Hochschule ...

Studiengang ...

Masterthesis

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

Zur Erlangung des akademischen Grades
Master of Science

vorgelegt im Sommersemester 2018

von
Johannes Busam

Erstbetreuung: ...

Zweitbetreuung: ...

Inhaltsverzeichnis

1 Einleitung

1.1 Allgemeines

1.2 Problemstellung

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

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

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

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

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

und Informationen auch in anderen spezialisierten Analysetools weiterzuverarbeiten.

1.3 Zielsetzung

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

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

Die Basis bildet das verteilte Dateisystem *HDFS*[1], welches die Daten redundant auf allen Knoten des Computer-Clusters speichert. Der Zugriff auf die Daten kann über unterschiedliche Komponenten erfolgen.
So findet beispielsweise meist Apache Spark™ bei der Prozessierung und Analyse der Daten Anwendung. Auch in der Masterthesis soll Apache Spark für anfallende Analysezwecke verwendet werden. Hierbei übernimmt der Ressourcenmanager *YARN*[2] die Bereitstellung und Verteilung von verfügbarer Rechenleistung.

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

Im Rahmen dieser Thesis soll die reine Datenanalyse vorerst auf grundlegende Operationen basieren. Unter anderem sollen beispielsweise folgende Informationen zu einzelnen Dateien ermittelt werden: Name, vollständiger Pfad, Hashsumme, Dateityp, Größe, Zeitpunkt der letzten Änderung und Erstellung.
Es soll auch eine Volltextsuche auf den Daten möglich sein. Darauf aufbauend soll der Nutzer beispielsweise gleiche Dateien und Verbindungen zwischen den einzelnen Beweismitteln erkennen können.
Optional könnte die Analyseplattform gezielt nach IP-Adressen, URLs, E-Mail-Adressen oder Positionsdaten[3] suchen. Diese Operationen sollen erst implementiert werden, wenn im Rahmen der Thesis noch weitere Bearbeitungszeit vorhanden ist. Sie sind vorerst nicht

[1] HDFS ist die Abkürzung für *Hadoop Distributed File System*.
[2] YARN ist die Abkürzung für *Yet Another Resource Negotiator*.
[3] Beispielsweise könnten Geopositionen oder Ortsnamen aus Dateien extrahiert werden. Diese Daten könnten dann mit ihrem geografischen Bezug auf einer Karte dargestellt werden.

Gegenstand der Thesis.

Die Resultate durchgeführter Datenanalysen sollen dem Nutzer bereitgestellt werden. Hierzu soll eine prototypische Implementierung entwickelt werden, deren grafische Oberfläche die fachlichen Aspekte der forensischen Analyse widerspiegelt. Der Forensiker soll Analyseaufgaben konfigurieren und starten können. Nach der Prozessierung soll er die Resultate der Analysen direkt einsehen können.
Allerdings ist es nicht das Ziel dieser Thesis, detaillierte Konfigurationsmöglichkeiten und unterschiedlichste Visualisierungen zu implementieren. Dies würde den Rahmen der Arbeit übersteigen. Der Fokus dieser Fachanwendung liegt bei der schlichten Anzeige der Analyseergebnisse. In diesem Kontext soll auch geprüft werden, ob existierende Programme zur Datenvisualisierung im Hadoop-Umfeld wiederverwendet werden können.

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

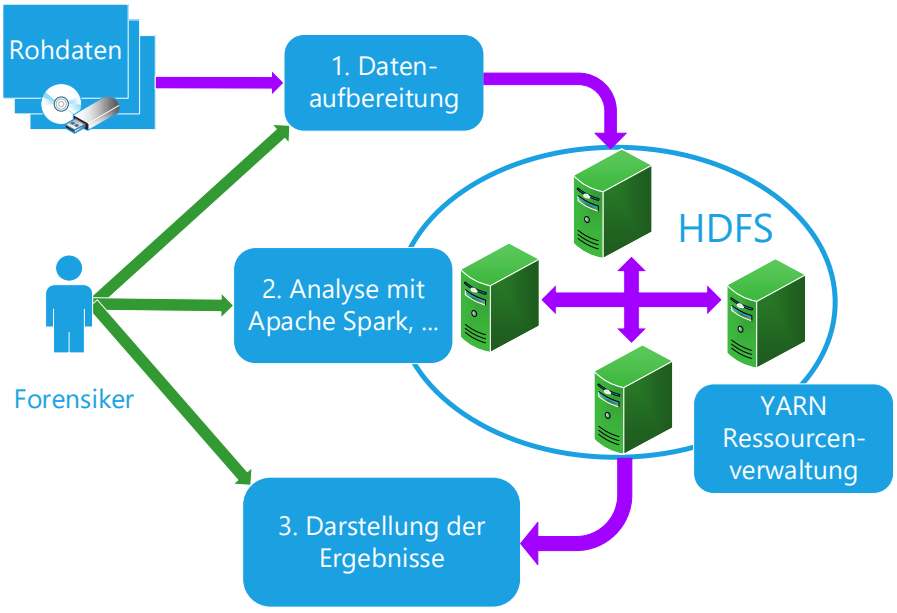

Abbildung 1.1: Datenverarbeitung im Hadoop-Umfeld

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

1.4 Weitere Aspekte der Thesis

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

durch die Nutzung des Apache Hadoop Frameworks abgedeckt werden und welche technischen oder organisatorischen Regelungen getroffen werden müssen.
Das System muss gegen fremden Zugriff gesichert sein. Es muss zu jeder Zeit ersichtlich sein, welche Personen zu welchem Zweck auf das System zugreifen.

