Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Ditmar Lange

Bewertung von Graphdatenbanksystemen durch den Entwurf und die Implementierung eines Filmempfehlungssystems

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Ditmar Lange

Bewertung von Graphdatenbanksystemen durch den Entwurf und die Implementierung eines Filmempfehlungssystems

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Olaf Zukunft
Zweitgutachter: Prof. Stefan Sarstedt

Eingereicht am: 14. November 2018

Ditmar Lange

Thema der Arbeit

Bewertung von Graphdatenbanksystemen durch den Entwurf und die Implementierung eines Filmempfehlungssystems

Stichworte

Anfragesprache, ArangoDB, Empfehlungssystem, Graphdatenbank, Inhaltsbasiert, Kante, Knoten, Kollaborativ, Lasttest, Neo4j, OrientDB, Pattern-Matching

Kurzzusammenfassung

Diese Bachelorarbeit vergleicht die Graphdatenbankmanagementsysteme (DBMS) Neo4j, OrientDB und ArangoDB anhand eines selbstentworfenen Filmempfehlungssystems, das die Konzepte der inhaltsbasierten und kollaborativen Empfehlung umsetzt. Dazu werden Daten von MovieLens benutzt, die Filme und deren Genres, Themen, und Bewertungen beinhalten. Das Ergebnis dieses Vergleichs ist, dass Neo4j in den wichtigsten Kategorien die besten Resultate erzielt hat, ArangoDB beim Datenimport, Speicherbedarf und bei der Benutzerkapazität überzeugte, aber ansonsten die schlechtesten Berechnungszeiten hatte, und OrientDB immer schlechter als Neo4j war.

Ditmar Lange

Title of the paper

Evaluation of Graphdatabases by designing and implementing a Movie Recommendation Engine

Keywords

ArangoDB, Colaborative, Content-based, Edge, Graphdatabase, Load testing, Neo4j, OrientDB, Pattern-Matching, Query-Language, Recommendation-Engine, Vertex

Abstract

This thesis compares the graph-database management systems (DBMS) Neo4j, OrientDB, and ArangoDB, based on a movie recommendation engine that uses both concepts of content-based and collaborative filtering. To do this, the data was taken from MovieLens. This dataset includes the movies and their genres, themes and ratings. The result of this comparison is that Neo4j showed the best results in the most relevant categories. ArangoDB showed the biggest user capacity, needed the least storage space and had the fastest import speed, but otherwise needed the longest time for the calculations. Neo4j always outperformed OrientDB.

Inhaltsverzeichnis

1. Einleitung **1**
- 1.1. Motivation . . . 1
- 1.2. Ziel . . . 2
- 1.3. Aufbau . . . 2

2. Grundlagen **4**
- 2.1. Graphen . . . 4
  - 2.1.1. Definition . . . 4
  - 2.1.2. Gerichteter Graph . . . 4
  - 2.1.3. Property-Graph . . . 5
  - 2.1.4. Vollständiger Graph . . . 5
  - 2.1.5. Teilgraph . . . 5
- 2.2. Graphdatenbanksystem . . . 6
  - 2.2.1. Neo4j . . . 6
  - 2.2.2. OrientDB . . . 8
  - 2.2.3. ArangoDB . . . 9
- 2.3. Empfehlungssystem . . . 10
  - 2.3.1. Kollaborativ . . . 10
  - 2.3.2. Inhaltsbasiert . . . 11
  - 2.3.3. Hybrid . . . 12

3. Filmempfehlungssystem **14**
- 3.1. Anforderungen . . . 14
- 3.2. Architektur . . . 14
  - 3.2.1. Graphdatenbanksysteme . . . 15
  - 3.2.2. Algorithmen . . . 15
  - 3.2.3. Daten . . . 20
- 3.3. Tests . . . 23
  - 3.3.1. Datenimport . . . 23
  - 3.3.2. Empfehlungsalgorithmen . . . 23

4. Implementierung **24**
- 4.1. Neo4j . . . 24
  - 4.1.1. Datenimport . . . 24
  - 4.1.2. Empfehlungsalgorithmen . . . 26

4.2. OrientDB 27
4.2.1. Datenimport 27
4.2.2. Empfehlungsalgorithmen 29
4.3. ArangoDB 31
4.3.1. Datenimport 31
4.3.2. Empfehlungsalgorithmen 31

5. Vergleich der Graphdatenbanksysteme **33**
5.1. Aufbau und Durchführung 33
5.2. Ergebnisse 34
5.2.1. Speicherbedarf 34
5.2.2. Zeitmessungen 34
5.2.3. Lasttests 38
5.3. Auswertung 39
5.3.1. Objektive Bewertung 39
5.3.2. Subjektive Bewertung 41

6. Fazit **43**

A. Anhang **45**

Abbildungsverzeichnis **56**

Tabellenverzeichnis **57**

Listings **58**

Literaturverzeichnis **59**

1. Einleitung

Dieses Kapitel dient als Einführung in diese Bachelorarbeit. Hier wird die Motivation, die zur Auswahl dieses Themas geführt hat, beschrieben, sowie das Ziel dieser Arbeit und dessen Aufbau.

1.1. Motivation

Die große Aufstieg von Video-On-Demand Plattformen wie Netflix oder Amazon Prime haben dazu geführt, dass den Benutzern eine enorm große, und daher auch unübersichtliche, Anzahl von Filmen jederzeit zur Verfügung steht. Hier ist es wichtig, dass den Benutzern Filme vorgeschlagen werden, damit sie schnell etwas finden, das ihnen gefällt. Dies ist nicht nur für den Benutzer ein Vorteil, sondern auch für die Filmplattform, denn sollte der Kunde des Öfteren nichts finden, oder sogar Filme sehen die ihm nicht gefallen, könnte er daraufhin als unzufriedener Kunde sein Abonnement kündigen. Schon im Oktober 2013 war es so weit, dass 35% von Amazon's Einnahmen aus Produkten entstanden, die Benutzern mittels eines Empfehlungssystems vorgeschlagen wurden. Auch 75% der auf Netflix gestreamten Videos wurden dem Benutzer vorgeschlagen (vgl. MacKenzie u. a. (2013)).

Da die meisten dieser Plattformen mehrere Millionen Benutzer haben, die mehrere Bewertungen abgeben (z.B. Netflix hat 117.600.000 Benutzer (Stand: Januar 2018, vgl. Molla (2018)), fällt die Analyse dieser Daten zweifelsfrei in den Bereich Big-Data. Für Filmempfehlungen, oder Empfehlungen generell, sind Graphdatenbanksysteme am besten geeignet, da Kunden und deren Vorlieben besonders gut als Graphen dargestellt und gespeichert werden können. Graphdatenbanksysteme erlauben dies und ersparen dem Entwickler des Empfehlungssystems die vielen JOIN-Operationen, die bei einer relationalen Datenbank nötig wären, um Beziehungen wie in einem Graphen zu speichern.

Das bekannteste Beispiel im Bereich der BigData-Graph-Datenbankmanagementsysteme (DBMS) ist Neo4j (vgl. Neo4j (2018)). Dabei gibt es aber auch andere Graph-DBMS. Um zu erfahren ob Neo4j wirklich die beste Graph-DBMS ist, oder nur einen guten Ruf genießt

während andere Graph-DBMS gleichwertig oder sogar besser sind, sollten Neo4j und zwei weitere Graph-DBMS verglichen werden.

OrientDB (vgl. Orientdb (2018b)) und ArangoDB (vgl. ArangoDB (2018)) wurden zum Vergleich ausgewählt, da sie laut Predictive-Analytics-Today (2018) im Trend sind, und sich beide auf ihren Websites auch direkt mit Neo4j vergleichen.

1.2. Ziel

Ziel dieser Bachelorarbeit ist es, die drei Graph-DBMS (Neo4j, OrientDB, und ArangoDB) anhand eines einfachen (aber gut funktionierenden) Filmempfehlungssystems neutral zu vergleichen, um anhand dieses Vergleiches feststellen zu können, welche dieser Graph-DBMS am besten für eine solche Aufgabe geeignet ist. Dazu sollen gemessen werden, wie lang der Datenimport bei verschiedenen Datengrößen dauert und die Bearbeitungsdauer der Algorithmen und der Speicherplatzbedarf der Graph-DBMS. Auch Lasttests sollen durchgeführt werden. Die Benutzerfreundlichkeit ist ebenfalls ein Aspekt, der mit dieser Arbeit verglichen werden soll.

Es gibt auch Aspekte von Graph-DBMS, die von einem Filmempfehlungssystem überhaupt nicht in Anspruch genommen werden, z.B. das Suchen des kürzesten Pfades. Der Vergleich dieser Eigenschaften entfällt dadurch in dieser Arbeit. Nur die Eigenschaften, die einem Filmempfehlungssystem dienen, sollen in dieser Arbeit verglichen werden.

1.3. Aufbau

Nachdem dieses Kapitel die Motivation und das Ziel dieser Bachelorarbeit erläutert hat, liefert Kapitel 2 die Grundlagen: Es wird das generelle Konzept eines Graphen erläutert, eine Einführung in die Graphdatenbanksysteme gegeben sowie die Funkionsweise von Empfehlungssystemen erklärt.

In Kapitel 3 wird die Architektur des Filmempfehlungssystems vorgestellt, das zum Vergleich der DBMS verwendet werden soll. Dies beinhaltet die Anforderungen an das System, den Aufbau der Testumgebung sowie den Ablauf der Algorithmen. Daraufhin wird auch der Datensatz beschrieben, mit dem das System arbeitet.

In Kapitel 4 werden die Implementierung des Datenimports und der Empfehlungsalgorithmen für die drei Graph-DBMS vorgestellt.

In Kapitel 5 werden die Graph-DBMS dann verglichen. Hier werden mehrere Zeitmessungen für den Datenimport und die Empfehlungsalgorithmen vorgenommen sowie der Speicherbedarf der Graphdatenbanksysteme verglichen und Lasttests (Kapazitätentests sowie Stresstests) durchgeführt. Anhand dieser gesammelten Messungen werden Schlussfolgerungen gezogen, die die relative Eignung der drei DBMS für ein Filmempfehlungssystem feststellen. Auch die subjektiven Aspekte der Graph-DBMS (Bewertungen der Benutzungsoberfläche, Anfragesprache, usw.) werden in diesem Kapitel verglichen.

Diese Arbeit endet mit Kapitel 6, in dem das Fazit den Inhalt dieser Bachelorarbeit zusammenfasst und die Stärken und Schwächen der drei Graph-DBMS abschließend erläutert werden.

2. Grundlagen

In diesem Kapitel werden einige Begriffe und Konzepte erklärt, die für das Verständnis dieser Arbeit von großer Wichtigkeit sind. Vor allem die Konzepte von Graphen, Graphdatenbanksystemen und Empfehlungssystemen werden hier beschrieben. Der Abschnitt über Graphen berührt nur die oberflächlichen Eigenschaften eines Graphen, da sich die detaillierten Aspekte, wie z.B. Traversierung oder Speichermethode bei den verschiedenen Graph-DBMS unterscheiden, und somit erst in der Vorstellung der drei Graphdatenbanksysteme beschrieben werden. Daraufhin folgt eine Einführung in die Funktionsweise von Empfehlungssysteme, mit einem besonderen Fokus auf den drei wichtigsten Empfehlungsarten: kollaborativ, inhaltsbasiert und hybrid.

2.1. Graphen

Graphen sind in der Mathematik weit verbreitet. Es gibt sogar ein ganzes Themengebiet, die Graphentheorie, dass sich nur mit Graphen befasst, und viele, die sich auf diesem Gebiet spezialisiert haben, wie z.B. Schneeweiss (1985) und Diestel (2010). Die folgenden Abschnitte erläutern das Grundprinzip der Graphen, sowie einige für diese Arbeit relevanten Graphtypen.

2.1.1. Definition

Ein Graph besteht aus Ecken (oder Knoten) und Kanten.

Formell: $G = (V, E)$.

V ist hier die Menge der Knoten, und E die Menge der Kanten. Laut Schneeweiss (1985) (S. 2) können Knoten kantenfrei sein, aber jede Kante "verläuft" zwischen zwei Knoten.

2.1.2. Gerichteter Graph

Graphen können gerichtet oder ungerichtet sein. "Während gerichtete Kanten eine einseitige Beziehung von einem Knoten zu einem anderen darstellen, beschreibt eine ungerichtete Kante

eine beidseitige Beziehung zwischen zwei Knoten." (Edlich u. a. (2010), S. 172). Ein Beispiel für eine gerichtete Kante wäre die 'hat gekauft' Beziehung, bei der ein Kunde ein Produkt gekauft hat, das Produkt aber nicht den Kunden gekauft hat. Eine ungerichtete Kante wäre z.B. 'Freundschaft', bei der beide betroffene Knoten/Personen den anderen als ihren Freund sehen.

Diestel (2010) beschreibt dies auf Seite 30 formell: "Ein gerichteter Graph ist ein Paar (V, E) disjunkter Mengen (von Ecken und Kanten) zusammen mit zwei Funktionen $init : E \to V$ und $ter : E \to V$, die jeder Kante e eine Anfangsecke $init(e)$ und eine Endecke $ter(e)$ zuordnen; die Kante e heißt dann von $init(e)$ nach $ter(e)$ gerichtet. Man beachte, dass ein gerichteter Graph zwischen zwei Ecken x, y durchaus mehrere Kanten haben kann. Solche Kanten nennt man Mehrfachkanten; haben zwei Mehrfachkanten die gleiche Richtung (etwa beide von x nach y), so sind sie parallel. Ist $init(e) = ter(e)$, so ist e eine Schlinge."

2.1.3. Property-Graph

Die Knoten und Kanten können auch Eigenschaften ("Properties") besitzen. In dem Fall spricht man von einem "Property-Graph". So kann z.B. der Knoten, der eine Person darstellen soll, die Eigenschaften der Person speichern (Name, Alter, usw.), und die 'hat gekauft'-Kante z.B. den Preis, der bezahlt wurde, als 'Property' besitzen.

2.1.4. Vollständiger Graph

Sollten alle Knoten miteinander verbunden sein, d.h. von jedem Knoten geht eine Kante zu jedem anderen Knoten, so ist dies ein vollständiger Graph. "Einen vollständigen Graphen auf n Ecken bezeichnen wir als K^n; ein K^3 ist ein Dreieck" (Diestel (2010), S.3)

2.1.5. Teilgraph

Ein Teilgraph, auch Subgraph oder Untergraph genannt, ist ein Ausschnitt eines größeren Graphen (Supergraph oder Obergraph). Alle Knoten und Kanten des Teilgraphen müssen im Supergraphen enthalten sein, es können jedoch beliebig viele Knoten und Kanten des Supergraphen fehlen.

Formell: Gilt $V' \subseteq V$ und $E' \subseteq E$, so ist G' ein Teilgraph von G (und G ein Obergraph von G'), geschrieben $G' \subseteq G$. (vgl. Diestel (2010), S.3).

2.2. Graphdatenbanksystem

Um die eben beschriebenen Graphen besonders effizient speichern und bearbeiten zu können, wurden Graphdatenbanksysteme entwickelt.

Graph-DBMS spezialisieren sich, im Gegensatz zu relationalen Datenbanken, auf vernetzte Informationen und deren möglichst effiziente Traversierung. Sie vermeiden vor allem teure Datenbankoperationen wie z.B. mehrere rekursiv verschachtelte JOINs (vgl. Edlich u. a. (2010) S. 170). Wichtige Aspekte von Graphdatenbanken sind die Art, in der die Knoten und Kanten gespeichert werden, die Art der Traversierung, und wie stark skalierbar sie sind. Hier unterscheiden sich die verschiedenen Graph-DBMS, weswegen dies für jede der drei Graphdatenbanksysteme gesondernd erklärt werden wird. Generell ist es jedoch so, dass Graph-DBMS zwar die komplexesten Daten der NoSQL-Big Data Datenbanken bearbeiten und speichern können, jedoch am schlechtesten skalierbar sind, weswegen sie weniger Daten speichern können als z.B. Dokumentenbasierte Datenbanken, Key-Value Datenbanken oder Spaltenbasierte Datenbanken. Die schlechte Skalierbarkeit liegt daran, dass sich die Graphdatenbanken nur schwer partitionieren lassen, da die Daten alle durch Kanten verbunden sind, und Kanten nicht ohne Start- oder Zielknoten bleiben können. Eine Partitionierung funktioniert eigentlich nur, falls der Graph mehrere komplett disjunkte Teilgraphen besitzt.

