HUGO

Highly Useful Graphical Output

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

Gruppenmitlieder:

Ben Lohrengel Matr. Nr.:

André Meier Matr. Nr.:

Prüfer

Erstprüfer: Lars George

Zweitprüfer:

Abgabedatum: 25.02.2016

Inhaltsverzeichnis

[Einleitung 3](#_Toc443987422)

[Problemstellung 3](#_Toc443987423)

[Zielsetzung 4](#_Toc443987424)

[Input 4](#_Toc443987425)

[User Interface 5](#_Toc443987426)

[Output 5](#_Toc443987427)

[Verwendete Tools 6](#_Toc443987428)

[Xuggler 6](#_Toc443987429)

[Hadoop File System (HDFS) 7](#_Toc443987430)

[HBase 7](#_Toc443987431)

[Map Reduce 7](#_Toc443987432)

[Oozie 8](#_Toc443987433)

[Implementierung 8](#_Toc443987434)

[Prozessübersicht 8](#_Toc443987435)

[User Interface 9](#_Toc443987436)

[Prozessphasen 9](#_Toc443987437)

[Einlesen und Zerlegen der Videostreams 9](#_Toc443987438)

[Analyse der Frames 10](#_Toc443987439)

[Ergebnispräsentation 12](#_Toc443987440)

[Ergebnisse 12](#_Toc443987441)

[Visionen 12](#_Toc443987442)

[Fazit 12](#_Toc443987443)

Abbildung 1 Logo Wikimedia Commons 4

Abbildung 2 Xuggle Logo 6

Abbildung 3 Prozessübersicht des HUGO-Projektes 8

# Einleitung

## Problemstellung

Das Internet enthält heute eine unvorstellbare Menge an Videos. Im Dezember 2014 wurden alleine auf der Videoplattform YouTube, pro Minute 300 Minuten Material hochgeladen. All diese Videos sind einmalig und identifizieren sich aus einer bestimmten Kombination von Pixeln. Die darin enthaltenen Farbwerte bzw. RGB-Werte machen den gesamten Videostream unverwechselbar.

Es ergibt sich nun die Fragestellung inwieweit das Bildmaterial der Videos auf seine Farbwerte 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 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 Viedostream herunter laden und ihn 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.

## Zielsetzung

Ziel dieses Projektes namens “Highly Useful Graphical Output” kurz HUGO, ist es nun, die zuvor beschriebene Vorgehensweise zu realisieren. Im Folgenden werden der erwartet Input sowie Output und das User Interface näher spezifiziert.

### 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 Wikipedias 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.

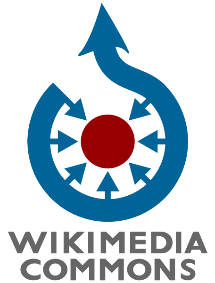


Abbildung Logo Wikimedia Commons

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 verwendet.

### User Interface

Als zentrale Anlaufstelle zur Interaktion mit dem HUGO-Projekt, dient eine einzige Webseite. Hier soll zunächst eine Eingabemaske zu sehen sein. Der zuvor kopierte Link zum Videostream ist dort einzufügen. Im nächsten Schritt ist der Nutzer dazu angehalten anzugeben welches Ergebnis er haben möchte. Es kann entweder die dominierenden Farbe oder die Durchschnittsfarbe ermittelt werden. Nach der Betätigung eines Buttons zur Bestätigung, wird das entsprechende Video im Hintergrund vom System heruntergeladen und lokal gespeichert. Anschließend soll die interne Verarbeitung ebenfalls unsichtbar durchgeführt werden. Für den Nutzer ist auf der Webseite ein Lade- bzw. Fortschrittsbalken zu visualisieren, anhand dieser den Prozessfortschritt erkennet. Zudem kann die Beendigung einzelner Prozessabschnitte, wie bspw. der erfolgreiche Download durch Ausgabe eines Textes dem Nutzer erscheinen. Zum Abschluss der Verarbeitungskette sind die Ergebnisse in Form einer Grafik wieder auf der Webseite angezeigt.

### Output

In den vorherigen Abschnitten erfolgte die Definition des erwarteten Inputs, der zukünftig verwendeten Quelle sowie der graphischen Oberfläche. In diesem Abschnitt soll es nun kurz um eine Beschreibung des erwarteten/möglichen Ergebnisses bzw. Outputs gehen.