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

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

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

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

1.5 Aufbau

In Kapitel 1 wird die grundlegende Problemstellung bei der forensischen Analyse beschrieben, welche diese Masterthesis lösen soll. Darauf folgt die Zielsetzung der Thesis, welche als möglicher Lösungsvorschlag zur beschriebenen Problemstellung gilt.
In Kapitel 2 folgt das allgemeine Entwicklungsvorgehen. Darin ist auch der aktuelle Projektplan enthalten, welche die Arbeitspakete definiert. Zusätzlich wird der Umgang mit Quellcode und Konfigurationsdateien als Open-Source Projekt beschrieben.

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

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

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

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

2 Vorgehen

2.1 Projektplanung

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

Der zweite Teil behandelt die Rohdatenspeicherung im HDFS und eine Datenaufbereitung. Es soll geprüft werden, welche Struktur der Daten für eine optimale Speicherung und Verarbeitung im Hadoop-Framework erforderlich ist. Dieser Teil beansprucht abermals vier Wochen. Am Ende dieses Arbeitspaketes soll ein erster Zwischenbericht erstellt werden, welcher die bisherigen Ergebnisse enthält (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. Das Ergebnis ist eine Sammlung von Programmen, welche mit Apache Spark auf den Daten ausgeführt werden können. Darüber hinaus soll ermittelt werden, welche Möglichkeiten zur Ausführung dieser Spark-Anwendungen bestehen.[3] Ein weiterer Aspekt der Datenanalyse beschäftigt sich mit den Möglichkeiten, wie die Ergebnisse persistiert werden können.[4] Im Anschluss soll die Performanz der Algorithmen geprüft werden. Hier bietet sich der Vergleich zu herkömmlichen Analyseprogrammen an. Denn schließlich hat diese Thesis auch das Ziel, bei großen Datenmengen schneller Ergebnisse zu liefern als die herkömmlichen Analysewerkzeuge auf einem einzelnen Analyserechner. Für dieses Arbeitspaket sind sieben Wochen eingeplant (siehe Abbildung 2.3). Darauf folgt ein zweiter Zwischenbericht.

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

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

[2] 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.

[3] Hier könnte beispielsweise das Projekt Apache Livy nützlich sein.

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

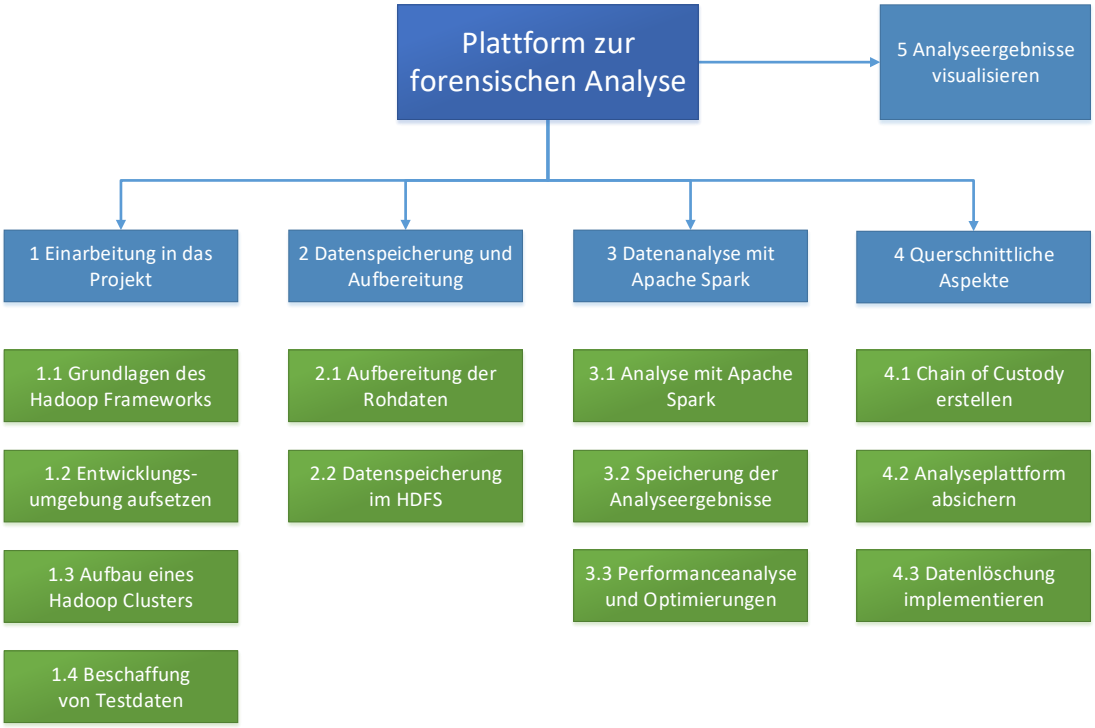

Abbildung 2.1: Arbeitspakete der Masterthesis

plant (siehe Abbildung 2.4).

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

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

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

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

2.2 Entwicklungsumgebung

Der Aufbau eine 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 physikalischem Apache Hadoop Cluster basieren, um mögliche Infrastrukturprobleme identifizieren zu können und die Performanz zu testen.

