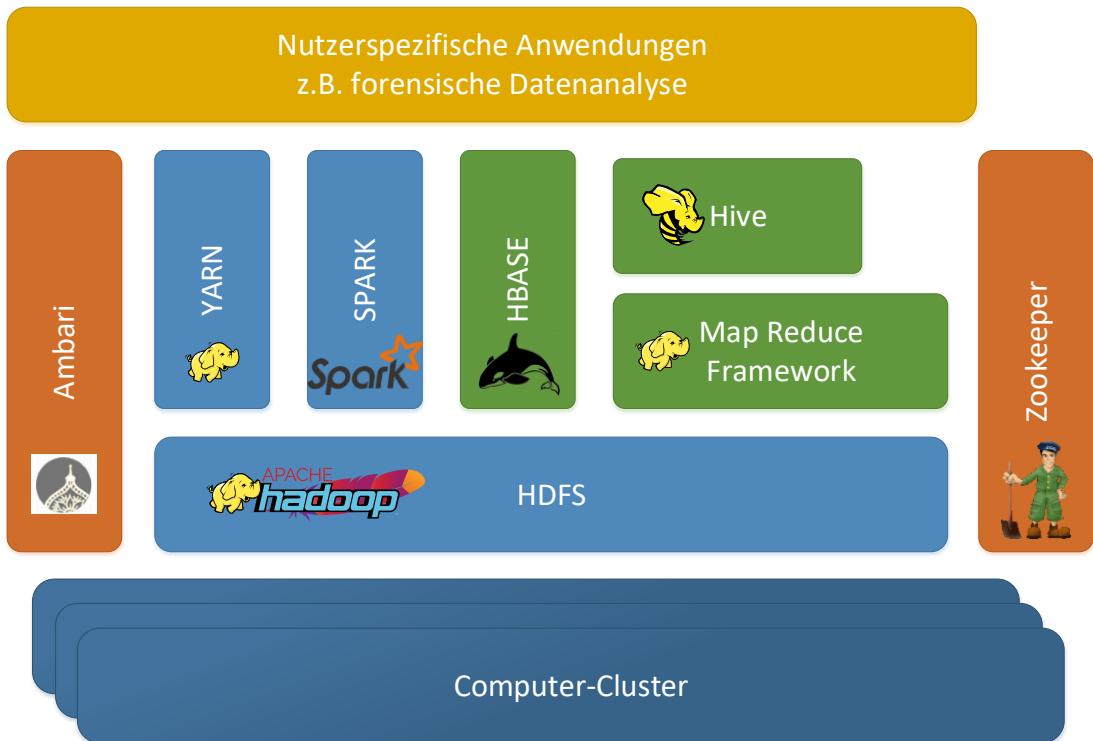


Abbildung 3.1: Apache Hadoop Ökosystem (Vgl. [4],[11]. Siehe Kapitel A.2)

oder weitere Funktionen anbieten. So wird beispielsweise in dieser Thesis *Apache Spark<sup>TM</sup>* 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.<sup>3</sup>

Des Weiteren bietet *Apache Hive<sup>TM</sup>* eine Möglichkeit Dateien im HDFS mithilfe einer SQL ähnlichen Syntax (HiveQL) 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<sup>®</sup>* 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 ZooKeeper<sup>TM</sup>* genutzt. Mit ZooKeeper ist es auch möglich Konfigurationen und Änderungen im Cluster zu verteilen. Zum eigentlichen konfigurieren und überwachen des Hadoop-Clusters wird *Apache Ambari<sup>TM</sup>* genutzt.

Zusätzlich existieren dutzende weitere Projekte die auf dem Hadoop-Ökosystem aufbauen oder sich integrieren lassen. Mit *Apache Accumulo<sup>®</sup>* existiert eine weitere Key-Value Datenbank, welche eine Alternative zu HBASE bietet. *Apache Livy* kann zur Ausführung von

<sup>3</sup>Durch das In-Memory Computing ist Apache Spark deutlich schneller als das bereits erwähnte Hadoop Map-Reduce.

Apache Spark Anwendungen über eine REST-Schnittstelle genutzt werden.<sup>4</sup> Apache NiFi hingegen ermöglicht das Aufbereiten von Daten und organisiert Datenimporte.

Im Rahmen dieser Thesis wird auch das bekannte Open-Source Projekt *Apache Solr<sup>TM</sup>* verwendet, um innerhalb des Hadoop-Ökosystems eine Datenindexierung für eine Volltextsuche durchzuführen.

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 darauf spezialisiert haben das Hadoop-Ökosystem inklusive weiterer Komponenten zu einzelnen Analyseplattformen zusammenzufassen. Sie bieten dafür kostenpflichtigen Support an, wobei diese Plattformen auch kostenfrei betrieben werden können. So wird im Praxisteil der Masterthesis beispielsweise die *Hortonworks Data Platform (HDP)* des Unternehmens *Hortonworks* genutzt.

### 3.2 Apache Hadoop HDFS

Das Hadoop Distributed File System (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 im HDFS 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 einem Knotenausfall. In großen Computer-Clustern mit mehreren hunderten Knoten ist ein Ausfall eines Knoten kein Sonderfall sondern die Regel. Daher muss es sich 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 Terabyte 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 Algorithmen vorgenommen werden, resultieren in neuen Datensätzen.

Darüber hinaus gilt das Prinzip der Datenlokalität. Algorithmen werden dort ausgeführt,

---

<sup>4</sup> *Representational State Transfer (REST)* bezeichnet ein Programmierparadigma in verteilten Systemen. Hierbei werden Ressourcen über das Hypertext Transfer Protocol (HTTP) angefordert, gespeichert und verarbeitet.

wo die Daten liegen, um das Transportieren von Daten über das Netzwerk zu vermeiden.[16]

Der Aufbau eines HDFS bildet eine Master-Slave Architektur aus einem *Name Nodes* und mehreren *Data Nodes*. Der Name Node ist einmalig im verteilten System vorhanden und enthält alle Metainformationen zu den Dateien. Eine Datei selbst wird in ein oder mehrere Blöcke aufgeteilt und auf mehreren Data Nodes gespeichert. Der Name Node organisiert diese Speicherung und bestimmt, wo welche Daten persistiert werden. Über den Name Node 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 eine Datei im HDFS unter `/home/foo.txt` gespeichert werden. Dazu kann ein HDFS-Client genutzt werden, welcher Zugang zum Hadoop-Cluster hat. Der HDFS-Client speichert zuerst die Metadaten der Datei auf dem Name Node. Der Name Node kennt die Größe der Datei und entscheidet, in wie viele Blöcke sie unterteilt werden soll. Er ermittelt für jeden einzelnen Block, auf welchen Data Nodes dieser Block gespeichert werden soll. Diese Blockaufteilung und die Zuordnung zu den Data Nodes werden an den HDFS-Client zurückgeschickt. Dieser übermittelt die Blöcke an einen der Data Nodes. Sobald ein Data Node einen Block empfangen hat, schickt er diesen Block auch an die Knoten, welche eine Replikation des Blocks speichern sollen. Die Data Nodes stehen in Kontakt zum Name Node und senden Informationen über ihren Zustand und die momentan gespeicherten Blöcke. Der Name Node bekommt dadurch auch mit, wann ein Data Node ausfällt.

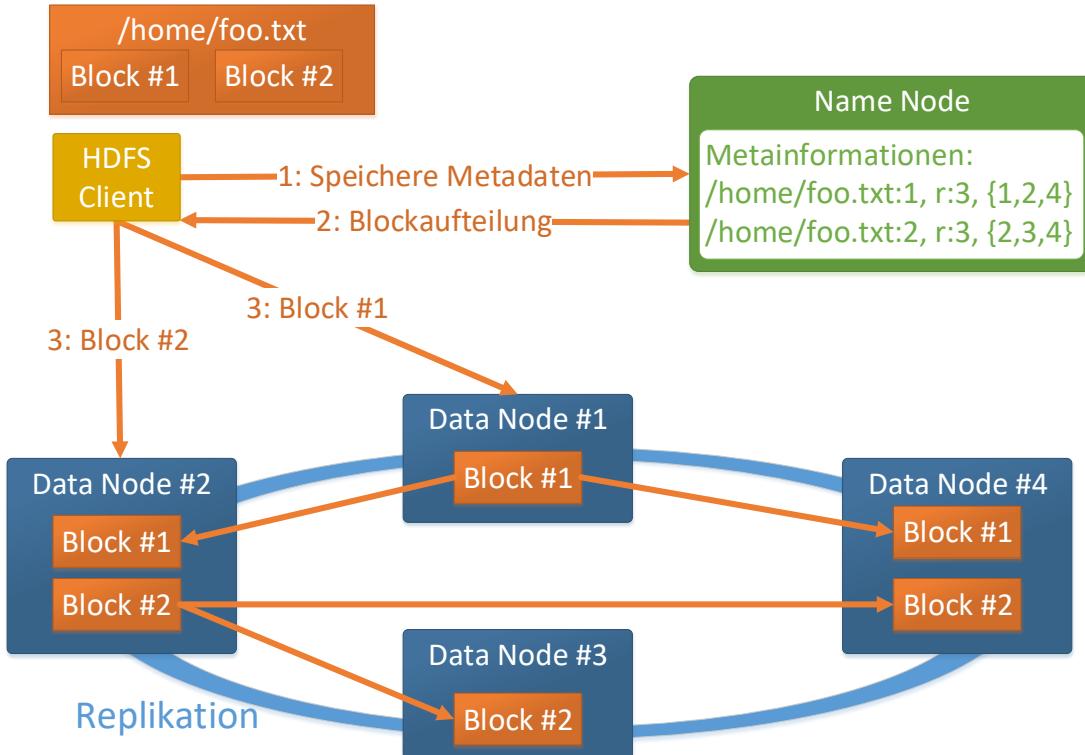


Abbildung 3.2: HDFS - Datenspeicherung im Verbund (Vgl. [16],[11])

Ein Block ist in der Standardkonfiguration 128 MB groß. Er kann bis zu 512 MB Blockgröße konfiguriert werden. Bei der Replikation wird der gleiche Block in unterschiedlichen Data Nodes angelegt.

Es ist nicht erlaubt den gleichen Block mehrmals im gleichen Data Node zu replizieren. Dies würde ja auch keinen Sinn ergeben, da die Replikation vor Datenverlust bei Ausfällen von einzelnen Knoten schützen soll. Wichtig hierbei ist auch, dass im Produktivsystem auf jedem physikalischen Knoten auch nur ein Data Node oder ein Name Node läuft. Denn würden beispielsweise mehrere Data Nodes auf dem gleichen physikalischen Knoten laufen, so wäre bei einem Ausfall nicht garantiert, dass die Dateiinhalte auch noch auf mindestens zwei anderen verfügbaren Knoten gespeichert sind. 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 Knoten in einem gemeinsamen Rack gruppiert sind. 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 einzelnen Testumgebungen ist es aber durchaus möglich einen Name Node und einen Data Node oder mehrere Data Nodes gemeinsam auf einem physikalischen Knoten zu installieren. Allerdings greifen die Mechanismen für eine Toleranz gegenüber Hardwareausfällen dann nicht mehr.

Wie in Abbildung 3.2 ersichtlich, ist der Name Node die Schlüsselstelle im HDFS-Cluster. Zusätzlich existiert ein sogenannter *Secondary Name Node*, der den (First) Name Node beim Speichern von Daten in gewisser Hinsicht unterstützt. Der Name Node hält die Metainformationen des HDFS im Arbeitsspeicher. Zusätzlich existieren zwei Dateien, das sogenannte *FsImage* und ein *EditLog*, welche persistent auf der Festplatte gespeichert sind. Das FsImage selbst beschreibt einen Zustand der Dateisystemmetainformationen zu einem gewissen Zeitpunkt (Checkpoint). 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, welche wiederum als neu FsImages gespeichert werden. Dieser Mechanismus wurde implementiert, um bei einem Neustart des Name Nodes die Startzeit zu optimieren. Dadurch kann die Verfügbarkeit des gesamten HDFS verbessert werden. Der *Secondary Name Node* unterstützt also den (First) Name Node, er kann ihn aber nicht ersetzen. Bei einem Ausfall wäre das HDFS nicht mehr einsatzbereit. Um dieses Problem zu umgehen, kann ein sogenannter *Standby Name Node* konfiguriert werden. Dieser kann einspringen, sobald der erste Name Node ausgefallen ist. Allerdings muss er extra konfiguriert werden.[11, S. 40 ff.]

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

### 3.3 Apache Hadoop YARN

YARN (Yet Another Resource Negotiator) ist eine Ressourcenverwaltung, welche die verfügbaren Ressourcen innerhalb des Hadoop Clusters organisiert und die Ausführungsreihe von Jobs plant und überwacht. Es gibt einen separaten *Resource Manager*, welcher

nur die Ressourcen verwaltet. Auf jedem Knoten, der auch Datenverarbeitungen durchführt, ist ein *Node Manager* installiert. 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.

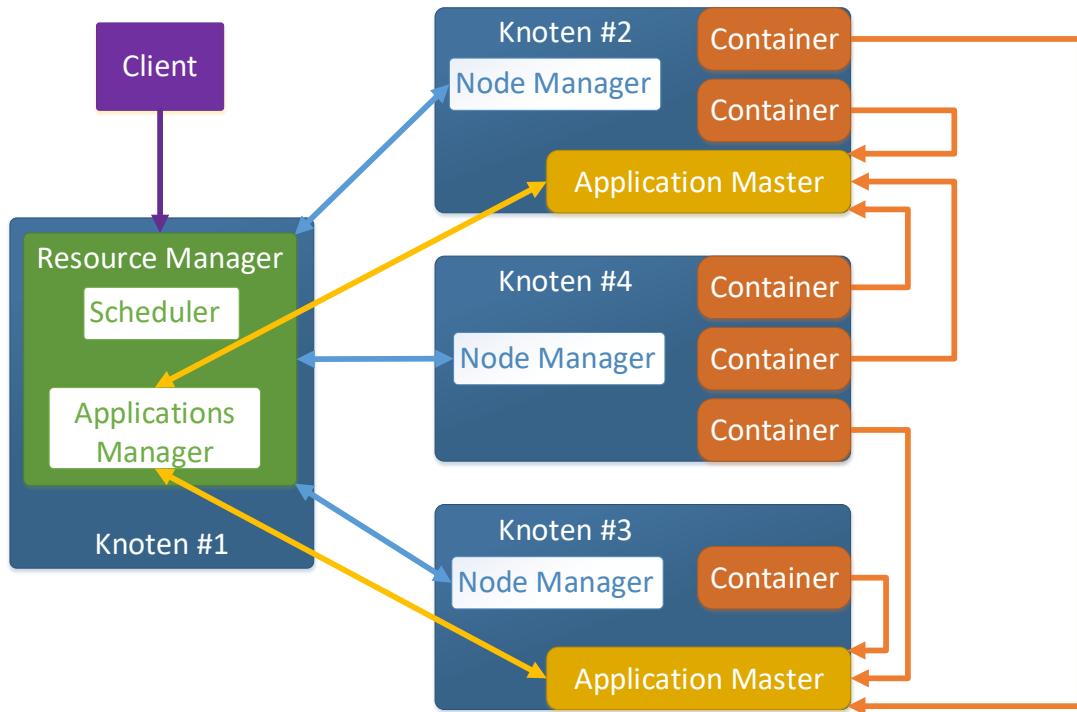


Abbildung 3.3: Ressourcenverteilung mit YARN (Vgl. [15],[11])

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.<sup>5</sup> Derzeit werden für den Container die Anzahl der CPU-Cores (Ausführbare CPU-Threads) und die Größe des nutzbaren Arbeitsspeichers definiert.[11, S. 48 ff.]

Wenn nun eine Anwendung über YARN im Cluster ausgeführt werden soll, dann sendet ein Client eine Anfrage an den Ressourcen 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 kennt die Anwendung nicht, noch weiß er wie diese

<sup>5</sup>In der Praxis ist es meistens weniger, da entsprechende Ressourcen für das darunter liegende Betriebssystem und YARN selbst reserviert werden.

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 der Application Master 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 Master an, wie viele Container (inklusive Arbeitsspeicher und CPU) er benötigt. Zusätzlich übermittelt er die Dateiblöcke, welche er aus dem HDFS braucht und auf welchen Knoten er wie viele Container starten will. So würde der Application Master auf dem Knoten #2 (siehe Abbildung 3.3) einen Container auf dem Knoten #2 und zwei Container auf dem Knoten #4 mit beispielsweise einem GB Arbeitsspeicher und einem 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.<sup>6</sup>

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

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

Nach der Ausführung der einzelnen Tasks innerhalb der Container und dem Abschluss des Jobs, schickt der Application Master über den Application Manager die Ergebnisse zurück zum Client.<sup>7</sup> Danach meldet er sich beim Resource Manager. Zuletzt gibt der Resource Manager allozierte Ressourcen frei.

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 auszusteuren, um den höchstmöglichen Datendurchsatz zu erreichen. Der Fair-Scheduler prüft hingegen, dass jedem Job die gleichen Ressourcen zugeteilt werden, um möglichst alle Jobs parallel bedienen zu können.

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 Spark

*Apache Spark<sup>TM</sup>* ist ein Projekt zur verteilten Verarbeitung von großen Datenmengen. Mit Apache Spark können verschiedene Algorithmen und Verarbeitungsschritte über eine einheitliche Programmierschnittstelle auf gespeicherte Daten angewendet werden. Spark

---

<sup>6</sup>Wobei 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.

<sup>7</sup>Hierbei werden die fachlichen Ergebnisse meistens als Datei im HDFS gespeichert.

selbst kümmert sich um die Verteilung, Ausführung und Überwachung der Applikationen zur Datenverarbeitung.[6, S. 2]

Apache Spark ist mittlerweile schon fast der Standard, wenn es im Hadoop-Umfeld um die Datenverarbeitung geht. Es löst damit auch das ursprünglich verwendete MapReduce-Framework von Hadoop ab, denn Spark bietet einen enormen Geschwindigkeitsvorteil gegenüber dem MapReduce-Framework. Dies lässt sich auf eine intelligente Ausführung einzelner Verarbeitungsschritte und diverse Optimierungen zurückführen.[11, S. 148 ff.]

Spark ermöglicht vielseitige Einsatzzwecke. So wird die klassische Datenverarbeitung von statischen Datenmengen<sup>8</sup> unterstützt, aber auch die Verarbeitung von dynamischen Datenmengen (Streaming-Data).<sup>9</sup> Auch die Prozessierung von Graphen-Strukturen und das maschinelle Lernen werden unterstützt.[11, S. 152]

Darüber hinaus steht es dem Anwender frei, ob er seine Applikationen in Scala, Python oder Java schreibt. Bei den Interpreter-Sprachen Scala und Python gibt es eine Spark-Shell zur interaktiven Datenverarbeitung und Analyse. Diese Vielseitigkeit macht sich auch in unzähligen Projekten und Programm-Bibliotheken bemerkbar, welche rund um Apache Spark entwickelt werden. Es existieren diverse Anbindungen zu Datenquellen, die sogenannten *Spark-Connectoren*. Damit können beliebige Datenspeicher als Datenquelle verwendet werden. Beispielsweise können Daten aus dem HDFS geladen werden, aber auch direkt aus Datenbanken wie HBASE, Cassandra, Neo4j oder Elasticsearch, welches zur Datenindexierung genutzt werden kann.

Abbildung 3.4 zeigt die Ausführung einer Spark-Applikation innerhalb eines Hadoop-Clusters mit YARN und skizziert den physikalischen Kontext im Cluster. Dieser Aufbau beschreibt im Kontext der Thesis den primären Anwendungsfall zur Datenverarbeitung. Apache Spark könnte auch vollständig unabhängig von dem Hadoop-Framework in einem eigenen Spark-Cluster ausgeführt werden und bietet dafür auch einen eigenen Ressourcenmanager. Allerdings wird innerhalb des Hadoop-Umfelds die Spark-Ausführung mit dem bereits erwähnten Ressourcenmanager YARN durchgeführt (siehe Kapitel 3.3). Dies hat auch den Vorteil, dass YARN die Ressourcen auf den einzelnen Knoten besser verwaltet kann. Denn wenn der Spark-Ressourcenmanager parallel zu YARN auf den gleichen Knoten genutzt werden würde, so könnte dies zu Ressourcen-Engpässen führen. Denn die Ressourcenmanager würden nicht miteinander kommunizieren und die Last der ausgeführten Anwendungen im Cluster könnte nicht gleichmäßig verteilt werden. Aus diesem Grund ist es ratsam YARN auch die Ausführung von Spark-Anwendungen im Cluster zu überlassen.

Wie in Abbildung 3.4 ersichtlich, wird die Ausführung einer Spark-Anwendung über den YARN-Ressourcenmanager gestartet. Es gibt hierbei unterschiedliche Varianten, wie eine Spark-Anwendung ausgeführt werden kann. Im konkreten Fall wird das *Spark-Submit* Kommando genutzt. Letztlich handelt es sich hierbei um einen Konsolenbefehl, welcher

---

<sup>8</sup>In diesem Kontext ist die simple Ausführung einer Anwendung auf eine bereits existierende Datenmenge gemeint, welche am Ende ein definiertes Ergebnis liefert.

<sup>9</sup>Bei der Datenverarbeitung von Streaming-Data wächst die zu verarbeitende Datenmenge dynamisch an und die ausgeführte Anwendung verarbeitet die neu hinzugekommenen Daten. Ein Beispiel wäre das Filtern von Tweets auf Twitter nach bestimmten Merkmalen, wobei auch neu hinzukommende Tweets bearbeitet werden und nicht nur die Tweets, welche beim Startzeitpunkt der Anwendung bereits existieren.

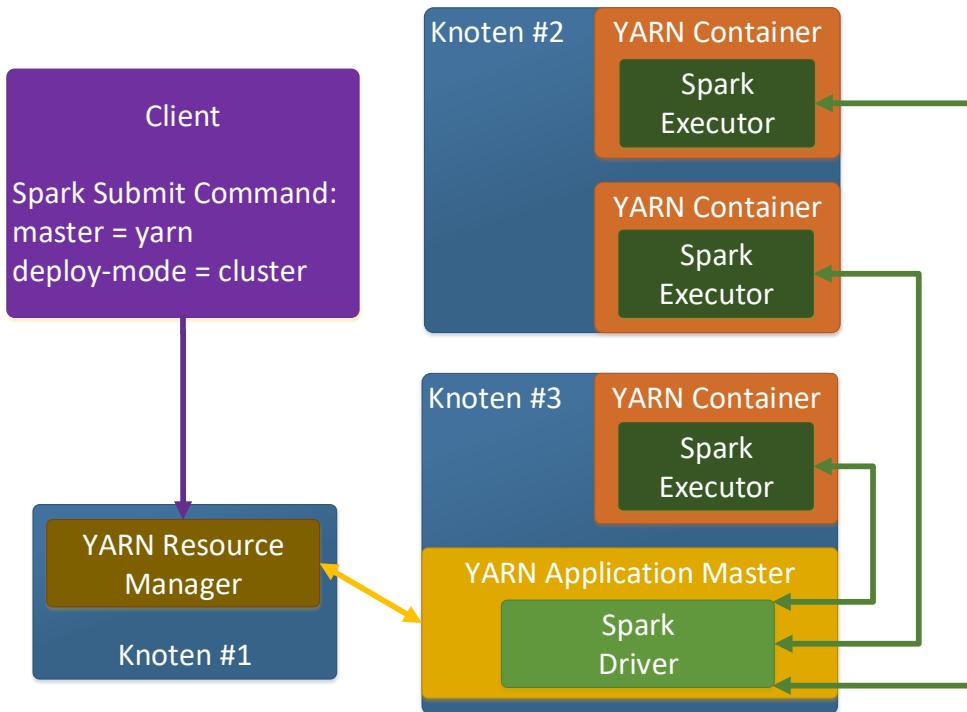


Abbildung 3.4: Spark Datenverarbeitung im Cluster

die Anwendung<sup>10</sup> selbst entgegen nimmt und über diverse Parameter konfiguriert werden kann. So kann unter anderem der Master und Deploy Mode so konfiguriert werden, dass YARN die Ressourcen der Applikation verwaltet. Wie in Abbildung 3.3 (siehe Kapitel 3.3) bereits beschrieben, wird bei YARN ein Application Master erstellt, welcher wiederum diverse Ausführungscontainer auf den einzelnen Knoten anfordert und diese überwacht. Bei der Ausführung einer Spark-Anwendung werden diese Komponenten wiederverwendet und kapseln letztlich die fachlichen Komponenten von Spark.