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

1. 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. Zuletzt ist dann die so entstandene Grafik auf der Weboberfläche auszugeben. Der Nutzer erhält damit das von ihm gewünschte Ergebnis und die Arbeit des HUGO-Projektes ist beendet.

* Webseite
* Video
* Blocksize
* Analyseverfahren
  + Durchschnittsfarbe
  + Dominante Farbe

# Verwendete Tools

## 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. Dem User wird dann die Startseite präsentiert, auf welcher er einen Video Downloadlink, eine sogenannte Blocksize und das Analyseverfahren („averageColor“, „dominantColor“) angeben werden angeben kann. Über den Button Analyse wird der ganze Prozess dann gestartet, indem ein Python Script aufgerufen wird.

Dieses nimmt die Benutzereingaben entgegen und speichert diese in einer „data.txt“ Datei, welche anschließend in das HDFS gelegt 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 einem dann die ID des gestarteten Workflows zurück.

Das Python Script übernimmt neben den REST Calls auch die Aufgabe eine Webseite zu erstellen. Die zeigt dem User den Fortschritt des Workflows in einem Fortschrittsbalken an. Außerdem wird der aktuelle Job der gerade ausgeführt wird 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.

Sobald der Workflow Abgeschlossen ist, wird die Schleife abgebrochen. Das Python Script holt sich dann die „data.txt“ aus dem HDFS zurück, welche jetzt den Pfad zu dem Outputimage enthält. Über „hadoopy.get(imagepath, outputpath)“ wird das Outputimage in das lokale Dateisystem geladen um es anschließend in die Website einzubinden und damit dem User abschließend präsentieren.

In den folgenden zwei Unterkapitel werde die genutzen Technologien und Tools beschrieben, die für das User Interface/Website genutzt wurden.

### Apache Webserver

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

### Phyton

Als Scriptsprache für die Website wurde Python genutzt, weil sie einfach zu verstehen und eine gute Untersützung für REST POST und GET für Oozie bietet, dessen vorgehen auch auf der Oozie Website dokumentiert ist. Für eine einfache Nutzung von POST requests wurde zusätzlich noch „python-requests“[[1]](#footnote-1) installiert. Um mit dem HDFS zu arbeiten wurde „hadoopy“[[2]](#footnote-2) installiert.

## Xuggler

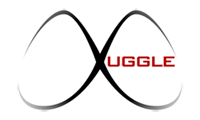
Xuggler definiert sich selber als einfachen Weg Videostreams zu dekomprimieren, zu manipulieren und wieder zu komprimieren. Es handelt sich dabei um eine Java-Bibliothek und ist unter der GPL Version 3 Lizenz frei verfügbar. Verwendet wird die leistungsstarke FFmpeg media handling library.

Abbildung Xuggle Logo

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

## Hadoop File System (HDFS)

* Für Videodateien und Frames

## 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, stellt kein Hindernis dar, da ausschließlich Strings und Integerwerte 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 Datenmengen 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.

## Map Reduce

Im Rahmen des Projekts HUGO erfolgt die Farbanalyse der einzelnen Frames durch MapReduce. Die Entscheidung liegt darin, dass MapReduce als Hadoops Hauptverarbeitungsengine hervorragend die Integration gespeicherter Daten in die Verarbeitung ermöglicht. Die Aufteilung in die verschiedenen Phasen zeigen die Auslegung auf paralelle Datenverarbeitung von großen Datenmengen. Das Aufteilen von Videos in Frames, die Speicherung dieser sowie die Analyse jedes einzelnen Frames 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.

## OutputGenerator

Abschließend wurde ein sogennanter Outputgenerator erstellt. Hier bei handelt es sich um einen Algorithmus der aus den Ergebnisdaten des MapReduce ein Bild erstellt. Die Ergebnisse des MapReduce werden wie zuvor beschrieben in HBase abgelegt.