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

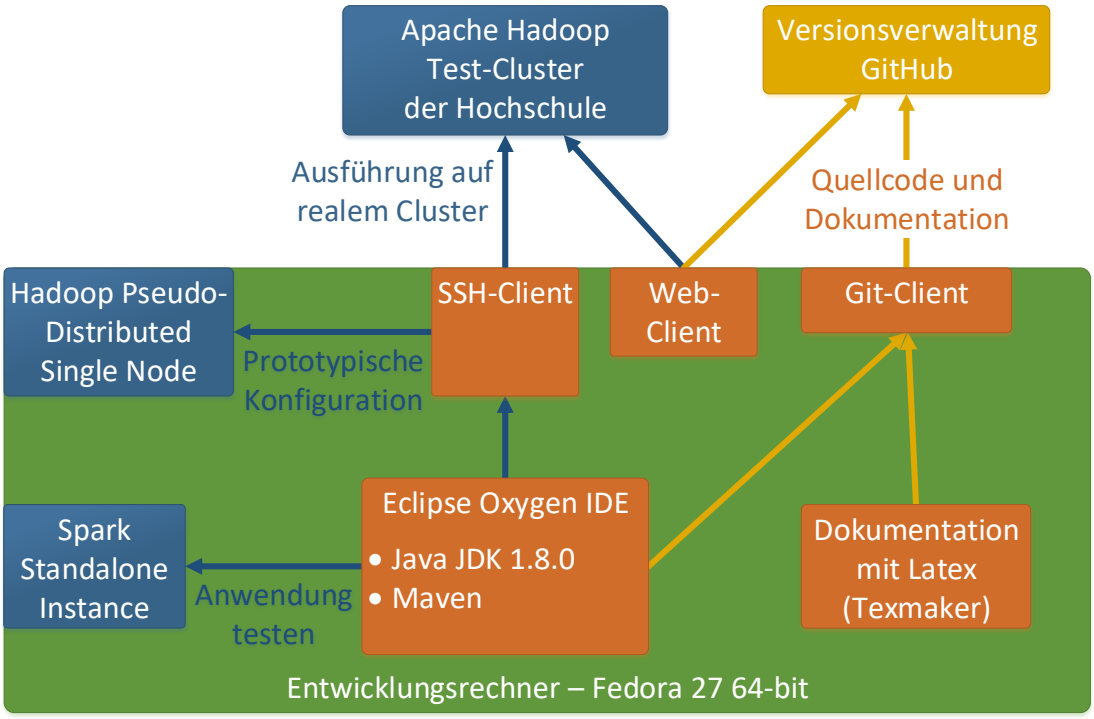

Abbildung 2.5: Komponenten der Entwicklungsumgebung

Zur Entwicklung der forensischen Analyseprogramme wird *Eclipse Oxygen* genutzt. Die Anwendungen selbst werden in Java geschrieben.[5] Zum Bauen der ausführbaren Java-Archive (JAR-Dateien) wird *Maven* verwendet. Mit Maven können weitere Java-Bibliotheken in eigenen Programmen auf einfache Weise wiederverwendet werden.[6]
Um die gebauten Java-Programme zur Datenanalyse schnell testen zu können, kann auf dem lokalen Entwicklungsrechner eine Apache Spark Standalone Instanz gestartet werden. Diese dient ausschließlich zur simplen Ausführung von Spark-Applikationen. Hierbei arbeitet die Instanz direkt auf dem lokalen Dateisystem und nutzt kein HDFS. Darüber hinaus wird nicht YARN, sondern ein bei Apache Spark mitgelieferter Ressourcenmanager genutzt.[7]

[5] Wobei auch Python oder Scala als Programmiersprache genutzt werden kann.
[6] Diese können über ein zentrales Repository, dem sogenannten *Maven Central Repository* aus dem Internet geladen werden (siehe `https://search.maven.org/`).
[7] Siehe Kapitel 3 für eine detaillierte Erklärung von Apache Hadoop.

Analog zur Spark Standalone Instanz kann auch ein Hadoop Pseudo-Distributed Single Node auf dem lokalen Rechner gestartet werden.[8] Mithilfe diese Single-Nodes können spezifische Konfiguration des HDFS oder des Ressourcenmanagers YARN ausprobiert werden. Letztendlich kommen diese lokale Hadoop und Spark Instanzen aber schnell an ihre Grenzen. Daher werden spezifische Konfigurationen und fertiggestellte Analyseprogramme auch auf einem realen Apache Hadoop Test-Cluster durchgeführt. Dort kann das Zusammenspiel zwischen Hadoop und Spark nachvollzogen werden. Auch entsprechende Last- und Performance-Tests sind nur auf dem Hadoop Test-Cluster sinnvoll.
Um mit dem Test-Cluster arbeiten zu können, wird ein SSH-Client benötigt. 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*[9] enthält die schriftliche Ausarbeitung der Thesis und den Quellcode als Latex-Projekt. Als Entwicklungsumgebung wird *Texmaker* genutzt. Über den Link `https://github.com/jobusam/foam-thesis` ist der aktuelle Stand der Arbeit jederzeit einsehbar.[10]

- Das Projekt *foam-processing-spark* enthält den Quellcode zur Auswertung mit Apache Spark™ . 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.

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

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

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

[9] Die Abkürzung *foam* oder auch *foAm* steht für ***fo****rensische* ***A****nalyseplattfor****m***

[10] 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.

2.3 Testdatengenerierung

Für den Aufbau einer forensischen Analyseplattform sollen entsprechende Testdaten generiert werden.

3 Grundlagen von Apache Hadoop®

3.1 Hadoop® Framework

Apache Hadoop ist ein etabliertes Java-Framework zur verteilten Speicherung und Verarbeitung von Daten. Durch die parallele Ausführung von Algorithmen eignet sich ein Hadoop-Cluster für rechenaufwendige Datenanalysen. Ein primäres Paradigma ist das Konzept der *Datenlokalität*. Die auszuführenden Programme werden auf die Knoten verteilt, auf welchen auch die Daten liegen. Ressourcenintensive Datentransporte sollen weitgehend vermieden werden.[1, S. 20 ff.]
Das Framework ist für die Ausführung auf Standardhardware konzeptioniert. Es wird also keine verhältnismäßig teure Spezialhardware benötigt. Das Cluster besteht aus vielen einzelnen Knoten mit Standardhardware, welche im Verhältnis zu Spezialhardware günstiger und leicht ersetzbar ist. Der Ausfall einzelner Knoten ist die Regel und wird bei der Datenhaltung entsprechend berücksichtigt.