So gibt es bei Spark einen sogenannten *Driver*, welcher im Yarn Application Master läuft und wiederum die sogenannten *Executor* aussteuert. Diese laufen wiederum gekapselt in einzelnen YARN Containern auf den Knoten. Ein Spark Executor entspricht aus Betriebsystemsicht eines Knotens der Ausführung einer Java Virtual Machine (JVM) in einem eigenständigen Prozess. Aufgrund der Kapselung durch YARN können die einzelnen JVM-Prozesse überwacht werden und zur Not auch beendet werden, falls sie zu viel Ressourcen auf den Knoten anfordern.

Spark und YARN müssen entsprechend konfiguriert werden, damit die Anwendungen auch korrekt im Cluster skaliert werden können. Bei YARN und auch Spark beziehen sich die Ressourcen auf die Anzahl der genutzten CPU-Cores und die Größe des genutzten Arbeitsspeichers.

Gerade wenn YARN einzelne Application-Container stoppt, weil sie zu viel Arbeitsspeicher benötigen deutet dies auf eine falsche Konfiguration oder falsche Programmierung der

---

<sup>10</sup>Beispielsweise ist dies bei einer Spark-Anwendung in Java ein herkömmliches *Java Archiv* im *JAR*-Dateiformat.

Spark-Anwendungen hin. Um das Problem zu lösen wird gerne der nutzbare Arbeitsspeicher pro Executor höher konfiguriert. Dies ist in den meisten Fällen jedoch der falsche Ansatz, da hierdurch kritische Probleme in der Programmlogik der Anwendung oftmals nur kaschiert werden.

Daher ist es auf jeden Fall auch sinnvoll bei der Anwendungsentwicklung relativ kleine Cluster mit geringen Ressourcen zu nutzen, denn auch dort müssen die Anwendungen fehlerfrei ausführbar sein. Lediglich die Ausführungsgeschwindigkeit sollte sich in kleinen Clustern verlangsamen. Aus diesem Grund werden im Rahmen dieser Thesis auch die Spark-Anwendungen auf einem einzelnen Knoten getestet, um Programmfehler besser und frühzeitiger erkennen zu können. Einzelheiten zu den Programmierparadigmen und den grundlegenden Datenstrukturen können in Kapitel 5 nachgelesen werden.

### 3.5 Apache HBASE

*Apache HBASE*<sup>®</sup> 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.<sup>11</sup>

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 bei einem 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. [4, 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. Ein vereinfachtes Beispiel einer möglichen Datenstruktur wird in Abbildung 3.5 dargestellt.

Anhand der Abbildung werden einige Eigenschaften von HBASE sichtbar. So existieren zwei *Column Families* mit den Namen *metadata* und *content*. Die Column Family *meta-*

---

<sup>11</sup>Der Name *HBASE* basiert auf der Kombination von *Hadoop* und *Database*.

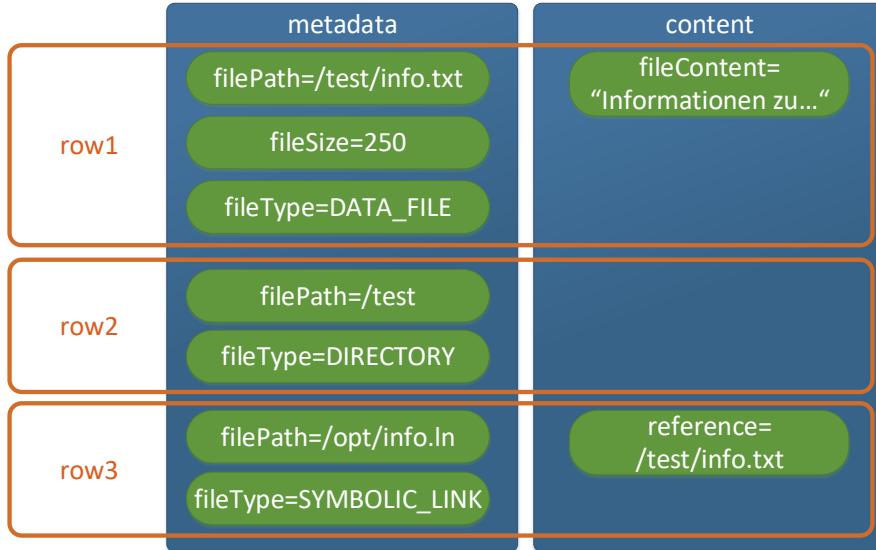


Abbildung 3.5: Schema-Beispiel einer HBASE Tabelle nach [4]

*data* enthält wiederum die Spalten *filePath*, *fileSize* und *fileType*. Die Werte werden alle als Binär-Inhalt gespeichert. Eine Zeile hat einen eindeutigen Spaltenschlüssel, wie zum Beispiel *row1*. Über diesen Schlüssel können die spezifischen Inhalte der einzelnen Spalten für eine bestimmte Zeile erfragt werden. Interessant hierbei ist, dass die Spaltenwerte optional sind und nicht für jede Zeile existieren müssen. So hat eine Datendatei einen konkreten Inhalt, welcher in der Spalte *fileContent* der Spaltenfamilie *content* gespeichert wird. Ein Verzeichnis hingegen hat keinen Dateinhalt. Daher ist in der zweiten Zeile auch kein Inhalt in der Spalte *fileContent* abgelegt. In der dritten Zeile hingegen, wird ein symbolischer Link gespeichert. Dieser wiederum hat auch keinen Inhalt in der Spalte *fileContent*. Dafür wird aber die Referenz auf die Originaldatei in einer weiteren Spalte gespeichert. Aufgrund der Gruppierung und Speicherung in Column Families benötigen leere Spaltenwerte auch keinen Speicherplatz. Ein einzelne Zelle beschreibt letztlich den Wert einer Spalte für eine konkrete Zeile. Hierbei wird zu jeder Zelle auch ein Zeitstempel gespeichert. Durch die Zeitstempel können auch ältere Werte einer Zelle ausgelesen werden. So wird bei einer Modifikation einer konkreten Zelle der Wert inklusive eines neuen Zeitstempels geschrieben. Es ist jedoch immer noch möglich, ältere Zustände der Zelle zu lesen.

Die einzelnen Spalten innerhalb einer Spaltenfamilie müssen nicht bei der Erstellung einer Tabelle bekannt sein. Lediglich die Spaltenfamilien müssen initial angegeben werden und können später auch nicht mehr geändert werden. Somit kann nachträglich die Tabelle um weitere Spalten erweitert werden.[10, S. 577]

Die Skalierbarkeit und die Partitionstoleranz wurden bei der Entwicklung von HBASE berücksichtigt. 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. Abbildung 3.6 zeigt die physikalische Aufteilung.

Auf den einzelnen Knoten im Computer-Cluster existieren sogenannte *Region Server*. Diese Region Server speichern jeweils unterschiedliche Teile der in HBASE angelegten Tabellen.

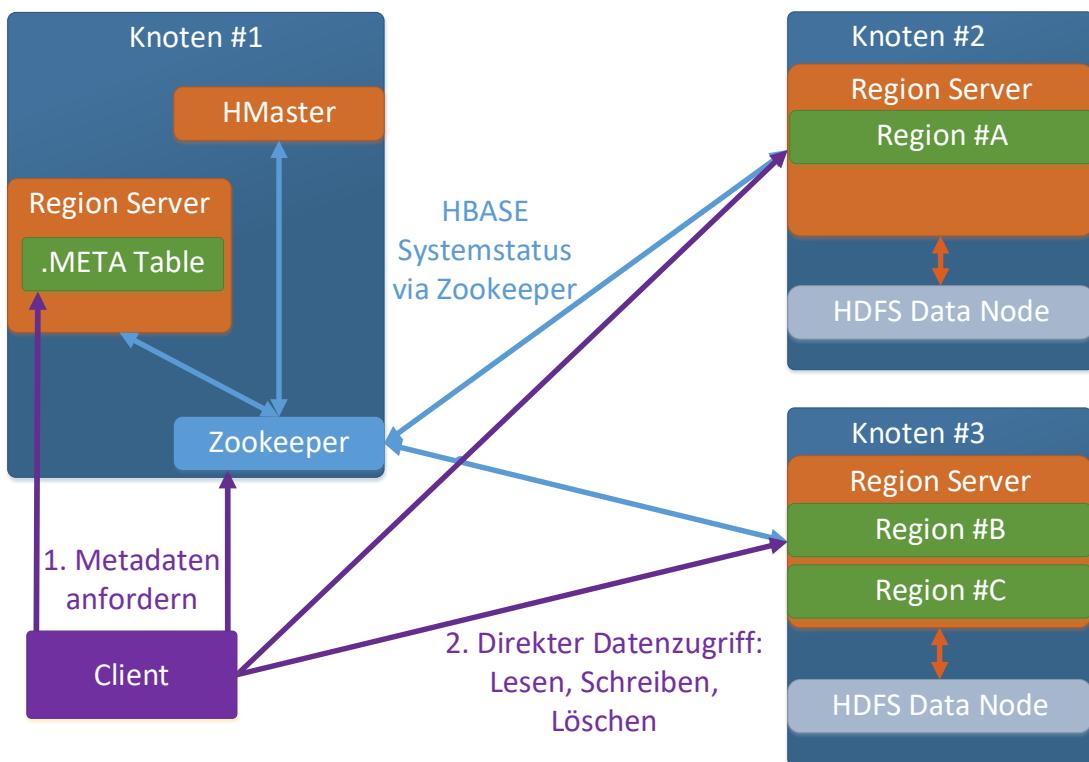


Abbildung 3.6: HBASE Datenspeicherung im Cluster

Eine Tabelle wird hierbei anhand der Zeilenschlüssel in mehrere Bereiche, den sogenannten *Regions* unterteilt. Beispielsweise könnte die Region A (siehe Abbildung 3.6) alle Daten einer Tabelle der Zeilen 1 bis 3000 enthalten. Die Region B wiederum enthält alle Daten der gleichen Tabelle aber von den Zeilen 3001 bis 8540. Und die Region C könnte die Daten der Zeilen 1 bis 22300 von einer anderen Tabelle enthalten. Somit kann ein Region Server mehrere Regions von gleichen oder auch unterschiedlichen Tabellen verwalten. Das Prinzip der Datenlokalität greift auch bei HBASE. So sollte auf jedem Knoten, auf dem ein RegionServer läuft, auch ein HDFS Data Node existieren. Dieser speichert die Daten der einzelnen Regions als Dateien im HDFS.

Wenn neue Tabellen erstellt werden, oder einzelne Regions zu groß werden, dann koordiniert eine übergeordnete Instanz die Umverteilung von Daten und die Erstellung neuer Regions. Diese Instanz ist bei HBASE der sogenannte *HMaster*. Es ist ein leichtgewichtiger Prozess, welcher auf einem beliebigen Knoten im Cluster läuft und über Apache ZooKeeper auch den Status der einzelnen Region Server überwacht.<sup>12</sup> Um die Ausfallsicherheit zu gewährleisten ist auch dieser Prozess redundant ausgelegt.<sup>13</sup> Darüber hinaus kümmert sich der HMaster-Prozess auch um die Restrukturierung bei Teilausfällen einzelner Region Server.

Wenn nun ein Client auf die Daten einer Tabelle zugreifen möchte, muss dieser wissen auf welchem Knoten die Daten abgelegt sind. Hierfür verbindet sich der Client zuerst mit

<sup>12</sup>Die Funktionsweise von Apache ZooKeeper wird in Kapitel 3.6 näher erläutert.

<sup>13</sup>Über ZooKeeper kann immer der primäre HMaster-Prozess ermittelt werden. Ist der aktive HMaster nicht mehr erreichbar, schaltet sich ein Backup-HMaster ein.

ZooKeeper und erfährt darüber, welcher Region Server die sogenannte *Meta-Tabelle* speichert.[10, S. 579]

Diese Meta-Tabelle enthält Informationen über alle Tabellen in HBASE inklusive der Region-Server und deren Regions die sie bereitstellen. Anhand dieser Metadaten kann der Client dann direkt die benötigten Daten an den entsprechenden Region Servern anfordern. Dieses Vorgehen scheint für den Zugriff auf wenige Daten etwas aufwendig. Es skaliert aber sehr gut bei großen Datenmengen, da kein Flaschenhals vorhanden ist. Normalerweise speichert der Client die angeforderte Meta-Tabelle temporär, so dass er bei nachfolgenden Anfragen direkt auf die entsprechenden Region Server zugreifen kann.

Im Rahmen dieser Thesis wird HBASE verwendet, um Metadaten zu speichern. Diese werden wiederum mit Apache Spark ausgelesen und verarbeitet. Hierbei kann das Prinzip der Datenlokalität sehr gut genutzt werden. Denn in der Theorie ist es durchaus möglich die HDFS Data Nodes, die Spark Worker und die HBASE Region Server getrennt auf unterschiedlichen Knoten auszuführen. Aber gerade dies macht keinen Sinn, da sonst immer wieder Daten über das Netzwerk transportiert werden müssen und dieses dann zum Flaschenhals der Verarbeitung wird.

Sinnvoller ist es HDFS Data Nodes auf den Knoten auszuführen, wo auch die Region Server ausgeführt werden. Darauf aufbauend sollten auch die Spark Worker auf den Knoten gestartet werden, auf welchen die Region Server laufen. Dadurch können im besten Fall die Daten, welche durch Apache Spark benötigt werden, direkt von dem lokalen HBASE Region Server bereitgestellt werden. Dieser erhält die Daten wiederum von dem lokalen Data Node. 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.<sup>14</sup> Die Koordinierung dieser Komponenten ist allerdings entsprechend komplex. Hierbei können bei der Entwicklung von Anwendungen viele Fehler gemacht werden, die dieses Prinzip der Datenlokalität aushebeln können. Daher müssen die Implementierungen genau geprüft werden, ob sie das Prinzip der Datenlokalität korrekt umsetzen.

### 3.6 Apache ZooKeeper

Innerhalb eines Computer-Clusters zur verteilten Datenverarbeitung existiert oftmals das Problem, dass sich die einzelnen Komponenten koordinieren müssen. Beispielsweise teilt HBASE die Daten auf mehrere RegionServer auf, welche wiederum auf den einzelnen Knoten ausgeführt werden. Doch welche Instanz koordiniert diese Aufteilung? Ein anderes Problem ist der Datenzugriff. Ein Client möchte eine Zeile einer bestimmten Tabelle auslesen. Woher weiß der Client, welchen konkreten Knoten er anfragen muss, um genau diese Zeile zu erhalten?

In den meisten Fällen existiert hierzu eine bestimmte Instanz, welche die Koordinierung der Knoten übernimmt. Bei HBASE gibt es den HMaster, welcher die Koordinierung übernimmt. Das Problem ist hierbei, dass auch der Knoten auf dem diese Master-Instanz läuft ausfallen kann und das komplette System zum erliegen bringt. Diese Verwaltungsinstanzen im Allgemeinen sind kritische Komponenten und können als Single-Point-of-Failure zu einem Stillstand des kompletten Systems führen. Um solche Totalausfälle zu vermeiden, müssen bestimmte Automatismen definiert werden, wie sich die Knoten selbst organisieren können, um einen Ausfall beliebiger Knoten zu überstehen.

---

<sup>14</sup>Sie auch Kapitel 3.3 und Kapitel 3.4.

An dieser Stelle bietet *Apache ZooKeeper<sup>TM</sup>* Mechanismen an, wie sich die einzelnen Knoten in einem verteilten System organisieren und Informationen verteilt synchronisieren können. Aus logischer Sicht stellt ZooKeeper einen Service zu Verfügung. Dieser Service ermöglicht das Speichern von Informationen als strukturiertes Verzeichnis mit einzelnen Dateien. Bei den Informationen handelt sich normalerweise um wichtige Konfigurationen, welche im Cluster verteilt werden müssen und zentral über ZooKeeper aktualisiert werden können. Jeder Client, der sich mit ZooKeeper innerhalb des Computer-Clusters verbindet, sieht die gleiche Konfiguration und kann bei Bedarf auch bestimmte Konfigurationen ändern.[12, S. 4 ff]

Aus Sicht des Entwicklers, stellt dieser Service immer die aktuellen Informationen im Cluster zu Verfügung. Wie der Service dies bewerkstelligt ist ein Implementierungsdetail. Neben dem bereits erwähnten Konfigurationsmanagement bietet ZooKeeper einen Naming-Service an oder ermöglicht es auch den Live-Status einzelner Knoten zu überwachen.[5]

Wie bei HBASE in Kapitel 3.5 bereits erwähnt wird, kann sich ein Client mit ZooKeeper verbinden und erhält über ZooKeeper den aktuellen Knoten, welcher die Metadaten zu allen Tabellen in HBASE speichert. Ein anderes Beispiel zeigt die Backup-Instanz des HMaster-Prozesses. Dieser prüft über ZooKeeper den Status des primären HMaster-Prozesses und wird informiert, wenn letzterer nicht mehr verfügbar ist. Daraufhin propagiert sich die Backup-Instanz als neuen HMaster-Prozess im Cluster, um so die Funktionsfähigkeit von HBASE aufrecht zu erhalten. Abbildung 3.7 zeigt die Verzeichnisstruktur, welche stark an ein Verzeichnis eines Dateisystems erinnert. Die einzelnen Einträge werden *ZNodes* genannt.<sup>15</sup>

```
[zk: localhost:2181(CONNECTED) 5] ls /hbase
[replication, meta-region-server, rs, splitWAL, backup-masters, table-lock,
 flush-table-proc, region-in-transition, online-snapshot, master, running,
 recovering-regions, draining, namespace, hbaseid, table]
```

Abbildung 3.7: Gespeicherte Informationen von HBASE in ZooKeeper

Normalerweise ist Zookeeper auf mehreren Knoten im Cluster installiert. Sie bilden ein sogenanntes *ZooKeeper Ensemble* und bestehen zumeist aus 3 oder 5 Einzelinstallationen auf beliebigen Knoten. Ein ZooKeeper Ensemble, welches für die gleiche Anwendungsdomäne zuständig ist, wird auch *Quorum* genannt. Innerhalb eines Quorums gibt es einen Leader und mehrere Follower, die sich die gleichen Konfigurationsinformation teilen. Fällt der Leader aus, können die Follower einen neuen Leader bestimmen.[5]

### 3.7 Apache Solr und Lucene

*Apache Solr<sup>TM</sup>* ist eine skalierbare und performante Plattform zur Datensuche. Die Daten werden vorher indexiert und können effizient durchsucht werden.[8]

Mithilfe von Apache Solr können zum Beispiel eine Volltextsuche für Dokumente oder eine Produktsuche eines Webshops implementiert werden.

---

<sup>15</sup>Siehe Link <http://zookeeper.apache.org/doc/r3.5.4-beta/zookeeperOver.html>. Letzter Zugriff: 26.7.2018.

Im Allgemeinen ist ein Suchsystem in mehrere Analyseschritte aufgeteilt. Ein Großteil dieser Verarbeitungsschritte wird nicht von Solr selbst übernommen. Vielmehr baut Solr auf dem Open-Source Projekt *Apache Lucene™* auf und erweitert Lucene letztlich um eine skalierbare Infrastruktur und diverse Schnittstellen zur Verarbeitung der Daten.

Zu Beginn liegen beliebige Daten in Form von Dateien oder Dokumente vor. Diese Daten durchlaufen eine textuelle Aufbereitung bevor sie indexiert werden können.

Bei der Textanalyse wird der relevante Text aus den Daten extrahiert. Hierbei hängt die Extraktion der Daten auch davon ab, ob die Daten strukturiert, semistrukturiert oder unstrukturiert sind. Anhand der Dateiformate können diverse Bibliotheken genutzt werden, um Daten zu extrahieren. Beispielsweise kann mithilfe von *Apache PDFBox®<sup>16</sup>* Text aus PDF-Dokumenten extrahiert werden. Mithilfe der *Geospatial Data Abstraction Library (GDAL)<sup>17</sup>* können Geopositionen aus den EXIF-Metadaten von JPEG-Bildern extrahiert werden. Mithilfe von *Tesseract OCR<sup>18</sup>* kann lesbarer Text direkt aus Bildern extrahiert werden.[8, S. 39]

Im nächsten Schritt wird der extrahierte Text aufbereitet. Die Aufbereitung hängt stark von dem spezifischen Anwendungsfall und den Daten selbst ab. Im Allgemeinen werden Satzzeichen und überflüssige Füllwörter entfernt und Großbuchstaben werden in Kleinbuchstaben umgewandelt. Darüber hinaus kann auch eine Stammformreduktion durchgeführt werden. Darauf aufbauend kann der Text weitergehend analysiert werden. Beispielsweise könnten Redewendungen erkannt oder Synonyme einzelner Wörter identifiziert werden.[8, S.44]

Nach der Aufbereitung des Textes erfolgt die Erstellung eines sogenannten *Inverted Indexes*. Ein *Inverted Index* ist ähnlich aufgebaut, wie ein Stichwortverzeichnis. Jedes Wort wird darin mit den Verweisen zu den Vorkommen in den einzelnen Dokumenten versehen. Auf Basis dieses Indexes können sehr schnell alle Dokumente gefunden werden, welche das gesuchte Wort enthalten.[8, S. 47]

Im nächsten Schritt können Suchanfragen auf die erstellten Indizes durchgeführt werden. Aber auch hier gibt es unterschiedliche Modelle, wie die relevanten Dokumente ermittelt werden und in welcher Reihenfolge die Ergebnismenge zurückgeliefert wird.

Es gibt das sogenannte *Boolean Model* auf Basis der booleschen Algebra. Beispielsweise könnte eine forensische Suchanfrage lauten: Suche alle Bilder oder Videos, welche das Wort *Unfall* im Dateinamen haben aber nicht kleiner sind als 1 MB. Diese Suchanfrage wird in einen booleschen Ausdruck überführt und die Dateien werden zurückgeliefert. Allerdings liefert dieses Modell keine Aussage über die Reihenfolge der Ergebnismengen. Hierzu gibt es komplexere Suchmodelle, wie das sogenannte *Vector Space Model*. Es basiert auf gewichteten Faktoren und ermöglicht es für jedes Dokument der Ergebnismenge die Trefferwahrscheinlichkeit zu ermitteln. Nach dieser Bewertung wird die Reihenfolge der Ergebnisse ermittelt. Beispielsweise sollte ein Dokument eine höhere Bewertung erhalten, wenn das gesuchte Wort mehrfach im Dokument vorhanden ist.[8, S. 47 ff]

Solr besitzt einen *Cloud-Mode*, welcher es ermöglicht die indexierten Daten verteilt auf mehreren Knoten zu speichern. Abbildung 3.8 skizziert diese Verteilung im Computer-Cluster.

---

<sup>16</sup>Siehe Link: <https://pdfbox.apache.org/>. Letzter Zugriff: 01.08.2018.

<sup>17</sup>Siehe Link: <https://www.gdal.org/>. Letzter Zugriff: 01.08.2018.

<sup>18</sup>Siehe Link: <https://github.com/tesseract-ocr/tesseract>. Letzter Zugriff: 01.08.2018.

