HUGO

Highly Useful Graphical Output

Durchgeführt im Rahmen des Moduls „Big Data Engineering“ des Master Studiengangs Wirtschaftsinformatik an der Fachhochschule Münster.

Gruppenmitglieder:

Ben Lohrengel Matrikelnummer: 872164

André Meier Matrikelnummer: 872175

Prüfer:

Lars George

Abgabedatum: 25.02.2016

Vorwort

Die vorliegende Dokumentation beschreibt das Projekt HUGO (Highly Useful Graphical Output), welches im Rahmen des Mastermoduls Big Data Engineering an der Fachhochschule Münster im Wintersemester 2015/16 durch André Meier und Ben Lohrengel durchgeführt wurde.
Dabei wurden folgende Aufgabenteile durch André Meier realisiert:

- Design und Implementierung der Website
- HBase Konfiguration und Implementierung
- Implementierung der Videostreamspeicherung
- Outputgenerierung
- Implementierung des Workflows durch Oozie

Während Ben Lohrengel für folgende Punkte verantwortlich war:

- Implementierung der MapReduce Jobs
- Zerlegung einer einzelnen Videodatei
- Speicherung der Bilder in HDFS

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einleitung

1.1 Problemstellung

Das Internet enthält heute eine unvorstellbare Menge an Videos. Im Juli 2015 wurden allein auf der Videoplattform YouTube pro Minute 400 Minuten Material hochgeladen [1]. All diese Videos identifizieren sich aus einer bestimmten Kombination von Pixeln.

Es ergibt sich nun die Fragestellung, inwieweit das Bildmaterial der Videos auf seine Farb- bzw. RGB (Rot, Grün, Blau)-Werte hin analysiert und ausgewertet werden können. Als Ergebnis sollen zwei Farbwerte visualisiert werden. Zum einen ein durchschnittlicher Farbwert, basierend auf allen im Stream vorkommenden Pixel, und zum anderen der am häufigsten vorkommende, also dominierende Farbwert. Das Ziel ist es, nicht nur eine diese Berechnungen durchzuführen, sondern auch eine Plattform für Nutzer zu schaffen, auf der sie die Ermittlung durchführen lassen können.

Dazu ist es zunächst nötig, den Link einer, prinzipiell beliebigen, Videoquelle einzugeben. Nach der Betätigung eines Buttons soll sich das System eigenständig den Videostream herunterladen und lokal ablegen. Die darauffolgende interne Verarbeitung soll mit verschiedenen Tools aus dem Bereich des Big Data Engineering erfolgen. Zum Abschluss des Analyseprozesses soll dem Nutzer das Ergebnis in Form einer Durchschnittsfarbe sowie die ermittelte dominierende Farbe auf der Plattform präsentiert werden.

1.2 Zielsetzung

Ziel des Projektes “Highly Useful Graphical Output” (kurz HUGO) ist die zuvor beschriebene Vorgehensweise zu realisieren. Im Folgenden werden der erwartet Input sowie Output und das User Interface näher spezifiziert.

1.2.1 Input

Zu Beginn des Prozesses wird ein Videostream benötigt, den ein Nutzer auf seine dominante bzw. Durchschnittsfarbe analysieren lassen möchte. Dabei müssen jedoch zwei Voraussetzungen erfüllt sein.

1. Der Stream muss online verfügbar und über einen direkten Downloadlink erreichbar sein.

2. Die Onlineplattform auf der sich der Stream befindet muss über eine frei zugängliche API erreichbar sein.

Ist beides erfüllt, so kann der Nutzer den http-Link zum Video kopieren und in der Eingabemaske des HUGO einfügen.
Jedoch wird bei den meisten Videoplattformen der zuvor erwähnte Punkt 2 leider nicht erfüllt. In der Regel sind es datenschutzrechtliche Gründe, die große Anbieter wie YouTube dazu bewegen keine freie API mehr anzubieten. Die Wikimedia Foundation hingegen bietet auf ihrer Website „wikimedia.org“ Videostreams an, die basierend auf freien Lizenzen genutzt und vor allem mittels ihrer API „MediaWiki“ von Drittsystemen verwendet werden können.

Wikimedia Commons

Wikimedia definiert sich als weltweite Bewegung für freies Wissens. Unter dem Dach der gemeinnützigen Wikimedia Foundation werden verschiedenste Wikimedia-Projekte wie Wikipedia angeboten. Darunter befindet sich ebenfalls Wikimedia Commons, welche zum Ziel hat, als zentrales Medienarchiv in verschiedenen Sprachen für die Wikipedias zu dienen. Die Medienplattform stellt somit gemeinfreie und frei-lizensierte Medieninhalte wie Bilder, Audio- und Videodateien bereit. Derzeit werden über 30 Millionen Dateien zur Verfügung gestellt.