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

Das Outputimage hat immer ein Verhältnis von 1 zu 4 um eine einheitliche Darstellung der Ergebnisse zu gewährleisten. Um die Arraygröße zu berechnen wird zu erst die Anzahl der Bilder (breite des Outputimage „width“) durch 4 geteilt und gerundet. Das ergibt die Höhe des Outputimage in Pixeln und wird in einer Variable „height“ persistiert. Anschließend wird die Höhe mit der Breite Multipliziert und der Array in dieser Größe initialisiert. Über eine Funktionen werden 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 einen Offset in einer Integer Variable abgespeichert und anschließend zu dem zuvor initalisierten Array hinzugefügt. Da über dieses Verfahren nur die erste Zeile des Bildes befüllt wird, müssen über eine Schleife auch noch die verbleibenden Zeilen der selben Spalte mit dem selben Farbwert befüllt werden. Im folgenden ein Beispiel, wie der Array befüllt wird:

* 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 vom ersten Frame 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

Zuordnung der Arraywerte zu den Outputimage Pixeln

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Frame 1 | Frame 2 | Frame 3 | Frame 4 | Frame 5 | Frame 6 | Frame 7 | Frame 8 |
| Pixel 0,0  ArrayIndex = 0 | Pixel 0,1  1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Pixel 1,0  8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |

Um die Bilder auch in einer gewissen Größe auf der Website präsentieren zu können, müssen diese bei Bedarf noch hochskaliert werden. Das bedeutet z.B., dass ein Bild mit Größe 8x2 (wie das Beispiel zuvor), auf eine Größe von 800x200 hochskaliert wird. Hierfür wurde ebenfalls ein Algorithmus entwickelt, welcher den zuvor erstellten Array verarbeitet. Dazu werden die Pixel eines Frames einfach nur repliziert, dass sie statt einem Pixel breit, 100 pixel bereit werden. Dementsprechend werden sie auch auch in der höhe 100x repliziert.

## Oozie

Bei Oozie handelt es sich um ein Workflowverwaltungssystem, welches für dieses Projekt verwendet wurde. 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 entwickelt.

Der Implemtierungsverlauf Orientiert sich an Vorlesungsinhalten. Dazu wurde die notwendigen Dateien (job.properties, workflow.xml, librarys) 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 Testdurchläufen, wurde das Start des Workflows auf die Website ausgelagert. Dieses Vorgehen wird in Kapitel <ToDo>user Interface / Website erläutert.

Der Workflow wird durch einen REST Call von der Webseite intalisisiert. 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 sobald er fertig seine Ergebnisse zurück, die für den nächsten Workflow Step als Input notwendig sind. Dadurch ist eine Datenübergabe zwischen den Workflow Steps und die Zwischenspeicherung der Inputparamter gewährleistet. Der erste Workflow Step ist die Überprüfung, ob das gesuchte Video bereits analysiert wurde, indem der Key in HBase abgefragt wurde. Wenn das Video noch nicht runtergeladen, wird dieses in das HDFS geladen und anschließend mit Xuggle in seine Frames zerlegt. Die Frames werden dann vom MapReduce Job analysiert und die Ergebnisse in der HBase Datenbank persitiert. Der Outpugenerator verarbeitet die Daten aus dem HBase zu diesem Video dann zu einem Outputimage und speichert dieses im HDFS und schreibt den Dateipfad zurück in die data.txt. Die Website holt sich die „data.txt“ zurück ins Lokale Dateisystem und bezieht anschließend auch das Outputimage, welches dann auf der Website angezeigt wird.

# Implementierung

In den vorherigen Kapiteln haben wir zunächst die Idee des HUGO-Projektes zusammen mit dem gelieferten Input sowie den zu erwartenden Output kennengelernt. Darauf folgte eine Einführung in die Tools die im Rahmen dieses Projektes zum Einsatz kommen werden.

Im Folgenden Kapitel soll es nun um die konkrete Implementierung von Hugo gehen. Zunächst wird dabei eine Prozessübersicht gezeigt. Mit ihr sollen vorab die Zusammenhänge zwischen den einzelnen Modulen bzw. Prozessschritten visuell dargestellt werden. Darauf erfolgt dann eine detaillierte Beschreibung der einzelnen Prozessphasen. Beginnend mit dem einlesen des Videostreams, folgt das Zerlegen der Videostreams sowie die Analyse der so ermittelten Frames und schließt mit der Ergebnispräsentation.

## Prozessübersicht

Abbildung 3 zeigt die Prozessübersicht des gesamten HUGO-Projekts. Es verdeutlicht die Zusammenhänge zwischen den im Anschluss beschriebenen einzelnen Phasen.