Typische Anwendungen für Graph-DBMS sind Systeme für Betrugserkennung, Fahr- und Flugplanoptimierung, Ausbreitungsvorhersagen, Verkehrsleitsysteme, Empfehlungssysteme und die Analyse Sozialer Netze (vgl. Edlich u. a. (2010), S. 170).

2.2.1. Neo4j

Neo4j (Neo4j (2018)) ist eines der ältesten Graphdatenbanksysteme (2007), und gleichzeitig auch eines der beliebtesten (vgl. Predictive-Analytics-Today (2018)). Die Beliebtheit liegt zum Teil an “der hohen Performance, des schmalen API sowohl für Java als auch viele anderen Sprachen und der kompakten Installation” (Edlich u. a. (2010), S. 184). Bei der ‘hohen Performance’ ist vor allem der Performance-Vorteil von Neo4j gegenüber anderen Graph-DBMS gemeint. Andere Datenbanksysteme sind zwar performanter als Neo4j, wenn auch der deutlich komplexere Implementierungsaufwand bei hoch vernetzten Daten in regulären Datenbanksystemen diesen Performance-Vorteil überwiegend wieder ausgleicht.

Speicherung

Laut Hunger (2014) (S. 19) nutzt Neo4j, anders als andere Graphdatenbanken, einen eigenen, optimierten Persistenzmechanismus für die Speicherung und Verwaltung der Graphdaten. Die Persistenzschicht, die mittels Java NIO (Datei-Memory-Mapping usw.) implementiert wurde, nutzt Blöcke fester Größe zum Abspeichern von Knoten und Verbindungsinformationen. Dadurch kann Neo4j die unteren Schichten optimal auf seine Bedürfnisse optimieren.

Traversierung

Die Traversierung bei Neo4j funktioniert im Prinzip wie eine Breitensuche, bei der aber bestimmte Kanten und Knoten nicht berücksichtigt werden. Bei einer Anfrage wie z.B. der auf einen Film basierenden kollaborativen Filmempfehlung (siehe Abbildung 3.2.) würde die Traversierung folgendermaßen aussehen:

1. Traversierung beginnt beim Startknoten (Film)
2. Alle eingehenden 'RATED'-Kanten werden gesammelt
3. Es werden alle Kanten, die kein Rating von 5.0 haben, fortan ignoriert.
4. Es werden alle Knoten (Benutzer), die am anderen Ende der Kanten sind, gesammelt.
5. Es werden alle ausgehenden 'RATED'-Kanten gesammelt.
6. Es werden alle Kanten, die kein Rating von 5.0 haben, fortan ignoriert.
7. Es werden alle Knoten (Filme), die am anderen Ende der Kanten sind, gesammelt, und als Ergebnis ausgegeben.

Skalierbarkeit

Neo4j ist, wie für Graphdatenbanken üblich, nicht partitionierbar. Replikation erfolgt nach dem Master-Slave Prinzip, bei dem, laut Junghanns (2014) (S.51), die Datenbasis im Sinne der vollständigen Replikation auf allen Rechnern im Cluster gespeichert ist. Schreibzugriffe werden ausgehend von einem Master-Rechner koordiniert und Änderungen werden an mehrere Slave-Rechner weitergegeben. Der Ausfall einzelner Rechner wird von der Architektur toleriert und eine horizontale Skalierbarkeit von Lesezugriffen wird gewährleistet.

Im Detail sieht die Ausführung eines Schreibzugriffs folgendermaßen aus:

1. Ein Schreibzugriff kann an jedem Rechner des Systems erfolgen. Bei einem Slave wird dies mit dem Master synchronisiert.

2. Der Master speichert diese Transaktion und informiert den Client über Erfolg oder Misserfolg. Somit ist die Änderung schon direkt auf zwei Rechnern umgesetzt worden.

3. Daraufhin werden die Slaves asynchron über den Schreibzugriff informiert. Das Eventual Consistency Prinzip wird hier verwendet, bei der fehlgeschlagene Nachrichten kein Problem verursachen, da die Datenbank sicherstellt, dass die Synchronisation später stattfindet.

Durch das Eventual Consistency Prinzip kann es dazu kommen, dass bei einem Lesezugriff die Informationen nicht auf dem aktuellsten Stand sind. Soll dies vermieden werden, müsste der Rechner eine "Pull"-Aktion durchführen, die aber die Antwortzeiten vergrößern würde.

Fällt ein Slave aus, wird er als inaktiv markiert und einfach bei Reaktivierung synchronisiert. Fällt aber der Master aus, wird nach dem Paxos-Protokoll (Demokratische Abstimmung aller Rechner) ein neuer Master gewählt. Hierbei kann es eventuell zum Datenverlust kommen, wenn der Ausfall noch vor der Synchronisation eines Zugriffs passiert.

2.2.2. OrientDB

OrientDB ist ein von Orient Technologies entwickeltes Datenbankmanagementsystem. "Die OrientDB ist ein Vertreter der Multimodel-Datenbanken, die Konzepte vieler NoSQL-Datenbanken vereint. Ihre Basis ist eine Dokumentendatenbank, welche aber in letzter Zeit durch graphorientierte Erweiterungen von sich reden gemacht hat." (Edlich u. a. (2011), S. 329) Insofern ist OrientDB kein reines Graphdatenbankmanagementsystem. Dennoch ist dies ein Bereich, in dem OrientDB vordringt, und den Vorreitern, wie Neo4j, Konkurrenz machen möchte. Auch OrientDB wurde, wie Neo4j, in Java implementiert.

Speicherung

Laut Junghanns (2014) (S. 62) wird für die Speicherung ein dokumentenorientiertes Speicherkonzept eingesetzt. Die Speicherung der Informationen erfolgt grundlegend in Form von Dokumenten und ein darauf aufbauendes graphenorientiertes Datenmodell ermöglicht die Definition von Beziehungen zwischen den Dokumenten.

Ein Dokument besteht aus Schlüssel-Wert Paaren. Schlüssel sind immer Strings, während die Werte einen beliebigen Typ haben können. OrientDB unterstützt eine besonders große Anzahl von Typen (mehr als die anderen hier untersuchten Graphdatenbanken). Selbst ein Dokument kann als Wert angegeben werden.

Die Informationen der Knoten im Graph (z.B. über ausgehende und eingehende Kanten) werden auch in Schlüssel-Wert-Paaren gespeichert. Laut Junghanns (2014) (S. 71) werden Knoten und attributierte Kanten auf Dokumente abgebildet, die innerhalb der Speicherschicht als Record bezeichnet werden. Ein Record ist genau einem Cluster zugeordnet. Ein Cluster ist die physische Gruppierung von Records anhand einer definierten Eigenschaft. Die Eigenschaft kann der Typ oder ein beliebiges Attribut des Dokumentes sein. Bei der Verwendung des Typsystems wird für jede Klasse automatisch ein Cluster erzeugt.

Traversierung

Ähnlich wie Neo4j benutzt OrientDB das Breitensuche-Verfahren bei der Graphtraversierung.

Skalierbarkeit

Das gesamte Prinzip der Skalierbarkeit ist bei OrientDB genau wie bei Neo4j entwickelt worden.

2.2.3. ArangoDB

ArangoDB ist das Graphdatenbanksystem der 2014 in Köln gegründeten Firma ArangoDB Inc., deren Hauptsitz nun in San Francisco liegt (vgl. ArangoDB (2018)). Ähnlich wie OrientDB, ist ArangoDB auch eine Kombination aus Graphdatenbank mit Dokumentenbasierter Datenbank. ArangoDB hat ein sehr flexibles Datenmodell, wodurch man es sogar als Key-Value Datenbank benutzen kann. Anders als Neo4j und OrientDB wurde ArangoDB nicht in Java, sondern in C und C++ implementiert (vgl. Kumar und Babu (2015), S. 29).

Speicherung

Laut Kaltenmaier (2016) (S. 67) ist die Datenhaltung grundsätzlich persistent, basiert jedoch auf der Verwaltung von Collections in Form von „memory mapped data files“. Diese werden gemäß der Speicherverwaltung des Hostbetriebssystems in den physikalischen Hauptspeicher ein- und ausgelagert.

Traversierung

Wie Neo4j, traversiert auch ArangoDB die Graphen mittels der Breitensuche.

Skalierbarkeit

Anders als die anderen beiden Datenbanken, ist bei ArangoDB keine Replikation möglich. Alle Daten werden auf einer Festplatte gespeichert.

2.3. Empfehlungssystem

Klahold (2009) (S.1) definiert das Empfehlungssystem als ein System, das einem Benutzer in einem gegebenen Kontext aus einer gegebenen Entitätsmenge aktiv eine Teilmenge „nützlicher" Elemente empfiehlt.

Das System hat dafür Zugriff auf eine Entitätsmenge und auf bestimmte Daten, die den Kontext, bzw. die Situation des Benutzers, darstellen. Anhand der Situation des Benutzers soll das System die möglichst am besten passenden Objekte aus der Entitätsmenge auswählen und sie anschließend dem Benutzer vorschlagen. Der Inhalt der Entitätsmenge variiert von System zu System: Es könnten z.B. Produkte (www.amazon.de), Fotos (photoree.com), Lieder (www.lastfm.de), etc. sein (vgl. Klahold (2009), S. 1).

Um die richtigen Empfehlungen berechnen zu können, muss das Empfehlungssystem nach bestimmten Verfahren gehen. Die drei typischen Verfahren sind 'Collaborative-Filtering' (kollaborativ), 'Content Based Filtering' (inhaltsbasiert), und der hybride Ansatz, der die beiden vorherigen Ansätze kombiniert (vgl. Klahold (2009), S. 2).

2.3.1. Kollaborativ

Beim kollaborativen Ansatz wird das Verhalten der Benutzer analysiert (vgl. Klahold (2009), S. 62). Anstatt die ähnlichsten Produkte zu suchen, sucht das System nach den sich am ähnlichsten verhaltenden Benutzerprofilen. Bei diesem Ansatz können aber drei wesentliche Probleme vorkommen:

Kaltstart-Problem Ein mögliches Problem des kollaborativen Ansatzes ist das 'Kaltstart-Problem'. Da das System nur auf dem früheren Verhalten der Benutzer basiert, kann es beim Start des Systems erhebliche Probleme durch den Mangel an einer relevanten Anzahl an

Benutzern geben. Das System braucht eine 'kritische Menge' an Daten und Bewertungsaktionen, um funktionieren zu können (vgl. Klahold (2009), S. 66).

Spärlichkeit Ein weiteres Problem des kollaborativen Ansatzes ist, dass bei einer großen Entitätsmenge (die meistens vorhanden ist), die meisten Benutzer über 98% der Elemente nicht bewertet haben, z.gr.T. sogar über 99,5% (vgl. Claypool u. a. (1999)). Das führt dazu, dass die Benutzerprofile große Unterschiede aufweisen, da sich die Bewertungen auf zu viele unterschiedliche Elemente verteilen, wodurch die Qualität der Empfehlungen dieses Ansatzes sehr leiden kann (vgl. Klahold (2009), S. 66).

Lemming-Effekt Neuere Elemente könnten bei einem kollaborativen Empfehlungssystem stark benachteiligt werden, da ältere Elemente eine größere Wahrscheinlichkeit haben (wegen der längeren Zeit, die sie verfügbar waren und den Benutzern schon vorgeschlagen werden konnten), mehrere Bewertungen bekommen zu haben, wodurch sie nun immer mehr Benutzern vorgeschlagen werden. Neue Elemente haben noch keine Bewertungen bekommen, und werden deshalb auch nicht vorgeschlagen. "Neue Objekte haben kaum eine Chance in die Liste der empfohlenen Empfehlungselemente aufzusteigen" (vgl. Klahold (2009), S. 67).

2.3.2. Inhaltsbasiert

Beim inhaltsbasierten Ansatz sind nur die Eigenschaften der Elemente wichtig. Das Verhalten der anderen Benutzer wird nicht in Betracht gezogen. Welche Eigenschaften eines Elements benutzt werden sollen, wird durch eine Eigenschaftsanalyse ermittelt. Laut Klahold (2009) (S.42) bestimmt die Eigenschaftsanalyse (auch als Feature Selection bezeichnet) die charakteristischen Eigenschaften von Empfehlungselementen in einer Weise, die einen algorithmischen Vergleich zwischen Empfehlungselementen ermöglicht. Die Laufzeitkomplexität wird reduziert indem die Eigenschaftsanalyse ein möglichst gutes Verhältnis zwischen der Menge der Eigenschaften und der dadurch gegebenen diskriminierenden Wirkung bezüglich der Empfehlungselemente erzielt.

Dieses Verfahren kann aber auch möglicherweise falsche Empfehlungen berechnen, wenn z.B. jemand eine Kettensäge gekauft hat und ihm daraufhin viele Kettensägen empfohlen werden, obwohl er ja nun keine mehr braucht. Auch ist z.B. das ähnlichste griechische Restaurant für den Kunden unbrauchbar, wenn es auf einem anderen Kontinent liegt. Außerdem würde ein Benutzer, der noch nie in einem griechischen Restaurant war, auch nie eines empfohlen bekommen, obwohl er es möglicherweise mögen würde. Ein kollaboratives System

könnte herausfinden, dass Benutzer, die in Hamburg im Steakhaus und in türkischen Restaurants waren, oft auch griechische Restaurants in Hamburg mögen. Solche Schlussfolgerungen bleiben beim inhaltsbasierten Ansatz aus. Hier würden dann nur Steakhäuser und türkische Restaurants empfohlen werden. Diese ganze Problematik nennt man 'Überspezialisierung' (Overspecialization) (vgl. Adomavicius und Tuzhilin (2005)).

2.3.3. Hybrid

Hybride Empfehlungssysteme kombinieren die beiden genannten Ansätze. Diese Kombination beider Ansätze fällt aber in jedem System etwas anders aus. Es können beide Ansätze vollständig implementiert und dann anhand einer Metrik die besten Empfehlung beider Systeme ausgewählt werden, oder einer der Ansätze hauptsächlich genommen werden, mit einem kleineren Anteil des anderen Ansatzes. Außerdem könnte ein hybrides System spezifisch für den Anwendungsfall entscheiden, welcher Ansatz sinnvoller ist (Adomavicius und Tuzhilin (2005)). In den folgenden Absätzen werden einige Möglichkeiten, beide Ansätze zu kombinieren, im Detail vorgestellt.

Beide Ansätze separat und ein "Best-Of" der Empfehlungen Bei dieser Option werden zwei unabhängige Empfehlungssysteme implementiert, die separat Empfehlungen vorschlagen (eines inhaltsbasiert, das andere kollaborativ). Eine vordefinierte Metrik (z.B. dem Ausgabe-Durchschnitt des Benutzers ähnlichster Preis oder einfach die besten drei Empfehlungen jedes Systems) würde entscheiden, welche der Empfehlungen dem Benutzer dann vorgeschlagen werden würden.

Ein hauptsächlicher Ansatz mit dem anderen Ansatz in geringer Form kombiniert Hier würde die Berechnung der Empfehlungen nach einem Ansatz gehen, in dem einige Elemente oder Benutzer zuerst vom anderen Ansatz gefiltert würden. Dies könnte z.B. bedeuten, dass zwar kollaborativ berechnet wird, aber nur unter ähnlichen Benutzern, oder inhaltsbasiert berechnet wird, aber nur mit Elementen, die wenigstens ein ähnlicher Benutzer auch positiv bewertet hat. Somit könnten z.B. die griechischen Restaurants, die auf einem anderen Kontinent liegen, ausgeschlossen werden.

Andere hybride Systeme Empfehlungssysteme können beide Ansätze auch in jeder weiteren Kombination benutzen. Ein System, das beide Ansätze beherrscht, könnte auch anhand des Anwendungfalles autonom entscheiden, welcher Ansatz sinnvoller ist (z.B. bei Produktkategorien, in denen konstant neue Produkte erscheinen, inhaltsbasiert; bei anderen kollaborativ,

um den Lemming-Effekt (siehe 2.3.1) zu umgehen) . Jedes Empfehlungssystem arbeitet unterschiedlich, aber die meisten verwenden ein hybrides System in irgendeiner Form.