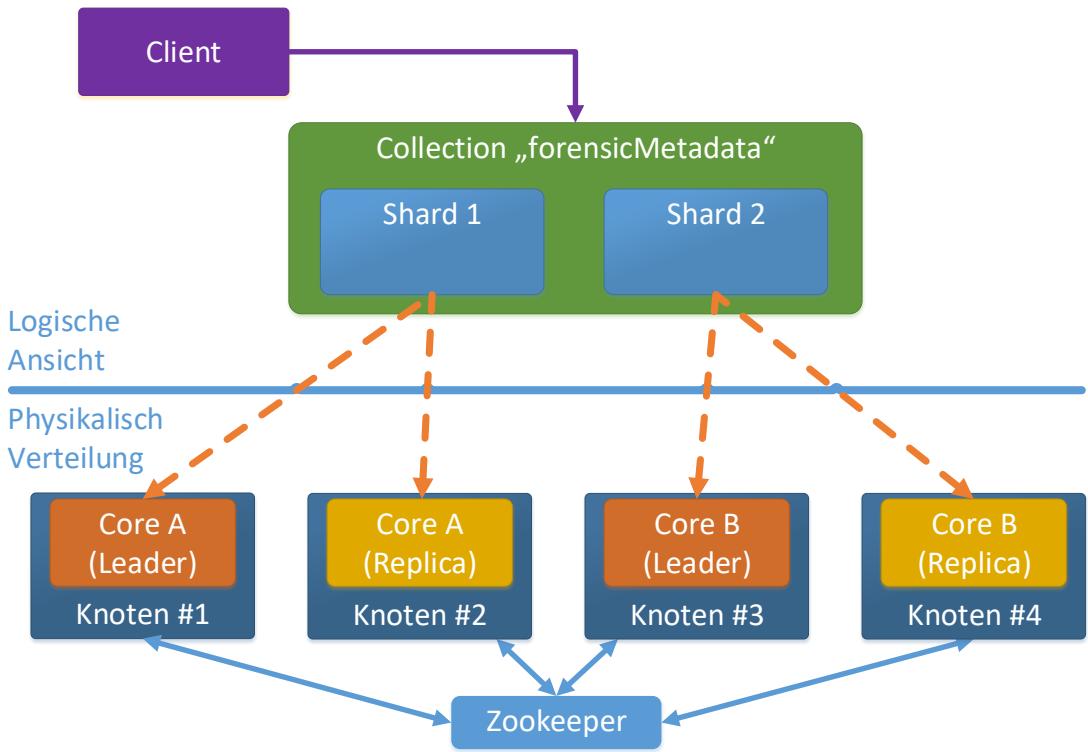


Abbildung 3.8: Solr Cloud-Mode im Cluster

Dieser Cloud-Mode wird auch bei der Integration in die Analyseplattform genutzt.<sup>19</sup> Solr kann als zusätzliches Software-Paket in das Hadoop-Ökosystem integriert werden.<sup>20</sup> Es können beliebige Daten in der Solr-Cloud indexiert werden. Darüber hinaus nutzt Solr im Cloud-Mode das bereits beschriebene Zookeeper-Projekt<sup>21</sup> zur Koordinierung und Überwachung der einzelnen Knoten.

Innerhalb des Cloud-Modes werden die oben beschriebenen Indizes (Inverted Index) in mehrere sogenannte *Shards* aufgeteilt. Ein einzelner Shard ist hierbei ein Teil des Indexes, welcher unabhängig von den anderen Shards durchsucht werden kann. So kann bei einer Suchanfrage die Suche parallel auf alle Shards verteilt werden. Einzelne Shards können wiederum auf anderen Knoten repliziert werden, um die Ausfallsicherheit zu gewährleisten. Solr spricht hierbei von Cores. Ein Core ist die physikalische Repräsentation eines logischen Shards auf einem konkreten Knoten. Alle logischen Shards bilden zusammen den Index, welcher bei Solr als sogenannte *Collection* dargestellt wird.[3]

Jedes Dokument, welches in einer Collection indexiert werden soll, wird einem der logischen Shards zugewiesen. Hierbei wird zuerst anhand eines eindeutigen Dokumentenschlüssels entschieden in welchen Shard das Dokument aufgenommen werden soll.<sup>22</sup> Ein logischer Shard

<sup>19</sup>Siehe Kapitel 5.

<sup>20</sup>Zum Beispiel liefert die Hortonworks Dataplatform ein passendes Software-Paket zur Installation von Solr. Primär wird dieses Paket genutzt, um eine performante Suche im HDFS-Dateisystem zu ermöglichen. Die Solr-Installation kann aber auch für andere Zwecke genutzt werden.

<sup>21</sup>Siehe Kapitel 3.6.

<sup>22</sup>Der Dokumentenschlüssel kann zum Beispiel von der Hashsumme der Datei abgeleitet werden.

wird abermals aufgeteilt in mehrere Replicas, den bereits erwähnten Cores auf den einzelnen Knoten. Für jeden logischen Shard wird ein *Leader*-Knoten bestimmt.<sup>23</sup>. Dieser Leader-Knoten speichert einen physikalischen Core des logischen Shards und indexiert im nächsten Schritt das Dokument. Sobald der Index neu aufgebaut wurde, wird die Aktualisierung an die anderen Knoten weitergesendet, welche wiederum eine Replica (Core) des logischen Shards speichern.[18, S. 867 ff]

Apache Solr und Lucene sind beide in Java implementiert. Daher gibt es auch einen Java Client, zur Durchführung von Suchanfragen. Andererseits existiert auch eine REST-Endpunkt, der für Anfragen genutzt werden kann. Damit können Anfragen simpel und schnell in beliebige Websites integriert werden oder direkt mit dem Kommandozeilentool *curl* durchgeführt werden. Hierbei können die Anfragen in XML oder JSON formatiert sein. Ein simples Beispiel zeigt nachfolgende Anfrage (siehe Abbildung 3.9).

```
> curl "http://localhost:8983/solr/forensicMetadata/query?q=*shot*"
{
  "responseHeader": {
    "status": 0,
    "QTime": 20,
    "params": {
      "q": "*shot*"
    }
  },
  "response": {
    "numFound": 1,
    "start": 0,
    "maxScore": 1.0,
    "docs": [
      {
        "filePath": ["Screenshot from 2018-05-13 05-55-23.png"],
        "fileType": ["DATA_FILE"],
        "fileSize": [360377],
        "owner": ["johannes"],
        "group": ["johannes"],
        "permissions": ["[OWNER_READ, GROUP_WRITE, OWNER_WRITE, OTHERS_READ, GROUP_READ]"],
        "lastModified": ["2018-05-13T03:55:31.613Z"],
        "lastAccessed": ["2018-05-13T03:55:31.613Z"],
        "created": ["2018-05-13T03:55:31.613Z"],
        "mediaType": ["image/png"],
        "fileHash": ["c894fcc169d7971bfc4fc6fee973a852333db485fcf1ac772fd2e573f9701fa7baa9c7180d0c1ad402efd89ae1aed4e873e1d4d02b024b4cffae415dacc25fcf"]
      }
    ],
    "id": "row4",
    "_version_": 1607729961236430848
  }
}
```

Abbildung 3.9: Solr-Suchanfrage via *curl*

In dem Beispiel ist die Solr-Instanz unter <http://localhost:8983> erreichbar. Die Suchanfrage selbst wird auf der Collection *forensicMetadata* ausgeführt. Es soll in allen Feldern nach einem Vorkommen von dem Wort *shot* (*q=\*shot\**) gesucht werden. Die Antwort ist im JSON-Format definiert. Unter anderem wird die Anzahl der Treffer (*numFound:1*) mitgeliefert. Bei dieser konkreten Anfrage, wurde ein Bild mit dem Dateinamen *Screenshot...* gefunden.

---

<sup>23</sup>Diese Leader werden mittels Zookeeper im Cluster propagiert.

# 4 Datenspeicherung

## 4.1 Allgemeiner forensischer Analyseprozess

Im Praxisteil dieser Arbeit soll eine Analyseplattform auf Basis von Apache Hadoop aufgebaut werden. Diese Analyseplattform dient zur Auswertung sichergestellter Asservate und dem Auffinden von Beweisen in großen Datenmengen. Hierbei behandelt die Analyseplattform aber nur einen Teil der Arbeitsvorgänge während einem forensischen Analyseprozess.

Abbildung 4.1 skizziert einen allgemeinen forensischen Analyseprozess für digitale Beweismittel.[14, S.16] Die grün hinterlegten Schritte definieren den Arbeitsbereich, bei welchen die hier entwickelte forensische Analyseplattform den Forensiker unterstützen kann.

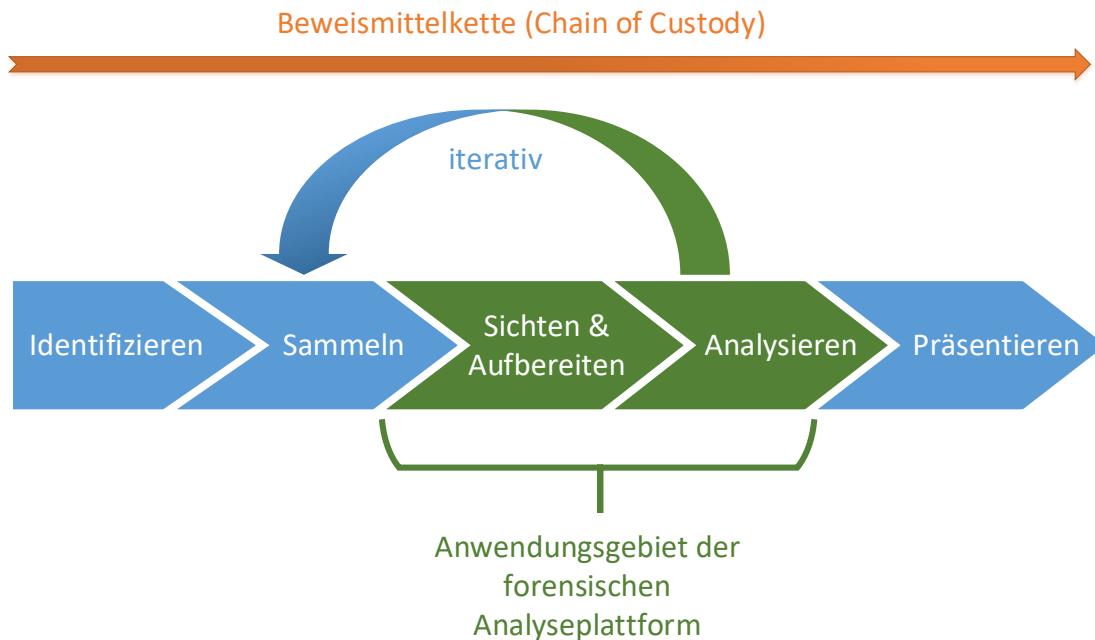


Abbildung 4.1: Forensischer Analyseprozess für digitale Beweismittel (Vgl. [14, S.16])

Zu Beginn existiert ein Vorfall oder ein Tatverdacht für eine Straftat. In einem nachfolgenden Schritt ermittelt beispielsweise die Staatsanwaltschaft. Im Ermittlungsverlauf wird

darauf der Tatort untersucht oder auch bei Tatverdächtigen nach Hinweisen für die Tat und gegebenenfalls deren Tathergang gesucht. Dieser Schritt beschreibt die Identifikationsphase aus Abbildung 4.1. Hierbei geht es um die Identifikation von potentiellen Beweismitteln, welche sichergestellt oder beschlagnahmt werden sollen.[14, S. 17-24].

In der Forensik wird häufig hypothesenbasiert vorgegangen. So entwickelt der Ermittler eine Hypothese, wie eine möglich Straftat begangen wurde und wer diese begangen haben könnte. Darauf aufbauend überlegt er sich, welche Spuren für oder gegen diese Hypothese sprechen und welche möglichen Beweismittel eben diese Spuren enthalten könnten (z.B. Kommunikationsdaten auf dem PC oder Bildmaterial auf einem Mobiltelefon).

Im zweiten Schritt aus Abbildung 4.1 geht es um das Vereinnahmen von potentiellen Beweismitteln. Hierbei werden identifizierte Datenträger und Geräte sichergestellt oder beschlagnahmt. In dieser Phase werden beispielsweise auch schon forensisch korrekte Datenträgerabbilder erstellt, auf welchen dann später eine Datenanalyse ausgeführt werden kann.[14, S. 24-33] In vielen Fällen, beispielsweise bei Unternehmensservern, werden nicht die Geräte selbst sichergestellt, sondern nur wichtige Teile der Daten forensisch korrekt kopiert. Auch hier findet teilweise schon eine Vorselektion statt, welche Daten benötigt werden und welche Daten für den konkreten Fall irrelevant sind.

Die dritte Phase aus Abbildung 4.1 behandelt das Sichten und Aufbereiten der sichergestellten Daten. Gerade bei großen unstrukturierten Datenmengen erfolgt in dieser Phase auch eine Vorselektion, um die Datenmenge nochmals einzuschränken. Dies kann automatisiert oder auch manuell erfolgen.[14, S. 33-39]

An dieser Stelle beginnt das Anwendungsgebiet der hier entwickelten forensische Analyseplattform. So können alle gesammelten Daten in die forensische Analyseplattform importiert und prozessiert werden. Darauf kann über eine allgemeine Suche nach spezifischen Stichworten, Hashsummen oder Zeitpunkten gesucht werden. Dies soll dem Ermittler das Auffinden von fallrelevanten Daten erleichtern. Die meisten bekannten forensischen Analysetools bieten ebenfalls eine Stichwortsuche an, weil damit große Datenmengen schnell nach bestimmten Kriterien gefiltert werden können.[2, S. 116-123]

In dieser Phase werden aber auch gelöschte, verschlüsselte oder verschleierte Daten wiederhergestellt oder entschlüsselt, sofern dies möglich ist. Eine klassische Methode ist beispielsweise auch das sogenannte *File Carving* auf Datenträgern.[14, S. 38-39]<sup>1</sup> Diese Art von Datenaufbereitung beherrscht die hier entwickelte forensische Analyseplattform derzeit noch nicht.<sup>2</sup>

Auch in der anschließenden Analysephase kann die hier entwickelte forensische Analyseplattform genutzt werden (siehe Abbildung 4.1). In dieser Phase werden die aufbereiteten Daten detailliert analysiert, um Informationen zu erhalten die für oder gegen einen bestimmten Tathergang sprechen.[14, S. 39-45] Anhand dieser Informationen werden die eingangs beschriebenen Hypothesen zu möglichen Tathergängen verifiziert. Bei einer Analyse werden aus den Daten komplexe Zusammenhänge und Beziehungen erarbeitet. Beispielsweise werden aus den Rohdaten Kommunikationsverläufe auf Basis von E-Mails oder

<sup>1</sup>Beim File Carving wird versucht logisch zusammenhängende Daten allein anhand des Dateiinhalts zu rekonstruieren, ohne die Dateisystemmetadaten zu nutzen. Die Methode wird gerne angewendet, wenn das Dateisystem nicht wiederherstellbar ist oder wenn gelöschte Dateien auf bereits freigegebenen Speicherbereichen gesucht werden.

<sup>2</sup>Es wäre aber durchaus möglich diverse Methoden der Datenaufbereitung zu implementieren. Siehe auch Kapitel 10.

zeitliche Abläufe basierend auf Zeitstempelanalysen erstellt.[14, S. 33-39] Hierbei geht es auch darum die Aussagekraft eines potentiellen Beweismittels zu ermitteln. So könnte beispielsweise urheberrechtsverletzendes Material auf einem Datenträger eines PCs gefunden werden. Allerdings kann die Aussagekraft dieses potentiellen Beweismittels sehr gering sein, wenn das System bereits durch Schadsoftware kompromittiert wurde. Einige Analyseschritte können automatisiert werden und sind daher prädestiniert für die hier entwickelte Analyseplattform (siehe auch Kapitel 5).

Bei der Analyse können wieder Verbindungen zu neuen potentiellen Beweismitteln gefunden werden, welchen dann wieder über den forensischen Analyseprozess vereinnahmt und aufbereitet werden. Daher ist der Prozess auch iterativ anzusehen. Gerade durch die Nutzung der forensischen Analyseplattform sollen diese Iterationen verkürzt werden, indem durch die parallelisierte Prozessierung Zeit eingespart werden soll.

In der letzten Phase des Analyseprozesses müssen die Ermittlungsergebnisse visuell aufbereitet werden, um sie auch vor Gericht präsentieren zu können. Letztlich muss ein Analysebericht erstellt werden, welcher einerseits die Ergebnisse enthält und andererseits nachvollziehbar beschreibt, wie diese Analyseergebnisse zustande gekommen sind.<sup>3</sup>[14, S. 45-47] Viele Analysetools, unter anderem auch das hier genutzte Referenztool *Autopsy*, ermöglichen die halbautomatische Erstellung von Analyseberichten. Die hier entwickelte Analyseplattform kann dies derzeit noch nicht. Bei einer Weiterentwicklung der Analyseplattform wäre diese Funktionalität aber durchaus brauchbar.

Ein primärer Aspekt bei dem allgemeinen forensischen Analyseprozess aus Abbildung 4.1 ist letztlich die Dokumentation der Beweismittelkette. Mit ihr steht und fällt die Aussagekraft der Ergebnisse aus einer forensischen Analyse.<sup>4</sup> Daher muss bei der Dokumentation der Beweismittel lückenlos festgehalten werden, was mit letzteren passiert ist. Nachfolgende Liste liefert hierzu wichtige Kriterien, die festgehalten werden müssen:

- Die Ermittler, welche das Beweismittel sichergestellt und später analysiert haben.
- Die Prozesse, Datenaufbereitungen und Analysen, welche durchgeführt wurden.
- Die Zeitpunkte der Sicherstellung, Aufbereitung und der Analyse.
- Die Umstände, wie das Beweismittel sichergestellt wurde.
- Gründe, wieso das Beweismittel sichergestellt wurde.
- Transportwege und Lagerstätten der Beweismittel.
- Personen die Zugang zu den Beweismittel hatten und allgemeine Informationen, wie die Beweismittel vor unbefugtem Zugriff geschützt wurden.

Ein sehr wichtiger Punkt bei der Dokumentation der Beweismittelkette ist die Verifikation der Datenintegrität der Asservate während des gesamten Analyseprozesses. Dies wird primär durch die Prüfung mittels kryptografischer Hashes erreicht. Letztlich dienen diese zum Schutz vor einer unbeabsichtigten oder beabsichtigten Modifikation des Beweismittels.

---

<sup>3</sup>Der Bericht sollte so geschrieben sein, dass andere Parteien oder Ermittler die gleichen Ergebnisse reproduzieren können.

<sup>4</sup>In der Vergangenheit gab es immer wieder Fälle, bei welchen eine fehlerhaft dokumentierte Beweismittelkette zu sehr fragwürdigen Aussagen auch bei ursprünglich eindeutigen Beweislagen führte.

Das Ändern von Asservaten oder deren Kopien sollte eigentlich immer vermieden werden. In vielen Fällen ist dies jedoch notwendig, um beispielsweise defekte Datenstrukturen wiederherstellen zu können. Bei solchen bewussten Datenänderungen ist eine entsprechende Dokumentation notwendig.

Bei der Analyse mit der hier entwickelten Analyseplattform muss auch die Beweismittelkette entsprechend dokumentiert werden. Diese Thematik wird im Rahmen dieser Thesis zumindest in der Theorie nochmals diskutiert. Die derzeitige Implementierung der forensischen Analyseplattform behandelt diesen Aspekt noch nicht. Aber auch hier könnte in einer Weiterentwicklung eine Beweismittelkette erstellt werden, welche die einzelnen Verarbeitungsprozesse, Zeitpunkte und Nutzerzugriffe dokumentiert.

## 4.2 Herkömmliches Analysevorgehen

Um die fachlichen Anforderungen an das Analyse-System herauszuarbeiten, soll das herkömmliche Analysevorgehen mit einem vergleichbaren Open-Source Analysewerkzeug betrachtet werden.

Die Ausgangslage liefern einige Datenträgerabbilder aus diversen Testszenarien. Diese Datenträgerabbilder sind bezogen auf den forensischen Analyseprozess aus Abbildung 4.1 die Grundlage für die dritte Phase - dem Sichten und Aufbereiten der Daten.

Bei gängigen Analysevorgehen werden beispielsweise die Datenträgerkopien mit Betriebssystemprogrammen unter Linux oder mithilfe des Open-Source Analysetools *Autopsy*<sup>5</sup> unter Windows analysiert. Im kommerziellen Bereich existieren etliche weitere Analyse-Tools mit größerem Funktionsumfang. Nachfolgend wird Autopsy als Referenzsystem unter Windows betrachtet, da es eines der bekanntesten Analysewerkzeuge unter den kostenfreien Open-Source Programmen ist.

Die Datenträgerabbilder können in unterschiedlichen Dateiformaten vorliegen. Mithilfe klassischer Open-Source Anwendungen, wie beispielsweise *dd*<sup>6</sup> können Abbilder im sogenannten *RAW*-Format erstellt werden. Von *dd* existiert auch eine forensische Variante *dcfldd*, welche beim Kopieren auch noch Hashsummen zur Verifikation berechnet.<sup>[7]</sup> Andere Tools, wie der *FTK-Imager* können auch Datenträgerabbilder in speziellen Container-Formaten erstellen und lesen. Zum Beispiel gibt es das *EnCase Physical*-Format mit der Dateiendung *.e01* oder das *Advanced Forensic Format* mit der Endung *.aff*.<sup>[7]</sup> Diese Formate unterstützen eine bessere Extraktion von Metadaten oder bieten eine zusätzliche Datenkompression oder Verschlüsselung der darin gespeicherten Dateien an.<sup>[14, S. 35]</sup>

Es wird auch unterschieden, ob es sich um ein vollständiges Datenträgerabbild handelt oder um ein logisches Dateiarchiv. Bei dem vollständigen Datenträgerabbilder werden auch nicht allokierte Speicherbereiche innerhalb des Dateisystems, der Partition oder des Datenträgers gesichert. Hier können sich potentiell versteckte und gelöschte Dateifragmente befinden. Auf diesen Datenträgerabbilder kann auch das bereits beschriebene File Carving ausgeführt werden, um gelöschte Dateien wiederherzustellen.

---

<sup>5</sup>Siehe Link: <https://www.sleuthkit.org/autopsy/>. Letzter Zugriff: 10.8.2018. Es wird die Version 4.7.0 in der 64-bit Variante unter Windows 10 Pro genutzt.

<sup>6</sup>*dd* ist ein bekanntes Werkzeug zum Kopieren von Daten, welches unter den meisten Unix-basierten Betriebssystemen läuft. Damit können auch ganze Partitionen in einzelne logische Dateien kopiert werden.

<sup>7</sup>Siehe Link: <https://support.accessdata.com/hc/en-us/articles/222778608-What-Image-Formats-Do-AccessData-Products-Support->. Letzter Zugriff: 4.4.2018.

Ein logisches Dateiarchiv 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 Dateiarchive sind die bekannten Archiv-Formate 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 Analyseplattform untersucht werden.

Nachfolgend wird nun gezeigt wie ein Datenträgerabbild in einem konkreten Beispiel mit Autopsy (Version 4.7.0 64-bit) unter Windows 10 geöffnet und analysiert wird.

Zu Beginn wird bei Autopsy ein neuer Fall erstellt. Hierbei kann ein Name für den Fall angegeben werden und ein Verzeichnis, worin Autopsy die Anwendungsdaten der Analyse speichert. Danach können noch einige optionale Informationen, wie beispielsweise der Bearbeiter, Adressdaten, die Organisation und eine kurze Beschreibung angegeben werden (siehe Abbildung 4.2). Bei der Erstellung des Falls wird unter anderem eine neue Falldatenbank angelegt. Diese Datenbanken werden alle lokal auf dem Analyserechner gespeichert.

Im nächsten Schritt kann dem Fall eine neue Datenquelle hinzugefügt werden. Es werden folgende Typen von Datenquellen unterstützt:

- Ein Datenträgerabbild im *RAW*-Format, Encase-Format oder ein Abbild einer virtuellen Maschine (z.B. von Virtual Box).
- Ein lokales Laufwerk (z.B. eine externe Festplatte).
- Logische Dateien aus einem verfügbaren Dateisystem (z.B. ein beliebiger Ordner).
- Ein Abbild eines beliebigen Speicherbereiches in einer Datei.