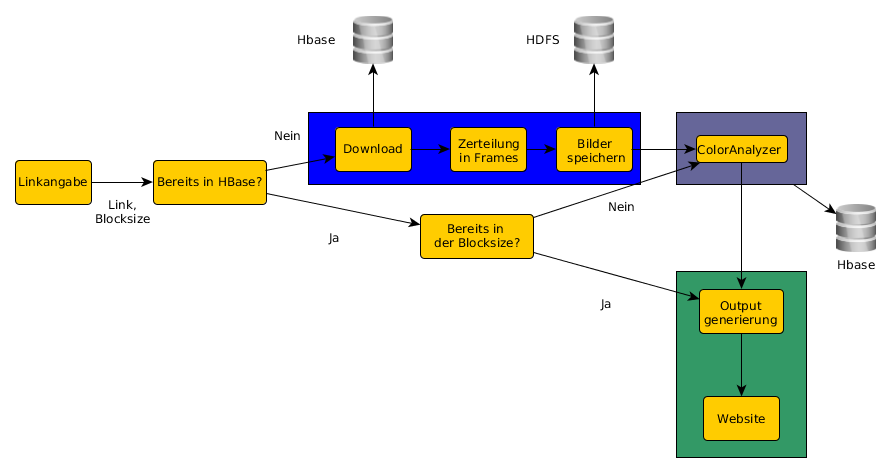


Abbildung Prozessübersicht des HUGO-Projektes

## User Interface

## Prozessphasen

### Einlesen und Zerlegen der Videostreams

Wenn der auf der Website angegebene Link nicht als Key in der HBase Datenbank zu finden ist, wird der Download gestartet. Dies geschieht über einen FSDataOutputStream, der das Video in das festgelegte Verzeichnis "hugo/Videos/filesName" schreibt, wobei FilesNames aus dem Hyperlink extrahiert wird.  
  
<TODO: Welches Ablageformat der Videos?> <@André: TODO Offset beschreibung>  
  
Um das Video in Frames zu zerlegen, wird das Framework Xuggler genutzt. Auf das Framework wird in Kapitel <TBD> eingegangen.  
Der Verweis auf das Video erfolgt als Inputargument in der Mainmethode. Das Video wird temporär zwischengespeichert, um das Extrahieren aus der Offsetdatei zu vereinfachen.  
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 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 die Komprimierung verlustfrei erfolgt und durch Xuggler problemfrei umgesetzt 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.

### Analyse der Frames

#### MapReduce Jobs

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 Blockgröße 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, das sie unbrauchbar werden. Aus diesem Grund 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. Bei MR1 wird ebenso die Combinerklasse festgelegt.

Im Folgenden wird auf die einzelnen MapReduceJobs eingegangen.

#### Ermittlung des durchschnittlichen Werts – MR1

Rechenschema am Beispiel

Als Beispiel dient in folgenden das erste Bild des 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 Rot Wert, da Blau und Gelb beide 0 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

Mapper

Der Mapper erhält als Parameter die angegebene Blockgröße 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 Codetyp (R, G oder B), dem Wert (je nach Blockgröße 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>.

Combiner

Der Combiner aggregiert die Häufigkeit eines "gesehenen" Farbcodes, sortiert nach Grundfarbe. Das heißt, er zählt 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>.

Reducer

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, der Grundfarbe 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“ wird für das erste Bild automatisch angehangen.

#### Ermittlung der dominanten Farbe – MR2

Mapper

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.

Reducer

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>.

### Ergebnispräsentation

* Prüfen
* Ablaufmodell
* Download
* Zerlegen
* MapReduce Job

Out Image generieren und abspeichern

# Ergebnisse

# Visionen

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 Blockgröße geschehen.

In der aktuellen Version von HUGO wird bereits geprüft, ob das Video zur angegebenen Blockgröße bereits analyisiert wurde. Eine weitere Idee ist, dass abhängig von der Antwort ein dezidierter Prozess angestoßen wird: Ist das Video bereits in der angebenen Blockgröße analysiert worden, so wird das Bild nur noch aus HBase ausgesucht und angezeigt, wobei der gesamte Analyseprozess umgegangen wird. Dieser wird gestartet, wenn das Video noch nicht in der angegebenen Blockgröße 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 MapReduce jobs übersetzten.

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 MapReducejobs gleich beblieben.

# Fazit

<http://de.statista.com/statistik/daten/studie/207321/umfrage/upload-von-videomaterial-bei-youtube-pro-minute-zeitreihe/>

Glossar RGB

1. http://docs.python-requests.org/en/master/ [↑](#footnote-ref-1)
2. https://github.com/bwhite/hadoopy.git [↑](#footnote-ref-2)