Abbildung 1: Logo Wikimedia Commons. Quelle: (1)

Die Gemeinfreiheit sowie das freie Lizensierungsmodell gestatten es, Videoframes ohne die Verletzung von Persönlichkeits- und Datenschutzrechten sowie des Urheberrechts herunterzuladen und nicht-kommerziell zu nutzen. Die Videoinhalte können dabei über einen direkten Link erreicht werden. Das bedeutet, dass der Link nicht, wie bei anderen Plattformen, auf die Webseite des Videos führt, sondern direkt zum Download verwendet werden kann. Somit erfüllt Wikimedia Commons die beiden zuvor erwähnten Voraussetzungen für eine verwendbare Videoquelle. Dadurch wird diese Plattform im Folgenden als Videoherkunft für das HUGO-Projekt als alleiniges Input verwendet.

1.2.2 User Interface

Als zentrale Anlaufstelle zur Interaktion mit dem HUGO-Projekt dient eine einzige Webseite. Hier ist zunächst eine Eingabemaske zu sehen. Der zuvor kopierte Link zum Videostream ist dort einzufügen. Im nächsten Schritt ist der Nutzer dazu angehalten, anzugeben, welches Ergebnis er haben möchte. Die Optionen sind die Ermittlung der dominierenden Farbe oder die Durchschnittsfarbe. Ferner kann der Nutzer eine im Folgenden genannte „Bucketgröße“ angeben. Diese gruppiert die Farbwerte in Blöcke. Wird beispielsweise der Wert 10 als Bucketgröße angegeben, werden alle Farbwerte auf den unteren Zehnerwert gerundet. Diese Option wird geboten, um etwas mehr Flexibilität bei den Analyseverfahren zu erreichen.

Nach der Bestätigung über einen Button wird das entsprechende Video im Hintergrund vom System heruntergeladen und lokal persistiert. Anschließend soll die interne Verarbeitung ebenfalls unsichtbar durchgeführt werden. Für den Nutzer ist auf der Webseite ein Lade- bzw. Fortschrittsbalken zu sehen, anhand dessen der Prozessfortschritt erkennbar ist. Zudem kann der Abschluss einzelner Prozessabschnitte, wie bspw. der erfolgreiche Download, durch Ausgabe eines Statustexts nachvollziehbar gemacht werden. Zum Abschluss der Verarbeitungskette sind die Ergebnisse in Form einer Grafik wieder auf der Webseite angezeigt.

1.2.3 Output

In den vorherigen Abschnitten erfolgte die Definition des erwarteten Inputs, der zukünftig verwendeten Quelle sowie der graphischen Oberfläche. In diesem Abschnitt werden die erwarteten/möglichen Ergebnisses bzw. Outputs beschrieben.
Nachdem eine Videoquelle erfolgreich eingelesen, verarbeitet und analysiert wurde, soll auf der zentralen Weboberfläche die Ausgabe erfolgen. Abhängig von der zuvor getroffen Wahl des Nutzer, sind die Farbwerte hinsichtlich eines Ergebnisses zu ermitteln und in Form einer Grafik zu visualisieren. Dabei stehen folgende Analysemöglichkeiten zu Verfügung.

1. Durchschnittsfarbe (average Color)

   Ziel der Ergebnisform „Durchschnittsfarbe“ ist den durchschnittlichen Farbwert pro Frame zu ermitteln.

2. Dominierende Farbe (dominant Color)

   Bei der Ergebnisform „Dominierende Farbe“ handelt sich um die Ermittlung jenes Farbwertes pro Frame, der am häufigsten vorkommt.

Unabhängig von dem gewünschten Ergebnis sind die ermittelten RGB-Werte eines jeden Frames abzulegen. Zur abschließenden Darstellung müssen die einzelnen Durchschnitts- bzw. dominierenden Farbwerte nacheinander als Pixel ausgegeben und in einer Grafik zusammengefasst werden. Die so entstandene Grafik bildet somit eine chronologische Aneinanderreihung der Farbwerte eines Videostreams. Abschließend ist dann die so entstandene Grafik auf der Weboberfläche anzuzeigen. Der Nutzer erhält damit das von ihm gewünschte Ergebnis und der Workflow des HUGO-Projektes ist beendet.

2. Verwendete Tools

2.1 User Interface - Website

Um den Usern eine komfortable Eingabemöglichkeit und Darstellung der Ergebnisse zu gewährleisten, wurde eine eigene Webseite entwickelt. In dem Testsystem ist diese über localhost:81 zu erreichen. Zur Realisierung der Website wurden ein Apache Webserver sowie Python als Scriptsprache genutzt, welche im Folgenden beschrieben werden.

2.1.1 Apache Webserver

Die Cloudera VM hat bereits eine Webserver, die allerdings für das „cloudera.quickstart“ User Interface genutzt wird. Daher wurde ein zweiter Webserver (Apache) installiert, welcher über einen anderen Port erreichbar sein ist. Der Webserver wurde via Command Line installiert und für das Projekt entsprechend konfiguriert.