3. Filmempfehlungssystem

Das Filmempfehlungssystem ist ein häufig genanntes Beispiel für eine “Recommendation Engine”, und auch eines, das von sehr vielen Menschen täglich benutzt wird. “Recommendation Engines”, bzw. Empfehlungssysteme, sind bekannt als eine der größten Stärken der Graphdatenbanken. Durch die Möglichkeit, das Verhalten der Benutzer wie ein Netzwerk darstellen zu können, eignen sich Graphdatenbanksysteme besonders gut dafür, aus dem Benutzerverhalten zu lernen, und dadurch Empfehlungen machen zu können (vgl. Temme (2015)).

In diesem Kapitel wird ein selbst entworfenes Filmempfehlungssystem beschrieben. Es ist weiterhin nicht Ziel dieser Arbeit, ein perfektes Filmempfehlungssystem zu entwickeln (obwohl es trotzdem funktionsfähig sein und gute Empfehlungen berechnen soll), sondern durch sechs Algorithmen, die auf den drei in Kapitel 2.3 vorgestellten Ansätzen basieren (inhaltsbasiert, kollaborativ und hybrid), die drei Graph-DBMS neutral zu vergleichen.

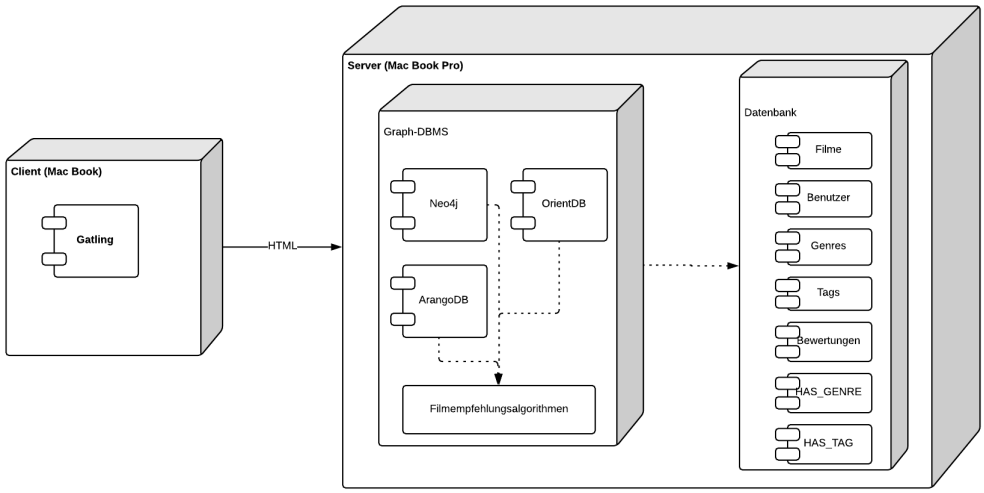
3.1. Anforderungen

Das Filmempfehlungssystem sollte Filmempfehlungen machen können, sowohl für einen Benutzer als auch für einen Film. Das soll mit drei verschiedenen Methodiken geschehen: inhaltsbasiert, kollaborativ, und einer Kombination dieser beiden Ansätze (hybrid). Diese sind die traditionellen Ansätze aller Empfehlungssysteme (vgl. Klahold (2009), S. 2).

3.2. Architektur

In diesem Abschnitt wird die Architektur der Testumgebung beschrieben, mit der die drei Graphdatenbanksysteme verglichen werden (siehe auch Abb. 3.1). Die Testumgebung besteht aus den drei Graph-DBMS, die auf einem Rechner laufen, dem Datensatz, mit dem sie arbeiten, und den Algorithmen, durch die sie verglichen werden. Ein Programm, das die Messungen registriert oder misst, gibt es nicht, da die drei Graph-DBMS den Zeitaufwand jeder Anfrage immer anzeigen. Diese werden also manuell notiert. Bei den Lasttests wird ein zweiter Rechner einbezogen, der als Client mittels des Programms ‘Gatling’ die Anfragen an den Server sendet.

Abbildung 3.1.: Deploymentsicht des Testsystems



3.2.1. Graphdatenbanksysteme

Die drei Graphdatenbanken Neo4j, OrientDB, und ArangoDB laufen auf einem MacBook Pro. Es läuft nichts anderes im Hintergrund. Während der Tests ist der Datensatz nur dem momentan getesteten DBMS bekannt. Die anderen zwei Datenbanken sind leer und nicht in Betrieb. Somit hat jedes Datenbanksystem beim Testen genau die gleiche Menge an freiem Speicher.

Bei den Lasttests kommen die Anfragen von einem zweiten Rechner über eine Internetverbindung. Hier agiert das MacBook Pro als Server.

3.2.2. Algorithmen

Um die Graph-DBMS zu vergleichen, wurde ein einfaches funktionsfähiges Filmempfehlungssystem entworfen, das aus sechs Algorithmen besteht. Diese Algorithmen basieren auf den zwei wichtigsten Konzepten der Empfehlungssysteme (inhaltsbasiert und kollaborativ) und auf zwei Use-Cases: Der Anfrage nach Filmempfehlungen basierend auf einen Film, und der Anfrage nach Filmempfehlungen basierend auf allen Bewertungen eines Benutzers.

Die beiden Empfehlungs-Konzepte wurden für diese Arbeit gewählt, nicht nur weil sie die typischen Ansätze bei Empfehlungssystemen darstellen, sondern auch weil sie die Graphdatenbankensysteme auf folgende Aspekte testen:

1. Wie verändern sich die Antwortzeiten beim Erforschen des exakt gleichen Teilgraphen innerhalb von Graphen verschiedener Größen (inhaltsbasiert)?
2. Wie verändern sich die Antwortzeiten bei Teilgraphen verschiedener Größen (kollaborativ)?

Da die meisten Empfehlungssysteme hybride Algorithmen benutzen, wird auch die Performance bei einer solchen Vorgehensweise getestet.

Kollaborativ

Bei kollaborativen Filmempfehlungen sind die Bewertungen der Benutzer entscheidend. Genres und Tags sind hier irrelevant. Es geht hier also nicht darum, den ähnlichsten Film zu finden, sondern den, den die Benutzer, die diesen Film gut bewerteten, auch gut bewerteten.

Filmbasiert Für die kollaborative Filmempfehlung eines Films werden alle Benutzer gefiltert, die den Film mit fünf Sternen bewertet hatten, und daraufhin alle von diesen Benutzern ebenfalls mit fünf Sternen bewertete Filme gesammelt (s. Abb. 3.2).

Anschließend zählt das System, wie oft jeder dieser Filme eine Fünf-Sterne-Bewertung von einem dieser Benutzer bekommen hat, und die Filmliste, absteigend sortiert, ausgegeben (Beispiel s. Abb. 3.3).

Abbildung 3.2.: Kollaborative Filmempfehlung für einen Film

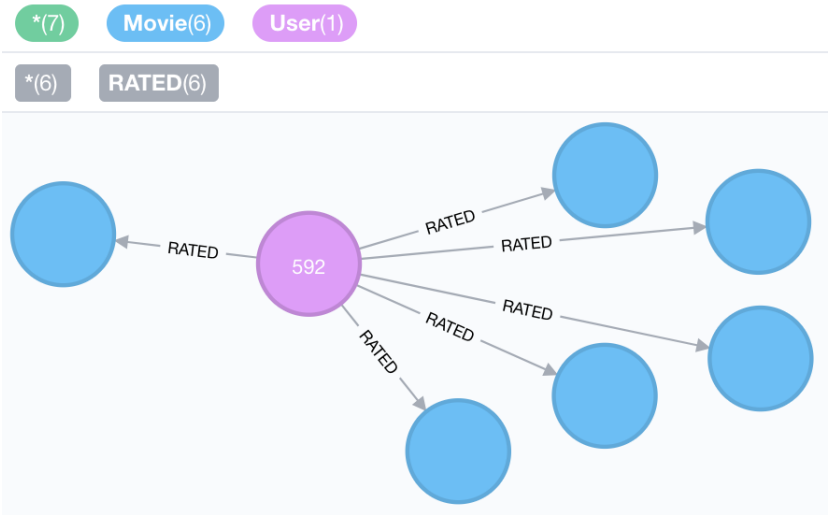
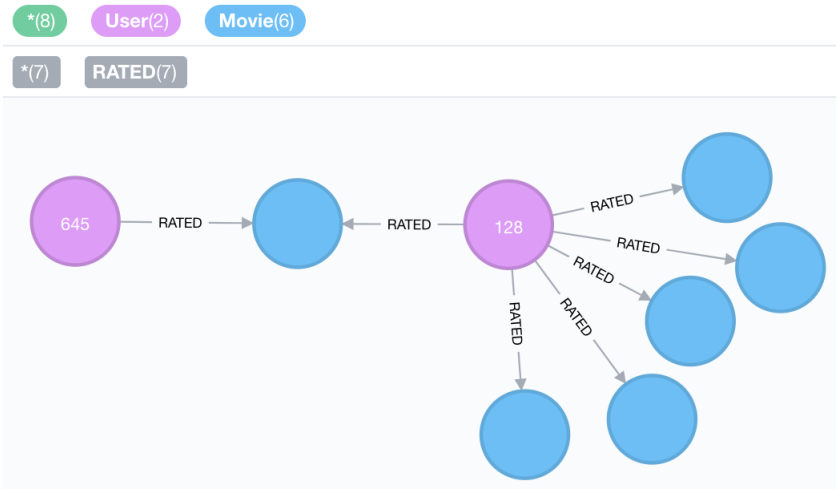


Abbildung 3.3.: Ausgabe für Kollaborative Filmempfehlung für den Film “Star Wars: Episode IV”

Film	Filmempfehlung	Vorkommnisse
"Star Wars: Episode IV - A New Hope (1977)"	"Star Wars: Episode V - The Empire Strikes Back (1980)"	69
"Star Wars: Episode IV - A New Hope (1977)"	"Star Wars: Episode VI - Return of the Jedi (1983)"	55
"Star Wars: Episode IV - A New Hope (1977)"	"Raiders of the Lost Ark (Indiana Jones and the Raiders of the Lost Ark) (1981)"	45
"Star Wars: Episode IV - A New Hope (1977)"	"Shawshank Redemption, The (1994)"	39
"Star Wars: Episode IV - A New Hope (1977)"	"Pulp Fiction (1994)"	37
"Star Wars: Episode IV - A New Hope (1977)"	"Matrix, The (1999)"	37
"Star Wars: Episode IV - A New Hope (1977)"	"Godfather, The (1972)"	36
"Star Wars: Episode IV - A New Hope (1977)"	"Schindler's List (1993)"	29
"Star Wars: Episode IV - A New Hope (1977)"	"Usual Suspects, The (1995)"	28
"Star Wars: Episode IV - A New Hope (1977)"	"Fargo (1996)"	28

Benutzerbasiert Bei der kollaborativen Filmempfehlung für einen Benutzer ist die Prozedur fast identisch wie bei der Filmempfehlung für einen Film, nur dass der Startknoten nicht ein Film ist, sondern ein Benutzer. Es wird für jeden der vom Benutzer mit fünf Sternen bewerteten Filme die kollaborative Filmempfehlung für den Film ausgeführt (s. Abb. 3.4). Anschließend werden alle Ergebnisse addiert und in absteigender Reihenfolge ausgegeben. Filme, die der Benutzer schon bewertet hatte, werden nicht berücksichtigt. Dies ist der aufwändigste Algorithmus, da er den größten Teilgraphen betrachtet. Die Filme werden nämlich nicht auf inhaltliche Ähnlichkeit gefiltert, wodurch die Menge an betrachteten Filmen hier am größten ist.

Abbildung 3.4.: Kollaborative Filmempfehlung für einen Benutzer



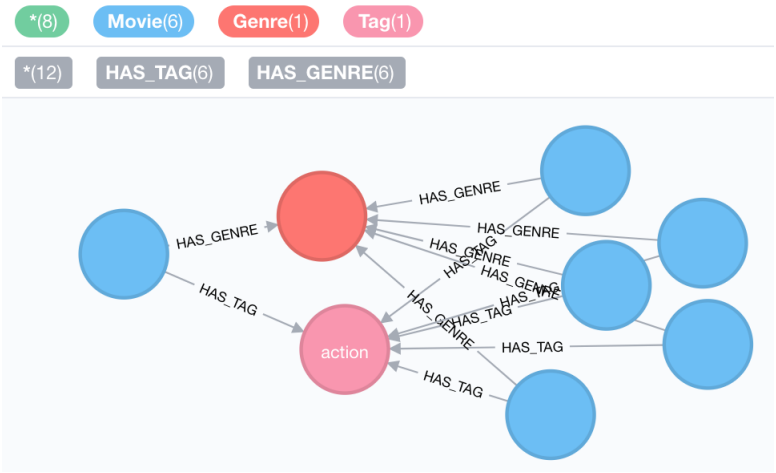
Inhaltsbasiert

Bei inhaltsbasierten Filmempfehlungen spielen Bewertungen von Benutzern keine Rolle. Hier zählt nur der Inhalt des Film, der hier durch die Genres und Tags des Films dargestellt wird. Es geht also darum, den ähnlichsten Film zu finden.

Filmbasiert Um für einen Film den ähnlichsten Film zu finden, werden alle Pfade im Graphen gesucht, die zwei Filme durch ein Genre und einen Tag mit einer Relevanz von über 0.97 verbinden (s. Abb. 3.5). Dieser Wert von 0.97 könnte natürlich geändert werden, dementsprechend, wie sehr sich die Filme ähneln sollen. In diesem Fall wurde ein sehr hoher Wert ausgewählt.

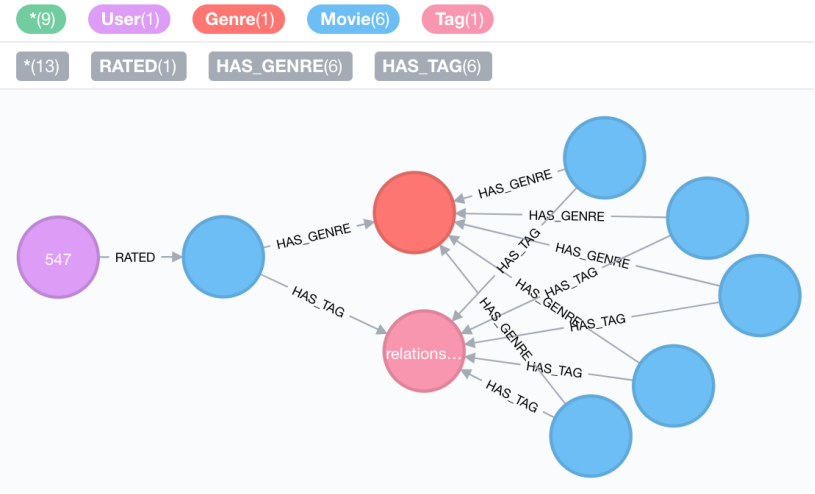
Das System zählt daraufhin, wie oft jeder gefundene Film Teil einer dieser Pfade ist. So könnte ein Film z.B. drei Genres und 10 Tags gemeinsam haben, während ein anderer nur ein Genre und einen Tag teilt. Die Filme werden dann absteigend sortiert. Das stellt dann die Liste der ähnlichsten Filme für einen Film dar.

Abbildung 3.5.: Inhaltsbasierte Filmempfehlung für einen Film



Benutzerbasiert Bei der inhaltsbasierten Filmempfehlung für einen Benutzer werden alle Filme gefiltert, die der Benutzer mit fünf Sternen bewertet hat. Für jeden dieser Filme wird die inhaltsbasierte Filmempfehlung ausgeführt (s. Abb. 3.6), und die Ergebnisse aller dieser Filme addiert, sortiert und ausgegeben. Filme, die der Benutzer schon bewertet hatte, werden auch hier nicht berücksichtigt.