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

Die Basis bildet das verteilte Dateisystem *Hadoop Distributed File System (HDFS)*, welches die Daten redundant auf allen Knoten des Computer-Clusters speichert. Hierbei besteht das Computer-Cluster selbst aus mehreren Knoten, auf welchen vorzugsweise ein Linux-Betriebsystem, wie beispielsweise CentOS, arbeitet.
Der Ressourcenmanager *YARN (Yet Another Resource Negotiator)* ist für die Verteilung und Bereitstellung von verfügbarer Rechenleistung verantwortlich.
Die dritte Komponente ist das *Hadoop® Map-Reduce Framework*. Hadoop Map-Reduce kann zur Datenverarbeitung genutzt werden. Hierbei werden Algorithmen parallel auf den Knoten prozessiert und die Ergebnisse im Anschluss zusammengetragen. Die einzelnen Zwischenergebnisse werden alle im HDFS abgelegt.[2]

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

[1] 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.

[2] 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 explizit verwendet wird.

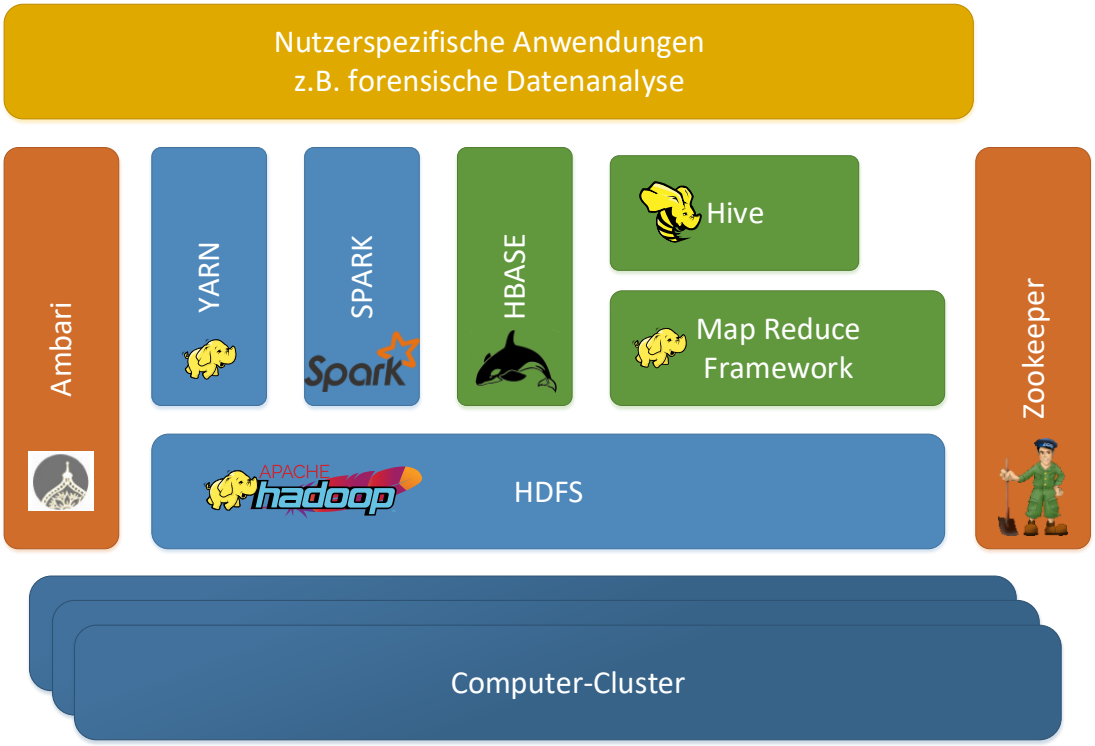

Abbildung 3.1: Apache Hadoop Ökosystem (Vgl. [1],[3]. Siehe Kapitel A.2)

oder spezielle Aufgaben durchführen.
So wird beispielsweise in dieser Thesis *Apache Spark*^{TM} bei der Prozessierung und Analyse der Daten genutzt. Der Vorteil von Apache Spark ist eine performante Datenverarbeitung, da einerseits die Daten verteilt verarbeitet werden und andererseits Zwischenergebnisse und temporäre Daten im Arbeitsspeicher der einzelnen Rechenknoten gehalten werden.[3]
Desweiteren bietet *Apache Hive*^{TM} eine Möglichkeit Dateien im HDFS mithilfe einer SQL ähnlichen Syntax[4] abzufragen. Hierbei nutzt die Komponente wiederum das Map-Reduce Framework von Hadoop. Apache Hive ist jedoch keine reine Datenbank, sondern arbeitet auf den Dateien im HDFS.
Apache HBASE® hingegen ist eine spaltenorientierte Key-Value Datenbank. Sie wurde eigens für Apache Hadoop implementiert, um große Datenmengen performant zu speichern.

Das Hadoop-Ökosystem als Ganzes muss auch konfiguriert und überwacht werden. Um die Verfügbarkeit einzelner Instanzen zu gewährleisten und gegebenenfalls redundante Verarbeitungswege anzubieten, wird *Apache Zookeeper*^{TM} 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*^{TM} genutzt.