2.1.2 Python

Als Scriptsprache für die Website wurde Python genutzt, da diese einfach zu verstehen ist und eine gute Unterstützung für REST POST und GET für Oozie bietet. Ein entsprechendes Vorgehen ist auf der Oozie Website dokumentiert. Für eine einfache Nutzung von POST

Requests wurde zusätzlich noch „python-requests“[1] installiert. Um via Python mit dem HDFS arbeiten zu können, wurde zusätzlich „hadoopy“[2] installiert.

2.2 Xuggler

Xuggler[3] definiert sich selber als einfachen Weg, Videostreams zu dekomprimieren, zu manipulieren und wieder zu komprimieren. Es handelt sich dabei um eine Java-Bibliothek und ist unter der GPL Version 3 Lizenz frei verfügbar. Verwendet wird die leistungsstarke FFmpeg media handling library[4].

Abbildung 2: Xuggler Logo. Quelle: (2)

Im Rahmen des HUGO-Projektes wird Xuggler dazu verwendet, einen zuvor geladenen und persistierten Videostream in seine einzelnen Frames zu zerlegen. Anschließend werden diese Frames im HDFS abgelegt und mit MapReduce analysiert.

2.3 Hadoop Distributed File System (HDFS)

HDFS (Hadoop Distributed File System) ist das verteilte Datensystem von Hadoop und ist durch den Master-Slave Ansatz vor allem für die Speicherung großer Datenmengen geeignet. Daher wird es für die Ablage der Video- und Bilderdateien wird im Rahmen des Projekts HUGO verwendet.

2.4 HBase

Die Anforderungen für das Projekt HUGO an ein Datenspeicherungssystem sind hohe Performance für CRUD (Create, Read, Update, Delete) Operationen, da dort die Metadaten des Videos abgelegt werden. Eben diese sind der Hauptfokus von HBase.

HBase ist ein verteiltes, spaltenorientiertes und multidimensionales Speichersystem und ist auf hohe Performance und Verfügbarkeit ausgelegt. Zwar unterstützt es keine Transaktionen, diese sind jedoch für HUGO nicht notwendig. Auch der Fakt, dass HBase Byte Arrays nutzt,

---

[1] HTTP://DOCS.PYTHON-REQUESTS.ORG/EN/MASTER/
[2] HTTPS://GITHUB.COM/BWHITE/HADOOPY.GIT
[3] HTTP://WWW.XUGGLE.COM/XUGGLER
[4] HTTP://WWW.FFMPEG.ORG/

stellt kein Hindernis dar, da ausschließlich Strings und Integer Werte gespeichert werden, die problemlos in Byte Arrays geparst werden können.
Ein weiterer Vorteil ist das System der Spaltenfamilien: Damit konnten die verschiedenen Analysemöglichkeiten als Spaltenfamilie gewählt und die Farbräume darunter spaltenweise gespeichert werden.
HBase ist auf kleine Datensätze ausgelegt. Daher werden nur die bereits oben genannten Metadaten in Form von Strings und Integerwerten in HBase gespeichert, während für die Persistierung der Videostreams HDFS genutzt wird.

2.5 Map Reduce

Im Rahmen des Projekts HUGO erfolgt die Farbanalyse der einzelnen Frames durch MapReduce. Die Entscheidung ist darin begründet, dass MapReduce als Hadoops Hauptverarbeitungsengine hervorragend die Integration im HDFS gespeicherter Daten in die Verarbeitung ermöglicht. Die Aufteilung in die verschiedenen Phasen zeigt die Auslegung auf parallele Datenverarbeitung von großen Datenmengen. Die Zerlegung von Videos in Frames, deren Speicherung sowie Analyse ist ein ebensolches Problem: Große Datenmengen müssen rechenintensiv verarbeitet werden. Dies funktioniert mit MapReduce nahtlos.
Ein weiteres Argument liegt im Komfort des MapReduce Frameworks für den Programmierer. Dieser muss nur die Funktionen des Mappers, des Reducers - optional des Combiners - und des Drivers implementieren, während MapReduce das Managen von Clustern und der Koordinierung von Jobausführungen zwischen den Knoten komplett selbst übernimmt.

2.6 Outputgenerator

Abschließend wurde ein sogenannter Outputgenerator erstellt. Hierbei handelt es sich um einen selbstentwickelten Algorithmus, der aus den Ergebnisdaten des MapReduce ein Bild erstellt, welche, wie zuvor beschrieben, in HBase abgelegt sind. Die genaue Implementierung ist in Kapitel 2.6 erläutert.

2.7 Oozie

Der Workflow des Projekts HUGO wird durch das Workflowverwaltungsystem Oozie gemanaged. Zum einen gibt es einen Workflow Editor, über den verschiedene Hadoop

Aktionen verknüpft verwenden können. Dieser wurde erstellt um das Oozie Projekt zu erstellen. Dieses stellt eine Workflow.xml Datei bereit, in welcher der Ablauf beschrieben wird. Diese Konfigurationsdatei wurde händisch befüllt.
Der Implementierungsverlauf orientiert sich an den in der Vorlesung geschildeten vorgehen. Dazu wurden die notwendigen Dateien (job.properties, workflow.xml, libraries) im HDFS abgelegt. Der Oozie Job kann dann entweder über die CLI oder die REST API gestartet werden. Für Testzwecke wurde der Oozie Workflow erst über CLI gestartet und der Jobstatus über das Webinterface überprüft. Das Webinterface stellt auch ein Errorlog bereit, welches zur Fehlerbehandlung Anhaltspunkte gab, um diese zu beheben.
Nach erfolgreicher Implementierung des Workflow und der Testdurchläufe, wurde der Start des Workflows auf die Website ausgelagert. Das neue Vorgehen wird in Kapitel 3.2 erläutert.