Im konkreten Fall wird ein Datenträgerabbild als Datenquelle hinzugefügt. Dabei muss auch noch eine Zeitzone angegeben werden. Diese Zeitzone kann entscheidend für die Analyse der Zeitstempel auf einem Datenträger sein. In einem *ext*-Dateisystem bei gängigen

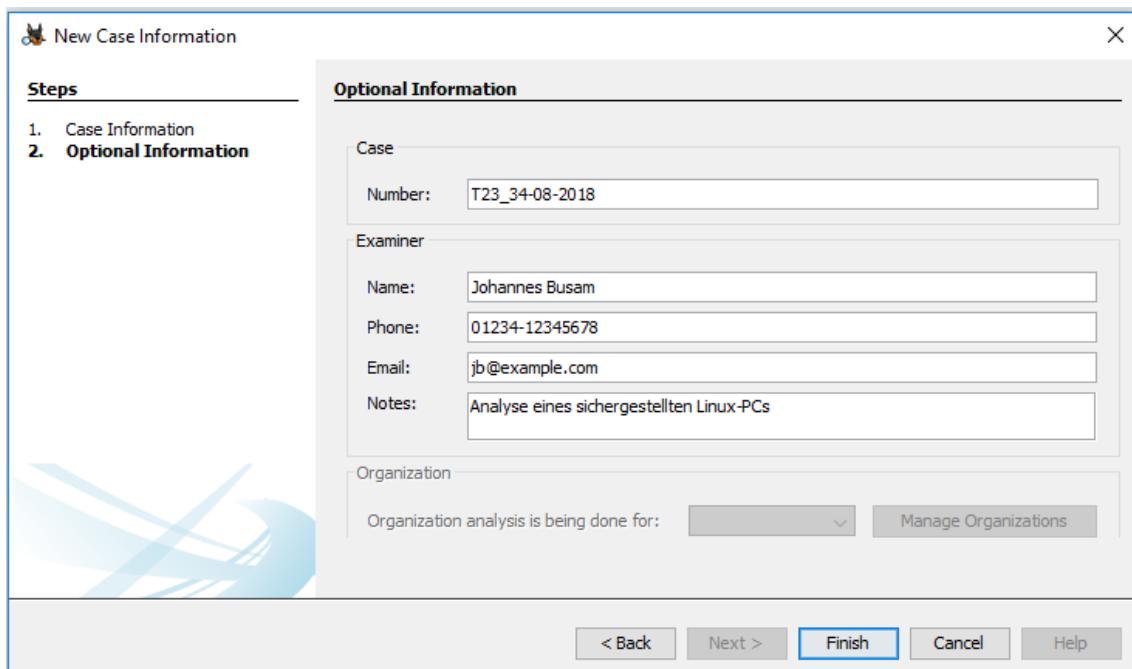


Abbildung 4.2: Erstellung eines neuen Falls mit Autopsy

Linux-PCs werden die Zeitstempel als Anzahl der Sekunden seit dem 1. Januar 1970 in der Zeitzone UTC gerechnet.[1, S. 326]<sup>8</sup> Bei den älteren *FAT*-Dateisystemen hingegen wird die Zeit ohne Zeitzone gespeichert.[1, S. 192-194] Je nachdem in welcher Zeitzone das Betriebssystem konfiguriert wurde, welches die Dateien in dem *FAT*-Dateisystem änderte, ergeben sich zeitliche Unterschiede. Daher muss bei Zeitstempeln auch später bei der Anzeige der Analyseergebnisse immer auch auf die Zeitzone geachtet werden.<sup>9</sup>

Im nächsten Schritt können diverse Module zur Datenaufbereitung aktiviert werden. Diese Module dienen zur automatischen Datenaufbereitung und werden ausführlich in Kapitel 5.1 beschrieben. In dem Kapitel erfolgt auch der Vergleich zur der automatisierten Auswertung mit der hier entwickelten Analyseplattform.

Nachdem die entsprechenden Module ausgewählt wurden, beginnt Autopsy die Datenquelle zu analysieren. Dies läuft vollständig im Hintergrund ab und der Nutzer kann parallel hierzu die Datenquelle manuell analysieren.

Das Importieren einer Datenquelle bei Autopsy besteht letztlich aus dem Erstellen eines Falls und der Angabe einiger Konfigurationsmöglichkeiten. Die Datenquelle wird hierbei nicht in einen internen Anwendungsordner kopiert, sondern während der Analyse auf dem Rechner bereitgestellt (entweder als Datenträgerabbild oder direkt als externer Datenträger). In dem Anwendungsordner von Autopsy werden Metainformationen, wie beispielsweise die Indexierung bestimmter Daten, abgespeichert um während der Analyse schneller

<sup>8</sup>Diese Zeitdefinition entspricht der sogenannten Unixzeit.

<sup>9</sup>Ein weiteres interessantes Problem ergibt sich auch bei der Sicherung der Beweismittel an einem Tatort. Auch dort ist nie garantiert, dass alle gesicherten Beweismittel überhaupt zeitlich synchronisiert sind. Gerade in Kombination mit Netzwerkverbindungsdaten können bei einer fehlenden Zeitsynchronisation kritische Abläufe zeitlich versetzt sein. Daher muss gerade die Aussagekraft von Zeitstempel immer kritisch betrachtet werden.

darauf zugreifen zu können. Prinzipiell ist dieses Vorgehen sehr gut, da der Ermittler direkt mit der Arbeit beginnen kann und keine Daten kopiert werden müssen.<sup>10</sup>

### 4.3 Umsetzung in der forensischen Analyseplattform

Im vorangegangen Kapitel wurde bereits beschrieben, wie Datenträgerabbilder als sogenannte Datenquellen in Autopsy importiert werden. Das Datenträgerabbild oder der extern angeschlossene Datenträger wird nicht in einen internen Anwendungsordner kopiert. Autopsy arbeitet direkt auf den Daten, um ein unnötiges Kopieren von Daten und dessen Ressourcenaufwand zu vermeiden.

Im Vergleich hierzu arbeitet die hier entwickelte forensische Analyseplattform auf einem eigenen Computer-Cluster basierend auf mehreren Knoten und nicht nur auf einem einzelnen Analyserechner. Daher müssen die forensisch relevanten Daten zuerst über das Netzwerk in die Analyseplattform importiert werden.

Da die Analyseplattform auf dem Hadoop-Framework aufbaut, bildet der Kern der Datenspeicherung das HDFS.<sup>11</sup> Hierbei geht es nicht nur darum, wie die Daten im Hadoop-Framework verwaltet werden, sondern vielmehr um die Art und Weise, wie Daten forensisch korrekt gespeichert werden können.

Zur Speicherung der Daten des Datenträgerabbildes im Hadoop-Framework gibt es mehrere Möglichkeiten, deren Vor- und Nachteile nachfolgend dargestellt werden sollen.

### 4.4 Variante 1 - Datenträgerabbild im HDFS speichern

Die naheliegende Variante zur Speicherung der Asservate, wäre die Datenträgerabbilder direkt im HDFS abzuspeichern. Allein die Größe der Abbilder ist nicht problematisch. Um eine entsprechende Aufteilung kümmert sich das HDFS. Allerdings hat die Lösung den entscheidenden Nachteil, bei der Weiterverarbeitung der Daten.

Auf Betriebssystemebene können solche Datenträger mit mehreren Partitionen und unterschiedlichen Dateisystemen interpretiert und eingebunden werden. Dabei 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 zum Beispiel ext4 oder NTFS enthalten. Dieses Dateisystem enthält die eigentlichen Dateien, welche logisch zusammengesetzt werden müssen.

Viele Betriebssysteme bieten hier bereits eine weitreichende Unterstützung zum Lesen und Schreiben dieser Dateisysteme. Hierzu können die Dateisysteme einzelner Partitionen des Datenträgerabbildes *gemountet* werden.

Aber innerhalb des Hadoop-Frameworks findet sich keine Unterstützung zum Lesen von beliebigen Dateisystemen. Denn normalerweise nutzen Java-Applikationen ein definierte

---

<sup>10</sup>Das Erstellen eines Datenträgerabbildes aus dem originalen Asservat wurde hierbei schon vorher durchgeführt.

<sup>11</sup>Siehe Kapitel 3.2.

Schnittstelle auf Basis von Dateien, die wiederum vom Betriebssystem bereitgestellt werden. Um die logischen Dateien aus dem Datenträgerabbild extrahieren zu können, müsste für jedes einzelne Dateisystem eine eigene Implementierung in Java geschrieben werden. Und diese Implementierung müsste dann auch noch für das HDFS-Dateisystem optimiert sein.

Darüber hinaus wäre das Extrahieren der Dateien aus einem Dateisystem auf einem Datenträgerabbild, gespeichert im HDFS, auch nicht 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 im ext4-Dateisystem eine Datei lesen zu können, sollten zumindest die Dateisystemmetadaten verfügbar sein. Darüber hinaus kann der Dateiinhalt einer einzelnen Datei verstreut innerhalb des Dateisystems liegen. Je nach Grad der Fragmentierung des Dateisystems, müssten auf einem Data-Node innerhalb des Clusters schlimmstenfalls dutzende weitere Blöcke anderer Knoten nachgeladen werden, um den Inhalt einer einzelnen Datei zu verarbeiten. Dies würde das Prinzip der Datenlokalität aushebeln.

Abbildung 4.3 skizziert diese verstreute Aufteilung einer Datei im physikalischen Hadoop-Cluster.

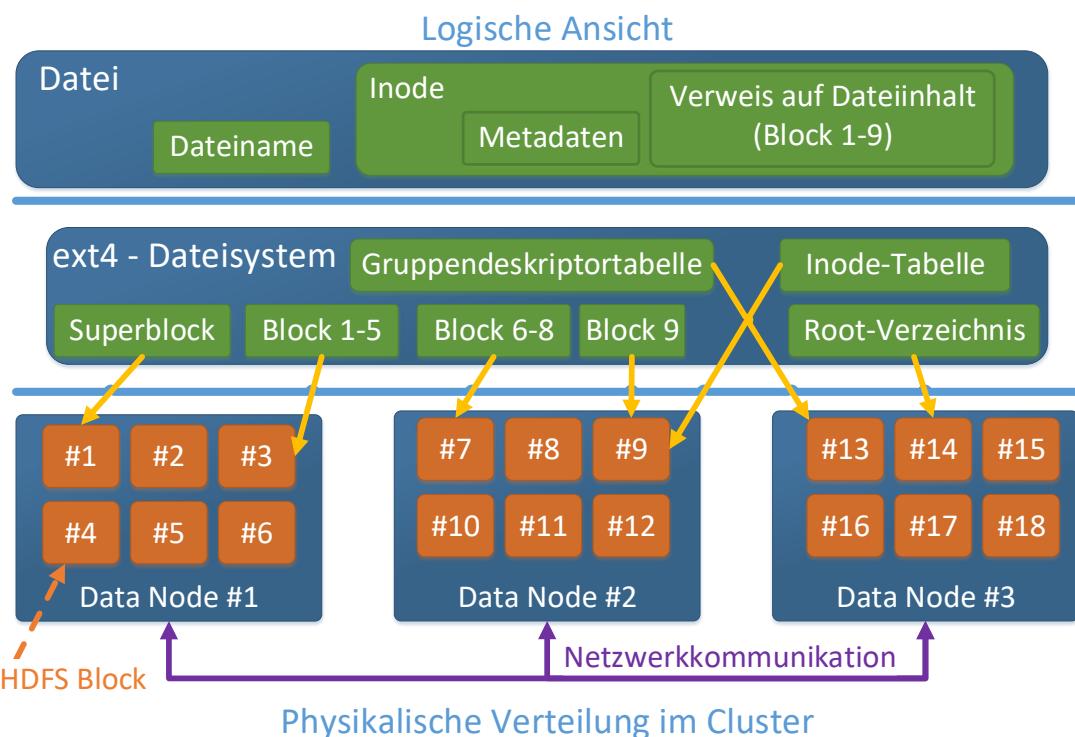


Abbildung 4.3: Aufteilung der Daten des Datenträgerabbildes im Hadoop-Cluster

Das Abbild zeigt eine Datei, welche aus logischer Sicht einen Dateinamen, Metadaten (beispielsweise Zugriffsrechte) und einen Inhalt besitzt. Der Inhalt ist in 9 Blöcke zu je-

weils 2048 Byte aufgeteilt.<sup>12</sup> Um nun den Inhalt aus dem ext4-Dateisystem einer Datei auszulesen wird zuerst der Superblock benötigt. Dieser enthält allgemeine Informationen zum Dateisystem. Darauf wird die Gruppendedeskriptortabelle benötigt, um auf einzelne Blockgruppen zuzugreifen.<sup>13</sup> Über eine Blockgruppe kann wiederum auf die Inode-Tabelle zugegriffen werden. Diese speichert die Metadaten einzelner Dateien als sogenannte *Inodes* ab. Ein Inode-Eintrag hält wiederum Verweise auf die Blöcke, welche den Dateinhalt beschreiben. Der Dateiname ist wiederum in dem logisch übergeordneten Verzeichnis gespeichert. Das oberste Verzeichnis ist das Wurzelverzeichnis. Dieses enthält die Namen der Kind-Dateien und entsprechende Verweise zum Inode.

Beim Auslesen einer Datei müssen nun etliche Speicherstellen innerhalb des Dateisystems gelesen und interpretiert werden. Angenommen, dass ein konkretes ext4-Dateisystem in einer 100 GB großen Partition gespeichert wird, so wird diese große Datei im HDFS-Dateisystem in 800 große Blöcke zu je 128 MB aufgeteilt und auf den einzelnen Knoten des Clusters gespeichert. Hierbei kann aber kein Einfluss darauf genommen werden, wo welche Blöcke mit welchem Inhalt gespeichert werden. Letztlich bedeutet dies wiederum, wenn in einem Apache Spark Executor zu Verarbeitung der Daten eine Datei des Knotens gelesen werden soll, müssen schlimmstenfalls etliche Datenblöcke von anderen Knoten angefordert werden. Dadurch wird auf Netzwerkebene unnötig viel Last erzeugt. Letztlich gilt für das Prinzip der Datenlokalität, dass die einzelnen Blöcke im HDFS möglichst unabhängig voneinander verarbeitet werden können müssen. Diese Problematik trifft übrigens nicht nur bei der Ext-Dateisystemfamilie auf sondern auch bei anderen Dateisystemen.

Aus den oben genannten Gründen ist die direkte Speicherung von Datenträgerabbildern im HDFS nicht geeignet für die Analyse im Hadoop-Cluster.

## 4.5 Variante 2 - Logische Dateien im HDFS speichern

In dieser Variante wird das Beweismittel auf dem lokalen Analyserechner gemountet. Darauf aufbauend werden alle Dateien auf logischer Ebene direkt in das HDFS importiert. Damit ist die gesamte Dateisystemstruktur aus dem Datenträgerabbild im HDFS abgelegt. Einzelne Dateien aus dem Datenträgerabbild sind nun auch als einzelne Dateien im HDFS gespeichert und können unabhängig voneinander prozessiert werden. Darüber hinaus ist die Datenstruktur im HDFS unabhängig von dem Dateisystem des importierten Datenträgerabbildes.

Damit sind die Nachteile der vorangegangenen ersten Variante aus Kapitel 4.4 behoben. Allerdings ist das bereits erwähnte File Carving, beziehungsweise das Auffinden von gelöschten Dateien nun nicht mehr im Hadoop-Framework möglich. Denn bei dieser Variante werden ja nur die Dateien aus dem Dateisystem in das Hadoop-Framework importiert und nicht allokierte Speicherbereiche werden nicht weiter untersucht. Theoretisch wäre es aber auch möglich zusätzlich das vollständige Datenträgerabbild in das HDFS zu importieren, um dann später freie Speicherbereiche analysieren zu können. Im Rahmen dieser Thesis wird diese Einschränkung jedoch vorerst akzeptiert. Eine File Carving ist daher mit der hier entwickelten forensischen Analyseplattform noch nicht möglich. Andererseits können nun in Analogie zur Referenzsoftware Autopsy auch beliebige Verzeichnisse als Datenquelle in

---

<sup>12</sup>Die Blockgröße wird hierbei vom ext4-Dateisystem bestimmt und ist die kleinste allozierbare Einheit im Dateisystem. In der physikalischen Aufteilung im HDFS gibt es auch Blöcke, welche aber eine Größe von 128 MB aufweisen (orange in Abbildung 4.3).

<sup>13</sup>Das Dateisystem ist in mehrere autarke Bereiche unterteilt, welche für sich genommen eigenständig Daten einer Teilmenge aller Dateien vorhalten. Dies sind die sogenannten Blockgruppen.

die Analyseplattform geladen werden.<sup>14</sup>

Interessant an dieser Variante ist das Verhalten des HDFS-Dateisystems bezüglich der Metadaten und der unterschiedlichen Größen von Dateien. Zur Analyse der Dateien auf dem Datenträger werden auch die Metadaten zu den Dateien benötigt. Beim Importieren muss darauf geachtet werden, dass alle Metadaten des lokalen Dateisystems im Datenträgerabbild unverändert in das HDFS kopiert werden. Im HDFS werden auch Metadaten zu den einzelnen Dateien gespeichert. Möglicherweise könnten diese Dateiattribute wiederverwendet werden, um die Metadaten aus dem originalen Dateisystem zu speichern. Welche Metadaten bereits im HDFS mit angelegt werden, zeigt Abbildung 4.4 anhand eines Ausschnitts aus der Web-Repräsentation eines HDFS.

## Browse Directory

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

Abbildung 4.4: HDFS - Dateieigenschaften

Daraus ist ersichtlich, dass jede Datei entsprechende Dateirechte hat und einem Nutzer und einer Gruppe zugeordnet ist. Zusätzlich wird die Größe und der Zeitstempel der letzten Änderung gespeichert.<sup>15</sup> 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 abgelegt werden und damit auf keinem DataNode explizit repliziert werden. Auch die Blockgröße ist in der Standardkonfiguration auf 128 MB festgelegt. Wie im Grundlagenkapitel 3.2 erwähnt, werden die Dateien in mehreren Blöcken zu maximal 128 MB gespeichert und auch repliziert.<sup>16</sup>

Bezogen auf klassische Dateisysteme entspricht ein Block im HDFS einem Block im Ext4-Dateisystem oder einem Cluster im NTFS-Dateisystem. Es ist letztlich die kleinste allozierbare Dateneinheit im Dateisystem.[1, S.129-140] 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.[12, S. 16-17]

Mit dem Befehl aus Listing 4.1 können lokale Dateien in das HDFS importiert werden.

<sup>14</sup>Siehe Kapitel 4.2.

<sup>15</sup>Bei Dateiverzeichnissen ist die Größe 0 Byte.

<sup>16</sup>Diese Blockgröße kann aber konfiguriert werden.

```

1 # hdfs dfs -put [source] [destination]
2 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.<sup>17</sup> Ratsam ist dies jedoch nicht, da der Nutzer, die Gruppe und die dazugehörigen Zugriffsrechte in einem produktiven HDFS-Cluster verwendet werden, um Zugriffsbeschränkungen einzelner Nutzer und Anwendungen auf Dateien im HDFS umzusetzen. Die Metadaten des originalen Dateisystems sollten daher mit einer anderen Methode im HDFS gespeichert werden.

Eine bessere Möglichkeit bietet das HDFS mithilfe von erweiterten Dateiattributen. Diese können beliebige Informationen zu einer Datei speichern. Mit nachfolgenden Befehlen kann beispielsweise der Zeitstempel der Erstellung einer Datei aus dem ursprünglichen Dateisystem im HDFS als erweitertes Attribut gespeichert und ausgelesen werden. Hierbei kann der Name des Attributs (*user.ntfs.creationtime*) und dessen Inhalt (2018-04-07T11:14:42,798583789+02:00) frei gewählt werden.

```

1 # Create custom file attribute
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 ist es also prinzipiell möglich, alle Metadaten des ursprünglichen Dateisystems als erweiterte Dateiattribute im HDFS zu speichern. Allerdings müssen diese Metadaten für jede Datei im HDFS eingetragen werden.

Zusätzlich müssen beim Verarbeiten der Daten mit Apache Spark die Metadaten auslesbar sein. Dieses Auslesen ist umständlich aber möglich.<sup>18</sup>

Es gibt jedoch zwei entscheidende Nachteile bei dieser Variante, die beide den gleichen Ursprung haben. Der erste Nachteil ist, dass jede einzelne Datei aus dem ursprünglichen Dateisystem als getrennte eigenständige Datei in das HDFS hochgeladen werden muss. In einem einzigen Dateisystem können Millionen kleine Dateien gespeichert sein, die alle auch in das HDFS importiert werden müssen. Allerdings ist das HDFS primär für größere Dateien (in der Größenordnung einer Blockgröße von 128 MB) ausgelegt und kann viele kleine Dateien nicht wirklich effizient speichern und bereitstellen. Dies ist dem Umstand geschuldet, dass alle Metadaten auf dem Name Node gespeichert werden und nur der Dateiinhalt im Cluster aufgeteilt wird.[12, S. 16] Wenn nun viele kleine Dateien gespeichert werden, dann steigt der Speicherverbrauch im NameNode an und beeinträchtigt die Performance des Systems. Der zweite Nachteil verstärkt dieses Problem. Denn wenn nun auch

<sup>17</sup>Hierzu kann dem *put*-Befehl aus Listing 4.1 der Parameter *-p* mit übergeben werden.

<sup>18</sup>Siehe Metadaten-Extraktion im Projekt *foam-processing-spark* unter <https://github.com/jobusam/foam-processing-spark>.

noch die Metadaten aus dem ursprünglichen Dateisystem als erweiterte Metadatenattribute im HDFS gespeichert werden, dann benötigt der Name Node noch mehr Ressourcen. Darüber hinaus wäre beispielsweise die Verarbeitung der Metadaten mit Apache Spark nur bedingt parallelisierbar, da diese Daten immer zuerst an den einen Name Node angefordert und über das Netzwerk zu den einzelnen Spark Executoren gesendet werden müssten. Erst dann könnten letztere Executoren die Daten parallel verarbeiten. Abbildung 4.5 skizziert nochmals den Datenfluss bei dieser Lösung.

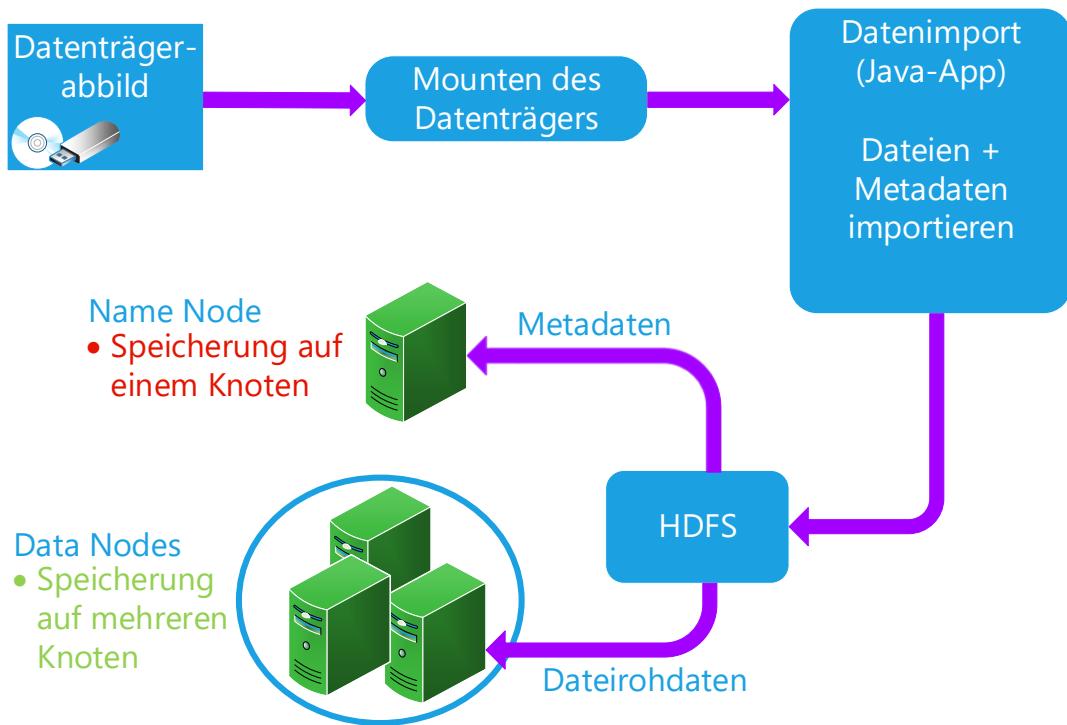


Abbildung 4.5: Datenspeicherung mit erweiterten Dateiattributen im HDFS

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. Nicht zuletzt werden alle Datei-Metadaten physikalisch im Name Node gespeichert. Dadurch benötigt der Name Node mehr Speicher und könnte zu einem Flaschenhals im System werden. Zudem wäre die parallelisierte Datenverarbeitung eingeschränkt, da die Metadaten immer zuerst von dem Name Node angefordert werden müssen. Aus diesen Gründen ist die Variante 2 nicht akzeptabel.