Abbildung 3.6.: Inhaltsbasierte Filmempfehlung für einen Benutzer



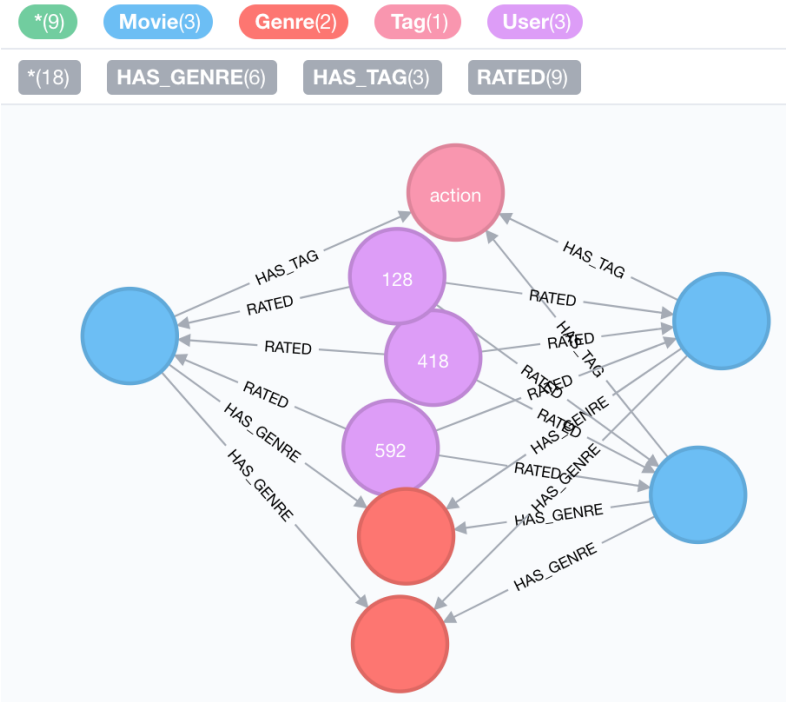
Hybrid

Diese dritte Art der Filmempfehlung vereint den inhaltsbasierten mit dem kollaborativen Ansatz. Hier muss ein Film sowohl gute Bewertungen als auch eine Übereinstimmung von Genre und Tag vorweisen, um als Filmempfehlung ausgewählt werden zu können.

Filmbasiert Zuerst werden, wie bei der kollaborativen Filmempfehlung, die Filme gefiltert, die eine Fünf-Sterne-Bewertung von den Benutzern bekommen hatten, die diesen Film auch mit fünf Sternen bewertet hatten. Auf dieser Filmliste sind dann nur noch die Filme verzeichnet, die mindestens ein Genre und einen Tag (hier Relevanz von mind. 0.97) als Gemeinsamkeit mit dem Film nachweisen (s. Abb. 3.7). Auch hier wird gezählt, in wie vielen Kombinationen jeder gefundene Film vorkommt, z.B. vereinen zwei Filme ein Genre, vier Tags und 24 Fünf-Sterne-Bewertungen der selben Benutzer, dann wird dieser Film in 96 dieser Patterns vorgefunden. Die Filme werden dann dementsprechend absteigend sortiert und ausgegeben.

Benutzerbasiert Hier wird für jeden Film, den der Benutzer mit fünf Sternen bewertet hatte, die zuvor beschriebene hybride Filmempfehlung durchgeführt (s. Abb. 3.8), die daraus resultierenden Ergebnisse addiert und ausgegeben. Filme, die der Benutzer schon bewertet hatte, werden hier ebenfalls nicht berücksichtigt.

Abbildung 3.7.: Hybride Filmempfehlung für einen Film



3.2.3. Daten

Die für diese Bachelorarbeit benutzten Daten wurden von movieLens bereitgestellt. movieLens betreibt eine Website, auf der Benutzer Filme bewerten können. Die Benutzer sind anonym und die aus diesen Bewertungen resultierenden Daten werden öffentlich zugänglich gemacht (s. Grouplens (2018)).

Der Datensatz beinhaltet die Filme, Tags, Bewertungen sowie die Genres und Benutzer. Diese werden in den folgenden Abschnitten im Detail beschreiben.

Filme

Die Filmdatei beinhaltet 45.897 Filme. Diese werden durch eine eindeutige ID, dem Namen des Films (mit Erscheinungsjahr in Klammern) und allen seinen Genres beschrieben. Bei mehreren Genres werden die verschiedenen Genres durch ein '|' getrennt. Da die Genres der Filme als eine Beziehung zwischen dem Film und dem jeweiligen Genre importiert werden sollen, wurde mittels eines selbst-programmierten Java-Programms eine zweite csv-Datei erstellt, in dem pro Zeile nur ein Film und ein Genre sind (s. Abb. 3.9 und 3.10).

Abbildung 3.8.: Hybride Filmempfehlung für einen Benutzer

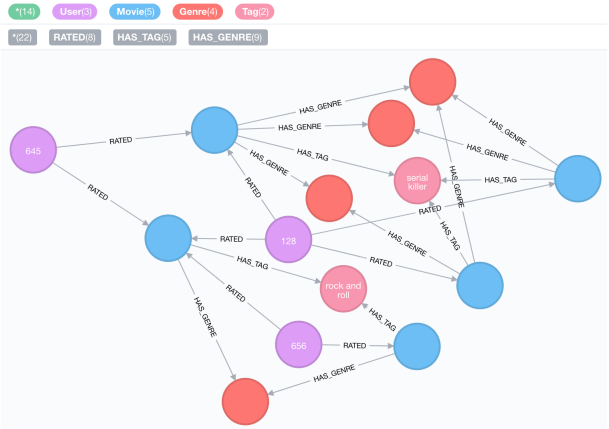


Abbildung 3.9.: Format der Original 'movies.csv' Datei

```
movieId,title,genres
1,Toy Story (1995),Adventure|Animation|Children|Comedy|Fantasy
2,Jumanji (1995),Adventure|Children|Fantasy
3,Grumpier Old Men (1995),Comedy|Romance
```

Tags

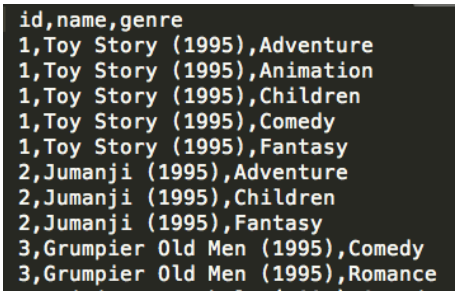
Obwohl Genres einen Film recht gut kategorisieren, ist diese Kategorisierung doch sehr grob. Die Tags bieten eine deutlich detailliertere Kategorisierung der Filme an. Tags können ein Thema des Films sein, der Name eines bekannten Regisseurs oder Schauspielers, der Ort der Handlung, oder ähnliches.

Einige Beispiele für Tags aus der von Grouplens (2018) bereitgestellten Datei sind: 007, dance, gambling, hackers, macabre, narrated, palestine, rabbits, sacrifice, talking animals, undercover cop, vampire, war, zombie.

Für die Tags gibt es zwei Dateien. Eine beinhaltet die Liste von 1128 Tags (tagId, tag). Die zweite verbindet die Filme mit den Tags (movieId, tagId, relevance). Alle Filme werden mit jedem Tag verbunden und an jede dieser erstellten 'HAS-TAG' Kanten wird ein Wert 'relevance' angefügt, der beschreibt, wie relevant der Tag für diesen Film ist. Der Wert geht von 0 (absolut irrelevant) bis 1 (absolut relevant).

Beispiel: Der Tag '007' hat für den Film 'Goldfinger' eine Relevanz von 0.9997499999999999. Der Tag '19th century' hat bei 'Goldfinger' nur eine Relevanz von 0.02250000000000002. Ein

Abbildung 3.10.: Format der bearbeiteten ‘movies.csv’ Datei für die Erstellung von Film-zu-Genre Beziehungen



halbwegs relevanter Tag bei diesem Film wäre beispielsweise ‘action packed’ mit einer Relevanz von 0.63625, da es sich zwar um einen Action-Film handelt, er aber im Vergleich zu den heutigen Action-Filmen eher ruhig verläuft.

Da die Verbindung jedes Films mit jedem Tag unnötig speicherlastig ist, und da die in dieser Arbeit benutzten Algorithmen nur Tags mit einer Relevanz von über 0.97 berücksichtigen, wurde die csv-Datei bearbeitet. Sie enthält nun nur noch die Verbindungen zwischen Film und Tag, die eine Relevanz von über 0.97 haben, wodurch die Datei von über 12.000.000 Beziehungen auf 16.651 reduziert wurde.

Bewertungen

Die Bewertungen bei MovieLens sind Bewertungen von 0.5 (sehr schlecht) bis 5 (sehr gut) in Schritten von 0.5. Jede Bewertung geht von einem bestimmten Benutzer aus (Benutzer besteht nur aus userId. Weitere Informationen über den Benutzer gibt es nicht.) zu einem bestimmten Film. Die Datei enthält auch einen Timestamp für jede Bewertung, dieser wird in dieser Untersuchung aber nicht beachtet. Es gibt mehrere Bewertungsdateien mit variierenden Größen. Hier werden die Dateien mit Größe 100.000 (Bewertungen), 1.000.000, und 10.000.000 benutzt.

Genres und Benutzer

Es gibt keine csv-Dateien für die Genres und die Benutzer. Da es keine persönlichen Informationen über die Benutzer gibt, kann eine csv-Datei mit den Zahlen 1 bis 268654 (Anzahl der Benutzer) manuell erstellt werden, ebenso für die Genres eine csv-Datei mit den 20 Genres.

Tabelle 3.1.: Zusammenfassung der Attribute jedes Datensatzes für Knoten

Filme	Tags	Genres	Benutzer
movieId title genres	tagId tag	genreId genre	userId

Tabelle 3.2.: Zusammenfassung der Attribute jedes Datensatzes für Kanten

HAS_TAG	HAS_GENRE	RATED
movieId tagId relevance	movieId title genre	userId movieId rating

Eine Zusammenfassung über die Attribute aller Knoten und Kanten ist in den Tabellen 3.1 und 3.2 dargestellt.

3.3. Tests

Die Korrektheit des Imports und der Algorithmen des Filmempfehlungssystems muss getestet werden. Die folgenden Abschnitte beschreiben diese Tests.

3.3.1. Datenimport

Um zu testen, ob der Datenimport korrekt erfolgt ist, wird die Anzahl der durch den Import erstellten Elemente angezeigt. Diese muss der Anzahl der Zeilen der csv-Datei entsprechen, damit der Test erfolgreich ist. Dieser Test ist für jede importierte csv-Datei durchzuführen.

3.3.2. Empfehlungsalgorithmen

Die Empfehlungsalgorithmen gelten als korrekt durchgeführt, wenn alle drei Graphdatenbanksysteme dasselbe Ergebnis anzeigen. Außerdem muss für jedes System der Algorithmus für einen sehr kleinen Teilgraphen manuell ausgerechnet werden (also z.B. für einen Benutzer mit nur drei Bewertungen), um zu überprüfen, ob das Ergebnis korrekt ist.

4. Implementierung

Dieses Kapitel beschreibt die Implementierung des Datenimports sowie der Algorithmen des Filmempfehlungssystems. Statt der Anzeige des Codes jeder Anfrage folgt eine Beschreibung, wie für jedes Graphdatenbanksystem die Algorithmen angegeben werden müssen sowie ein konkretes Beispiel zur Veranschaulichung der Syntax.

4.1. Neo4j

Neo4j benutzt die eigene Anfragesprache 'Cypher'. Alle Anfragen können direkt über die Benutzeroberfläche eingegeben werden.

4.1.1. Datenimport

Beim Datenimport ist es wichtig, nach dem Import der Knoten Indizes festzulegen, da dies den Import der Kanten extrem beschleunigt. Hier müssen 3 Indizes erstellt werden:

- create index on :Movie(id)
- create index on :User(id)
- create index on :Tag (id)

Dadurch ist der Kantenimport schneller, da Neo4j die Film-, Benutzer-, und Tagknoten intern schneller findet, um sie daraufhin mit der Beziehung (z.B. Bewertungs-Beziehung von Benutzer zu Film) zu verbinden.

Die csv-Dateien müssen in einem fest vorgegebenen Ordner gespeichert werden, damit Neo4j darauf zugreifen kann.

Daraufhin wird in der Neo4j Benutzeroberfläche (erreichbar unter 'localhost:7474') der Ladebefehl eingegeben, mit dem Namen der csv-Datei, gefolgt vom Befehl zur Erstellung, ganz nach Bedarf. In Listing 4.1 findet sich ein Beispiel für die Erstellung von Filmknoten.

Listing 4.1: Beispiel: Filmknotenimport Neo4j

```
1 USING PERIODIC COMMIT
2 LOAD CSV WITH HEADERS FROM ‘‘file:///movies.csv’’ AS row
3 create (movie:Movie {id: toInteger(row.movieId), title: row.title})
```

‘row’ bezeichnet jede Zeile der csv-Datei. In dem obigen Beispiel ist es so, dass ein Film der Klasse ‘Movie’ erstellt wird, mit den Werten ‘id’, der in der csv-Zeile unter ‘movieId’ aufgeführt ist, und ‘title’, der in der csv-Datei auch unter ‘title’ zu finden ist. Die Werte bzw. Properties, der Knoten sind in geschweiften Klammern anzugeben.

Alle Werte der csv-Datei liest Neo4j als String. Sollte also ein Wert als Integer oder Float gespeichert werden, so muss eine toInteger oder toFloat Methode benutzt werden.

Das gleiche Verfahren muss für die Genres, Benutzer, und Tags wiederholt werden.

Um Kanten zu erstellen, ist das Verfahren ähnlich, nur der Befehl zur Erstellung der Kante unterscheidet sich von dem zur Erstellung eines Knotens (s. Listing 4.2).

Listing 4.2: Beispiel: Kantenimport Neo4j

```
1 USING PERIODIC COMMIT
2 LOAD CSV WITH HEADERS FROM ‘‘file:///genome-scores.csv’’ AS row
3 MATCH (m:Movie {id:toInteger(row.movieId)})
4 MATCH (t:Tag {id:toInteger(row.tagId)})
5 MERGE (m)-[:HAS_TAG {relevance:toFloat(row.relevance)}]->(t);
```

Zuerst müssen die beiden zu verbindenden Knoten gefunden werden. Daraufhin wird die Kante zwischen den Knoten erstellt. Der Kante können auch Properties angefügt werden.

Folgende Beziehungen/Kanten müssen importiert werden:

- (Movie)-[:HAS_GENRE]->(Genre)
- (Movie)-[:HAS_TAG {relevance}]->(Tag)
- (User)-[:RATED {rating}]->(Movie)

4.1.2. Empfehlungsalgorithmen

Um die Empfehlungsalgorithem zu implementieren, wird "Pattern Matching" ausgeführt. Dies wird in Neo4j gemacht indem der Pfad bildlich eingegeben wird, mit Knoten in runden Klammern und den Kanten in eckigen Klammern (s. Listing 4.3).

Listing 4.3: Beispiel: Pfad in Neo4j

```
(n:Movie)-[:HAS_GENRE]->(g:Genre)
```

Sollten noch Properties beim Pattern-Matching berücksichtigt werden, so wird am Anschluss des Pfades dies noch angegeben. Alternativ kann die Property auch in geschweiften Klammern in die Klammern geschrieben werden (s. Listing 4.4).

Listing 4.4: Beispiele: Pattern-Matching mit Properties in Neo4j

```
(n:Movie)-[a:HAS_TAG]->(b:Tag) where a.relevance > 0.97

(n:Movie)<-[r:RATED {rating:5.0}]-(user:User)
```

Wenn ein Pfad-Element zum wiederholten Mal angegeben wird, so ist es nicht mehr notwendig die Klasse anzugeben. Sollte also zum Beispiel Film n wiederholt im Pfad vorkommen, so würde man nur (n) schreiben.

Schließlich muss angegeben werden, welche Werte von den gefundenen Pfaden angezeigt werden sollen. Die Anzahl der Vorkommnisse eines Knotens wird mit der Methode count() auf den Bezeichner angewendet (z.B. RETURN user.userID, n.title, count(n)).

Die Ergebnisse können dann noch aufsteigend oder absteigend sortiert werden. Die Anzahl der ausgegebenen Elemente kann auch begrenzt werden.

In Listing 4.5 findet sich als konkretes Implementierungsbeispiel die kollaborative Filmempfehlung für den Benutzer mit ID 2.

Listing 4.5: Beispiel: Kollaborative Filmempfehlung in Neo4j

```
MATCH (u:User {id:2})-[:RATED {rating: 5.0}]->(n:Movie)<-[:RATED
{rating:5.0}]-(u2:User)-[:RATED{rating:5.0}]->(other:Movie)
WHERE NOT (other)<-[:RATED]-(u)
RETURN other.title, other.id, count(other) AS scoring
Order by scoring desc limit 10
```

4.2. OrientDB

OrientDB benutzt die eigene Anfragesprache “OrientDB SQL”. Die Algorithmen werden auf der Benutzungsoberfläche ausgeführt, der Import auf der Konsole.

4.2.1. Datenimport

Der Import von csv-Dateien ist in OrientDB nur indirekt möglich. Um csv-Dateien zu importieren, müssen zunächst ETL (Extractor Transformer and Loader) Dateien geschrieben werden (vgl. Orientdb (2018a)).