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

[3] Durch das In-Memory Computing ist Apache Spark deutlich schneller als das bereits vorgestellte Hadoop Map-Reduce.
[4] Dem sogenannten HiveQL.

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

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

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

3.2 Hadoop HDFS

Das Hadoop Distributed Filesystem (HDFS) ist ein verteiltes Dateisystem, welches die Grundlage zu Speicherung von Daten im Hadoop-Ökosystem bietet. Nachfolgende Zwecke soll es erfüllen.
Es soll ausfallsicher sein. In der Standardkonfiguration wird jede Datei dreifach auf unterschiedlichen physikalischen Knoten gespeichert. Damit kann selbt bei einem Ausfall von zwei Knoten immer noch auf die Datei zugegriffen werden. Darüber hinaus verteilt das HDFS die Dateien automatisch und regeneriert sich selbst nach Ausfällen von Knoten. In großen Computer-Clustern mit mehreren hunderten Knoten ist ein Ausfall eines Knoten kein Sonderfall sondern die Regel. Daher muss sich das HDFS selbst heilen können, um auch ohne manuelle Administration weiter verfügbar zu sein.
Das HDFS (und auch Hadoop im allgemeinen) soll horizontal skalierbar sein. Wird mehr Speicher benötigt, sollen einfach noch Knoten hinzugefügt werden können.
Das HDFS ist auf hohen Datendurchsatz und die Speicherung großer Datenmengen ausgelegt. So können einzelne Dateien mehrere Gigabyte bis hin zu Terrabyte groß seind 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 herkömlichen 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 Koherenzmodell. Dies fördert den Lesedurchsatz indem die Unterstützung der Modifikation von Daten stark eingechränkt wird. Ein wahlfreies Schreiben in eine existierende Datei wird beispielsweise nicht unterstützt. Änderungen an Daten, welche von

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

Algorithmen vorgenommen werden, resultieren in neuen Datensätzen.
Darüber hinaus gilt das Prinzip der Datenlokalität. Algorithmen werden dort ausgeführt, wo die Daten liegen, um das Verschieben von Daten über das Netzwerk zu vermeiden.[5]

Der Aufbau eines HDFS bildet eine Master-Slave Architektur aus *NameNodes* und *DateNodes*. Der NameNode is 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 DateNodes gespeichert. Der NameNode organisiert diese Speicherung und bestimmt, wo welche Daten persistiert werden. Über den NameNode selbst fließen aber keine Rohdaten von Dateiinhalten. Auf Dateisystemebene ist das HDFS wie gängige Dateisystem hierarchisch organisiert. Jede Datei wird über einen absoluten Pfad eindeutig bestimmt und erhält entsprechende Metadaten, wie Dateirechte und Zeitstempel.

Abbildung 3.2 verdeutlicht die Struktur im HDFS. Angenommen es soll die Datei `/home/foo.txt` gespeichert werden. Dies kann mit dem Terminalprogramm *hdfs* durchgeführt. Das Programm selbst ist hier der HDFS-Client und hat Zugang zum Hadoop-Cluster. Der HDFS-Client speichert zuerst die Metadaten der Datei auf dem Name Node. Der Name Node bekommt die Größe der Datei auch mit und entscheidet dann, in viele Blöcke sie unterteilt werden soll. Darauf hin ermittelt für jeden einzelnen Block, auf welchen Date Nodes dieser Block gespeichert werden soll. Diese Blockaufteilung und die Zuordnung zuden Date Nodes werden an den HDFS-Client zurückgeschickt. Dieser übermittelt die Blöcke an einen der Data Nodes. Sobald der erste Data Node einen Block hat, sorgt er dafür die Blöcke an die anderen Data Nodes weiterzuleiten. Die Data Nodes selbst stehen auch in Kontakt zum Name Node und reporten ihren Zustand und die momentan gespeicherten Blöcke. Der Name Node bekommt darüber auch mit, wenn ein Data Node ausfällt.
Ein Block hat in der Standardkonfiguration 128 MB. Er kann aber auch bis zu 512 MB Größe konfiguriert werden. Dies wirft die Frage auf, ob das HDFS gerade für sehr kleine Dateien, wie sie bei der Analyse von Datenträgern auch vorkommen, nicht zu viel Speicher verschwendet.
Hierbei wird der gleiche Block immer in unterschiedlichen Data Nodes angelegt. Es ist nicht erlaubt den gleichen Block mehrmals im gleichen Data Node zu replizieren. Daher ist die Anzahl der Replikationen auch kleiner gleich der Anzahl Data Nodes im System.
Wichtig hierbei ist auch, dass im Produktivsystem auf jedem physikalischen Knoten auch nur ein DataNode oder ein NameNode läuft. Denn würden beispielsweise mehrere DateNodes auf dem gleichen physikalischen Knoten laufen, so wäre bei einem Ausfall nicht mir garantiert, dass die Dateiinhalte auch noch auf mindestens zweie anderen Knoten laufen. Denn der Replikationsmechanismus im HDFS kann nicht erkennen, ob jeder Knoten physikalisch unabhängig arbeitet. Allerdings hat Hadoop eine sogenannte *Rack-Awareness*. So ist es möglich zu bestimmen, welche physikalischen Server in einem gemeinsamen Rack laufen. Abhängig davon, versucht das HDFS die Daten teilweise im selben Rack redundant zu speichern aber auch einige Replikationen außerhalb des Racks anzulegen. So kann auch der Ausfall eines Racks im Notfall kompensiert werden.
In Testumgebungen ist aber schön möglich sogar NameNode und mehrere DateNodes auf einem Knoten laufen zu lassen. Allerdings greifen die Mechanismen für eine Toleranz gegenüber Hardwareausfällen dann nicht mehr.