#### 4.6 Variante 3 - Speicherung in Dateicontainer

Die vorangegangene Variante überzeugt nicht, da die Speicherung vieler kleiner Dateien nicht effizient im HDFS durchgeführt werden kann. Da alle Metadaten einer Datei im Name Node abgelegt werden, bildet der Name Node bei vielen kleinen Dateien ein Flaschenhals. Zur Lösung dieser Problematik existieren im Hadoop-Umfeld diverse Dateicontainer. Diese

Dateicontainer können mehrere Dateien in einem strukturierten Format speichern. In Analogie zu bekannten Dateicontainern, wie beispielsweise *ZIP*-Archiven oder *TAR*-Archiven existieren im Hadoop-Umfeld *Sequence Files*, *RC/ORC Files*, *Avro-Files* oder *Parquet-Files*.<sup>[11, S. 296]</sup>

Ziel dieser Dateiformate ist es, viele kleine Daten in größere Dateien zu speichern, um eine bessere Parallelisierung und eine effiziente Speicherung in Hadoop zu ermöglichen. Oftmals unterstützen diese Formate auch eine Datenkompression, um gegebenenfalls Speicherplatz sparen zu können. Viel wichtiger ist jedoch die Teilbarkeit dieser Dateiformate. Wie bereits beschrieben, werden Dateien im HDFS in größere Blöcke geteilt und auf unterschiedlichen Knoten gespeichert. Innerhalb eines Blocks muss es also möglich sein, einzelne Einträge beziehungsweise Datensätze lesen zu können. Die oben erwähnten Hadoop Dateiformate unterstützen eben diese Teilbarkeit.<sup>[19]</sup>

Nachfolgend soll das *Sequence File*-Format in Abbildung 4.6 näher betrachtet werden.<sup>[20]</sup>

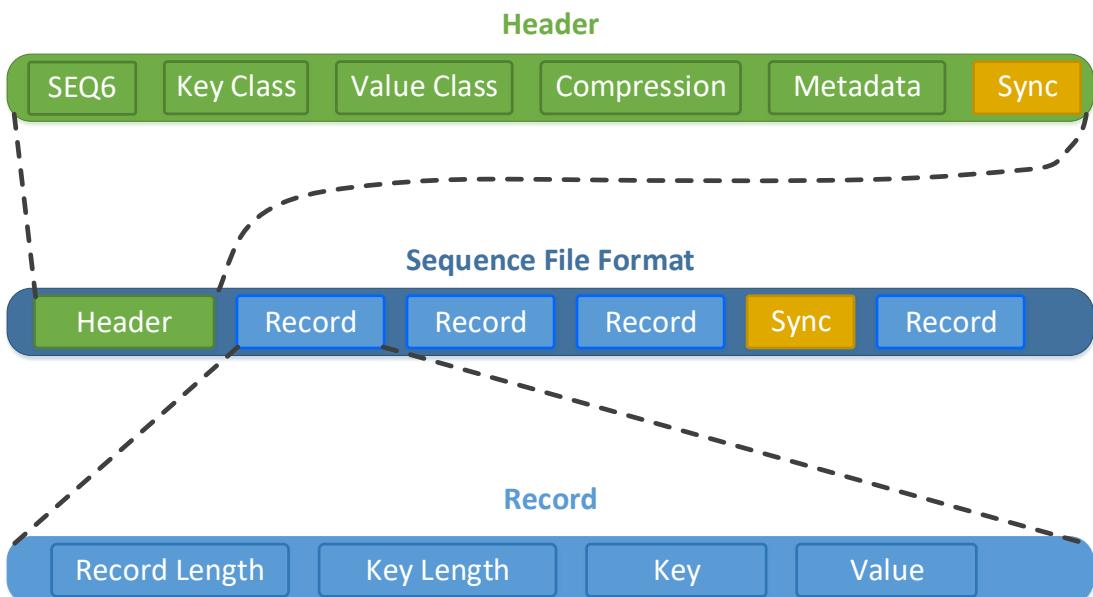


Abbildung 4.6: Sequence File Format (Vgl. [10, S. 134])

Das Sequence File Format besteht aus einem Header und mehreren Einträgen. Diese Einträge wiederum sind eigenständige Schlüsselwertpaare, welche die eigentlichen Daten beinhalten. Ein Schlüsselwertpaar, in Abbildung 4.6 auch *Record* genannt, enthält einen eindeutigen Schlüssel (*Key*) und einen Inhalt (*Value*). Zusätzlich wird am Anfang die Gesamtlänge

<sup>[19]</sup>Im Gegensatz hierzu sind beispielsweise die meisten Dateisysteme in einer Partition eben nicht teilbar. Denn wenn ein Block in der Mitte des Dateisystems ausgelesen werden soll, so können die Daten nicht ohne die Dateisystemmetadaten interpretiert werden. Daher ist auch die erste Variante aus Kapitel 4.4 eben nicht im HDFS anwendbar.

<sup>[20]</sup>Siehe auch [10, S. 134] und <https://hadoop.apache.org/docs/r2.7.5/api/org/apache/hadoop/io/SequenceFile.html>, Stand 18.8.2018.

des Schlüsselwertpaars und die Länge des Schlüssels in Bytes angegeben. Damit ist es möglich den Schlüssel sowie den Inhalt innerhalb des Sequence Files zu bestimmen. Allerdings ist dadurch noch nicht klar, wie der Schlüssel oder Inhalt zu interpretieren ist. Daher sind im Header des Sequence File Formats jeweils die Namen der Java-Klassen in den Feldern *Key Class* und *Value Class* gespeichert, die den Datentyp definieren. Im Header befinden sich auch Informationen, ob und in welcher Form eine Datenkompression auf die Daten angewendet wurde.

Zuletzt werden in unregelmäßigen Abständen sogenannte Sync-Felder abgespeichert. Diese dienen zur Unterstützung der eingangs erwähnten Teilbarkeit des Dateiformats. Eine Anwendung kann sich mithilfe der Sync-Felder von einer beliebigen Stellen aus innerhalb der Datei auf den Anfang eines Schlüsselwertpaars synchronisieren. Damit können auch Schlüsselwertpaare aus einem beliebigen HDFS-Block interpretiert werden.<sup>21</sup>

Prinzipiell können damit sehr kleine Dateien in mehrere Sequence Files strukturiert im HDFS gespeichert werden. Diese Variante enthält aber noch einige Hindernisse. Zuerst stellt sich hier die Frage, wie diese Sequence Files erstellt werden sollen. Hierbei müsste die Anwendung zum Datenimport auf dem Analyse-Rechner die Sequence Files zuerst lokal auf dem Rechner erstellen und danach in das HDFS hochladen. Gegebenenfalls müssten diese Sequence Files sogar noch auf dem lokalen Rechner persistent zwischengespeichert werden. Die Datenimport-Anwendung würde mehr Ressourcen benötigen.

Ein interessanteres Problem ist aber die Datenverarbeitung im HDFS. Angenommen es existieren nun Sequence Files mit den Dateiinhalten und den Metadaten in den Dateien. Mit Apache Spark können diese Sequence Files gelesen werden und beispielsweise die Hashtsummen und die Medientypen ermittelt werden.<sup>22</sup> Allerdings müssten beim Schreiben die Daten in neue Sequence Files geschrieben werden. Denn im HDFS ist es nicht möglich existierende Dateien wahlfrei zu modifizieren.[11, S. 42] Es können maximal neue Daten an das Dateiende einer Datei geschrieben werden. Damit müssten bei der Datenverarbeitung nochmals alle Daten neu geschrieben werden. Dies führt zu unnötigem Ressourcenverbrauch. Andererseits könnten die Rohdaten und die Metadaten in getrennte Sequence Files gespeichert werden. Damit müssten dann nur Sequence Files mit den Metadaten neu geschrieben werden. Es wäre auch denkbar alle neu gewonnenen Metadaten in eigenständige neue Sequence Files zu schreiben.

Allerdings könnte dies wiederum zu einer Art von Fragmentierung von logisch zusammenhängenden Daten führen. Denn nun sind die Metadaten und Rohdaten in mehrere Sequence Files aufgeteilt. Schlimmstenfalls könnten dadurch die Metadaten und Rohdaten, welche eine logische Datei repräsentieren, auf unterschiedlichen Knoten im Hadoop-Cluster liegen. Dies würde das Prinzip der Datenlokalität in gewissem Maße beeinträchtigen. Dies hängt aber auch stark davon ab, wie die Daten für die Verarbeitung angefordert werden. Ist es notwendig die Metadaten und die Rohdaten einer logischen Datei gemeinsam zu verarbeiten, oder können diese auch unabhängig voneinander auf getrennten Knoten prozessiert werden?

Zusammengefasst ist diese Variante technisch möglich. Allerdings besteht eben diese Problematik beim Speichern von neu gewonnenen Informationen. Letztlich ist es durchaus sinnvoll wenn die Rohdaten, die originalen Metadaten aus dem ursprünglichen Dateisystem und die neu gewonnenen Metadaten bei der Datenanalyse für eine logische Datei auch

---

<sup>21</sup>Voraussetzung hierbei ist allerdings, dass die Anwendung vorher schon die Datentypen der Schlüsselwertpaare kennt.

<sup>22</sup>Siehe auch Kapitel 5.

physikalisch zusammen gespeichert werden.<sup>23</sup>

Ein anderer Aspekt ist auch die Komplexität der Anwendung. Bei der praktischen Implementierung dieser Variante müsste immer betrachtet werden, wo nun welche Informationen liegen und wie letztlich alle Informationen zu einer Datei aus den Sequence Files zusammengezetzt werden müssen. Dadurch wäre die Implementierung schon sehr komplex. Darauf hinaus sollen ja nur kleine Dateien in Sequence Files abgelegt werden. Große Dateien hingegen könnten ja direkt im HDFS gespeichert werden. Dies würde die Anwendungskomplexität weiter erhöhen. Daher überzeugt auch diese Variante nicht zur Datenspeicherung.<sup>24</sup>

## 4.7 Variante 4 - Speicherung mit HBASE und HDFS

Die vorherige Variante beschreibt einen möglichen Ansatz zu Speicherung von kleinen Dateien. Allerdings wurde die Speicherung der Metadaten noch nicht optimal gelöst. Zumal beachtet werden sollte, dass während der Datenverarbeitung weitere Metadaten aus den Rohdaten ermittelt und gespeichert werden.<sup>25</sup>

Da es sich bei den Metadaten um strukturierte Daten handelt, wäre die Speicherung in einer Datenbank naheliegend. Im Hadoop-Umfeld kann hierzu die spaltenorientierte *NoSQL*-Datenbank *Apache HBASE* verwendet werden.<sup>26</sup>

Auch die Speicherung von kleinen Dateien könnte von HBASE übernommen werden. Große Dateien hingegen könnten direkt im HDFS gespeichert werden.

### 4.7.1 Speicherung kleiner Dateien

Diese Problematik von kleinen und großen Dateien wurde bereits in den vorherigen Varianten angesprochen. An dieser Stelle soll diese Thematik nochmals näher betrachtet werden. Nicht zuletzt soll anhand einiger Datenträgeranalysen ein Grenzwert ermittelt werden, nach welchem die Analyseplattform die Dateien entweder in HBASE oder im HDFS ablegt.

Wie bereits beschrieben, kann das HDFS mit großen und kleinen Dateien umgehen. Letztlich ist die Speicherung in große Dateien der primäre Anwendungsfall. Im Gegensatz dazu können viele kleine Dateien nicht effizient gespeichert werden. Es geht aber nicht darum, dass eine einzelne kleine Datei weniger effizient abgespeichert werden kann als eine große Datei. Vielmehr kann der Informationsgehalt einer einzeln großen Datei (beispielsweise als Sequence File) effizienter gespeichert werden, als der gleiche Informationsgehalt aufgeteilt in dutzende kleine Dateien.

Dies liegt daran, dass für jede Datei Metadaten gespeichert werden. Und diese Metadaten werden auf dem Name Node gespeichert und auch im Arbeitsspeicher vorgehalten.[16] Ein Eintrag ist beispielsweise ungefähr 150 bis 200 Bytes groß. Für eine Datei wird ein Metadateneintrag und ein Blockeintrag im Name Node angelegt (insgesamt 300-400 Byte). Wenn nun ein Million kleine Dateien abgespeichert werden, dann werden 300-400 MB an Arbeitsspeicher benötigt. Dies klingt eigentlich nach einem vertretbaren Ressourcenverbrauch.

---

<sup>23</sup>Zumindest sollten die Daten auf dem gleichen Knoten liegen, um Netzwerkverkehr zu vermeiden.

<sup>24</sup>Die Entscheidung, diese Variante nicht zu verfolgen, basiert nur auf den theoretischen Vorüberlegungen. Auf eine prototypische Implementierung dieser Variante wurde verzichtet, weil mit der vierten Variante zur Datenspeicherung ein Ansatz gefunden wurde, der auch schon in der Theorie mehr überzeugt also die Variante mit Sequence Files.

<sup>25</sup>Siehe Kapitel 5.

<sup>26</sup>Siehe Kapitel 3.5.

Letztlich geht es aber auch um die Netzwerklast. Denn bei der Verarbeitung der Daten werden auch eine Million Aufrufe an den Name Node gesendet, da nur er weiß, wo der Dateinhalt liegt. Wenn nun diese eine Million Dateien in ein einzelnes Sequence File gepackt werden, dann benötigt der Name Node nur 300-400 Byte Arbeitsspeicher und zur Datenverarbeitung werden weniger Netzwerkressourcen benötigt.

Nun stellt sich die Frage, ob eine Million Dateien realistisch anzusehen sind und wie groß kleine Dateien sind. Nachfolgende Abbildungen zeigen hier die Resultate einer Analyse der Dateigröße von diversen Datenträgerabbildern. Abbildung 4.7 zeigt hier die kumulierte Häufigkeit der Dateien unterteilt in mehrere Dateikategorien. Diese Kategorien sind logarithmisch nach dem dekadischen Logarithmus aufgeteilt.<sup>27</sup> Eine Kategorie beschreibt die Anzahl aller Dateien in einem Datenträgerabbild, welche kleiner als die angegebene Kategoriegröße ist. Zum Beispiel existieren auf dem Datenträgerabbild des Windows Systems knapp 400.000 Dateien die kleiner 10 Kilobyte sind.

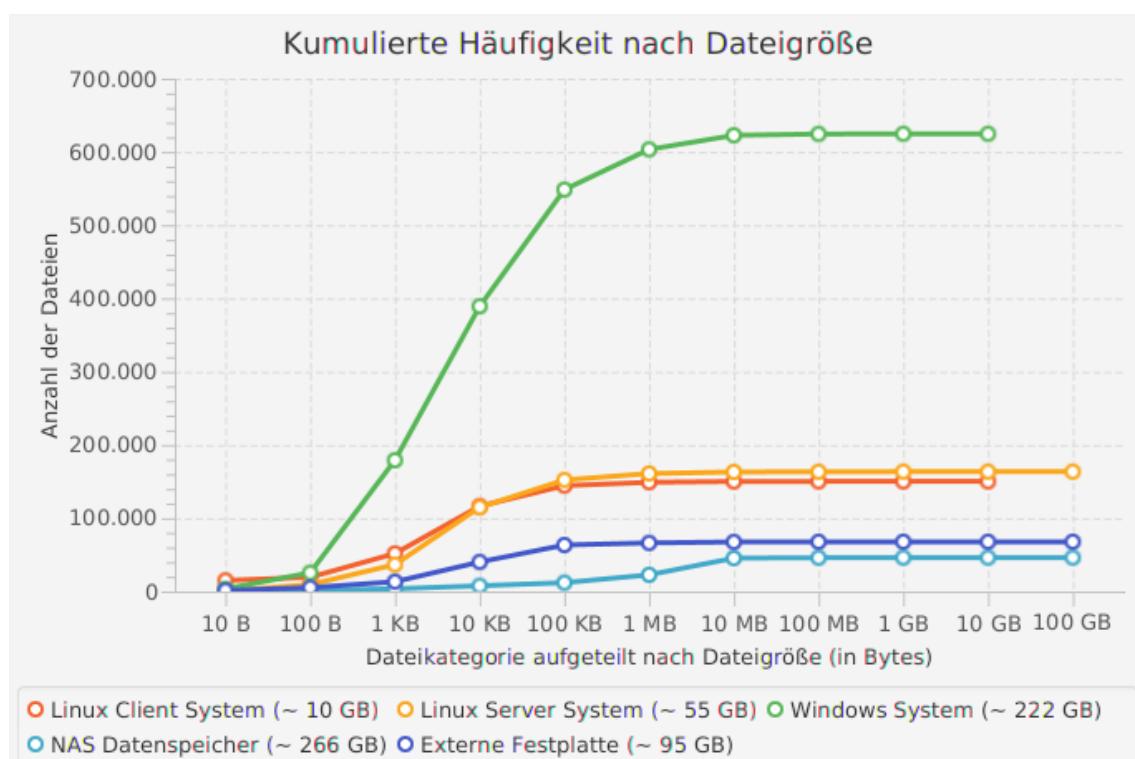


Abbildung 4.7: Kumulierte Häufigkeit nach Dateigröße

Nachfolgend werden die Testdaten beschrieben:

- Das *Linux Client System* ist ungefähr 10 GB<sup>28</sup> groß.  
Darauf ist ein Ubuntu-Betriebssystem installiert. Es wurde als Testdatenträgerabbild im Rahmen Masterthesis erstellt. Das Abbild enthält 150.229 Dateien.<sup>29</sup>

<sup>27</sup>Hierbei wurde die Ergebnismenge der Kategorien linear interpoliert. Der Quellcode zur Berechnung dieser Diagramme wird unter <https://github.com/jobusam/foam-data-analysis-ui> bereitgestellt.

<sup>28</sup>Die Größenangaben entsprechen den reinen Rohdaten der Dateien. Verzeichnisse und Dateisystemmetadaten sind nicht inkludiert.

<sup>29</sup>Es handelt sich ausschließlich um Datendateien. Verzeichnisse, Symbolische Links und Spezielle Dateien wurden nicht berücksichtigt.

- Das *Linux Server System* ist ungefähr 55 GB groß.  
Darauf ist ein CentOS-Betriebssystem installiert. Das System ist ein Name Node eines kleinen Hadoop-Clusters. Das Abbild enthält 163.555 Dateien.
- Das *Windows System* ist ungefähr 222 GB groß und ist ein reales Nutzersystem, welches seit mehreren Monaten genutzt wird. Das Abbild enthält 624.650 Dateien.
- Der *NAS Datenspeicher* entspricht einem QNAP-System mit ungefähr 266 GB an realen Rohdaten. Hierbei ist auf dem Datenträgerabbild kein Betriebssystem installiert. Es handelt sich hauptsächlich um Dokumente und Mediendateien. Das Abbild enthält 46.215 Dateien.
- Die *Externe Festplatte* mit ungefähr 95 GB Daten, wird als Backup für diverse Mediendateien genutzt. Das Abbild enthält 67.809 Dateien.

Da die forensische Analyseplattform mehrere Datenträgerabbilder speichern kann, sollte das System durchaus mehrere Millionen Dateien verarbeiten können. Abbildung 4.8 relativiert die Ergebnisse der einzelnen Abbilder.

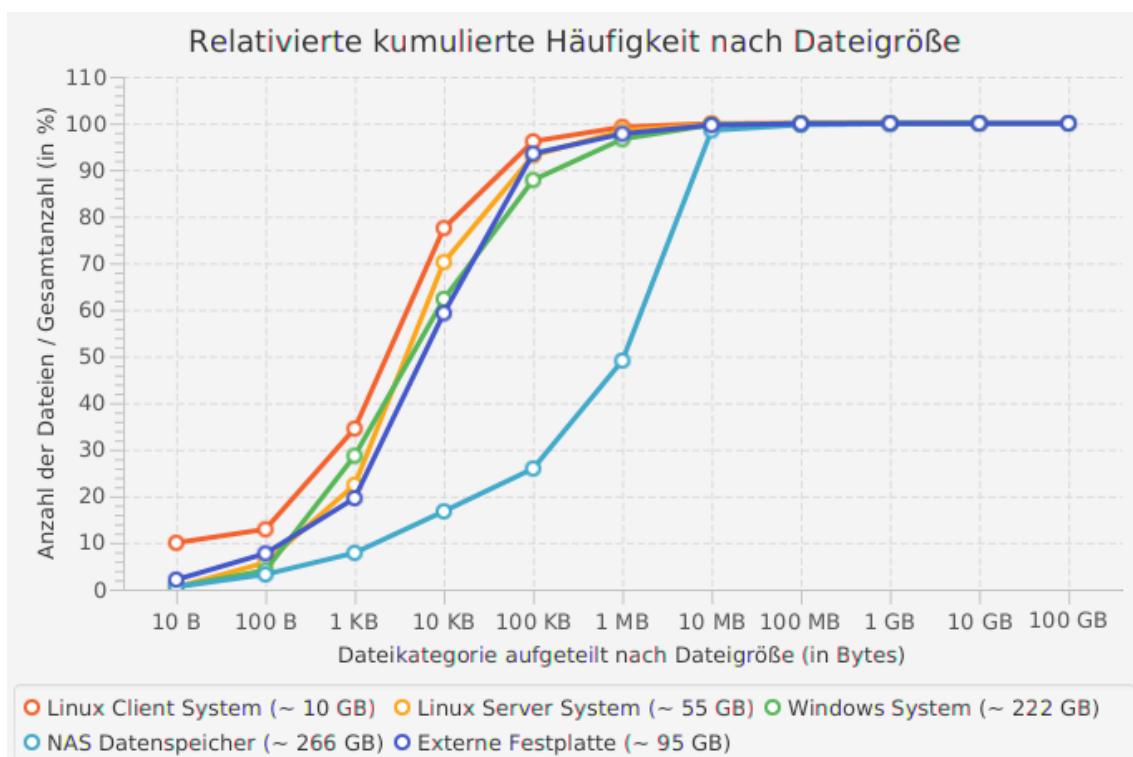


Abbildung 4.8: Relativierte kumulierte Häufigkeit nach Dateigröße

Anhand der relativierten kumulativen Häufigkeit aus Abbildung 4.8 wird klar, dass fast 80-90 % der Dateien kleiner 100 Kilobyte sind und mehr als 95% der Dateien kleiner 10 Megabyte sind. Allerdings sind diese Angaben mit Vorsicht zu genießen, denn je nach Anwendungsfall können Datenträger beliebige Dateien unterschiedlicher Größe speichern. Ist beispielsweise ein Betriebssystem auf dem Datenträger installiert, existieren allein durch das Betriebssystem tausende von Dateien mit minimaler Dateigröße. Umgekehrt enthält das Datenträgerabbild des NAS-Datenspeichers sehr viele Dateien zwischen 1 und 10 MB.

Dies liegt daran, dass von den 46.215 Dateien ungefähr 30.000 Dateien Fotos sind. Diese wiederum sind für gewöhnlich 500 Kilobyte bis 10 Megabyte groß. Die Kurve könnte allerdings anders aussehen, wenn beispielsweise auch Filme und Videos auf dem NAS gespeichert wären.

Die nachfolgende Abbildung 4.9 und die Abbildung 4.10 zeigen die absolute und relativierte kumulative Kategoriegröße der einzelnen Kategorien an. Die Kategoriegröße beschreibt die Gesamtgröße aller Dateien einer spezifischen Kategorie. Beispielsweise sind bei dem Windows System ungefähr 90 Gigabyte der Gesamtdatengröße in Dateien kleiner 10 Megabyte gespeichert.