Es handelt sich um JSON-Dateien, die die csv-Dateien lesen und in das JSON-Format für den Import umwandeln. Abb. 4.1 zeigt ein Beispiel einer ETL-Datei für den Import der Filme.

Abbildung 4.1.: ETL für den Import von Filmen in OrientDB

```
{
"config": {
"log": "info",
"parallel": false
},
  "source": { "file": { "path": "/Users/ditmarlange/Desktop/import/moviescombined2new.csv" } },
  "extractor": { "csv": {"separator": ",",
"columnsOnFirstLine": true,
"columns": ["movieId:integer",
"title:String", "genres:String"]} },
  "transformers": [
    { "vertex": { "class": "Movie" }, "skipDuplicates": true }
  ],
  "loader": {
    "orientdb": {
      "dbURL": "plocal:/Users/ditmarlange/Downloads/orientdb-community-importers-2.2.33/databases/Movies",
      "dbType": "graph",
      "classes": [
        {"name": "Movie", "extends": "V"},
      ], "indexes": [
      {"class":"Movie", "fields":["movieId:integer"], "type":"UNIQUE" }
      ]
    }
  }
}
```

Im “source”-Block wird der Pfad zur csv-Datei angegeben. Im “extractor”-Block wird angegeben, dass es sich um eine csv-Datei handelt, sowie das Format der Datei beschrieben (Trennzeichen, Spaltennamen, Datentypen).

Im “transformers”-Block wird die Klasse der zu erstellenden Knoten eingetragen.

Im “loader”-Block wird der Pfad zur OrientDB-Datenbank angegeben sowie die durch diesen ETL erstellten Klassen und die Indizes, die dabei erstellt werden sollen. Auch hier ist es wichtig

die IDs zu indizieren, damit der Import der Kanten danach schneller abläuft. Es kann hier auch angegeben werden, ob eine bestimmte Property “Unique” sein soll, damit es dann keine 2 Knoten der gleichen Klasse mit derselben Property gibt (hier bei der ID sinnvoll, denn sollten z.B. zwei verschiedene Benutzer oder Filme die selbe ID haben, so könnten Bewertungs-Kanten zwischen den verkehrten Benutzern und Filmen erstellt werden).

Für den Import der Kanten ist das Format der ETL-Datei ein wenig unterschiedlich (s. Abb. 4.2).

Abbildung 4.2.: ETL für den Import von Bewertungen in OrientDB

```
{
"config": {
"log": "info",
"parallel": false
},
"source": {
"file": {
"path": "/Users/ditmarlange/Desktop/import/ratings100k.csv"
}},
"extractor": {
"row": {
}},
"transformers": [{
"csv": {
"separator": ",",
"columnsOnFirstLine": true,
"columns": ["user:integer",
"movie:integer",
"rating:double"]
}},
{
"command": {
"command": "create edge RATED from (select from user where _key = ${input.user}) to (select from movie where movieId = ${input.movie}) set rating = ${input.rating}",
"output": "edge"
}},
  {
    "field": {
      "fieldName": "user",
      "expression": "remove"
    } },
  {
    "field": {
      "fieldName": "movie",
      "operation": "remove"
  }  },
  {
    "field": {
      "fieldName": "rating",
      "expression": "remove"
    }},
  {
    "field": {
      "fieldName": "_key",
      "operation": "remove"
    }},
  {
    "field": {
      "fieldName": "movieId",
      "operation": "remove"
    }},
  {
    "field": {
      "fieldName": "@class",
      "value": "RATED"
    }}],
"loader": {
"orientdb": {
"dbURL": "plocal:/Users/ditmarlange/Downloads/orientdb-community-importers-2.2.33/databases/Movies",
"dbType": "graph",
"standardElementConstraints": false,
"classes": [{
"name": "RATED", "extends": "E"}
]}}}
```

Die “config”- und “source” Blöcke bleiben unverändert. Im “extractor”-Block wird nun aber “row” anstatt “csv” angegeben.

Erst im “transformer”-Block wird angegeben, dass es sich um eine csv-Datei handelt, sowie deren Format. Die Klasse der zu erstellenden Kanten wird hier wiederum nicht angegeben.

Dafür folgt nun ein neuer “command”-Block, in dem ein OrientDB Befehl angegeben wird, als würde die Kante manuell in der OrientDB-Benutzungsoberfläche eingegeben werden (s. Abb. 4.2).

Daraufhin muss angegeben werden, dass es um das Erstellen einer Kante geht, gefolgt von mehreren “field”-Blöcken, in denen immer “fieldName”: vor den Klassen- oder Propertynamen steht, und darunter immer “expression”:“remove”.

Der letzte “field”-Block hat den “fieldName”: “@class”, gefolgt von “value” und dem Klassennamen der Kante.

Zuletzt kommt wieder der “loader”-Block mit den Verweisen zur OrientDB-Datenbank und zu der zu erstellenden Klasse (z.B. ‘RATED’).

Folgende ETL-Dateien müssen erstellt werden (Dateinamen sind variabel):

- movies.json
- users.json
- tags.json
- genres.json
- ratings.json
- hasgenre.json
- hastag.json

Um die ETL-Dateien auszuführen, muss in der Konsole in den bin-Ordner von OrientDB gewechselt werden und dann ./oetl.sh, gefolgt vom Pfad zur Datei ausgeführt werden. Beispiel: ./oetl.sh /Users/ditmarlange/Desktop/ratings.json

4.2.2. Empfehlungsalgorithmen

Das “Pattern-Matching” wird in OrientDB bildlich eingegeben. Zuerst muss angegeben werden, welche Werte erhoben werden sollen (z.B. m2.title). Anschließend wird der Pfad so, wie man ihn mit Linien zeichnen würde, angegeben. Es werden immer in geschweiften Klammern die Klasse des Knotens angegeben und, wenn benötigt, ein Bezeichner, um diesen Knoten noch zu referenzieren. Auch Properties können hier gefiltert werden (s. Listing 4.6).

Listing 4.6: Beispiel: Matching eines Knotens in OrientDB

```
{class:movie, as: m, where:(movieId = 616)}.
```

Ausgehende Kanten werden mit '.outE' angegeben, mit dem Namen der Kante in Klammern, und dannach, wenn notwendig, in geschweiften Klammern die Properties. Daraufhin folgt noch 'inV()', gefolgt vom nächsten Knoten im Pfad. Bei eingehenden Kanten ist dies genauso, nur dass '.inE' und 'outV()' angegeben werden. Sollten die Kante keine Properties benötigen, so kann auf das 'E' und auf das 'inV()' oder 'outV()' verzichtet werden.

Wenn der gesamte Pfad angegeben wurde, werden wieder die Werte, die anfangs für die Rückgabe angegeben wurden, angefügt. Die Ausgabe kann auch aufsteigend oder absteigend sortiert werden.

Das Zählen der Vorkommnisse eines Knotens gelingt durch die Methode 'count(*)', die als letzter Rückgabewert aufgeführt wird. Daraufhin wird die Anzahl der Pfade die den gewünschten Wert enthalten gezählt und ausgegeben.

In Listing 4.7 findet sich ein Implementierungsbeispiel für die hybride Filmempfehlung für einen Film.

Listing 4.7: Beispiel: Hybride Filmempfehlung in OrientDB

```
select m2.title, count(*) from (match
    {class:movie, as: m, where:(movieId = 616)}
    .inE('rated'){where: (rating = 5.0)}.outV()
    {class:user, as: u}
    .outE('rated'){where: (rating = 5.0)}.inV()
    {class:movie, as: m2}
    .out('hasgenre')
    {class:genre, as: g}
    .in('hasgenre')
    {as: m}
    .outE('hastag'){where: (relevance > 0.97)}.inV()
    {class: tag, as: t}
    .inE('hastag'){where: (relevance > 0.97)}.outV()
    {as: m2}
    return $paths)
group by m2.title order by count desc
```

4.3. ArangoDB

ArangoDB benutzt die eigene Anfragesprache AQL. Wie bei OrientDB werden die Algorithmen auf der Benutzungsoberfläche, der Datenimport aber auf der Konsole ausgeführt.

4.3.1. Datenimport

Für den Import in ArangoDB müssen die Header der csv-Dateien z.T. verändert werden. Wenn Knoten importiert werden sollen, muss der Header der ID "_key" lauten.
In Listing 4.8 findet sich ein Beispiel für einen Header für die Film-csv-Datei.

Listing 4.8: Beispiel: Benötigtes Header-Format der Film-csv-Datei für ArangoDB

```
1 _key,title,genres
```

Für den Import von Kanten/Beziehungen, müssen die Header '_from' und '_to' heißen. Unter '_from' wäre eine Property des Startknotens aufgeführt, unter '_to' eine Property des Zielknotens.

Außerdem muss vor dem Import von Kanten in der Benutzungsoberfläche von ArangoDB die Collection erstellt, und dabei 'waitForSync' auf 'true' gesetzt werden, damit die Daten permanent gespeichert werden.

Der Import geschieht dannach über die Konsole. Um Knoten zu importieren, muss ein Befehl ausgeführt werden, der folgendes Format hat (s. Listing 4.9).

Listing 4.9: Format des Knotenimports bei ArangoDB

```
1 arangoimp --file ''<Pfad zur csv-Datei'' --type csv
2 --from-collection-prefix <Klasse des Startknotens>>
3 --to-collection-prefix <Klasse des Zielknotens>>
4 --collection <<Klasse der zu erstellenden Kante,
5 genau wie in der Benutzeroberflaeche eingegeben>>
```

4.3.2. Empfehlungsalgorithmen

Um die Filmempfehlungsalgorithmen in ArangoDB zu implementieren, muss für jeden gesuchten Knoten eine For-Schleife erstellt werden. Für jede For-Schleife können dann die für den gesuchten Knoten gesuchten Properties gefiltert werden. Kanten werden mit 'INBOUND' oder 'OUTBOUND' angegeben. Beispiel:

Listing 4.10: Beispiel: Kanten-Matching in ArangoDB

```
FOR m, e IN 1..1 OUTBOUND u RATED
```

Hier ist ‘u’ der Ausgangsknoten, ‘m’ der Knoten, der durch die Kante RATED erreicht werden kann, und ‘e’ die Kante. Es kann dann sowohl nach Properties von m als auch von e gefiltert werden. Sollte die Kante keine relevanten Properties haben, so kann der Bezeichner der Kante auch entfallen.
‘1..1’ sagt hier aus, dass es eine direkte Kantenverbindung ist. Man könnte auch z.B. mit ‘1..3’ drei RATED Kanten traversieren. Dies ist aber für dieses Filmempfehlungssystem uninteressant.

Um die Anzahl der Vorkommnisse eines Knotens zu zählen, wird der Befehl ‘COLLECT’ angegeben, durch den diese Knoten in einer Liste gesammelt werden (z.B. collect title = m2 into list). Die Anzahl der Elemente jeder Liste sagt aus, wie oft der Knoten in einem Pfad gefunden wurde. Die Größe der Listen jedes gesammelten Knotens kann dann sortiert werden.

Letztlich müssen dann nur noch die gewünschten Rückgabewerte angegeben werden.

In Listing 4.11 findet sich ein Implementierungsbeispiel für die kollaborative Filmempfehlung für den Benutzer mit ID ‘2’:

Listing 4.11: Beispiel: Kollaborative Filmempfehlung in ArangoDB

```
FOR u IN User
FILTER  u._key == "2"
    FOR m, e IN 1..1 OUTBOUND u RATED
    Filter e.rating == 5
        For u2, e2 in 1..1 INBOUND m RATED
        filter e2.rating == 5
            for m2, e3 in 1..1 OUTBOUND u2 RATED
            filter e3.rating == 5 and m2.title != m.title
            collect title = m2 into list
            sort length(list) desc
            return {"title" : title.title, "genres": title.genres,
            "count": Length(list)}
```

5. Vergleich der Graphdatenbanksysteme

In diesem Kapitel werden die Graphdatenbanksysteme miteinander verglichen. Zunächst wird der Aufbau der Messungen beschrieben. Am Ende dieses Kapitels findet sich die Auswertung der gesammelten Daten.

5.1. Aufbau und Durchführung

Um sicherzustellen, dass der Vergleich neutral ist, wurden alle Zeitmessungen lokal auf demselben Rechner durchgeführt, auf dem ansonsten kein anderes Programm lief. Jede Datenbank hat den gleichen Speicherplatz auf der Festplatte zur Verfügung. Alle Anfragen wurden außerdem direkt auf der Datenbank ausgeführt, ohne jegliche zusätzliche Programme. Die Ergebnisse wurden auf einem separaten Rechner manuell notiert.

Für die Lasttests wurden die Anfragen von einem zweiten Rechner mittels des Tools “Gatling” (vgl. Gatling-Corp (2018)) über eine Netzverbindung gesendet, um sicherzustellen, dass jegliche Verzögerungen oder Fehler bei der Berrechnung der Algorithmen geschehen sind, und nicht schon beim Senden der Anfrage wegen Überlastung. Somit fungiert das MacBook Pro mit den Graphdatenbanksystemen als Server und der zweite Rechner (ein MacBook), der über die Internetverbindung mittels Gatling die Anfragen sendet, als Client. Die Internetverbindung ist das Haus-WLAN, das stabil ist. Außerdem wurde sichergestellt, dass kein anderes Gerät während der Lasttests mit dem WLAN verbunden war.

Es wurde jeder Empfehlungsalgorithmus auf jedem Graphdatenbanksystem getestet. Jeder Algorithmus wurde außerdem mit verschieden großen Teilgraphen getestet, in dem z.B. der Film mit den meisten oder wenigsten 5-Sterne Bewertungen zur Messung ausgewählt wurde, und auch der Benutzer mit den meisten oder wenigsten Bewertungen. Außerdem wurde jeder Algorithmus im Umfeld von 100.000, 1.000.000 und 10.000.000 Bewertungen ausgeführt. Jede Zeitmessung wurde drei Mal ausgeführt. Wenn eine Anfrage nach 10 Stunden keine Antwort zurückgegeben hatte, galt diese als gescheitert.

Tabelle 5.1.: Speicherbedarf

Bewertungen	Neo4j	OrientDB	ArangoDB
100.000	218,1 MB	205 MB	63,6 MB
1.000.000	587,5 MB	446,7 MB	300,3 MB
10.00.000	4,71 GB	3,08 GB	1,34 GB

Für die Lasttests wurden Kapazitätentests und Stresstests durchgeführt. Bei den Kapazitätentests wurden die Benutzer schrittweise hinzugefügt, während bei den Stresstests alle Benutzeranfragen gleichzeitig beim Graphdatenbanksystem ankamen. Dies wurde auch mit verschiedenen Mengen getestet, jedoch nur für die inhaltsbasierte Filmempfehlung des Films mit den meisten Tags/Genres (“Reservoir Dogs (1992)”), bei 100.000 Bewertungen. Getestet wurde auf die schnellste, langsamste und durchschnittliche Antwortzeit, sowie die Anzahl der gescheiterten Anfragen. Wenn die Antwortzeit über 100x des üblichen Zeitaufwands entsprach, galt die Anfrage durch einen Timeout als gescheitert.

5.2. Ergebnisse

Alle Messungen und Lasttests sind vollständig im Anhang zu finden. An dieser Stelle werden nur der Speicherbedarf der Datenbanken konkret gezeigt und die Ergebnisse zusammengefasst. Zur Visualisierung wurden sieben Diagramme erstellt, die einen Überblick über die Performance der Graphdatenbanksysteme geben. Die Graphen stellen die Summe der schnellsten Zeitmessungen der drei Teilgraphen (größter, mittlerer, kleinster) für jedes Graphdatenbanksystem bei sieben spezifischen Szenarien dar, die exemplarisch für die generelle Performance, die die Graphdatenbanken im Vergleich gezeigt haben, sind.

5.2.1. Speicherbedarf

Wie aus Tabelle 5.1 ersichtlich, hat Neo4j den größten Speicherbedarf der drei Graphdatenbanksysteme, gefolgt von OrientDB. ArangoDB ist sehr speichersparend.