Der Workflow wird durch einen REST Call von der Webseite initialisiert. In der job.Properties wird der Pfad zu der „data.txt“ hinterlegt, welche die Inputdaten des Users von der Webseite enthält. Jeder Workflow Step liest immer die aktuellen Daten aus dieser aus und schreibt nach dessen Abschluss seine Ergebnisse zurück, die als Input für den nächsten Step dienen. Somit ist die Datenübergabe zwischen den Workflow Steps und die Zwischenspeicherung der Inputparameter gewährleistet.
Der erste Step ist die Überprüfung, ob das gesuchte Video bereits analysiert wurde, indem der Key in HBase abgefragt wurde. Falls das Video noch nicht runtergeladen, wird dieses in das HDFS geladen und anschließend mit Xuggler in seine Frames zerlegt. Diese werden dann vom MapReduce Job analysiert und die Ergebnisse in der HBase Datenbank persistiert. Der Outputgenerator bezieht die Ergebnisdaten aus HBase, verarbeitet sie zu einem Outputimage, legt dieses im HDFS ab und schreibt diesen Dateipfad zurück in die „data.txt“. Die Website lädt sich die „data.txt“ ins lokale Dateisystem und bezieht anschließend auch das Outputimage, welches dann auf der Website angezeigt wird.

2.8 Travis CI

Um eine kontinuierliche Integration zu gewährleisten, wird Travis C [5] verwendet. Dies liegt zum einen an der kostenfreien Verwendungsmöglichkeit, zum anderen an der Einfachheit der Einbindung. Es funktioniert out-of-box durch die Anbindung an GitHub[6], sodass kein weiteres Hosting, Installation oder Konfiguration geschehen muss. Zwar bindet man sich damit in einem gewissen Maße an GitHub, jedoch wurde dort das Repository bereits vor der Nutzung von Travis CI angelegt. Somit wird nun bei jedem Push in das Repository automatisch ein Test seitens Travis durchgeführt, deren Ergebnis optional direkt via eMail an den Repository Owner gesendet wird.

[5] HTTPS://TRAVIS-CI.ORG/
[6] HTTPS://GITHUB.COM/

3. Implementierung

In den vorherigen Kapiteln wurde zunächst die Idee des HUGO-Projektes zusammen mit dem gelieferten Input sowie den zu erwartenden Output erläutert, gefolgt von einer Einführung in die im Projekt verwendeten Tools. Das folgende Kapitel beschreibt nun die konkrete Implementierung HUGO.

3.1 Prozessübersicht

Abbildung 3 zeigt die Prozessübersicht des gesamten HUGO-Projekts. Es verdeutlicht die Zusammenhänge zwischen den einzelnen, im Anschluss beschriebenen Phasen. Durchgezogene Linien visualisieren den Prozessverlauf, gestrichelte hingegen bedeuten eine Datenpersistierung.