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

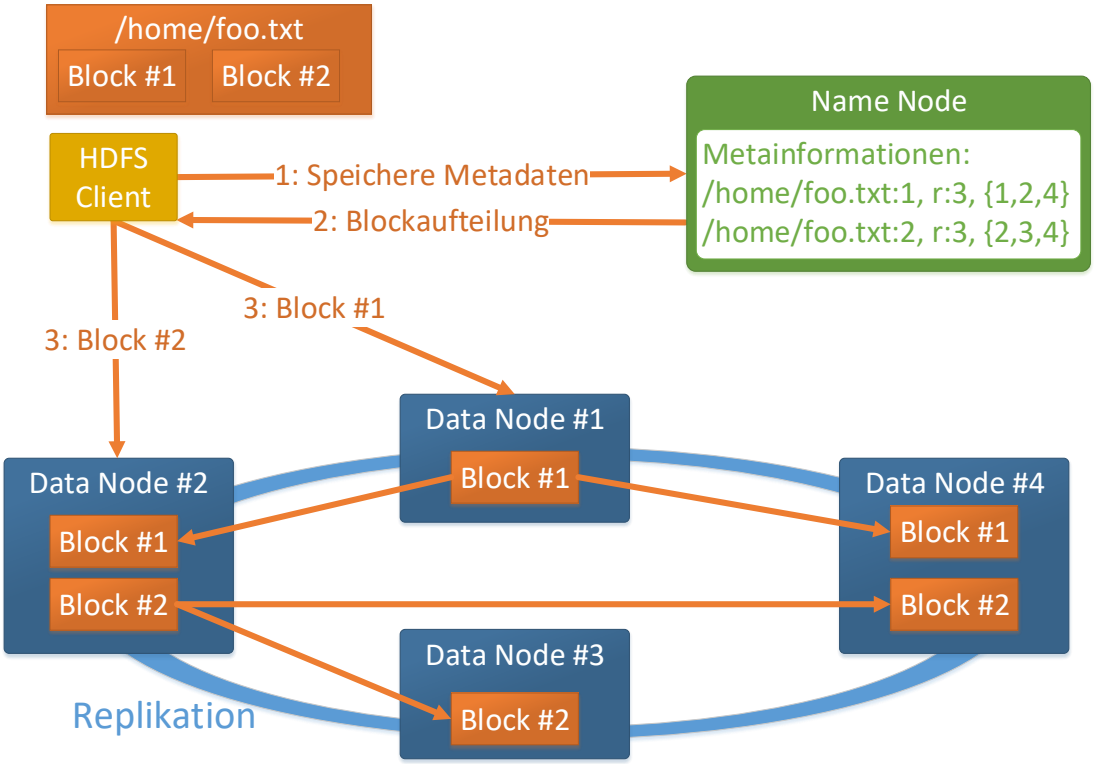

Abbildung 3.2: HDFS - Datenspeicherung im Verbund (Vgl. [5],[3])

tainformationen und erstellt daraus regelmäßig Checkpoints. Der Name Node hält die Metainformationen im Arbeitsspeicher. Es existiert aber auch eine Datei *FsImage* und ein *EditLog*, welche persistent auf der Festplatte gespeichert sind. Das FsImage selbst beschreibt einen Zustand der Dateisystemmetainformationen zu einem gewissen Zeitpunkt. Im EditLog befinden sich alle Änderungen seit dem letzten Checkpoint bis zum aktuellen Zeitpunkt. Der Secondary Name Node erstellt aus dem FsImage und dem EditLog regelmäßig neue Checkpoints, die dann der produktive Name Node bei einem möglichen Neustart wiederverwenden kann. Der *Secondary Name Node* unterstützt also den (First) Name Node, er kann ihn aber nicht ersetzen.
Daher ist es möglich auch einen sogenannten *Standby Name Node* zu konfigurieren. Dieser kann einspringen, sobald der erste Name Node ausgefallen ist. Allerdings muss dieser extra konfiguriert werden. Dafür kann aber dann der Secondary Name Node deaktiviert werden.[3, S. 88] **TODO: prüfen ob das wirklich stimmt!**