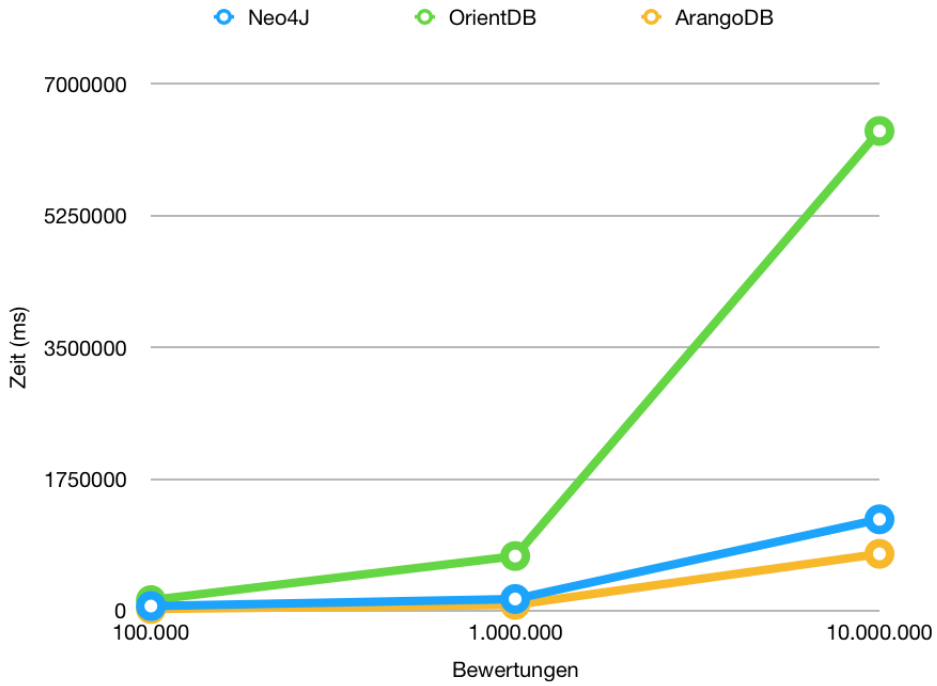
5.2.2. Zeitmessungen

Die exemplarischen Zeitmessungsergebnisse sind folgend in Diagrammen dargestellt. Die Zeit bei den Filmempfehlungen entspricht immer der Summe der drei Messungen (kleinster, mittlerer und größter Teilgraph).

Datenimport

Während bei 100.000 Bewertungen kein großer Unterschied zwischen den Graphdatenbanken erkennbar ist, wird bei größeren Datenmengen ersichtlich, dass OrientDB deutlich langsamer als die anderen beiden Graphdatenbanken importiert. ArangoDB hat sich als schnellste Graphdatenbank für den Datenimport erwiesen (s. Abbildung 5.1).

Abbildung 5.1.: Vergleich des Zeitaufwands für den Datenimport

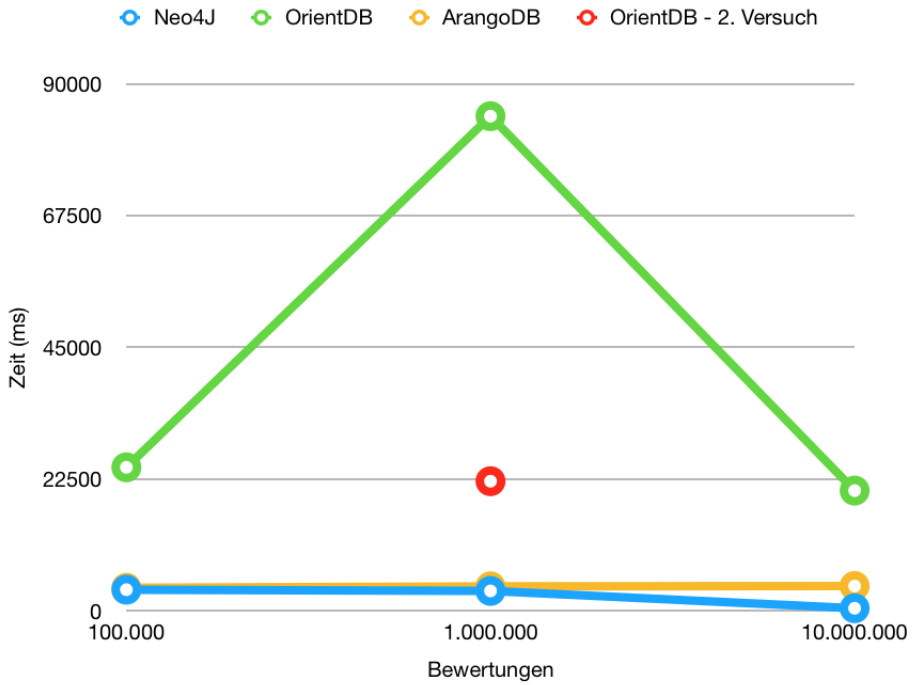


Filmempfehlungen

Bei den Filmempfehlungen variieren die Ergebnisse je nach Szenario. Jedes dieser Diargamme wird nachfolgend im Detail erklärt. Die in den Diagrammen dargestellte Bearbeitungszeit, ist die Summe der jeweils schnellsten Berechnung der Filmempfehlung für den Film/Benutzer mit dem größten Teilgraphen (z.B. meiste Bewertungen), dem kleinsten Teilgraphen und mittleren Teilgraphen.

Abbildung 5.2 Es ist ersichtlich, dass OrientDB die deutlich langsamste Graphdatenbank darstellt. Bei der Messung bei 1.000.000 Bewertungen ist offensichtlicherweise ein Messfehler

Abbildung 5.2.: Vergleich des Zeitaufwands für die inhaltsbasierte Filmempfehlung für einen Film



entstanden. Das könnte an einem vergessenem Neustart des OrientDB-Servers vor Durchführung der Messung liegen. Deshalb wurde die Messung ein zweites Mal durchgeführt. Neo4j und ArangoDB sind ähnlich schnell, wobei Neo4j bei den ganz großen Datenmengen die besten Ergebnisse zeigt. Bis auf den Messfehler kann aber bei allen Graph-DBMS festgehalten werden, dass beim gleichgroßen Teilgraph in größer werdenden Obergraphen die Antwortszeiten im Großen und Ganzen konstant bleiben.

Abbildung 5.3 Während Neo4j sehr gute und OrientDB gute Resultate zeigen, sind ArangoDBs Antwortzeiten exorbitant. Eine Anfrage, die wenige Sekunden dauern sollte, hat bei 10.000.000 Bewertungen bei ArangoDB über 10 Stunden gedauert. Da ArangoDB bei der inhaltsbasierten Filmempfehlung für Filme sehr gute Ergebnisse gezeigt hatte, sind diese Ergebnisse schwer nachzuvollziehen, da es sich hier eigentlich nur um die wiederholte Ausführung und Summierung dieser Filmempfehlungen handelt.

Abbildung 5.4 Bei dieser aufwändigen Aufgabe zeigt Neo4j wieder die schnellste Performance, während ArangoDB deutlich am langsamsten ist.

Abbildung 5.3.: Vergleich des Zeitaufwands für die inhaltsbasierte Filmempfehlung für einen Benutzer

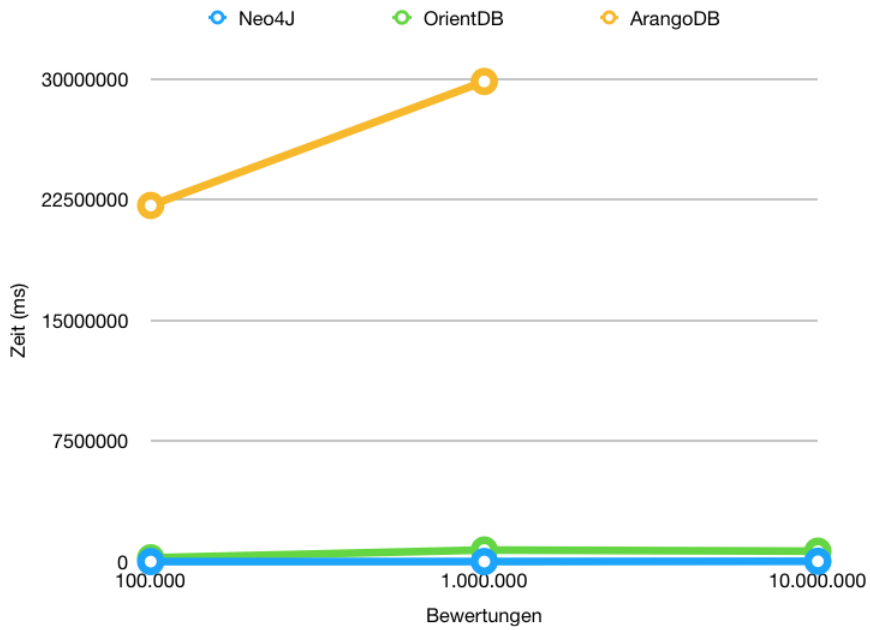
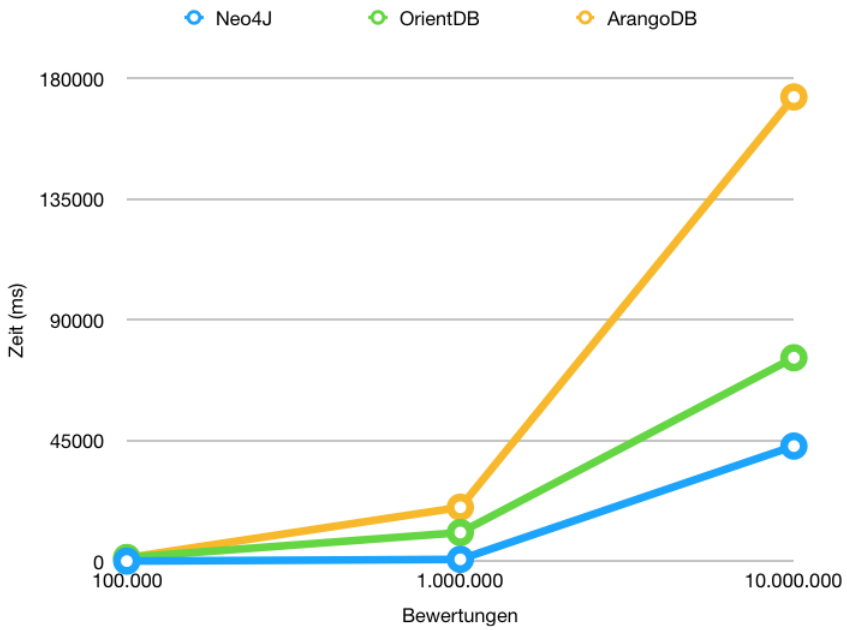


Abbildung 5.5 Dies ist der aufwändigste Algorithmus, der in dieser Arbeit getestet wurde. Bei 100.000 Bewertungen war ArangoDB am langsamsten, während auch hier Neo4j die schnellsten Zeiten aufweisen konnte. Ab 1.000.000 Bewertungen terminierte nur noch Neo4j in weniger als 10 Stunden. OrientDB und ArangoDB haben die 10 Std. überschritten.

Abbildung 5.6 Neo4j ist konstant schnell. Während die Antwortzeiten von OrientDB aber relativ linear wachsen, schießen die Antwortzeiten von ArangoDB bei 10.000.000 Bewertungen steil in die Höhe, wodurch auch bei diesem Vergleich ArangoDB nicht überzeugen kann.

Abbildung 5.7 Auch bei dem letzten Vergleich überzeugt Neo4j mit einer bei weitem besseren Performance. ArangoDB ist wieder mit Abstand das langsamste Graph-DBMS, und ab 1.000.000 Bewertungen gelingt es weder ArangoDB noch OrientDB, in unter 10 Stunden das Ergebnis zu berechnen.

Abbildung 5.4.: Vergleich des Zeitaufwands für die kollaborative Filmempfehlung für einen Film



5.2.3. Lasttests

In diesem Abschnitt werden die Ergebnisse der Lasttest zusammenfassend erläutert. Die detaillierten Ergebnisse sind im Anhang einsehbar.

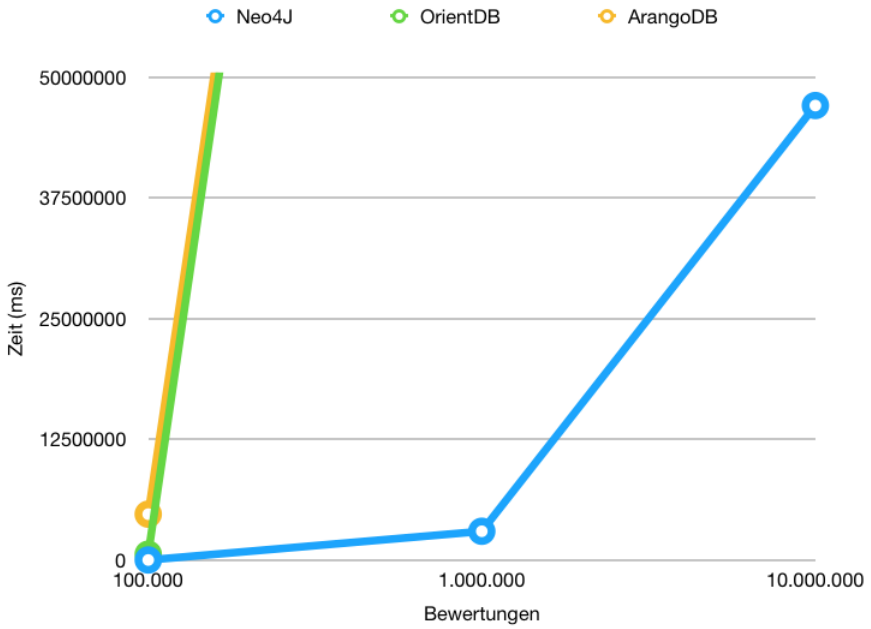
Kapazitätentest

Am besten hat ArangoDB beim Kapazitätentest abgeschnitten, bei einer maximalen Anzahl von Benutzern, die bei 8279 lag. Neo4j wies auch einen guten Wert vor, bei 7870 Benutzern. Bei OrientDB ist eine maximale Grenze von 1000 Benutzern vorgeschrieben, wodurch OrientDB diesen Vergleich klar verliert. Die Antwortszeiten waren bei Neo4j ausgezeichnet, bei ArangoDB mittelmäßig, und bei OrientDB sind alle Anfragen durch einen Timeout ergebnislos beendet worden.

Stresstest

Beim Stresstest hatte Neo4j hervorragende Ergebnisse. Nur eine von den insgesamt 11050 Anfragen ist gescheitert, alle anderen sind mit recht guten Antwortzeiten erfolgt. Ab 500

Abbildung 5.5.: Vergleich des Zeitaufwands für die kollaborative Filmempfehlung für einen Benutzer



Benutzern gab es bei ArangoDB Anfragen, die durch den Timeout beendet wurden, ab 750 Benutzern war es über die Hälfte der Anfragen. Auch ansonsten sind bei ArangoDB schon ab 200 Benutzern einige Anfragen komplett fehlgeschlagen. Wieder unbefriedigend zeigte sich OrientDB: Nicht eine einzige Anfrage wurde erfolgreich zurückgegeben.

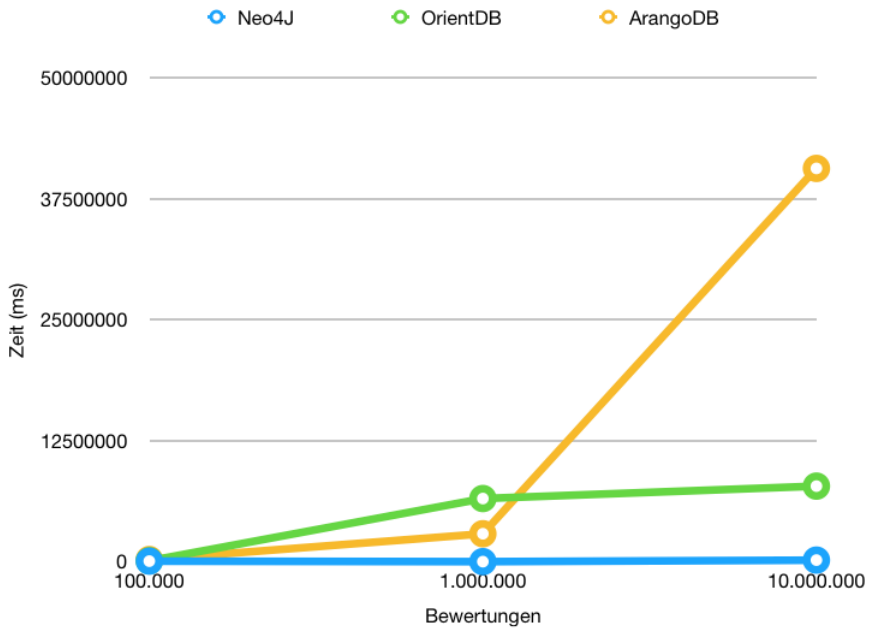
5.3. Auswertung

In dieser Auswertung werden die Ergebnisse der Untersuchung beurteilt. Zuerst werden die objektiv vergleichbaren Aspekte bewertet, bevor auf die subjektiven Aspekte eingegangen wird. Die subjektiven Aspekte basieren zwar auf Meinung, sind aber dennoch wichtig um zu entscheiden, welches der Graphdatenbankensysteme am besten für eine solche Aufgabe geeignet ist.