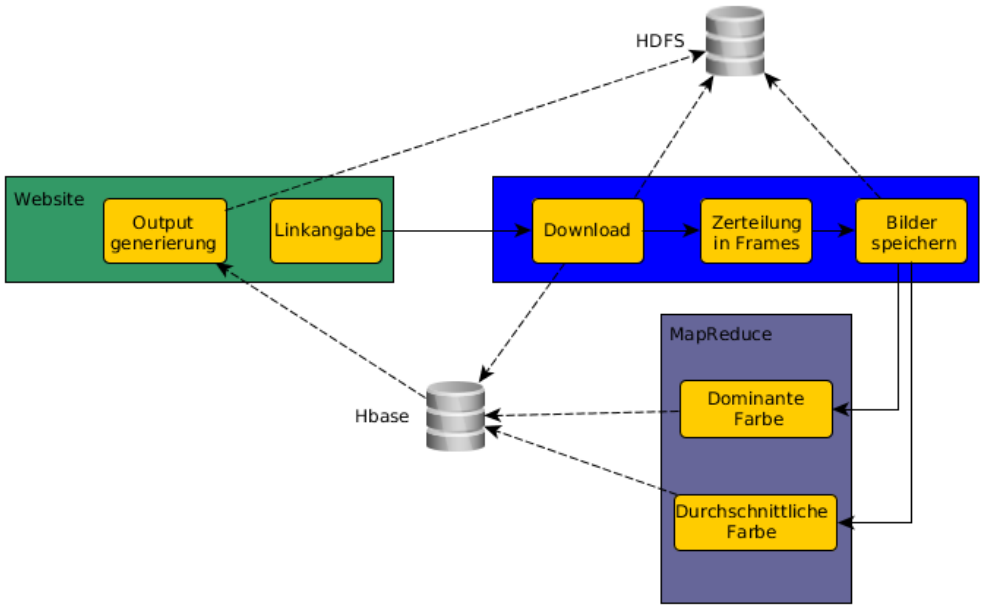

Abbildung 3: Prozessübersicht des HUGO-Projektes

Die Website stellt den Eintrittspunkt den Prozessablauf dar. Durch die Linkangabe wird der Videodownload gestartet. Das Video wird im HDFS abgelegt, während die Metadaten im HBase persistiert werden. Die Videos im HDFS werden in Offsetdateien gespeichert, wobei eine einzelne Offsetdatei bis zu 10GB groß werden kann. Ist dieses Limit erreicht, wird automatisch eine neue Offsetdatei begonnen. Unabhängig der Datengröße belegen Metadaten immer gleich viel Memory in der NameNode. Durch viele kleine Dateien wird die NameNode Memory immer voller, was die HDFS Skalierbarkeit und Performance negativ

beeinträchtigt [2]. Daher erfolgt die Speicherung der Videos in größeren Offsetdateien, statt viele kleinere Videodateien im HDFS abzulegen.
Nach einer erfolgreichen Zerlegung in Frames werden diese gespeichert; die Images wieder in HDFS, die Metadaten in HBase. Je nach der eingangs getroffenen Wahl wird nun entweder der MapReduceJob zur Ermittlung der durchschnittlichen Farbe oder der zur Bestimmung der dominanten Farbe angestoßen. Die jeweiligen Ergebnisse werden dann mit einem uniquem RowKey in HBase gespeichert.
Abschließend bezieht sich der Outputgenerator die Daten aus HBase und erstellt auf deren Basis dann das Outputimage, welches dem Anwender dann auf der Homepage angezeigt wird.

3.2 User Interface

Dem User wird über die in Kapitel 2.1 beschriebene Webseite eine die Startseite präsentiert, auf welcher er einen Video Downloadlink, die gewünschte Bucketgröße und das Analyseverfahren („averageColor“, „dominantColor“) angeben kann. Über den Button „Analyse“ startet der ganze Prozess, indem ein Python Script aufgerufen wird.
Dieses nimmt die Benutzereingaben entgegen und speichert sie in einer „data.txt“ Datei, welche anschließend im HDFS abgelegt wird. Über einen REST Call werden über eine „POST“ Methode die URL des REST Services „http://localhost:11000/oozie/v1/jobs?action=start“ und ein XML String mit den Einstellungen für den zu startenden Workflow übergeben. Die POST-Methode gibt die ID des gestarteten Workflows zurück.
Das Python Script übernimmt neben den REST Calls auch die Aufgabe der Webseiterstellung. Diese zeigt dem User den Fortschritt des Workflows in einem Fortschrittsbalken an. Außerdem wird der aktuell ausgeführte Job schriftlich angezeigt.
Über eine Schleife wird alle 5 Sekunden Status der Workflows über eine Rest Call abgefragt, die einen JSON String zurückgibt. Der JSON String enthält alle Informationen des Workflows.
Bei Abschluss des Workflows wird die Schleife abgebrochen. Das Python Script lädt dann erneut die „data.txt“ aus dem HDFS, welche jetzt den Pfad zu dem Outputimage enthält. Über „hadoopy.get(imagepath, outputpath)“ wird das Outputimage in das lokale Dateisystem geladen, sodass es anschließend in die Website eingebunden und damit dem User abschließend präsentiert werden kann.

3.3 Prozessphasen

3.3.1 Einlesen und Zerlegen der Videostreams

Falls der auf der Website angegebene Videolink nicht als Key in der HBase Datenbank zu finden ist, wird der Download gestartet. Dies geschieht über einen FSDataOutputStream. Anschließend wird das Video an die aktuelle Offsetdatei gehangen, die im Verzeichnis "hugo/Videos/filesName" liegt.

Ferner wird das einzelne Video temporär zwischengespeichert, um das Extrahieren aus der Offsetdatei zu umgehen. Die temporäre Datei wird mit Abschluss des Bearbeitungsschrittes gelöscht.

Zur Zerlegung des Videos in Frames wird Framework Xuggler genutzt, auf welches bereits in Kapitel 2.2 eingegangen wurde. Der Verweis auf das Video erfolgt als Inputargument in der Mainmethode. Xuggler zerlegt daraufhin das Video in einzelne Frames, für die Bearbeitung in BufferedImages vom Typ 3Byte BGR, also ein 8-Bit RGB BufferedImage ohne Alphachannel. Über die Systemzeit wird dann geprüft, ob eine festgelegte Zeit zwischen den Frames verstrichen ist, sodass ein neues Bild erstellt wird.