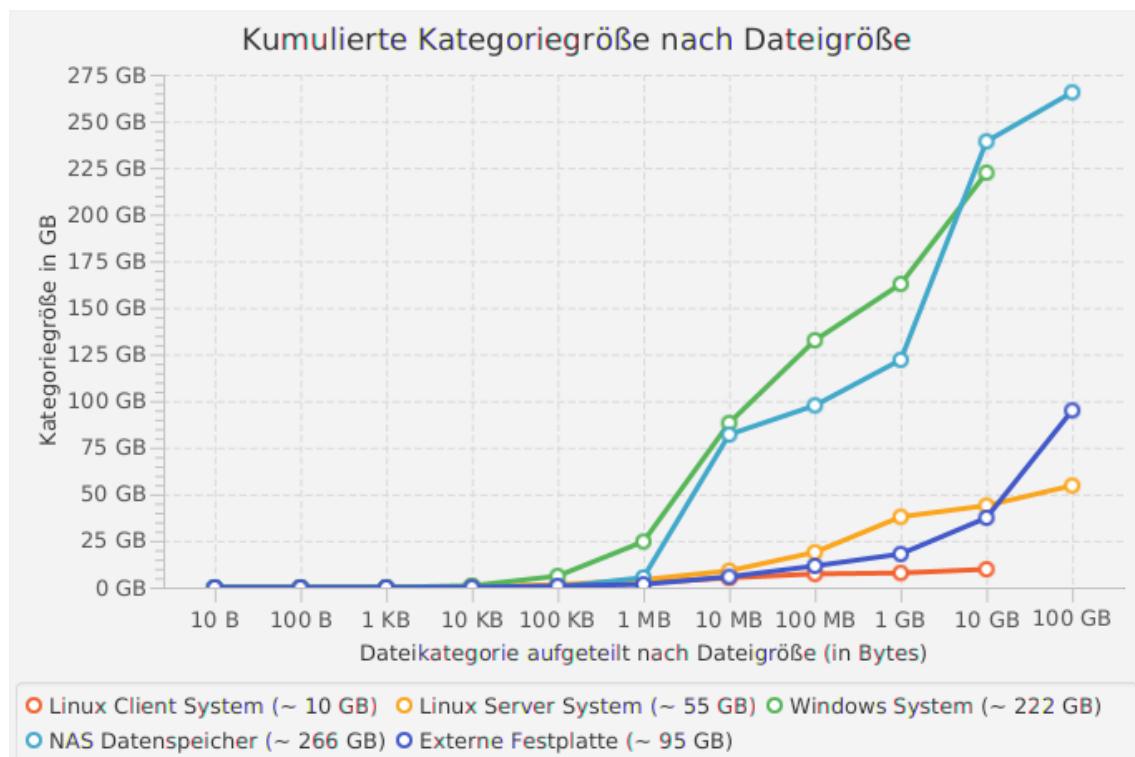


Abbildung 4.9: Kumulierte Kategoriegröße nach Dateigröße

Die relativierte kumulative Kategoriegröße in Abbildung 4.10 verdeutlicht den Kontrast in Bezug auf die relativierte kumulative Häufigkeit aus Abbildung 4.8. Während mehr als 95% aller Dateien kleiner 10 Megabyte sind, beansprucht dieser Anteil doch nur ungefähr 10 bis 55 % der Gesamtspeichergröße.

Anhand der Diagramme empfiehlt es sich den Grenzwert der Dateigröße zwischen 1 und 10 Megabyte zu definieren. Für die Implementierung im Rahmen der Thesis wird der Grenzwert für die forensische Analyseplattform auf 10 Megabyte gesetzt. Dies bedeutet, dass alle Dateien kleiner 10 Megabyte direkt in HBASE gespeichert werden. Darunter fallen beispielsweise auch größtenteils Fotos und Dokumente. Und nur die wenigen großen Dateien (größer 10 Megabyte) werden direkt im HDFS gespeichert.

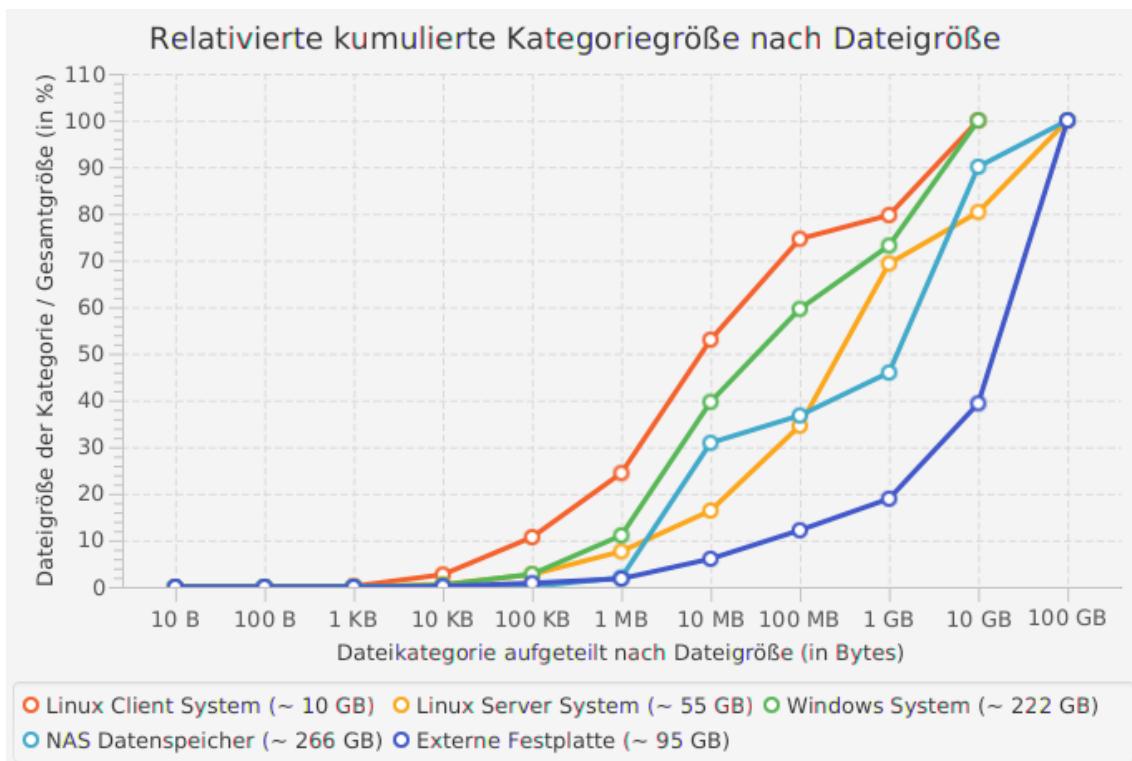


Abbildung 4.10: Relativierte kumulierte Kategoriegröße nach Dateigröße

#### 4.7.2 Anwendungimplemtenierung

Die Variante zu Datenspeicherung im HDFS-Dateisystem und der HBASE-Datenbank wird im Rahmen dieser Thesis implementiert. Das GitHub-Projekt *foam-data-import* enthält diese Anwendung.<sup>30</sup> Abbildung 4.11 skizziert die Datenaufbereitung und Speicherung in HBASE und im HDFS.

Der Datenimport ist aufgeteilt in zwei Schritte. Der erste Schritt ist das Mounten des Datenträgerabbildes (siehe Abbildung 4.11). Darüber hinaus müssen die Zugriffsrechte geprüft werden. Die eigentliche Datenimport-Applikation sollte aus sicherheitstechnischen Gründen nicht mit erhöhten Privilegien ausgeführt werden. Daher müssen beim Mounten der Abbilder entsprechende Vorkehrungen getroffen werden. Dieser Vorgang des Mountens muss derzeit manuell mit Betriebssystemwerkzeugen durchgeführt werden. In Kapitel 4.7.5 wird die detailliert beschrieben, wie die Zugriffsrechte berücksichtigt werden.

Im zweiten Schritt wird die Datenimport-Anwendung genutzt, um die Daten zu importieren. Die Applikation importiert ein vorgegebenes Verzeichnis. Dies kann entweder ein gemountetes Datenträgerabbild sein oder aber auch ein beliebiges logisches Verzeichnis.<sup>31</sup> Beim Importieren wird jede einzelne Datei des vorgegebenen Verzeichnisses analysiert. Es werden allgemeine Metadaten, wie Name, Größe, Zeitstempel, Zugriffsrechte und Dateityp ermittelt. Das Datenmodell wird detailliert in Kapitel 4.7.3 beschrieben. Abhängig von

<sup>30</sup>Siehe <https://github.com/jobusam/foam-data-import>.

<sup>31</sup>Damit können analog zu Autopsy unterschiedliche Datenquellen importiert werden. Beispielsweise könnte dies ein gemountetes Datenträgerabbild, ein lokaler Datenträger oder einfach nur ein beliebiges Verzeichnis sein. Siehe Kapitel 4.2.

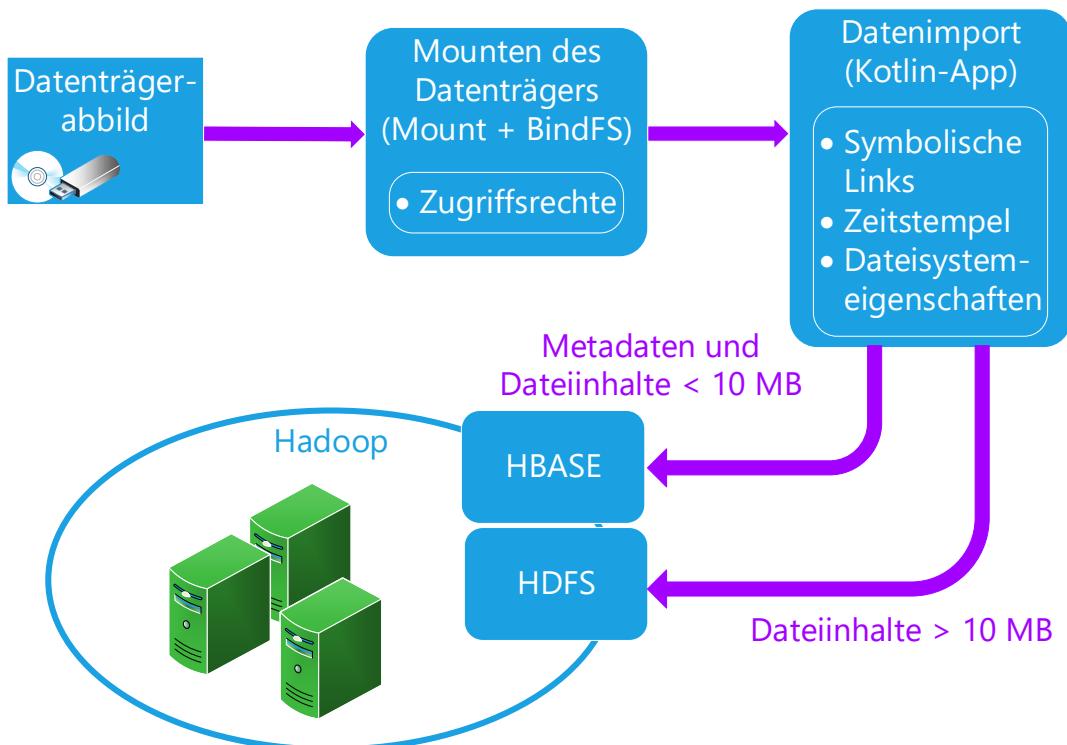


Abbildung 4.11: Datenimport in HBASE und HDFS

dem Datentyp handelt es sich um ein Verzeichnis, eine Datendatei oder um eine spezielle Datei, wie beispielsweise einen symbolischen Link. Wenn es eine Datendatei ist, dann wird die Dateigröße geprüft. Ist die Datendatei größer als 10 MB dann wird sie direkt im HDFS unter einem vorher konfigurierten Dateipfad abgelegt. Ist die Datei kleiner oder gleich groß, dann wird sie zusammen mit den Metadaten in HBASE gespeichert.

Die Applikation selbst parallelisiert die Metadatenextraktion und den Datenversand über das Netzwerk. Dennoch kann es bei großen Datenträgern durchaus lange dauern, da hier auch die Bandbreite des Netzwerks und vor allem auch die Lesegeschwindigkeit des Datenträgers eine Rolle spielen. Befindet sich beispielsweise das zu importierende Verzeichnis auf einem herkömmlichen Festplattenlaufwerk mit Magnetscheiben, ist die Lesegeschwindigkeit und damit auch der Datenimport oftmals um ein Vielfaches geringer als der Import von einer *Solid State Disk* (SSD).

Die Anwendung wird in *Kotlin* implementiert.<sup>32</sup> Diese Sprache wurde zur Entwicklung der Datenimport-App gewählt, um die Vorteile von Java zu nutzen und gleichzeitig neue Sprachkonstrukte verwenden zu können. So ist die entwickelte Applikation interoperabel und kann unter mehreren Betriebssystemen ausgeführt werden. Es muss lediglich ein *Java Runtime Environment (JRE)* installiert sein. Auch die Anbindung zum HDFS und zu HBASE kann einfach realisiert werden, da für beide Implementierungen eine Java-Bibliothek bereitsteht. Und letztlich ist es möglich alle benötigten Dateisystemmetadaten auch mit Java und somit mit Kotlin auszulesen.

Zum Bauen der Anwendung wird *Gradle* genutzt. Dies ermöglicht eine einfache Handha-

<sup>32</sup>Siehe auch Kapitel 2.2 zum allgemeinen Entwicklungsvorgehen.

bung von Third-Party-Bibliotheken und deren Versionierung. Darüber hinaus kann mithilfe von Gradle der Quellcode auch ohne Entwicklungsumgebung schnell und einfach gebaut werden. Somit könnte ein forensischer Ermittler die aktuelle Version aus der Versionsverwaltung unter dem Link <https://github.com/jobusam/foam-data-import> herunterladen und mit Gradle bauen.

Das Build-Artefakt selbst ist ein Dateiarchiv (ZIP/TAR). Es kann auf einem Analyserechner entpackt und ausgeführt werden. Hierbei wurde der Datenimport als Konsolenanwendung implementiert. Über mehrere Parameter kann der Import konfiguriert werden. Auf eine grafische Oberfläche wurde bewusst verzichtet. Durch die Ausführung als Konsolenanwendung kann das Programm auch sehr gut in andere Analyse-Skripte eingebettet werden.

Nachfolgende Abbildung 4.12 zeigt hier die Hilfeseite und listet alle konfigurierbaren Parameter auf.<sup>33</sup>

Beschreibung der Parameter:

- Bei dem Datenimport muss mindestens das Verzeichnis (*Input Directory*) angegeben werden, welches importiert werden soll.
- Mit der Option *-o*, *--hdfsBaseDirectory* kann angegeben werden, in welches HDFS-Verzeichnis die Dateien größer 10 MB gespeichert werden sollen.<sup>34</sup>
- Die Option *-x*, *--hbaseSiteXml* gibt den Dateipfad zur Konfiguration von HBASE an. Diese Datei kann von einem existierenden Hadoop-Cluster auf den lokalen Analyserechner kopiert werden und definiert eine Gruppe von Zookeeper-Endpunkten (Hostname inklusive Port), damit über Zookeeper die HBASE-Instanzen ermittelt werden können.<sup>35</sup>
- Die Option *-y*, *--hdfsCoreXml* gibt Analog zur HBASE-Konfiguration eine HDFS-Konfiguration des Cluster an. Auch hier muss wiederum der Endpunkt zum HDFS angegeben werden.<sup>36</sup>
- Mit der Option *-c*, *--caseNumber* kann eine Fallnummer angegeben werden. Damit können mehrere Asservate zu einem bestimmten Fall zugeordnet werden (siehe Kapitel 4.7.3).
- Mit der Option *-d*, *--caseName* kann zusätzlich ein Fallname angegeben werden.
- Die Option *-e*, *--examiner* kann den Namen des forensischen Analysten enthalten. Aktuell muss kein Name angegeben werden. Allerdings wäre dies später für eine automatische Generierung eines Reports zur Beweismittelkette (*Chain of Custody*) sinnvoll.<sup>37</sup>
- Mit der Option *-f*, *--exhibitname* kann zusätzlich ein Name des Asservats angegeben werden.

---

<sup>33</sup>Die Interpretation der Parameter wurde mit der Kotlin-Bibliothek *CLI KT* durchgeführt. Siehe Link: <https://ajalt.github.io/clikt/index.html>. Letzter Zugriff: 24.8.2018.

<sup>34</sup>Hierbei muss der aktuelle Nutzer des Analyserechners auch die Berechtigungen für das Schreiben in das angegebene HDFS-Verzeichnis besitzen.

<sup>35</sup>In Kapitel B.1 im Anhang wird eine minimale Konfigurationsdatei dargestellt.

<sup>36</sup>Siehe Kapitel B.1 im Anhang.

<sup>37</sup>Diese Funktionalität wird im Rahmen der Thesis jedoch nicht implementiert.

```

Usage: forensicdataimport [OPTIONS] INPUTDIRECTORY

Options:
  -v, --verbose           enable verbose mode
  -o, --hdfsBaseDirectory VALUE
                        contains the base directory in HDFS where
                        large files will be stored. The default is
                        /data/
  -x, --hbaseSiteXml VALUE
                        contains file path to hbase-site.xml
                        configuration file. If not given use
                        localhost:2181 for connecting to Zookeeper
  -y, --hdfsCoreXml VALUE
                        contains file path to Hadoop core-site.xml
                        configuration file. If not given use default
                        hdfs uri hdfs://localhost:9000
  -c, --caseNumber INT
  -d, --caseName TEXT
  -e, --examiner TEXT
  -f, --exhibitName TEXT
  -h, --help               Show this message and exit

Arguments:
  INPUTDIRECTORY  contains the path to local source directory that shall be
                  imported into forensic analysis platform

```

Abbildung 4.12: Datenimport-Ausführung in der Konsole

#### 4.7.3 Datenmodell

In Abbildung 4.13 wird das Datenmodell der forensischen Analyseplattform beschrieben. Dieses Modell wird in drei Tabellen in HBASE aufgeteilt. Es enthält neben der Speicherung der Metadaten auch das Datenmodell einer rudimentären Fallverwaltung.

Die Fallverwaltung ist notwendig um den Analysten die Möglichkeit zu geben, mehrere Asservate (z.B. Datenträgerabbilder) in die forensische Analyseplattform zu importieren. Dies ist ein wichtiger Bestandteil um Beziehungen zwischen den Asservaten identifizieren zu können. Ein Fall (*forensicCase*) besteht hierbei aus einer Fallnummer, einem Fallnamen und dem Namen des Auswerters. Die Fallnummer kann beim Datenimport als Parameter angegeben werden. Hierdurch kann der Analyst mehrere Asservate zu einem Fall importieren. Ein Fall kann daher auch mehrere Asservate (*forensicExhibit*) enthalten. Für diese Asservate kann wiederum ein beschreibender Name beim Import angegeben werden. Zusätzlich wird der Zeitpunkt beim Datenimport und das angegebene Basisverzeichnis im HDFS mit abgespeichert. Das Basisverzeichnis wird später bei der Datenverarbeitung ausgewertet, um die Dateien größer 10 MB im HDFS lokalisieren zu können.

Ein Asservat kann wiederum mehrere Daten (*forensicData*) enthalten. Dies sind die einzelnen Dateien und ihre Metadaten. Hierbei hat jeder Eintrag eine eindeutige Id, welche als Zeilenschlüssel in HBASE genutzt wird.<sup>38</sup> Das Attribut *filePath* enthält hierbei den vollständigen Dateipfad inklusive Dateinamen. Im Attribut *fileType* wird gespeichert, ob es sich um eine Datendatei, ein Verzeichnis, einen symbolischen Link und um eine andere spezielle Datei handelt. Abhängig von der Dateigröße wird bei kleinen Dateien der Dateiinhalt direkt in dem Attribut *fileContent* gespeichert. Bei großen Dateien hingegen, wird nur im Attribut *hdfsFilePath* auf den Dateipfad im HDFS referenziert, wo die Datei ge-

---

<sup>38</sup>In Abbildung 4.13 sind diese Zeilenschlüssel dunkelblau hinterlegt.

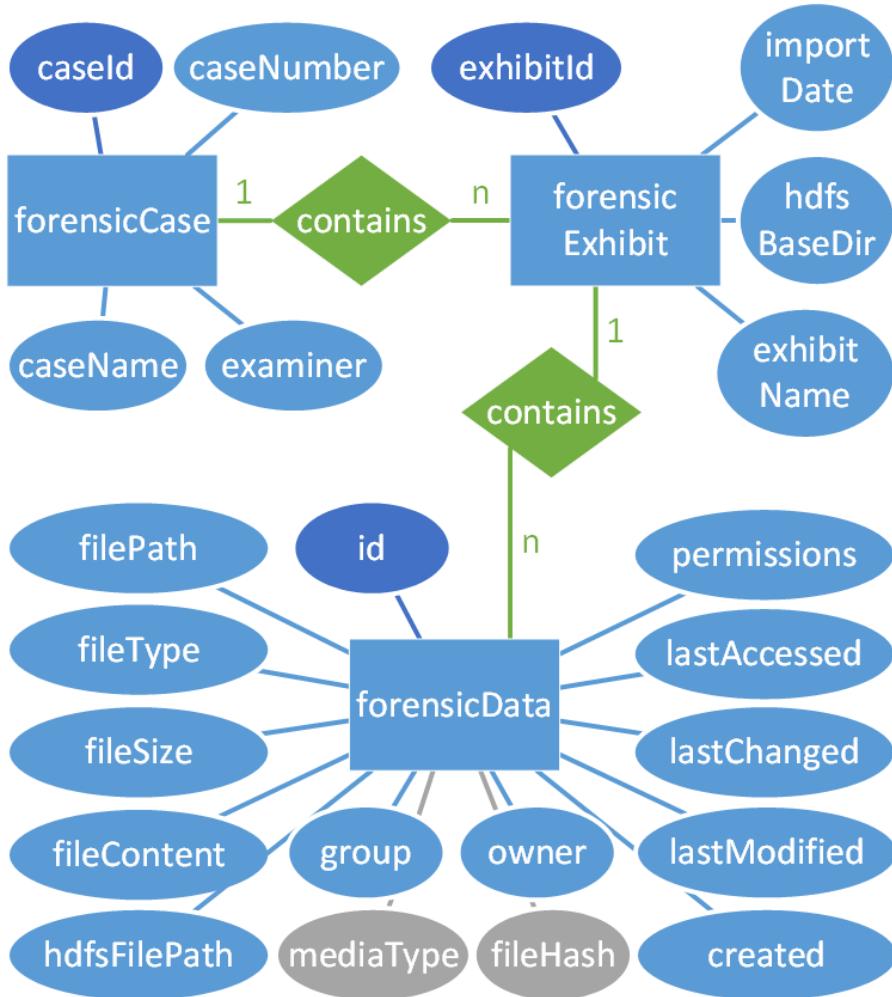


Abbildung 4.13: Datenmodell der forensischen Analyseplattform

speichert wird. Der Dateipfad im HDFS hingegen ist eine Kombination aus dem angegebenen *hdfsBaseDir* und dem Zeilenschlüssel (*id*) des Dateneintrags. Dadurch ist es auch möglich über eine Datendatei im HDFS deren Metadaten in HBASE zu identifizieren. Denn der Zeilenschlüssel in der HBASE-Tabelle *forensicData* ist innerhalb der forensischen Analyseplattform eindeutig.

Des Weiteren werden noch die Zugriffsrechte, die Besitzer, die Gruppe und die Zeitstempel abgespeichert. Je nach spezifischem Dateisystem des Asservats sind allerdings nicht immer alle Metadaten vorhanden.

Bei der anschließenden Datenverarbeitung in Kapitel 5 wird das bestehende Datenmodell um neue Metadaten erweitert.<sup>39</sup>

#### 4.7.4 Datenspezifische Aspekte

Bei der Implementierung des Datenimports müssen auch datenspezifische Aspekte berücksichtigt werden. Wie bereits erwähnt, werden die Daten auf logischer Dateiebene in das Hadoop-System importiert. Hierbei müssen spezielle Dateitypen berücksichtigt werden. Ein

<sup>39</sup>Hierzu gehören die Attribute *mediaType* und *fileHash*, welche in Abbild 4.13 grau hinterlegt sind.