5.3.1. Objektive Bewertung

Obwohl Neo4j den größten Speicherbedarf aufweist, hat Neo4j die allgemein beste Performance der drei Systeme gezeigt. Bei fast allen Filmempfehlungsalgorithmen hat Neo4j am

Abbildung 5.6.: Vergleich des Zeitaufwands für die hybride Filmempfehlung für einen Film

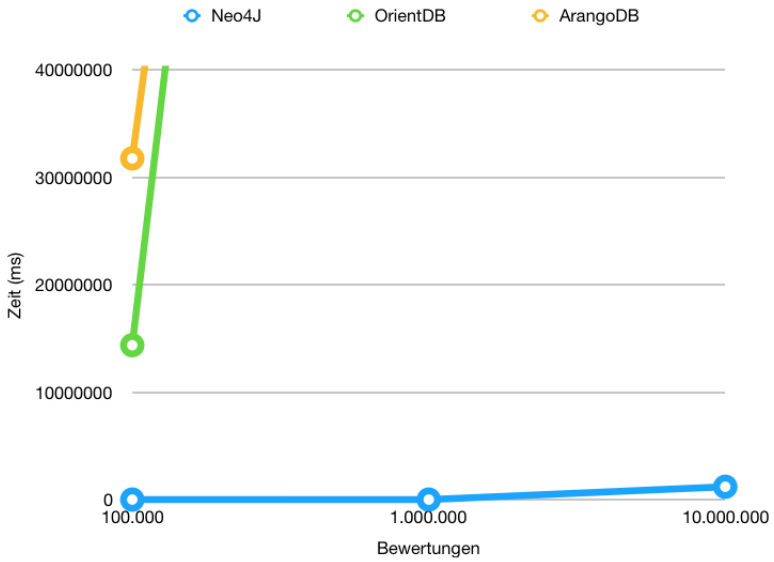


schnellsten das Ergebnis berechnet, und auch bei den Last- und Stresstests war Neo4j am souveränsten.

Allgemein am schlechtesten waren die Testergebnisse von ArangoDB. Die Berechnungszeiten für die Filmempfehlungen waren sehr schlecht, vor allem im Vergleich zu Neo4j. Unerklärlich ist auch, dass OrientDB (und teilweise auch ArangoDB bei etwas größeren Teilgraphen) bei komplexen Queries selbst an sehr simplen Anfragen scheiterte. Wenn z.B. die hybride Filmempfehlung eines durchschnittlichen Films etwa 43 Minuten dauert, und diese nun für einen Benutzer ausgeführt wird, der nur 3 Filme mit 5 Sternen bewertet hat, so sollte die Berechnungszeit hierfür ca. 3 x 43 Minuten dauern. Stattdessen war nach 10 Stunden immer noch kein Ergebnis berechnet worden. Dies weist darauf hin, dass diese Query zu komplex für OrientDB ist, was überraschte. Außerdem war das festgeschriebene Limit von 1000 Benutzern bei OrientDB sehr enttäuschend.

ArangoDB sticht positiv heraus, hinsichtlich des Speicherbedarfs und der Geschwindigkeit des Datenimports. Hier war ArangoDB herausragend. Bei den Last- und Stresstests war es positiv, dass ArangoDB die höchste Anzahl an Benutzern ertragen konnte, ohne abzustürzen,

Abbildung 5.7.: Vergleich des Zeitaufwands für die hybride Filmempfehlung für einen Benutzer



auch wenn die Antwortszeiten deutlich langsamer als bei Neo4j waren. Die Antwortszeiten der Filmempfehlungsalgorithmen waren allgemein bei ArangoDB und OrientDB extrem langsamer als bei Neo4j.

5.3.2. Subjektive Bewertung

In diesem Abschnitt werden die Aspekte bewertet, die eher meinungsbasiert sind. Nach monatelanger Benutzung dieser drei Graph-DBMS gewinnt man natürlich einen guten Eindruck darüber, welche der Graphdatenbanksysteme in jeder Hinsicht besser zu benutzen war. Die folgenden Feststellungen und Äußerungen basieren auf meiner persönlichen Meinung als Benutzer, der die drei Graphdatenbanksysteme für den selben Zweck intensiv benutzt hat.

Benutzungsoberfläche Die Benutzungsoberfläche der drei Graph-DBMS war ähnlich gut. Positiv bei Neo4j war, dass auch der Datenimport über die Benutzungsoberfläche geschah, während bei den anderen dies über die Konsole gemacht werden musste. Allgemein sticht hier aber keine der Datenbanken als besonders positiv oder negativ heraus.

Anfragesprache Die intuitivste Anfragesprache dieser Graphdatenbanken ist Cypher von Neo4j. Der Graph, der "gematcht" wird, wird praktisch genauso aufgeschrieben, wie man ihn malen würde. Dies ist leicht nachzuvollziehen, und leicht zu lernen und zu beherrschen.

Bei OrientDB ist es im Prinzip auch so, nur dass bei OrientDB sehr viel mehr Text benötigt wird, da z.B. der Neo4j-Pfeil '->' in OrientDB durch '.outE()' eher unschön dargestellt wird. Am unübersichtlichsten ist die Anfragesprache von ArangoDB, AQL, vor allem aufgrund der For-Schleifen.

Datenimport Während der Import von Daten in Neo4j besonders leicht ist, ist dieser bei den anderen beiden Graphdatenbanken eher unpraktisch. Besonders OrientDB ist beim Datenimport sehr unflexibel. Erstens bietet OrientDB keinen direkten Import von einer csv-Datei an, stattdessen muss, wie in Kapitel 4.2.1 beschrieben, eine ETL-Datei erstellt werden. Viel schlimmer ist aber, dass bei OrientDB die Daten absolut korrekt sein müssen. Sollte z.B. in der Bewertungen-Datei eine Bewertung eines Films aufgelistet sein, der nicht in der Datenbank vorhanden ist, so wird der gesamte Datenimport abgebrochen. Daraufhin müssen alle vor dem Fehler importierten Daten wieder gelöscht werden, der Fehler beseitigt werden, und der Import neu gestartet werden, in der Hoffnung, dass es nicht noch einen Fehler gibt. Alle Dateien müssen also absolut fehlerfrei sein. Dies ist ein großes Manko, gerade bei Big Data Datenbanken, bei denen die Daten nicht immer problemlos perfekt gesäubert werden können. Bei ArangoDB ist der Import extrem schnell, es muss nur darauf geachtet werden, dass die Header der csv-Dateien korrekt umbenannt werden, und dass in der Benutzungsoberfläche angegeben wurde, dass die Daten permanent gespeichert werden sollen. Ansonsten werden die Daten nämlich nur im RAM gehalten (siehe Kapitel 4.3.1).

Support Einer der größten Vorteile von Neo4j, ist dessen Beliebtheit. Durch die vielen Benutzer gibt es für die meisten Fragen schon Antworten auf mehreren Forum-Seiten (z.B. Stack-Exchange-Inc (2018)). Auch die Neo4j Website ist sehr inhaltsreich, da Hilfestellungen für sehr viele konkrete Probleme angeboten werden, und eine sehr umfangreiche Dokumentation auf der Website einsehbar ist. Es ist zweifelsfrei ein enormer Vorteil, dass Neo4j die größte Community aller Graph-DBMS hat (s. Edlich u. a. (2011), S. 302). Bei ArangoDB und OrientDB ist die Anzahl an hilfreichen Forumsbeiträgen sehr begrenzt, die Fragen müssen in ein Forum gestellt werden in der Hoffnung auf eine baldige, hilfreiche Antwort. Alternativ muss sehr viel experimentiert werden. Beides führt zu einem größeren Zeitaufwand.

6. Fazit

In diesem Fazit werden diese Bachelorarbeit resümmiert und die Ergebnisse des Vergleichs und die daraus resultierenden Schlüsse zusammenfassend dargestellt.

Nach einer Einführung in das Themengebiet der Graphentheorie, sowie das Prinzip der Graphdatenbanksysteme und Empfehlungssysteme in Kapitel 2, wurde in Kapitel 3 der eigene Entwurf eines Filmempfehlungssystems vorgestellt. Dieser wurde drei Mal implementiert, jeweils für Neo4j, OrientDB, und ArangoDB. Anhand dieses Filmempfehlungssystems wurden die drei Graphdatenbanksysteme verglichen. Der Speicherbedarf, die Bearbeitungsdauer und die bewältigte Anfrage- und Benutzerlast wurden notiert und auf Grund dieser Ergebnisse konnte geschlussfolgert werden, welches der genannten Graphdatenbanksysteme für ein Empfehlungssystem am besten geeignet ist.

Neo4j wird seiner Popularität absolut gerecht und schlägt in den meisten Aspekten, die für ein Empfehlungssystem wichtig sind, die beiden anderen Graphdatenbanken. Neo4j berechnet am schnellsten, verträgt die größten Datenmengen, ist sehr benutzerfreundlich und gerade wegen dieser Popularität ist bei Problemen sehr viel Hilfe im Internet zu finden.

ArangoDB hat in drei Bereichen die beste Leistung gezeigt: Speicherbedarf, Importschnelligkeit und Kapazitätstest. Alles davon ist bei Big-Data sehr wichtig. Dennoch ist durch die (manchmal extrem) langsamen Antwortzeiten ArangoDB nur mittelmäßig für ein BigData Empfehlungssystem geeignet. Außerdem ist es negativ, das ArangoDB keine Replikation erlaubt.

OrientDB konnte in keinem Bereich herausragen, und zeigte immer durchschnittliche bis sehr schlechte Werte. OrientDB ist für ein Big Data-Empfehlungssystem eher nicht geeignet.

Die Ergebnisse dieser Arbeit sind allerdings nicht als vollständige und abschließende Bewertung der Graphdatenbanksysteme zu sehen. Hier wurde nur in Hinsicht auf Empfehlungssysteme verglichen. Andere Aspekte von Graphdatenbanksystemen, die für andere Applikationen

von großer Wichtigkeit sein könnten, wie z.B. das Finden des kürzesten Weges, oder das Ausführen von Graphalgorithmen wie Dijkstra, wurden hier nicht verglichen. Das Ziel dieser Arbeit war die Eignung für ein Empfehlungssystem, bzw. Filmempfehlungssystem, zu beschreiben.

A. Anhang

Tabelle A.1.: Zeitmessungen Datenimport

Messung	Neo4j (ms)	OrientDB	ArangoDB
Filme	3078, 2167, 3568	2250, 5159, 5159	2499, 2256, 3260
Genres	144, 116, 188	198, 186, 198	587, 764, 866
HAS_GENRE	118181, 45738, 59439	46327, 24021, 23928	4065, 3871, 3959
Benutzer	3600, 4513, 5546	18833, 18679, 18445	7939, 25556, 7489
Tags	171, 174, 158	908, 830, 885	734, 2099, 730
HAS_TAG	1710, 1993, 2277	20065, 22192, 22652	1303, 1508, 1686
RATED (100.000 Bewertungen)	8477, 8958, 8965	79574, 77275, 75441	8386, 22970, 6515
RATED (1.000.000 Bewertungen)	102743, 103086, 102554	706631, 678034, 660659	66420, 627842, 62437
RATED (10.000.000 Bewertungen)	1199144, 1160640, 1184463	6937333, 6621936, 6307464	914772, 7249834, 739522

Tabelle A.2.: Zeitmessungen inhaltsbasiert (100.000 Bewertungen)

Messung	Neo4j	OrientDB	ArangoDB
Für den Film mit den meisten (20) Tags/-Genres (Reservoir Dogs (1992))	1687, 1215, 1197	16872, 14206, 13597	1775, 1762, 1805
Für einen Film mit 10 Tags/Genres (State of Grace (1990))	1233, 1243, 1259	12131, 9093, 9126	1887, 1874, 1845
Für den Film mit 3 Tags/Genres (mind. 3) (Sudden Death (1995))	1196, 1440, 1175	1830, 1814, 1824	329, 339, 341
Für den Benutzer mit den meisten 5-Sterne Bewertungen (ID:564)	1043, 673, 625	164501, 169304, 159774	16856123, 16840306, 16806382
Für einen Benutzer mit einer mittleren Anzahl an 5-Sterne Bewertungen (ID:30)	809, 548, 520	74399, 74576, 79149	5316635, 5318587, 5318281
Für den Benutzer mit den wenigsten 5-Sterne Bewertungen (mind. 3) (ID:645)	11, 9, 8	599, 613, 622	2859, 2862, 2906

Tabelle A.3.: Zeitmessungen kollaborativ (100.000 Bewertungen)

Messung	Neo4j	OrientDB	ArangoDB
Für den Film mit den meisten 5-Sterne Bewertungen (Shawshank Redemption, The (1994))	93, 75, 67	1106, 683, 625	796, 757, 748
Für einen Film mit einer mittleren Anzahl an 5-Sterne Bewertungen (American Beauty (1999))	68, 59, 55	577, 568, 590	600, 579, 575
Für den Film mit den wenigsten 5-Sterne Bewertungen (mind. 3) (Gone in 60 Seconds)	26, 23, 23	269, 273, 278	69, 66, 67
Für den Benutzer mit den meisten 5-Sterne Bewertungen (ID:564)	2634, 2308, 2426	453928, 437899, 429314	3314037, 3293289, 3300934
Für einen Benutzer mit einer mittleren Anzahl an 5-Sterne Bewertungen (ID:30)	3644, 3945, 3527	180754, 183828, 185825	1467404, 1457574, 1474817
Für den Benutzer mit den wenigsten 5-Sterne Bewertungen (mind. 3) (ID:645)	28, 34, 30	180, 184, 175	598, 581, 564

Tabelle A.4.: Zeitmessungen hybrid (100.000 Bewertungen)

Messung	Neo4j	OrientDB	ArangoDB
Für den Film mit den meisten 5-Sterne Bewertungen (Shawshank Redemption, The (1994))	20071, 18692, 18534	55528, 54264, 52596	132129, 131480, 131222
Für einen Film mit einer mittleren Anzahl an 5-Sterne Bewertungen (American Beauty (1999))	19364, 19617, 19186	41238, 43004, 46783	83108, 82778, 82841
Für den Film mit den wenigsten 5-Sterne Bewertungen (mind. 3) (Gone in 60 Seconds)	18806, 18865, 18940	572, 605, 579	4900, 4835, 4880
Für den Benutzer mit den meisten 5-Sterne Bewertungen (ID:564)	11428, 11323, 11080	6165294, 6168873, 6279120	22262944, 22299243, 22421857
Für einen Benutzer mit einer mittleren Anzahl an 5-Sterne Bewertungen (ID:30)	19594, 19935, 19665	8218935, 8233410, 8401456	9502673, 9574635, 9564972
Für den Benutzer mit den wenigsten 5-Sterne Bewertungen (mind. 3) (ID:645)	26, 23, 23	8090, 7961, 7998	8131, 7548, 7518

Tabelle A.5.: Zeitmessungen inhaltsbasiert (1.000.000 Bewertungen)

Messung	Neo4j	OrientDB	ArangoDB
Für den Film mit den meisten (20) Tags/-Genres (Reservoir Dogs (1992))	1644, 1097, 1112	1. Messung (Messfehler): 43449, 44098, 43380; 2. Messung: 15490, 11462, 10996	1874, 1863, 1854
Für einen Film mit 10 Tags/Genres (State of Grace (1990))	1199, 1185, 1180	1. Messung (Messfehler): 35147, 41801, 37837; 2. Messung: 9302, 9418, 9326	1974, 1968, 1958
Für den Film mit 3 Tags/Genres (mind. 3) (Sudden Death (1995))	1132, 1119, 1177	1. Messung (Messfehler): 5936, 7864, 6154; 2. Messung: 1832, 1912, 1899	391, 363, 368
Für den Benutzer mit den meisten 5-Sterne Bewertungen (ID:4277)	2868, 2408, 2337	529150, 595957, 574586	23462083, 23577685, 23528931
Für einen Benutzer mit einer mittleren Anzahl an 5-Sterne Bewertungen (ID:1698)	1302, 1310, 1298	245061, 189753, 203285	6410089, 6365095, 6376926
Für den Benutzer mit den wenigsten 5-Sterne Bewertungen (mind. 3) (ID:5899)	19, 11, 10	3166, 2598, 2398	4260, 4269, 4264