Die Bilder werden im Datenformat ".png" unter Angabe einer fortlaufenden Nummer in dem festgelegten Ordner "hugo/Frames/NameDesVideos" gespeichert, wobei NameDesVideos sich aus dem Namen der Videodatei (extrahiert aus der URL) sowie (bereinigtem) Timestamp zusammensetzt. Diese Kombination garantiert die Eindeutigkeit des Pfades, wobei zur Absicherung im Code noch das Vorhandensein des Pfades abgefragt und nötigenfalls gelöscht wird.

Die Speicherung der Bilder erfolgt im .png Dateiformat, da zum einen die Komprimierung verlustfrei erfolgt und zum anderen dieses Datenformat von Xuggler reibungslos unterstützt wird.

Final wird eine "Links.txt" im HDFS-Verzeichnis der Frames erstellt. In dieser wird, durch einen Zeilenumbruch getrennt, für jedes einzelne Bild der entsprechende Link gespeichert. Diese Datei wird dem MapReduceJob als Input zur Verfügung gestellt.

3.3.2 Analyse der Frames

Das HUGO-Projekt enthält zwei MapReduce Jobs: Einen, der die durchschnittliche Farbe eines Bildes ermittelt (im folgenden MR1 genannt), während der andere dessen dominante Farbe

bestimmt (im folgenden MR2 genannt). Der grundlegende Aufbau ist bei beiden gleich: Jeder hat eine Main- und Run-Methode, einen Mapper und einen Reducer. Es gibt jedoch die Ausnahme, dass der MR1 eine Combinerklasse besitzt.

Die Mainmethode beider Jobs erhält als Input eine Datei. Diese enthält die zu Workflowbeginn angegebene Bucketgröße und stellt diese per Parameterübergabe dem Mapper zur Verfügung. Zudem wählt die Mainmethode die letzte im HDFS modifizierte Datei in einem festgelegten Verzeichnis aus. Dies stellt sicher, dass automatisch die zuvor beschrieben "Links.txt" ausgewählt wird. Gemäß dem Workflow bildet ihre Erstellung den Abschluss des Zerlegungsprozesses.

Für jeden zu analysierenden Frame ist es notwendig, ausschließlich *einen* MapReduceJob zu starten. Andernfalls würde das verwendete Rechenschema die Ergebnisse derart verfälschen, dass sie unbrauchbar werden. Daher wird aus der "Links.txt" jede Zeile einzeln ausgelesen und ein exklusiver MapReduceJob gestartet. Zu beachten ist, dass die erste Zeile immer den Key für die Datenbank enthält, welcher per Parameterübergabe an den Reducer übergegeben wird.

Obligatorisch setzt die Run-Methode die Configuration, den Job, das FileInputFormat, die Mapperklasse und dessen Outputkeyklasse und -valueklasse sowie die Reduceklasse fest. Bei MR1 wird ebenso die Combinerklasse festgelegt.

Im Folgenden wird auf die einzelnen MapReduceJobs eingegangen.

3.3.2.1 Ermittlung des durchschnittlichen Werts – MR1

Rechenschema am Beispiel

Als Beispiel dient im Folgenden das erste Bild eines Videos mit der beispielhaften URL http://url.de/vid.ogg. Es besteht aus 4 Pixeln, wobei zwei Pixel den Farbwert R100 G0 B0 haben, ein Pixel den Wert R180 G0 B0 sowie ein Pixel den Wert R255 G0 B0 hat. In diesem Beispiel behandeln wir nur den roten Farbraum („R- Wert), da der blaue und grüne Farbraum null sind.

Zunächst wird gezählt, dass R100 zweimal vorhanden ist, R180 und R255 nur einmal. Danach wird der relative Anteil der Farbe an der Gesamtpixelzahl ermittelt, für R100 ist es 2/4 = 0.5, für R180 und R255 1/4= 0.25. Danach wird der relative Anteil mit dem Farbcode multipliziert, sodass man folgende Zahlen erhält: Für R100 50, für R180 45 sowie R255 abgerundet 63.

Summiert man diese Zahlen, erhält man das Gesamtergebnis der durchschnittlichen Farbe für alle R-Werte des Bildes, also gerundet 158. Als Gegenprobe kann können die Farbwerte summiert und durch die gesamte Pixelanzahl dividiert werden, was zum gleichen (abgerundeten) Ergebnis führt ((100+100+180+255) / 4) = 158).

Zusammenfassend ergeben sich also folgende Prozessschritte:

1. Häufigkeit der einzelnen Farbwerte ermitteln
2. Ermittlung des relativen Anteils des Farbcodes an der gesamten Pixelzahl
3. Zusammensetzung des Wertes durch Summierung der Anteil

Der Mapper erhält als Parameter die angegebene Bucketgröße sowie einen Verweis auf das zu analysierende Bild. Dieses wird aus dem HDFS geladen und für jeden Pixel via java.awt.Color.getRGB() jeder einzelner R, G und B Wert ermittelt. Der Mapperoutput wird als Key, bestehend aus dem Farbraum (R, G oder B), dem Wert (je nach Bucketgröße gerundet) sowie der gesamten Pixelanzahl des Bildes, und als Value 1 übergeben. Das Output beim o.g. Beispiel wäre <R100,4, 1,1> <R255,4, 1> <R180,4, 1>.