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 Analysestystem 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 Interpretation aber zu schwerwiegenden Fehlern der Analyseergebnisse führen, wenn Inhalte des Analyserechners verarbeitet werden, welche ursprünglich nicht auf dem Asservat vorhanden waren. Daher muss beim Import geprüft werden, ob die Datei einem symbolischen Link entspricht. Ist dies der Fall darf, der symbolische Link nur innerhalb des Asservats interpretiert werden.<sup>40</sup>

Auch beim NTFS-Dateisystem, welches vorzugsweise bei Windows genutzt wird, existieren spezielle Eigenschaften. Bei NTFS ist es möglich an Datendateien und sogar an Verzeichnissen sogenannte *Alternate Data Streams* anzuhängen. Diese können, wie jede andere Datei, beliebige Binärdaten enthalten.<sup>41</sup>

Ein weiterer Punkt sind auch die Zeitstempel. Das System kann letztlich die Zeitstempel zur Erstellung einer Datei (*created*), zur letzten Modifikation einer Datei (*lastModified*), zur letzten Modifikation der Metadaten einer Datei (*lastChanged*) und zum letzten Lesezugriff einer Datei (*lastAccessed*) speichern. Letztlich hängt es aber sehr stark von dem genutzten Dateisystem und auch sogar von dem Betriebssystem ab, welche Zeitstempel überhaupt geschrieben werden. Noch kritischer muss die Korrektheit der Daten geprüft werden. Derzeit liest die Datenimport-Anwendung diese Zeitstempel aus, falls sie vorhanden sind.

Analog zu diesen beschriebenen Fällen gibt es noch weitere dateisystemspezifische Eigenschaften, welche beim Datenimport und zukünftigen Weiterentwicklungen berücksichtigt werden sollten.

#### 4.7.5 Zugriffs und Ausführungsrechte

Ein weiterer Aspekt ist die Beschränkung der Dateizugriffe auf Basis der vorgegebenen Zugriffsrechte. Wie in Abbildung 4.11 in Kapitel 4.7.2 beschrieben wird, muss im ersten Schritt das Datenträgerabbild (beziehungsweise das Asservat) zuerst auf dem Analyserechner gemountet werden. Unter Linux kann dies im Normalfall nur mit erhöhten Administrator-Rechten (root) durchgeführt werden. Der forensische Analyst benötigt also zumindest auf seinem Analyse-Rechner privilegierte Ausführungsrechte.

Hier unterscheidet sich beispielsweise die Referenzanalysesoftware *Autopsy* von dieser forensischen Analyseplattform. Denn bei Autopsy unter Windows wird das Dateisystem auf Anwendungsebene direkt mit der Software analysiert. Das Betriebssystem selbst muss das Dateisystem nicht mounten und der Nutzer benötigt daher auch keine besonderen Systemprivilegien.

Beim Import von Dateien auf einem gemounteten Dateisystem des Datenträgerabbildes sind jedoch die Dateizugriffsrechte weitaus interessanter. Denn das Betriebssystem des Analyse-Rechners berücksichtigt diese Zugriffsrechte. Während diese Problematik bei NTFS-

---

<sup>40</sup>Derzeit werden logische Links in das System importiert. Jedoch wird ihre Referenz aktuell beim Import nicht interpretiert. Dies wäre bei einer Weiterentwicklung des Systems durchaus sinnvoll.

<sup>41</sup>Auch diese Eigenschaften werden im Rahmen dieser Thesis noch nicht beim Datenimport berücksichtigt.

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 Applikation nicht mit Root-Rechten ausgestattet werden, da dies im Fehlerfall zu unvorhergesehenen Rechteausweiterungen führen könnte und ein Sicherheitsrisiko für die Systemintegrität darstellen würde. 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 innerhalb des gemounteten Verzeichnisses unabhängig von deren Besitzer und Zugriffsrechten.

Eigentlich müsste beim Mounten des Dateisystems dem Betriebssystem mitgeteilt werden können, dass die Dateirechte des gemounteten Dateisystems ignoriert werden sollen. Diese Option existiert nicht.<sup>42</sup>

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 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 andere Alternative ist die Nutzung von Posix Capabilities<sup>43</sup>. Dies Variante ist prinzipiell unter CentOS/Fedora möglich. Zum Lesenden Zugriff auf Dateien muss die Posix Capability *CAP\_DAC\_READ\_SEARCH* gesetzt werden.

Mit nachfolgenden Kommando kann diese Capability für das Analyseprogramm gesetzt werden. Damit kann theoretisch auch ein nicht-privilegierter Nutzer lesenden Zugriff auf privilegierte Dateien erhalten.

```
1 setcap CAP_DAC_READ_SEARCH /bin/data.import
```

Listing 4.3: Befehl zum Setzen von Posix Capabilities

Allerdings funktioniert diese Alternative primär bei Binärprogrammen, jedoch nicht bei Shell-Skripten oder Java-Anwendungen.

Eine ähnliche Alternative zu den Posix Capabilities ist das Setzen des SUID-Bits als Unix-Dateirecht für die Programmdatei. Aber auch diese Möglichkeit funktioniert nur bei Binärprogrammen und nicht für interpretierte Skripte oder Java-Anwendungen, die wiederum in der Java Virtual Machine ausgeführt werden.

Zuletzt gibt es noch eine Variante, welche die Problematik mit den Dateirechten lösen kann. Mit dem Projekt *bindfs*<sup>44</sup> können unter Linux Dateisystemverzeichnisse neu gemountet

<sup>42</sup>Zumindest konnte keine funktionierende Variante gefunden werden. Siehe Man-Page des Mount-Befehls (geprüft unter Fedora 28).

<sup>43</sup>Siehe Manpages mit folgendem Befehl: *man 7 capabilities*.

<sup>44</sup>Siehe Link: <https://bindfs.org/>. Letzter Zugriff: 21.7.2018.

werden und ihre Zugriffsrechte verändert werden. Der nachfolgende Befehl mountet das existierende Verzeichnis mit den enthaltenen Dateien in einem neuen Verzeichnis und setzt 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. Das originale Datenträgerabbild wird nicht verändert. Der einzige Nachteil an dieser Lösung ist, dass der Besitzer und die Gruppe jeder einzelnen Datei im neu gemounteten Verzeichnis nun von dem Nutzer des Analysesystems überschrieben wurde. Dies bedeutet, dass die Attribute *Owner* und *Group* im Datenmodell der forensischen Analyseplattform derzeit nicht korrekt sind und daher auch nicht zur Datenverarbeitung genutzt werden können. Dieser Nachteil muss zukünftig behoben werden, damit die forensische Analyseplattform auch die Besitzer und Gruppen einer Datei korrekt auswerten kann. Beispielsweise könnte untersucht werden, ob die Implementierung von BindFS modifiziert werden kann, um den ursprünglichen Nutzer und die Gruppe möglicherweise als erweiterte Dateiattribute zu speichern. Diese erweiterten Dateiattribute könnten dann wieder beim Datenimport ausgelesen werden.

Alternativ könnte nach weiteren Möglichkeiten gesucht werden, wie dem Betriebssystem mitgeteilt werden kann, die Dateirechte für bestimmte gemountete Datenträger bei einem lesenden Zugriff zu ignorieren.

## 4.8 Fazit

Die vierte Variante überzeugt durch eine einfache Lösung zur Speicherung von vielen kleinen und großen Dateien. Darüber hinaus kann mithilfe der HBASE-Datenbank auch eine kleine Fallverwaltung implementiert werden, um mehrere Asservate eines Falls zu importieren.

Die prototypische Implementierung bestätigt die Machbarkeit zur Speicherung von großen semistrukturierten Datensätzen durch eine Kombination der Speicherung im HDFS und HBASE. Aufbauend auf dieser Implementierung und dem dargestellten Datenmodell können im nächsten Schritt der Datenverarbeitung weitere Informationen aus den Daten gewonnen werden.

# 5 Datenverarbeitung

## 5.1 Herkömmliches Analysevorgehen

In Kapitel 4.2 wurde bereits besprochen, wie bei der Open-Source Software *Autopsy* ein Fall angelegt und Datenträgerabbilder hinzugefügt werden können. Schon bei dem Importieren einer Datenquelle können diverse Module zur automatisierten Datenaufbereitung aktiviert werden. Dies entspricht der dritten Phase, dem *Sichten und Aufbereiten* aus dem allgemeinen forensischen Analyseprozess.<sup>1</sup> Diese Module werden nachfolgend beschrieben, um einen Vergleich zur hier entwickelten forensischen Analyseplattform zu ermöglichen.

Abbildung 5.1 zeigt die verfügbaren Module.

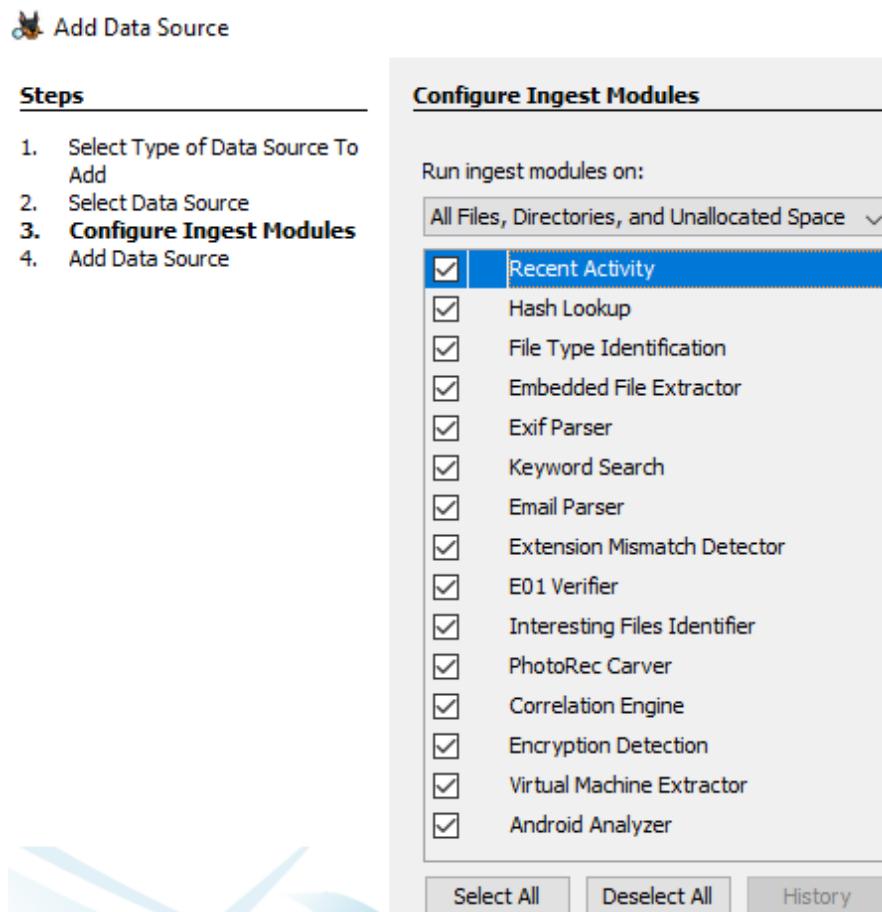


Abbildung 5.1: Module zur automatischen Datenverarbeitung bei Autopsy

<sup>1</sup>Siehe Abbildung 4.1 In Kapitel 4.1.

Nachfolgend werden die Module kurz beschrieben und mit der Funktionalität der hier entwickelten forensischen Analyseplattform verglichen. Zuerst werden die Module aufgelistet, welche ganz oder teilweise auch in der forensischen Analyseplattform implementiert sind.<sup>2</sup>

- Das *File Type Identification Modul* ermöglicht die Ermittlung von Medientypen und Dateiformaten, wie beispielsweise JPEG-Bilder. Autopsy nutzt hier die Open-Source Bibliothek Apache Tika<sup>TM</sup>. Ein gleichwertige Funktionalität wird auch für die forensische Analyseplattform entwickelt.<sup>3</sup>
- Das *Correlation Engine Modul* bietet eine Möglichkeit mehrere Fälle und ihre Datenquellen miteinander zu verknüpfen, um Beziehungen untereinander zu erkennen. Analog hierzu bietet auch die forensische Analyseplattform mit dem Auffinden von Datei-Duplikaten, eine einfache Funktion um Beziehungen zwischen mehreren Fällen und den Asservativen herzustellen.<sup>4</sup>
- Das *Keyword Search Modul* ermöglicht eine Schlüsselwortsuche. Hierzu werden Dateien mithilfe von Apache Solr indiziert. Darüber hinaus kann mithilfe diverser Regulären Ausdrücken nach bestimmten Strukturen, wie beispielsweise E-Mail- oder Web-Adressen gesucht werden. Die hier entwickelte Analyseplattform implementiert auch eine Textsuche auf Basis von Apache Solr. Derzeit ist es allerdings noch nicht möglich ganze Dateiinhalte zu indexieren.<sup>5</sup>

Nachfolgend werden die Module von Autopsy aufgelistet, welche noch nicht in der forensischen Analyseplattform vorhanden sind und in Weiterentwicklungen der Plattform implementiert werden könnten.

- Das *Recent Activity Modul* versucht anhand der Zeitstempel alle interessanten Tätigkeiten der letzten 7 Tage zu ermitteln, wie zum Beispiel besuchte Websites von diversen Browsern.
- Das *Hash Lookup Modul* ermöglicht es die Datei-Hashsummen mit bereits bekannten Hashsummen zu vergleichen. Damit könnten bekannte Betriebssystemdateien schnell gefiltert werden. Analog zu einem Virenschanner könnte dadurch auch bekannte böswillige Malware gefunden werden.
- Das *Embedded File Extractor Modul* ermöglicht das Entpacken von Archivdateien, wie beispielsweise ZIP-Dateien.
- Das *Exif Parser Modul* ermöglicht die Anzeige der sogenannten EXIF-Metadaten aus JPEG-Bilddateien. In der forensischen Analyseplattform könnte diese Funktionalität auch mit Apache Tika implementiert werden.
- Das *Email Parser Modul* extrahiert E-Mails aus diversen E-Mail Archiven, wie beispielsweise dem PST-Dateiformat, welches von Microsoft Outlook genutzt wird.
- Das *Extension Mismatch Detector Modul* ermittelt Dateien, deren Dateiendung nicht zum analysierten Medientyp auf Basis des Dateiinhalts passen. Dies könnte auf versteckte Dateien hinweisen. Ein ähnliche Funktion könnte in der forensischen Analyseplattform über die Volltextsuche implementiert werden, da die Dateiendung und der Medientyp bekannt sind.

---

<sup>2</sup>Siehe Link: <http://sleuthkit.org/autopsy/docs/user-docs/4.3/>. Letzter Zugriff: 09.08.2018.

<sup>3</sup>Siehe Kapitel 5.3.3.

<sup>4</sup>Siehe Kapitel 5.3.2

<sup>5</sup>Siehe Kapitel 5.3.4.

- Das *E01 Verifier Modul* prüft speziell für das forensische Analyseformat *E01* die Dateiprüfsummen mit den gespeicherten Prüfsummen. Damit könnten Änderungen an dem sichergestellten Datenträger ermittelt werden.
- Das *Interesting Files Identifier Modul* identifiziert interessante Dateien, welche vorher durch bestimmten Suchkriterien, wie beispielsweise Dateinamen und Datentypen, definiert wurden.
- Das *PhotoRec Carver Modul* ermöglicht das bereits beschriebene File Carving auf ungenutzten Speicherbereichen. Diese Funktionalität wird nicht von der forensischen Analyseplattform unterstützt, da ungenutzte Speicherbereiche aus den originalen Datenträgerabbildern nicht in das Analysesystem importiert werden.
- Das *Encryption Detection Modul* ermöglicht das Auffinden von verschlüsselten Containern. Hierbei wird nach Dateien gesucht, deren Inhalt eine hohe Entropie aufweisen.<sup>6</sup>
- Das *Virtual Machine Extractor Modul* kann Dateien aus Systemabbildern von virtuellen Maschinen extrahieren.
- Das *Android Analyzer Modul* ermöglicht das Extrahieren von bestimmten Informationen aus spezifischen Dateien des Android-Betriebssystems. So können beispielsweise Kontaktdaten und Nachrichten aus für Android spezifischen SQLite-Datenbanken ermittelt werden.

Nachdem die benötigten Module in Autopsy aktiviert und nach fallspezifischen Eigenschaften konfiguriert wurden, erfolgt die Datenaufbereitung im Hintergrund. Parallel hierzu kann der Analyst bereits die Rohdaten analysieren.<sup>7</sup>

## 5.2 Umsetzung in der forensischen Analyseplattform

### 5.2.1 Verarbeitung mit Apache Spark

Zur Datenverarbeitung in der forensischen Analyseplattform wird Apache Spark<sup>TM</sup> genutzt. Der physikalische Aufbau wurde bereits im Grundlagenkapitel 3.4 behandelt. In diesem Kapitel sollen primär die Algorithmen und die Verarbeitung der Daten aus logischer Sicht betrachtet.

Bei Apache Spark (Version 2.3.0) gibt es diverse Programmierschnittstellen (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. Listing 5.1 zeigt ein einfache Nutzung dieser RDDs, um alle Wörter aus einer Beispieldatei `history.txt` und deren Vorkommen aufzulisten. Die Implementierung kann abhängig von der Dateigröße und der Konfiguration von Spark verteilt auf mehreren Knoten ausgeführt werden, um die Datenverarbeitung zu beschleunigen. Die Daten werden aus dem HDFS gelesen und dort gespeichert.

---

<sup>6</sup>Die Entropie in der Informationstechnik ist ein Maß zur Bestimmung des mittleren Informationsgehalts pro Zeichen. Bei verschlüsselten Containern ist diese Entropie sehr hoch verglichen zu unverschlüsselten Dateien. TODO: Krypto-Buch als Quelle nutzen...

<sup>7</sup>Siehe Kapitel 6 zur Visualisierung von Daten.

```

1 //Datei aus dem HDFS lesen
2 JavaRDD<String> textFile = sc.textFile("hdfs://data/history.txt");
3 JavaPairRDD<String, Integer> counts = textFile
4     // Text separieren
5     .flatMap(s -> Arrays.asList(s.split(" ")).iterator())
6     // Wortvorkommen ermitteln
7     .mapToPair(word -> new Tuple2<>(word, 1))
8     .reduceByKey((a, b) -> a + b);
9 // Ergebnis in HDFS speichern
10 counts.saveAsTextFile("hdfs://...");
```

Listing 5.1: Beispielimplementierung eines Spark RDDs

Bei der Weiterentwicklung von Spark sind in neueren Versionen sogenannte *DataFrames* und *DataSets* hinzugekommen. Diese Datenstrukturen beschreiben eine Schnittstelle, welche der Sichtweise von Tabellen ähnelt. Die Implementierung dieser Typen nutzt wiederum die bereits beschriebenen RDDs.

Welche Datenstrukturen für die Anwendungsfälle von der forensischen Analyseplattform genutzt werden sollten, hängt primär von der Art der Daten ab. *DataFrames* und *DataSets* sind optimiert für strukturierte und semi-strukturierte Daten. Diese Daten lassen sich beispielsweise in Tabellenstrukturen einlesen und verarbeiten. Es gibt komplexe 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 Multimediateien und beliebigen Dateien im Allgemeinen.[13, 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 Dateframes auf diese Daten zuzugreifen. Letztlich kommt es aber auf die Anbindung zwischen Apache Spark und Apache HBASE an. Hierbei gibt es primär zwei unterschiedliche Connectoren<sup>8</sup>. Der *Hortonworks SHC Connector* ermöglicht die Interaktion mit Daten in HBASE und nutzt dafür die DataFrame/DataSet Datenstrukturen.<sup>9</sup>

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.<sup>10</sup> Daher wird die Datenverarbeitung mithilfe von Spark RDDs implementiert. Auch bei dem Auslesen von größeren Dateien aus dem HDFS werden Spark RDDs verwendet, da Spark selbst passende Methoden auf Basis von RDDs anbietet.

---

<sup>8</sup>Bei 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.

<sup>9</sup>Siehe <https://github.com/hortonworks-spark/shc>, Stand: 15.6.2018.

<sup>10</sup>Siehe <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.

## 5.3 Anwendungsfälle der Datenverarbeitung

### 5.3.1 Hashsummen ermitteln

Eine grundlegende Funktionalität einer forensischen Analysesoftware ist die Berechnung von Hashsummen, um die eben die Integrität fallrelevanter Dateien für zukünftige Analysen prüfen zu können. Darüber hinaus können auf Basis der Hashsummen sehr einfach Duplikate erkannt werden (siehe Kapitel 5.3.2).

In der derzeitigen Implementierung wird der SHA-512 Algorithmus genutzt, um eindeutige kryptografisch sichere Hashsummen zu berechnen. Bei Autopsy hingegen werden MD5-Hashsummen berechnet. Der MD5-Algorithmus gilt mittlerweile jedoch als kryptografisch unsicher, da Hash-Kollisionen in wenigen Stunden berechnet werden können.[17, S. 240-243]

Abbildung 5.2 verdeutlicht den Datenfluss bei der Ermittlung von Hashsummen.

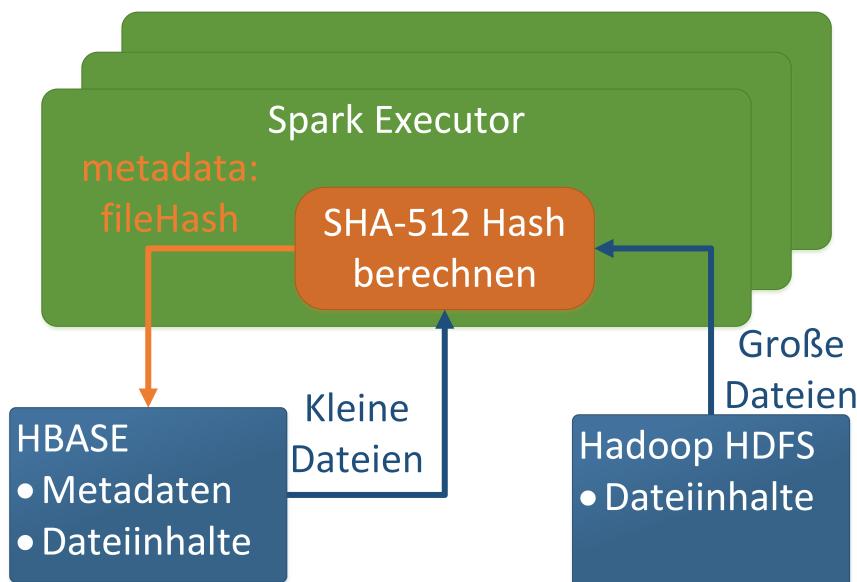


Abbildung 5.2: Datenfluss bei der Berechnung von Hashsummen

Die Datenverarbeitung von kleinen Dateien aus HBASE und großen Dateien aus dem HDFS wird unabhängig voneinander in eigenen Spark-RDDs ausgeführt. Aufgrund der unterschiedlichen Datenquellen existieren auch zwei unterschiedliche Datentypen zum Auslesen der Dateiinhalte. Daher wird die Prozessierung in getrennten Tasks innerhalb der Spark Executoren ausgeführt. Um dann später die Hashsumme einer großen Datei aus dem HDFS in dem dazugehörigen Eintrag in HBASE abzuspeichern, ist es notwendig den Zeilenschlüssel des Eintrags in HBASE als Dateiname im HDFS zu verwenden. Dadurch können die Dateien aus dem HDFS einem Eintrag in HBASE zugeordnet werden.

Der ermittelte Hashwert wird darauf in der Spalte `metadata:fileHash` in HBASE gespeichert.<sup>11</sup>

<sup>11</sup>Siehe auch HBASE-Datenmodell in Abbildung 4.13.

### 5.3.2 Duplikate erkennen

Aufbauend auf der Berechnung von Hashsummen können nun auch Datei-Duplikate ermittelt werden. Dadurch ist es möglich Beziehungen zwischen mehreren Asservaten herzustellen, wenn dort gleiche Dateien gefunden werden.

Derzeit werden diese Ergebnisse aber nur als Text im HDFS abgespeichert. In einer Weiterentwicklung könnten diese Informationen aber sehr gut in einer Graphendatenbank, wie beispielsweise *Neo4j* abgespeichert werden. Hierdurch wäre visuell sehr schnell sichtbar, welche Beziehungen zwischen den Asservaten bestehen.

Allerdings wäre es in diesem Kontext dann auch sinnvoll, zwischen technischen Betriebssystemdateien und fachlichen Nutzerdateien zu unterscheiden. Denn wenn zwei Asservate das gleiche Betriebssystem enthalten, existieren auch zugleich tausende Datei-Duplikate. Interessant ist diese Erkennung von Duplikaten aber gerade für fachliche Nutzerdaten, wie beispielsweise Bilder, Dokumente und Videos. Daran könnte später ermittelt werden, wie beispielsweise urheberrechtlich geschütztes Material über mehrere Systeme verbreitet wurde.