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

3.3 Hadoop YARN

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

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

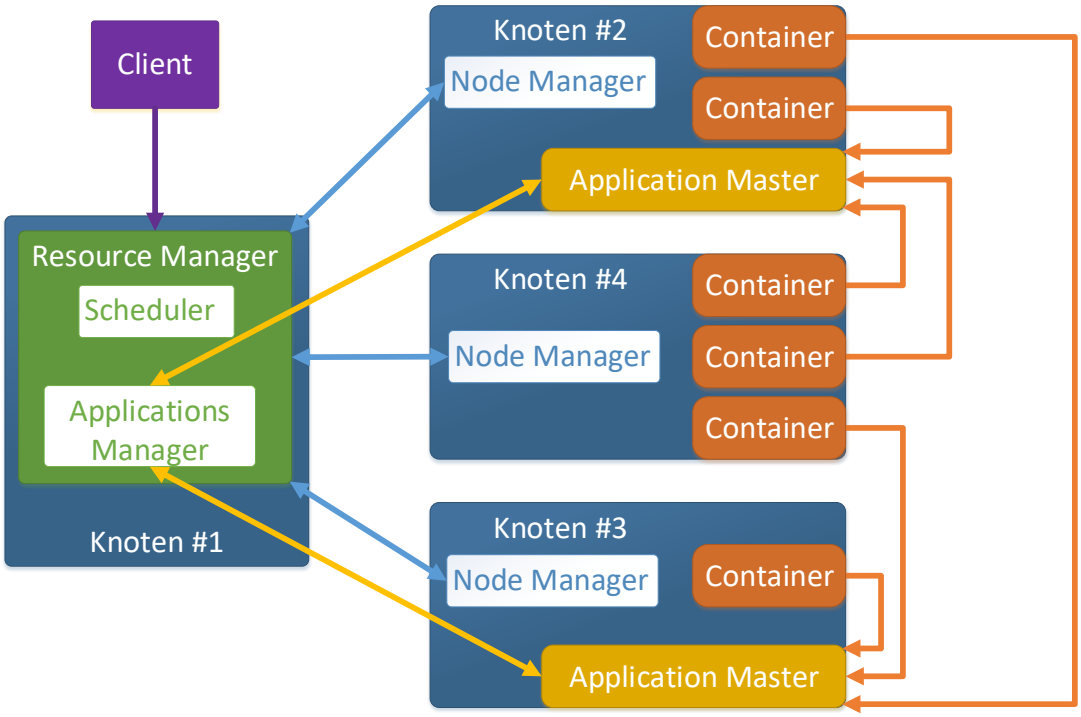

Abbildung 3.3: Ressourcenverteilung mit YARN (Vgl. [4],[3])

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

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

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

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

Der Application Master erhält dann, die Zustimmung von Resoure Manager, nachdem Scheduler die geforderten Resourcen entsprechend eingeteilt hat. Darauf fordert der Application Manager den Node Manager auf den jeweiligen Knoten auf, entsprechende Container zu erstellen.
Die einzelnen Node Manager stehen in Kontakt zum Resource Manager und reporten ihm, den aktuellen Status des Knoten und dessen Auslastung.
Nach der Ausführung der einzelnen Tasks innerhalb den Container und dem Abschluss des Jobs, schickt der Application Manager die Ergebnisse direkt zurück zum Client.[8]. Danach meldet er sich beim Resource Manager ab. Zuletzt kümmert sich der Resource Manager dann über das Freigeben von allokierten Resourcen.

Ähnlich wie beim Prozessschedulung in ein herkömmlichen Betriebssystem, gibt es auch für YARN unterschiedlicher Algorithmen, die festlegen, in welcher Reihenfolge und Zeitdauer die einzelnen Jobs ausgeführt werden. Bekannte Scheduler sind der *Fair Scheduler* und der *Capacity Scheduler*. Abhängig von der genutzten Plattform/Distribution einzelner Hersteller ist für YARN ein anderer Scheduler konfiguriert. In etlichen Fällen wird der Capacity Scheduler als Standard konfiguriert, da dieser versucht alle Knoten möglichst effizient auszusteuern um den höchstmöglichen Datendurchsatz durch erreichen. Der Fair-Scheduler prüft hingegen, dass jedem Job die gleichen Ressourcen zugeteilt werden, um möglichst alle Jobs parallel bedienen zu können.**TODO: Scheduling prüfen!**
In großen Clustern wird die Prozessierung in mehrere Sub-Cluster mit eigenen Resource Managern aufgeteilt. Diese Struktur findet sich in der Literatur unter *Federaded YARN* und soll die Skalierbarkeit von YARN in großen Clustern ermöglichen.

3.4 Apache Spark

3.5 Datenspeicherung in Datenbanken

[7] 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.

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

4 Aufbau einer Analyse-Plattform

4.1 Allgemeines

4.2 Datenspeicherung im HDFS

4.3 Datenverarbeitung mit Apache Spark™

4.4 Forensische Anforderungen

4.4.1 Plattform absichern

Ursprünglich spielt das Thema der Datensicherheit bei Apache Hadoop keine Rolle und gewann erst nach und nach an Relevanz. Anfänglich wurde immer angenommen, dass das Hadoop-Clusters aus vertrauenswürdigen Maschinen besteht, welche von vertrauenswürdigen Nutzern in abgesicherten Umgebungen verwendet wird[1]. Mittlerweile hat sich der Bedarf nach Sicherheit deutlich erhöht, da oftmals rießige vertrauliche Datensätze verarbeitet werden, welche bei Angriffen sehr schnell abfließen könnten.

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

Authentifizierung

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

[1] Vgl. https://www.infoq.com/articles/HadoopSecurityModel.
[2] https://knox.apache.org/

Datenverschlüsselung

Prinzipiell lässt sich die Datenverschüsselung 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 Verschlüsselungschlüssel nutzen. Die Verschlüsselung selbst kann direkt auf Betriebssyssystemebene beispielsweise auf LUKS aufbauen, oder sie findet auf logischer Ebene im HDFS statt.[2]

Darüber hinaus ist die Transportverschlüsselung auch möglich. So müssen die einzelnen Services wie Webzugriffe mit TLS verschlüsselt werden.[3]

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

4.5 Visualisierung der Ergebnisse

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

- Für jede Datei sollen Name, Pfad, Größe,Hashsumme, Dateityp, Owner und Group, Zugriffsrechte und die Zeitstempel der Erstellung und letzter Speicherung angezeigt werden.
- Nach all diesen Parametern kann auch gesucht werden.
- Auffinden von Duplikaten anhand der Hashsummen
- Indizierung für schnelle textbasierte Inhaltssuche?
- Zeitleiste? (wohl eher optional)
- Wordcloud, geographische Visualisierung, Flare-Chart, Tree-Map, Calendar-Chart als Timeline?
- Webframeworks wie `https://d3js.org/` [4]
- Neo4j
- Open Source Community Variante Helical Insight

[3] Weiter Infos unter: `https://www.infoq.com/articles/HadoopSecurityModel` und `https://community.hortonworks.com/articles/102957/hadoop-security-concepts.html`.