Der Combiner aggregiert die Häufigkeit eines "gesehenen" Farbcodes, sortiert nach Farbraum. Das heißt, es wird gezählt, wie häufig beispielsweise der Wert R10 gesehen wurde. Dies geschieht für alle im Mapper erkannten R, G und B Werte. Danach wird der relative Pixelanteil an der Gesamtpixelzahl ermittelt und anschließend mit dem Farbcode multipliziert. Dieser Wert wird als Value, zusammen mit dem Key R, G oder B an den Reducer weitergegeben. Ein Beispieloutput wäre <R, 50, 45, 63>.

Der Reducer erhält alle Farbwerte des Durchschnitts, gruppiert nach R, G bzw. B. Anschließend werden die Werte gruppiert aufsummiert, sodass nur noch ein einziger Farbwert bleibt. Dieser finale Wert wird dann in HBase gespeichert, mit dem Hyperlink des Videos als Key, averageColor als Columfamily, dem Farbraum als Qualifier sowie dem Farbwert dann als Value. Im Beispiel bleibend wäre dies <http://url.de/vid.ogg_0000001, averageColor, R, 158>. Der Anhang „0000001“ angehangen, da es sich um den ersten Frame es Videos handelt.

3.3.2.2 Ermittlung der dominanten Farbe – MR2

Der Mapper verhält sich nahezu identisch zu MR1. Das heißt, es wird ermittelt, dass ein gesamter Farbwert "gesehen" wurde. Allerdings wird diesmal als Value der gesamte Farbcode ausgegeben, sodass das Output beim o.g. Beispiel <R100 G0 B0, 1,1> <R255 G0 B0, 1> <R180 G0 B0, 1> ist.

Der Reducer ermittelt für jeden Key (also Farbwert), dessen Häufigkeit des Auftretens. Dabei wird gegen eine globale Variable maxValue geprüft, ob das Ergebnis größer ist. Falls dies der Fall ist, wird der zugehörige Farbwert, getrennt nach R, G oder B Wert in die HBase Table geschrieben. Am o.g. Beispiel orientiert, wäre dann der Output <http://url.de/vid.ogg_0000001, R, 100> < http://url.de/vid.ogg_0000001, dominantColor, G, 0> < http://url.de/vid.ogg_0000001, dominantColor, B, 0>.

3.3.3 Ergebnispräsentation

In Kapitel 2.6 wurde bereits erwähnt, dass das Output über einen eigens entwickelten Outputgenerator erstellt wird.

Wie in Abbildung 3 zu sehen ist, besteht ein Outputimage aus mehreren vertikalen Farbstrichen, die aneinandergereiht werden. Jeder vertikale Farbstrich repräsentiert dabei das Ergebnis („averageColor“ oder „dominantColor“) eines Frames.

Das Outputimage hat immer ein Verhältnis von 1:4, um eine einheitliche Darstellung der Ergebnisse zu gewährleisten. Zur Berechnung der Arraygröße wird zuerst die Anzahl der Bilder (Breite des Outputimage „width“) durch 4 geteilt und gerundet. Daraus ergibt sich die Höhe des Outputimage in Pixeln und wird in einer Variable „height“ persistiert. Anschließend werden Höhe und Breite multipliziert und das Array in dieser Größe initialisiert. Über eine Funktion wird dann je nach Auswahl des Benutzers die „dominantColor“ oder die „averageColor“ aus HBase je Bild ausgelesen. Die einzelnen Werte (R, G und B) werden über ein Offset in einer Integervariable abgespeichert und anschließend zu dem zuvor initialisierten Array hinzugefügt. Da über dieses Verfahren nur die erste Zeile des Bildes befüllt wird, müssen über eine Schleife auch noch die verbleibenden Zeilen derselben Spalte mit demselben Farbwert versehen werden. Im Folgenden erklärt ein Beispiel das beschriebene Verfahren:

- 8 Frames, die vom MapReduce analysiert wurden => 8 Spalten

- Anzahl der Zeilen = 8 / 4 (das Verhältnis) = 2 => 2 Zeilen
- Anzahl der Pixel = 8 Spalten * 2 Zeilen = 16 => Array mit 16 Werten
- Das Ergebnis des ersten Frames wird in dem Array unter dem Index 0 und 8 gespeichert, da diese beiden in dem Ergebnisimage eine Spalte ergeben.
- Ein Frame wird immer durch mehrere Pixel vertikal repräsentiert.