Ein anderer Aspekt, welcher in einer Weiterentwicklung der Plattform implementiert werden könnte, ist der Vergleich mit bekannten Hashsummen von existierenden Datensätzen. Dies wird bei Autopsy verwendet, um gegebenenfalls Malware zu erkennen oder automatisiert nach spezifischen Dateien zu suchen. Ein Beispiel hierzu ist die sogenannte *National Software Reference Library*, welche Hashsummen von bekannten Betriebssystem- und Anwendungsdateien enthält. Diese haben oftmals keinen forensischen Analysewert und könnten dann aus der Datenverarbeitung gefiltert werden.<sup>12</sup>[14, S. 36]

### 5.3.3 Medientypen erkennen

Das Erkennen von Medientypen ist ein elementarer Bestandteil bei der Datenanalyse im Allgemeinen. Autopsy nutzt hierbei die Open-Source Bibliothek Apache Tika<sup>TM</sup>. Dieses Projekt wird auch in der forensischen Analyseplattform genutzt und ermittelt den Dateitype anhand des Dateiinhalts und der gegebenen Dateiendung.

Abbildung 5.3 zeigt den Datenfluss bei der Ermittlung der Medientypen.

Prinzipiell kann Apache Tika den Medientyp auch nur anhand der Dateiendung oder anhand des Dateiinhalts erkennen. Wird allerdings nur der die Dateiendung angegeben, so könnten beispielsweise Bilddateien mit gefälschten Dateiendung nicht korrekt erkannt werden. Wird hingegen nur der Dateinhalt angegeben, so prüft Tika formatspezifische Bytesequenzen, welche oftmals am Dateianfang stehen. Bei Textdateien hingegen, kann es nur identifizieren, dass es sich um eine Textdatei handeln könnte. Gerade bei Textdateien macht es Sinn, die Dateiendung in die Analyse mit aufzunehmen, um ein genaueres Ergebnis zu erhalten. Dadurch kann Tika beispielsweise ermitteln, ob es sich um eine CSV-Datei oder eine HTML-Datei handelt. Allerdings sollten diese Ergebnisse kritisch hinterfragt werden. Wie bereits erwähnt, könnte eine gefälschte Dateiendung mit Textdateien zu einem falschen Ergebnis führen.

Das Ermitteln der Medientypen macht gerade bei der Datenvisualisierung Sinn, da damit

---

<sup>12</sup>Siehe Link: <https://www.nist.gov/itl/ssd/software-quality-group/nsrl-download/current-rds-hash-sets>. Letzter Zugriff: 9.9.2018.

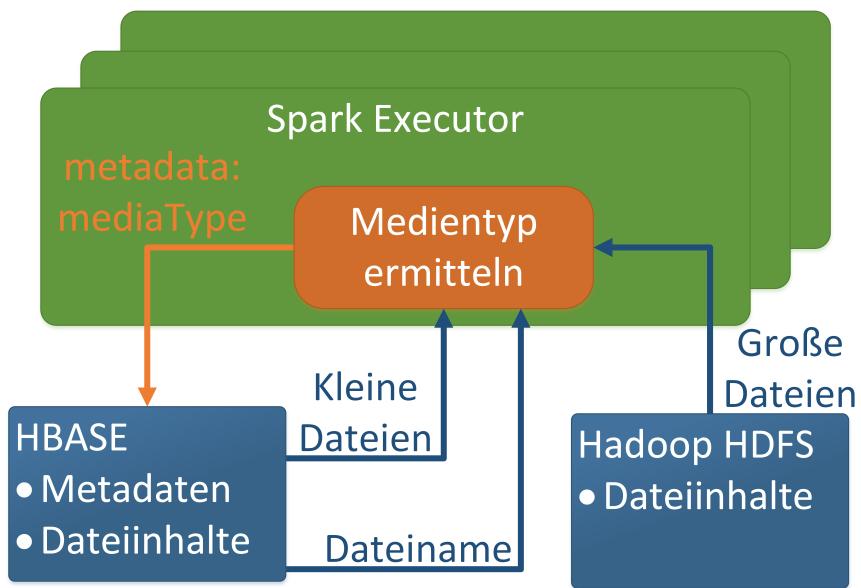


Abbildung 5.3: Datenfluss bei der Ermittlung von Medientypen

schnell nach allen vorhanden Bildern oder Videos gesucht werden kann.<sup>13</sup>

#### 5.3.4 Volltextsuche und Datenindexierung

Um auf die Daten in der forensischen Analyseplattform einfach zugreifen zu können, wäre eine Volltextsuche sinnvoll. Diese sollte performant arbeiten, damit Ergebnisse zu Suchanfragen von beliebigen Schlüsselwörtern in Realzeit angezeigt werden können. Um dies zu ermöglichen, können die Dateiinhalte und Metadaten indexiert werden.

Es gibt hierzu zwei mögliche Open-Source Projekte, die eine Datenindexierung anbieten. Dies sind *Apache Solr™* und *Elasticsearch™*. Beide bieten eine Indexierung auf Basis von Apache Lucene an. Apache Solr wurde bereits im Grundlagenkapitel 3.7 beschrieben. Beide Projekte bieten einen ähnlichen Funktionsumfang. Im Rahmen dieser Thesis wird jedoch Apache Solr aus nachfolgenden Gründen genutzt.

Apache Solr bietet einen Cloud-Modus an und nutzt Apache Zookeeper, um die einzelnen Instanzen zu koordinieren. Für Apache Solr existiert bereits eine Integration in das Hadoop-Framework und die hier verwendete *Hortonworks Data Platform*.<sup>14</sup>

Elasticsearch bietet auch einen Cloud-Modus und ist horizontal skalierbar. Allerdings wird ein eigener Mechanismus dafür genutzt und es existiert für die *Hortonworks Data Platform* kein Installationspaket zur Nutzung von Elasticsearch im Hadoop-Umfeld. Dies bedeutet, dass Elasticsearch eigenständig im Computer-Cluster ausgerollt und überwacht werden muss.

Ein weiterer Punkt ist die Absicherung der Cluster mithilfe von Kerberos. Prinzipiell können beider Projekte mit Kerberos abgesichert werden. Allerdings ist die Funktionalität bei Elasticsearch nur in einem kostenpflichtigen Zusatzpaket der Firma *Elastic*<sup>15</sup> enthalten. Zuletzt unterscheiden sich beide Projekte auch in der Art und Weise, wie die Daten ver-

<sup>13</sup>Siehe auch Kapitel 6.

<sup>14</sup>Siehe Link: [https://docs.hortonworks.com/HDPDocuments/HDPS-3.0.0/bk\\_solr-search-installation/content/ch\\_hdp-search.html](https://docs.hortonworks.com/HDPDocuments/HDPS-3.0.0/bk_solr-search-installation/content/ch_hdp-search.html). Letzter Stand: 9.9.2018.

<sup>15</sup>Siehe Link: <https://www.elastic.co/>. Letzter Stand: 9.9.2018.

teilt indexiert werden können. Bei Elasticsearch gibt es eine Bibliothek, welche die bereits bekannten *Apache Spark Resilient Distributed Datasets* (RDDs) direkt in Elasticsearch indexieren kann.<sup>16</sup> Diese Möglichkeit ist prinzipiell sehr gut, um extrahierte Daten direkt indexieren zu können.

Apache Solr hingegen bietet auch eine Möglichkeit mit Apache Spark Daten in die Solr-Cloud zu indexieren.<sup>17</sup> Allerdings ist die Schnittstelle etwas umständlicher nutzbar, da keine RDDs sondern die bereits beschriebenen *DateFrames* genutzt werden müssen.

Viel interessanter ist jedoch das Open-Source Projekt *Lily HBase Indexer*.<sup>18</sup> Damit können Daten direkt aus HBASE in den Solr-Cluster indexiert werden. Im Rahmen der Thesis wird letztes Projekt genutzt, um die Metadaten aus HBASE in Solr zu indexieren.

Ein weitere Anwendungsfall ist die Indizierung von Texten und Wörtern, welche aus den einzelnen Dateien extrahiert wurden.

Der Grund für eine Indizierung dieser Inhalte ist eine schnellere Suche nach beliebigen Wörtern, als bei der reinen Suche in HBASE. Zur Indizierung existieren zwei bekannte Projekt. Diese sind einerseits *Apache Solr* und andererseits *Elasticsearch*. Bei bauen wiederum auf das *Apache Lucene*-Projekt auf. Apache Solr ist ein Open-Source Projekt unter dem Dach der Apache Foundation. Wohingegen Elasticsearch auch als Open-Source Projekt entwickelt wird, jedoch primär von der kommerziellen Firma *Elastic???* verwaltet wird. Diese bietet gerade Zusatzpakete und Support gegen Bezahlung an. Siehe auch [8, S.7]

### Apache Solr

Spark-Connector von Databricks vorhanden. Der Connector selbst bietet aber vorzugsweise lesenden Zugriff.

Abbildung 5.4 stellt die physikalische Aufteilung mit dem hbase-indexer projekt

## 5.4 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.

### 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 fodere alle Daten an, um sie später im Spark-Executor auszuführen, oder ich versuche schon beim

<sup>16</sup>Siehe Link: <https://www.elastic.co/guide/en/elasticsearch/hadoop/current/spark.html>. Letzter Stand: 9.9.2018.

<sup>17</sup>Siehe Link: <https://github.com/lucidworks/spark-solr>. Letzter Stand: 9.9.2018

<sup>18</sup>Siehe Link: <https://github.com/NGDATA/hbase-indexer>. Letzter Stand: 9.9.2018.

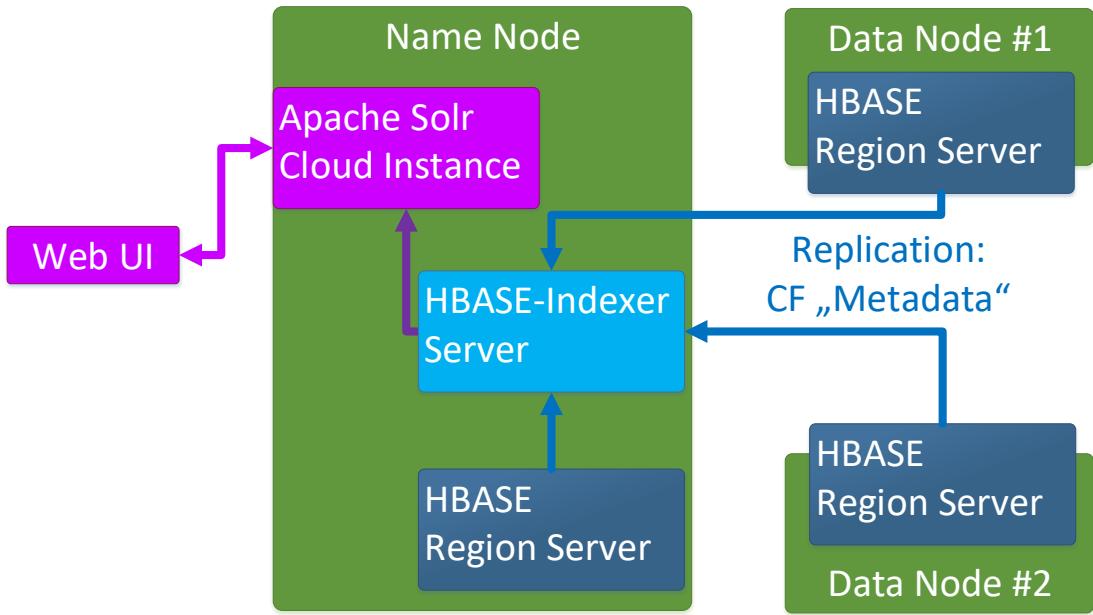


Abbildung 5.4: Indexierung von Daten aus HBASE in Solr

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 5.2). 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 3.4 (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 Executoren 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 Executoren im Arbeitsspeicher geladen wurden, nun jetzt an den Spark-Driver geschickt werden. Dieser sammelt sozusagen die Ergebnisse der Executoren 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 HbaseReader hbr = new HbaseReader(jsc, hbaseConfigFile);
2 JavaRDD<Metadata> forensicMetadata = hbr.getForensicMetadata();
3
4 // use collect() method
5 forensicMetadata.collect().stream()
6   .forEach(m -> LOGGER.info("Entry = .", m));
7
8 // use take(int amount) method
9 forensicMetadata.take(10).stream()
10  .forEach(m -> LOGGER.info("Entry = .", m));
11

```

Listing 5.2: Spark Java RDD collect()-Methode

## 5.5 Leistungsanalyse

TODO: Geschwindigkeit und Ressourcenverbrauch analysieren. Hierzu gehört auch die Performanz beim Datenimport! Maximal 3 Seiten.

# 6 Datenvisualisierung

TODO: 2-3 Seiten über mögliche Visualisierungen schreiben. Hauptsächlich theoretische Aspekte. Kurze Erklärung wie das Projekt *Banana for Solr* genutzt wurde um eine einfache UI bereitzustellen.<sup>1</sup>

Vergleich zu Autopsy

Duplicate Flag in DB schreiben, um über Banana UI alle Duplikate finden zu können... Möglichkeiten und Ideen zur Datenvisualisierung:

- 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/><sup>2</sup>
- Neo4j
- Open Source Community Variante Helical Insight
- Apache Superset für Visualisierung (siehe Ambari Cluster Services)
- Apache Grafana?
- GoJs incremental tree?

---

<sup>1</sup>Siehe Link: <https://github.com/lucidworks/banana>. Letzter Zugriff: 23.8.2018.

<sup>2</sup>Siehe Links: <https://bl.ocks.org/mbostock/4063550>, <https://bl.ocks.org/mbostock/5944371>, <https://bl.ocks.org/mbostock/1046712>, <https://bl.ocks.org/mbostock/4063269> und <http://xliberation.com/googlecharts/d3concept.html>. Letzter Zugriff: 25.7.2018

# 7 Forensische Anforderungen

## 7.1 Beweismittelkette

TODO: 1 Seite über die Beweismittelkette (Chain of Custody) im Rahmen des forensischen Analyseprozesses schreiben. Siehe auch Kapitel 4.1.

## 7.2 Plattform absichern

TODO: 2 Seiten über die Absicherung von dem Apache Hadoop Framework schreiben. Dieses Thema wird nur in der Theorie behandelt.  
Primär soll das Konzept von Kerberos beschrieben werden.

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 wird<sup>1</sup>. 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.

### 7.2.1 Authentifizierung

Standardmäßig wird Hadoop in Kombination mit Kerberos verwendet, um eine allgemeinen Zugriffsschutz zu ermöglichen.[9]  
Eine Alternative könnte hier auch Cloudera Sentry, Apache Ranger, Apache Atlas oder Apache Knox<sup>2</sup> sein.

---

<sup>1</sup>Vgl. <https://www.infoq.com/articles/HadoopSecurityModel>.

<sup>2</sup><https://knox.apache.org/>

### 7.2.2 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.[9]

Darüber hinaus ist die Transportverschlüsselung auch möglich. So müssen die einzelnen Services wie Webzugriffe mit TLS verschlüsselt werden.<sup>3</sup>

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

---

<sup>3</sup>Weiter Infos unter: <https://www.infoq.com/articles/HadoopSecurityModel> und <https://community.hortonworks.com/articles/102957/hadoop-security-concepts.html>.

# 8 Diskussion der Ergebnisse

TODO: Ergebnisse kritisch betrachten und entsprechende Nachteile aufzeigen.

- Datenimport dauert lange. Währenddessen können noch keine Daten analysiert werden. Ist Data Streaming eine Alternative?
- Probleme mit den Zugriffsrechten beim Mounten der Datenträger
- Arbeite das Hadoop-Framework überhaupt effizient?

# 9 Zusammenfassung

TODO: Zusammenfassung schreiben

# 10 Ausblick

TODO: Ausblick schreiben.

- Siehe Aspekte bei der Diskussion der Ergebnisse, die eventuell verbessert werden können.
- Hbase Indexer zu Solr überarbeiten.
- Datenvisualisierung
- Hadoop Absichern
- Reportgenerierung
- Beweismittelkette erstellen
- Forensisch korrektes Löschen von Daten im Hadoop-Umfeld
- Extraktion von IP-Adressen, Web-Adressen, E-Mail-Adressen oder Positionsdaten.

# Literatur

- [1] Brian Carrier. *File System Forensic Analysis*. 1. Auflage. Addison Wesley, 2005.
- [2] Eoghan Casey. *Handbook of Digital Forensics and Investigation*. 1. Auflage. Academic Press, 2009.
- [3] Steffen Hartmann. *Solr-Suchserver in die Cloud skalieren*. entwickler.de. 17. Juni 2013. URL: <https://entwickler.de/online/solr-suchserver-in-die-cloud-skalieren-136486.html> (besucht am 02.08.2018).
- [4] Jonas Freiknecht. *Big Data in der Praxis*. 1. Auflage. Hanser Verlag, 2014.
- [5] Saurav Haloi. *Apache zookeeper essentials: a fast-paced guide to using Apache ZooKeeper to coordinate services in distributed systems*. 1. Auflage. Packt Publishing, 2015.
- [6] Holden Karau u. a. *Learning Spark: Lightning-Fast Data Analysis*. 1. Auflage. O'Reilly, 2015.
- [7] Philip Polstra. *Linux Forensics*. 1. Auflage. Pentester Academy, 2015.
- [8] Dikshant Shahi. *Apache Solr: A Practical Approach to Enterprise Search*. 1. Auflage. Apress, 2015.
- [9] Ben Spivey und Joey Echeverria. *Hadoop Security: Protecting Your Big Data Platform*. 1. Auflage. O'Reilly, 2015.
- [10] Tom White. *Hadoop: The Definitive Guide*. 4. Auflage. O'Reilly, 2015.
- [11] Sam R. Alapati. *Expert Hadoop Administration*. 1. Auflage. Addison Wesley, 2016.
- [12] Benoy Antony u. a. *Professional Hadoop*. 1. Auflage. Wrox, a Wiley brand, 2016.
- [13] Krishna Sankar. *Fast Data Processing with Spark 2*. 3. Auflage. Packt Publishing, 2016.
- [14] André Årnes. *Digital Forensics*. 1. Auflage. Wiley, 2017.
- [15] 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).
- [16] 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).
- [17] Michael Kofler u. a. *Hacking and Security - Das umfassende Handbuch*. 1. Auflage. Rheinwerk Verlag, 2018.
- [18] o. V. *Apache Solr Reference Guide*. Version 7.4. Apache Software Foundation. 19. Juni 2018. URL: <http://www-eu.apache.org/dist/lucene/solr/ref-guide/apache-solr-ref-guide-7.4.pdf> (besucht am 03.08.2018).

# Abbildungsverzeichnis

1.1	Analysevorgehen	3
2.1	Arbeitspakete der Masterthesis	6
2.2	Projektplan Teil A - Einarbeitung und Rohdatenspeicherung (siehe Kapitel A.2)	7
2.3	Projektplan Teil B - Datenanalyse (siehe Kapitel A.2)	8
2.4	Projektplan Teil C - Querschnittliche Aspekte und Visualisierung (siehe Kapitel A.2)	9
2.5	Komponenten der Entwicklungsumgebung	10
3.1	Apache Hadoop Ökosystem (Vgl. [4],[11]. Siehe Kapitel A.2)	14
3.2	HDFS - Datenspeicherung im Verbund (Vgl. [16],[11])	16
3.3	Ressourcenverteilung mit YARN (Vgl. [15],[11])	18
3.4	Spark Datenverarbeitung im Cluster	21
3.5	Schema-Beispiel einer HBASE Tabelle nach [4]	23
3.6	HBASE Datenspeicherung im Cluster	24
3.7	Gespeicherte Informationen von HBASE in ZooKeeper	26
3.8	Solr Cloud-Mode im Cluster	28
3.9	Solr-Suchanfrage via <i>curl</i>	29
4.1	Forensischer Analyseprozess für digitale Beweismittel (Vgl. [14, S.16])	30
4.2	Erstellung eines neuen Falls mit Autopsy	35
4.3	Aufteilung der Daten des Datenträgerabildes im Hadoop-Cluster	37
4.4	HDFS - Dateieigenschaften	39
4.5	Datenspeicherung mit erweiterten Dateiattributen im HDFS	41
4.6	Sequence File Format (Vgl. [10, S. 134])	42
4.7	Kumulierte Häufigkeit nach Dateigröße	45
4.8	Relativierte kumulierte Häufigkeit nach Dateigröße	46
4.9	Kumulierte Kategoriegröße nach Dateigröße	47
4.10	Relativierte kumulierte Kategoriegröße nach Dateigröße	48
4.11	Datenimport in HBASE und HDFS	49
4.12	Datenimport-Ausführung in der Konsole	51
4.13	Datenmodell der forensischen Analyseplattform	52
5.1	Module zur automatischen Datenverarbeitung bei Autopsy	56
5.2	Datenfluss bei der Berechnung von Hashsummen	60
5.3	Datenfluss bei der Ermittlung von Medientypen	62
5.4	Indexierung von Daten aus HBASE in Solr	63

# Tabellenverzeichnis

# Listings

4.1	Befehl zum Speichern einer Datei im HDFS . . . . .	40
4.2	Befehl zum Hinzufügen und Auslesen von Metadaten . . . . .	40
4.3	Befehl zum Setzen von Posix Capabilities . . . . .	54
4.4	Nutzung von Bindfs zum Ändern von Dateirechten . . . . .	55
5.1	Beispielimplementierung eines Spark RDDs . . . . .	59
5.2	Spark Java RDD collect()-Methode . . . . .	64
B.1	Minimale Konfiguration der Datei <i>hdfs-core.xml</i> . . . . .	78
B.2	Minimale Konfiguration der Datei <i>hbase-site.xml</i> . . . . .	78

# Abkürzungsverzeichnis

**GDAL** Geospatial Data Abstraction Library

**HDFS** Hadoop Distributed File System

**HDP** Hortonworks Data Platform

**HTTP** Hypertext Transfer Protocol

**JAR** Java Archive

**JRE** Java Runtime Environment

**JVM** Java Virtual Machine

**NFS** Network File System

**NTFS** New Technology File System

**OCR** Optical Character Recognition

**REST** Representational State Transfer

**YARN** Yet Another Resource Negotiator

# A Allgemeines

## 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.<sup>1</sup>

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

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.<sup>3</sup> 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 aktiv<sup>4</sup>. 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.<sup>5</sup> Mit dem Werkzeug können Hashsummen berechnet oder auch Multimediateile 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

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

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

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

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

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

Analysen finden aber immer nur auf einem konkreten Analysecomputer statt. Ein paralleler 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](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](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.<sup>6</sup>

- Apache Ambari<sup>TM</sup> Logo von <https://ambari.apache.org/>, Stand 21.3.2018.
- Apache Hadoop<sup>®</sup> Logo von <https://hadoop.apache.org/>, Stand 21.3.2018.
- Apache Spark<sup>TM</sup> Logo von <https://spark.apache.org/>, Stand 21.3.2018.
- Apache HBASE<sup>®</sup> Logo von <https://hbase.apache.org/>, Stand 21.3.2018.
- Apache Hive<sup>TM</sup> Logo von <https://hive.apache.org/>, Stand 21.3.2018.
- Apache Zookeeper<sup>TM</sup> Logo von <https://zookeeper.apache.org/>, Stand 21.3.2018.

---

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

# B Datenimport

## B.1 Konfigurationsdateien

Nachfolgende Listings zeigen die minimale Konfigurationsdateien, welche beim Datenimport angegeben werden können. Diese werden genutzt um die Verbindung zum HDFS-Dateisystem und zur HBASE-Datenbank zu definieren.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
3 <!--
4   Following configuration describes how to connect to HDFS! -->
5 <configuration>
6   <property>
7     <name>fs.defaultFS</name>
8     <value>hdfs://localhost:9000</value>
9   </property>
10 </configuration>
```

Listing B.1: Minimale Konfiguration der Datei *hdfs-core.xml*

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
3 <!--
4   Following configuration describes how to connect to HBASE! -->
5 <configuration>
6   <property>
7     <name>hbase.zookeeper.quorum</name>
8     <value>localhost</value>
9   </property>
10  <property>
11    <name>hbase.zookeeper.property.clientPort</name>
12    <value>2181</value>
13  </property>
14 </configuration>
```

Listing B.2: Minimale Konfiguration der Datei *hbase-site.xml*