[4] Siehe auch `https://bl.ocks.org/mbostock/4063550` oder `https://bl.ocks.org/mbostock/5944371` oder `https://bl.ocks.org/mbostock/1046712` oder `https://bl.ocks.org/mbostock/4063269`. Letzteres wäre characteristisch für foAm. oder `http://xliberation.com/googlecharts/d3concept.html`

5 Zusammenfassung

6 Ausblick

Im praktischen Teil

Literatur

[1] Jonas Freiknecht. *Big Data in der Praxis*. 1. Auflage. Carl Hanser Verlag, 2014.

[2] Joey Echeverria Ben Spivey. *Hadoop Security: Protecting Your Big Data Platform*. 1. Auflage. O'Reilly Media, Inc, 2015.

[3] Sam R. Alapati. *Expert Hadoop Administration*. 1. Auflage. Addison Wesley, 2016.

[4] 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).

[5] o. V. *HDFS Architecture*. Version 3.0.0. Apache Software Foundation. 8. Dez. 2017. URL: https://hadoop.apache.org/docs/r3.0.0/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html (besucht am 17. 03. 2018).

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

A Anhang A

A.1 Analyse ähnlicher Projekte und Produkte

Im Bereich der IT-Sicherheit und Incident Response existiert für Unternehmensinfrastrukturen das Apache Projekt *Metron*, welches auf dem Hadoop Framework aufbaut.[1]
Ziel dieses Projektes ist es Sicherheitsvorfälle zu finden und zu analysieren. Hierbei kann Apache Metron auch mit Telemetriedaten umgehen.[2]
Eine entsprechende Abgrenzung zu diesem Projekt besteht aufgrund der unterschiedlichen Projektziele. Diese Thesis bezieht sich auf die forensische Analyse von Beweismitteln und informationstechnischen Systemen. Es ist nicht das Ziel Sicherheitsvorfälle in unternehmenskritischen Infrastrukturen zu analysieren.

Das Open-Source Framework *Turbinia* ist ein weiteres Projekt, welches ähnliche Ziele verfolgt.[3] Der Grundgedanke ist die Automatisierung und Skalierung forensischer Analysen in Computer-Clustern. Prinzipiell hat dieses Projekt das gleiche Ziel, wie diese Masterthesis. Aufwendige Analysen sollen parallelisiert verarbeitet werden, um sie schneller zu verarbeiten. Das Projekt ist aktiv[4]. Allerdings ist es jedoch in einer frühen Alpha-Phase und daher noch nicht ausgereift. Dieses Projekt basiert auch auf einer Master-Client Architektur. Es bietet aber keine Nutzung auf Basis eines verteilten Dateisystems an. Es muss dafür gesorgt werden, dass jeder Knoten auf alle verfügbaren Daten (Beweismittel) zugreifen kann. Im Rahmen dieser Thesis hingegen, wird durch die Nutzung von Apache Hadoop, eine verteilte Speicherung von Daten unterstützt. Darüber hinaus werden entwickelte Algorithmen dort ausgeführt, wo die Daten liegen und nicht umgekehrt.

Ein klassisches Analyse-Werkzeug in der Forensik ist *Autopsy*. Es basiert auf *The Sleuth Kit* und ist kostenlos.[5] Mit dem Werkzeug können Hashsummen berechnet oder auch Multimediadateien analysiert werden. Autopsy ist ein Single-Node Analyseprogramm und läuft vorzugsweise auf einem eigenen Analyserechner pro Nutzer.
Es gibt auch die Möglichkeit das Programm kollaborativ zu verwenden. Dabei gibt es einen zentralen Netzwerkspeicher, welcher alle Beweismittel enthält. Es ist möglich mit mehreren Nutzern parallel am gleichen Fall zu arbeiten und Analyseergebnisse in Echtzeit zu teilen. Diese Art der verteilten Analyse zeigt Ähnlichkeiten zu dieser Thesis auf.
Allerdings geht es bei diesem kollaborativen Ansatz vielmehr darum, an einem großen Fall mit mehreren Nutzern zu arbeiten und Ergebnisse einfacher zusammenzutragen. Einzelne

[1] Siehe https://metron.apache.org/ (Stand: 5.3.2018).
[2] Siehe https://www.heise.de/developer/meldung/Cybersecurity-Apache-Metron-wird-Top-Level-Projekt-3695901.html (Stand: 5.3.2018)
[3] Siehe https://github.com/google/turbinia (Stand 5.3.2018).
[4] Dies ist daran erkennbar, dass der letzte Commit in das Github-Repository am 26.01.2018 erfolgte.
[5] Siehe https://www.sleuthkit.org/autopsy/ (Stand 5.3.2018).

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

A.2 Lizenzierungen in dieser Arbeit

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

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

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

[6] Siehe auch https://www.apache.org/foundation/marks/, Stand: 21.3.2018.

B Hadoop Konfigurationen

B.1 Aufsetzen des aktuellen Hadoop-Frameworks

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

```
#Following code works for Fedora 27

#Create a new Hadoop user
sudo groupadd hadoop
sudo adduser -g hadoop hduser

#Change account
#get root
sudo -i
#change to hduser without setting any password for hduser
sudo -l hduser

#Create a ssh access without passwort!

ssh-keygen -t rsa -P '' -f ~/.ssh/id_rsa
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
chmod 0600 ~/.ssh/authorized_keys

#Store initial fingerprint
ssh localhost


#Unzip latest hadoop version (3.0.0)

#Change owner to hduser
sudo chown hduser:hadoop -R hadoop-3.0.0

#Configure


#Start HDFS
./sbin/start-dfs.sh
```

Listing B.1: Konfiguration des Hadoop-Frameworks