I	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
C	a	b	c	d	e	f	g	h	a	b	c	d	e	f	g	h

Abbildung 4: Array mit Pixeln

Aus dem Array wird mit Hilfe der Java AWT Funktion „.setImage()“ ein zuvor erstelltes BufferedImage mit Werten (Pixeln) gefüllt. Aus dem Array in Abbildung 4 ergibt sich dann folgendes Outputimage mit einer Größe von 8x2:

a	b	c	d	e	f	g	h
a	b	c	d	e	f	g	h

Abbildung 5: Image aus Pixelarray

Um die Bilder auch in einer gut erkennbaren Größe auf der Website präsentieren zu können, müssen diese bei Bedarf noch hochskaliert werden. Das bedeutet zum Beispiel, dass ein Bild mit Größe 8x2 (wie im Beispiel zuvor) auf eine Größe von 800x200 hochskaliert wird.
Hierfür wurde ebenfalls ein Algorithmus entwickelt, welcher das zuvor erstellte Array verarbeitet. Dazu werden die Pixel eines Frames einfach nur repliziert, sodass die Breite des Frames von 1 auf 100 erhöht wird. Entsprechend wird auch die Höhe um den Faktor 100 verändert.

4. Ausblick

Jedes Projekt hat weitere Ausbaumöglichkeiten, so auch in HUGO.
Die erste Idee ist, dem Anwender auf der Website eigens die Option zu geben, die Zeitintervalle zwischen extrahierten Frames auszuwählen.
Dies kann über eine Parameterweitergabe, ähnlich der Übergabe der Bucketgröße geschehen.

In der aktuellen Version von HUGO wird bereits geprüft, ob das Video zur angegebenen Bucketgröße bereits analysiert wurde. Allerdings ergibt sich daraus keine Konsequenz. Daher ist eine weitere Idee, dass abhängig von der Antwort ein dezidierter Prozess angestoßen wird: Ist das Video bereits in der angebenden Bucketgröße analysiert worden, so wird das Bild nur noch aus HBase ausgesucht und auf der Website angezeigt, wobei der gesamte Analyseprozess umgegangen wird. Dieser wird gestartet, wenn das Video noch nicht in der angegebenen Bucketgröße ausgewertet wurde.

Ferner ist die Implementierung einer Abstraktionsschicht mit Pig und Hive möglich. Pig würde genutzt, um die MapReduceabfrage via Pig Latin zu wrappen. Hive früg via HiveQL die Daten im HDFS ab und könnte ebenso die Daten Queries in MapReduceJobs übersetzten. Ferner könnten die MapReduce Jobs in Spark umgesetzt werden, um somit die Verarbeitungsgeschwindigkeit zu erhöhen.

Eine entferntere Vision ist die Ermittlung von durchschnittlichen und dominanten Tönen in Videos. Die Idee kam auf, da der Ablauf ähnlich wäre: Über einen Videolink wird das Video gespeichert und ein Wert pro einem zu definierenden Zeitintervall ausgewertet. Daraus kann dann der durchschnittliche oder dominante Wert, im Falle von Tönen die Frequenzen, ermittelt werden. Präsentiert wird dann das Ergebnis auch über eine Website.
Eine gänzlich neue Implementierung erforderte die Zerlegung des Videos in kleine Teile, aus denen dann der Ton extrahiert würde. Ebenso müsste eine Methode zur Einteilung eines bestimmten Tons bzw. zur Ermittlung des Durchschnittstons erarbeitet werden, wobei vom Prinzip her die MapReduce Jobs gleich belieben.

5. Fazit

Durch das Projekt HUGO konnte nicht nur ein theoretischer, sondern auch ein praktischer Einblick das Thema Big Data Engineering durch alle Teammitglieder gewonnen werden. Nach anfänglichen Schwierigkeiten sich die MapReduceverfahrensweise reinzudenken, konnten das Team auch aus seiner Sicht komplexere Problematiken lösen. Auch die stückweise

Erarbeitung einzelner Prozesse bis hin zu der Makropipeline konnten im Team zügig durchgeführt werden.
Eine Schwierigkeit stellte die Arbeit mit der VM Cloudera Quickstart dar. Nicht nur im Hinblick auf Performanceaspekte, sondern vor allem im Bereich Robustheit und Verlässlichkeit ist bei dieser definitiv Verbesserungspotential vorhanden.
Aus Zeitmangel wurde an einigen Stellen auf Performanceverbesserungen des Codes verzichtet, die, neben den im Ausblick bereits erwähnten Punkten, bei einer neuen Version umgesetzt werden können.

6. Literaturverzeichnis

[1] Statista (Juli 2015): Tubefilter. Durchschnittlicher Upload von Videomaterial bei YouTube pro Minute von in ausgewählten Monaten von Mai 2008 bis Dezember 2015 (in Stunden). http://de.statista.com/statistik/daten/studie/207321/umfrage/upload-von-videomaterial-bei-youtube-pro-minute-zeitreihe/. Abgerufen am 24.02.2016.

[2] Lublinsky, B, Smith, KT, Yakubovich, A (2013): Professional Hadoop solutions. Wrox Wiley, Indianapolis, Ind.

7. Bilderverzeichnis

(1) https://upload.wikimedia.org/wikipedia/commons/thumb/4/41/Commons-logo-en.svg/608px-Commons-logo-en.svg.png, Abgerufen am 24.02.2016

(2) http://www.xuggle.com/public/images/xuggle-logo.png, Abgerufen am 24.02.2016