Tabelle A.6.: Zeitmessungen kollaborativ (1.000.000 Bewertungen)

Messung	Neo4j	OrientDB	ArangoDB
Für den Film mit den meisten 5-Sterne Bewertungen (American Beauty (1963))	554, 513, 466	13656, 6517, 6091	12760, 12332, 12231
Für einen Film mit einer mittleren Anzahl an 5-Sterne Bewertungen (Casablanca (1942))	353, 315, 327	4258, 4369, 4406	9817, 7927, 7918
Für den Film mit den wenigsten 5-Sterne Bewertungen (mind. 3) (Wishmaster (1997))	24, 26, 23	446, 468, 455	240, 78, 69
Für den Benutzer mit den meisten 5-Sterne Bewertungen (ID:4277)	1900138, 1899121, 1921693	> 10 Std.	> 10 Std.
Für einen Benutzer mit einer mittleren Anzahl an 5-Sterne Bewertungen (ID:1698)	1117865, 1087173, 1079495	> 10 Std.	23822009, 23711183, 23905616
Für den Benutzer mit den wenigsten 5-Sterne Bewertungen (mind. 3) (ID:5899)	2902, 1709, 1505	12027, 8127, 8506	28773, 28506, 28382

Tabelle A.7.: Zeitmessungen hybrid (1.000.000 Bewertungen)

Messung	Neo4j	OrientDB	ArangoDB
Für den Film mit den meisten 5-Sterne Bewertungen (American Beauty (1963))	1698, 1426, 1285	4359776, 3934758, 4122148	1652630, 1622369, 1625965
Für einen Film mit einer mittleren Anzahl an 5-Sterne Bewertungen (Casablanca (1942))	1309, 1220, 1193	2618036, 2627491, 2592540	1261917, 1264433, 1275312
Für den Film mit den wenigsten 5-Sterne Bewertungen (mind. 3) (Wishmaster (1997))	1061, 1176, 1023	1069, 1151, 1209	5012, 5002, 4976
Für den Benutzer mit den meisten 5-Sterne Bewertungen (ID:4277)	11968, 12290, 12247	> 10 Std.	> 10 Std.
Für einen Benutzer mit einer mittleren Anzahl an 5-Sterne Bewertungen (ID:1698)	16227, 16224, 16764	> 10 Std.	> 10 Std.
Für den Benutzer mit den wenigsten 5-Sterne Bewertungen (mind. 3) (ID:5899)	351, 386, 350	> 10 Std.	1172897, 1174303, 1170614

Tabelle A.8.: Zeitmessungen inhaltsbasiert 10.000.000)

Messung	Neo4j	OrientDB	ArangoDB
Für den Film mit den meisten (20) Tags/-Genres (Reservoir Dogs (1992))	490, 386, 166	10267, 10131, 10112	1955, 1892, 1920
Für einen Film mit 10 Tags/Genres (State of Grace (1990))	354, 169, 133	8701, 8677, 8637	2587, 1994, 1978
Für den Film mit 3 Tags/Genres (mind. 3) (Sudden Death (1995))	139, 171, 149	2008, 1926, 1778	608, 367, 364
Für den Benutzer mit den meisten 5-Sterne Bewertungen (ID:18278)	14985, 14619, 15219	429906, 448902, 466999	> 10 Std.
Für einen Benutzer mit einer mittleren Anzahl an 5-Sterne Bewertungen (ID: 54903)	8182, 6219, 6161	224242, 217905, 219501	17488579, 17509603, 17483067
Für den Benutzer mit den wenigsten 5-Sterne Bewertungen (mind. 3) (ID: 6321)	150, 38, 34	855, 893, 972	4513, 4427, 4433

Tabelle A.9.: Zeitmessungen kollaborativ 10.000.000)

Messung	Neo4j	OrientDB	ArangoDB
Für den Film mit den meisten 5-Sterne Bewertungen (Shawshank Redemption, The (1994))	16808, 17842, 17031	43991, 40655, 40821	103649, 98980, 95161
Für einen Film mit einer mittleren Anzahl an 5-Sterne Bewertungen (Raiders of the Lost Ark (1981))	16663, 16790, 16415	40522, 35551, 34807	103861, 77905, 79482
Für den Film mit den wenigsten 5-Sterne Bewertungen (mind. 3) (New York Minute (2004))	9913, 9800, 10017	806, 502, 492	347, 104, 107
Für den Benutzer mit den meisten 5-Sterne Bewertungen (ID:18278)	37163769, 30162245, 30359622	> 10 Std.	> 10 Std.
Für einen Benutzer mit einer mittleren Anzahl an 5-Sterne Bewertungen (ID: 54903)	21158672, 17837538, 16893559	> 10 Std.	> 10 Std.
Für den Benutzer mit den wenigsten 5-Sterne Bewertungen (mind. 3) (ID: 6321)	8129, 7821, 7829	48883, 35488, 36169	132332, 124858, 127683

Tabelle A.10.: Zeitmessungen hybrid 10.000.000)

Messung	Neo4j	OrientDB	ArangoDB
Für den Film mit den meisten 5-Sterne Bewertungen (Shawshank Redemption, The (1994))	75376, 69066, 68343	5142136, 5189628, 5169232	15902972, 15813677, 15902342
Für einen Film mit einer mittleren Anzahl an 5-Sterne Bewertungen (Raiders of the Lost Ark (1981))	114037, 95352, 95052	2678281, 2680131, 2669810	25044316, 25032935, 24846968
Für den Film mit den wenigsten 5-Sterne Bewertungen (mind. 3) (New York Minute (2004))	1870, 236, 77	1456, 693, 572	2717, 2363, 2367
Für den Benutzer mit den meisten 5-Sterne Bewertungen (ID:18278)	682822, 697663, 683778	> 10 Std.	> 10 Std.
Für einen Benutzer mit einer mittleren Anzahl an 5-Sterne Bewertungen (ID: 54903)	526951, 516257, 519621	> 10 Std.	> 10 Std.
Für den Benutzer mit den wenigsten 5-Sterne Bewertungen (mind. 3) (ID: 6321)	26182, 23706, 23646	> 10 Std.	6333790, 6426127, 6441930

Tabelle A.11.: Kapazitätentest Neo4J für eine inhaltsbasierte Empfehlung für "Reservoir Dogs (1992)"

Anzahl Benutzer	Schnellste Antwort (ms)	Langsamste Antwort (ms)	ø Antwortzeit (ms)	% gescheiterte Anfragen	Anfragen pro Sekunde
1000 (100 pro Sekunde)	4162	58460	32361	0	14,493
2000 (100 pro Sekunde)	1173	105445	54756	0	15,944
3000 (100 pro Sekunde)	1281	145035	75634	0	16,37
4000 (100 pro Sekunde)	1488	187266	96945	0	17,524
5000 (100 pro Sekunde)	1644	231134	120064	0	17,724
10000 (100 pro Sekunde)	Server abgestürzt nach 95 Sekunden bei 7870 aktiven Benutzern und 1454 beendeten Anfragen				
1000 (10 pro Sekunde)	449	3375	967	0	16,912
5000 (500 pro Sekunde)	1426	283811	143243	1	17,272

Tabelle A.12.: Kapazitätentest OrientDB für eine inhaltsbasierte Empfehlung für “Reservoir Dogs (1992)”

Anzahl Benutzer	Schnellste Antwort	Langsamste Antwort	ø Antwortzeit	% gescheiterte Anfragen	Anfragen pro Sekunde
1000 (100 pro Sekunde)	>200000	>200000	>200000	1000 (durch timeout)	-
2000 (100 pro Sekunde) 3000 (100 pro Sekunde) 4000 (100 pro Sekunde) 5000 (100 pro Sekunde) 10000 (100 pro Sekunde)	Server akzeptiert höchstens 1000 Benutzer, und antwortet außerdem keine Anfrage in weniger als 200000ms				
1000 (10 pro Sekunde)	>200000	>200000	>200000	1000 (durch timeout)	-
5000 (500 pro Sekunde)	Server akzeptiert höchstens 1000 Benutzer, und antwortet außerdem keine Anfrage in weniger als 200000ms				

Tabelle A.13.: Kapazitätentest ArangoDB für eine inhaltsbasierte Empfehlung für "Reservoir Dogs (1992)"

Anzahl Benutzer	Schnellste Antwort	Langsamste Antwort	ø Antwortzeit	% gescheiterte Anfragen	Anfragen pro Sekunde
1000 (100 pro Sekunde)	34476	>300000	>300000	616 (timeout)	3,215
2000 (100 pro Sekunde)	34093	>300000	>300000	1620 (timeout)	6,231
3000 (100 pro Sekunde)	35831	>300000	>300000	2620 (timeout)	9,063
4000 (100 pro Sekunde)	37379	>300000	>300000	3622 (timeout)	11,73
5000 (100 pro Sekunde)	33390	>300000	>300000	4623 (timeout)	14,245
10000 (100 pro Sekunde)	Server reagiert nach 8279 Anfragen nicht mehr. 372 Anfragen wurden in weniger als 5 Minuten erfolgreich durchgeführt.				
1000 (10 pro Sekunde)	13341	>300000	>300000	577 (timeout)	2,5
5000 (500 pro Sekunde)	34013	>300000	>300000	4647 (4628 durch timeout)	16,138

Tabelle A.14.: Stresstest Neo4J für eine inhaltsbasierte Empfehlung für "Reservoir Dogs (1992)"

Anzahl Benutzer	Schnellste Antwort	Langsamste Antwort	ø Antwortzeit	Gescheiterte Anfragen	Anfragen pro Sekunde
100	497	6951	3340	0	14,5
200	709	11599	6016	0	17,833
500	764	27945	14017	0	17,517
750	769	41425	20746	0	18,071
1000	804	53356	27073	1	18,018
1500	907	80439	40715	0	18,193
2000	1109	107368	54072	0	18,255
5000	1867	266753	133828	0	18,348

Tabelle A.15.: Stresstest OrientDB für eine inhaltsbasierte Empfehlung für “Reservoir Dogs (1992)”

Anzahl Benutzer	Schnellste Antwort	Langsamste Antwort	ø Antwortzeit	Gescheiterte Anfragen	Anfragen pro Sekunde
100	>300000	>300000	>300000	100 (durch timeout)	-
200	>300000	>300000	>300000	200 (187 durch timeout)	-
500	>300000	>300000	>300000	500 (235 durch timeout)	-
750	>300000	>300000	>300000	750 (549 durch timeout)	-
1000	>300000	>300000	>300000	1000 (621 durch timeout)	-
1500 2000 5000	Server akzeptiert höchstens 1000 Benutzer, und antwortet außerdem keine Anfrage in weniger als 300000ms				

Tabelle A.16.: Stresstest ArangoDB für eine inhaltsbasierte Empfehlung für “Reservoir Dogs (1992)”

Anzahl Benutzer	Schnellste Antwort	Langsamste Antwort	ø Antwortzeit	Gescheiterte Anfragen	Anfragen pro Sekunde
100	32618	75605	56249	0	1,316
200	37111	164112	94752	21	1,331
500	33740	>300000	115944	148 (117 durch Timeout)	1,754
750	31327	>300000	>300000	416 (368 durch Timeout)	2,621
1000	39677	>300000	>300000	723 (677 durch Timeout)	3,447
1500	45764	>300000	>300000	1203 (1168 durch Timeout)	5.07
2000	36565	>300000	>300000	1672 (1635 durch Timeout)	6,719
5000	37418	>300000	>300000	4684 (4478 durch Timeout)	16,587

Abbildungsverzeichnis

3.1. Deploymentsicht des Testsystems 15
3.2. Kollaborative Filmempfehlung für einen Film 16
3.3. Ausgabe für Kollaborative Filmempfehlung für den Film “Star Wars: Episode IV” 17
3.4. Kollaborative Filmempfehlung für einen Benutzer 17
3.5. Inhaltsbasierte Filmempfehlung für einen Film 18
3.6. Inhaltsbasierte Filmempfehlung für einen Benutzer 19
3.7. Hybride Filmempfehlung für einen Film 20
3.8. Hybride Filmempfehlung für einen Benutzer 21
3.9. Format der Original ‘movies.csv’ Datei 21
3.10. Format der bearbeiteten ‘movies.csv’ Datei für die Erstellung von Film-zu-Genre Beziehungen 22

4.1. ETL für den Import von Filmen in OrientDB 27
4.2. ETL für den Import von Bewertungen in OrientDB 28

5.1. Vergleich des Zeitaufwands für den Datenimport 35
5.2. Vergleich des Zeitaufwands für die inhaltsbasierte Filmempfehlung für einen Film 36
5.3. Vergleich des Zeitaufwands für die inhaltsbasierte Filmempfehlung für einen Benutzer 37
5.4. Vergleich des Zeitaufwands für die kollaborative Filmempfehlung für einen Film 38
5.5. Vergleich des Zeitaufwands für die kollaborative Filmempfehlung für einen Benutzer 39
5.6. Vergleich des Zeitaufwands für die hybride Filmempfehlung für einen Film . . 40
5.7. Vergleich des Zeitaufwands für die hybride Filmempfehlung für einen Benutzer 41

Tabellenverzeichnis

3.1. Zusammenfassung der Attribute jedes Datensatzes für Knoten 23
3.2. Zusammenfassung der Attribute jedes Datensatzes für Kanten 23

5.1. Speicherbedarf 34

A.1. Zeitmessungen Datenimport 46
A.2. Zeitmessungen inhaltsbasiert (100.000 Bewertungen) 46
A.3. Zeitmessungen kollaborativ (100.000 Bewertungen) 47
A.4. Zeitmessungen hybrid (100.000 Bewertungen) 47
A.5. Zeitmessungen inhaltsbasiert (1.000.000 Bewertungen) 48
A.6. Zeitmessungen kollaborativ (1.000.000 Bewertungen) 48
A.7. Zeitmessungen hybrid (1.000.000 Bewertungen) 49
A.8. Zeitmessungen inhaltsbasiert 10.000.000) 49
A.9. Zeitmessungen kollaborativ 10.000.000) 50
A.10. Zeitmessungen hybrid 10.000.000) 50
A.11. Kapazitätentest Neo4J für eine inhaltsbasierte Empfehlung für “Reservoir Dogs (1992)” 51
A.12. Kapazitätentest OrientDB für eine inhaltsbasierte Empfehlung für “Reservoir Dogs (1992)” 52
A.13. Kapazitätentest ArangoDB für eine inhaltsbasierte Empfehlung für “Reservoir Dogs (1992)” 53
A.14. Stresstest Neo4J für eine inhaltsbasierte Empfehlung für “Reservoir Dogs (1992)” 53
A.15. Stresstest OrientDB für eine inhaltsbasierte Empfehlung für “Reservoir Dogs (1992)” 54
A.16. Stresstest ArangoDB für eine inhaltsbasierte Empfehlung für “Reservoir Dogs (1992)” 55

Listings

4.1. Beispiel: Filmknotenimport Neo4j 25
4.2. Beispiel: Kantenimport Neo4j 25
4.3. Beispiel: Pfad in Neo4j 26
4.4. Beispiele: Pattern-Matching mit Properties in Neo4j 26
4.5. Beispiel: Kollaborative Filmempfehlung in Neo4j 26
4.6. Beispiel: Matching eines Knotens in OrientDB 30
4.7. Beispiel: Hybride Filmempfehlung in OrientDB 30
4.8. Beispiel: Benötigtes Header-Format der Film-csv-Datei für ArangoDB 31
4.9. Format des Knotenimports bei ArangoDB 31
4.10. Beispiel: Kanten-Matching in ArangoDB 32
4.11. Beispiel: Kollaborative Filmempfehlung in ArangoDB 32

Literaturverzeichnis

[Adomavicius und Tuzhilin 2005] Adomavicius, Gediminas ; Tuzhilin, Alexander: Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions In: *IEEE Transactions on Knowledge and Data Engineering* Volume: 17 (2005) Issue: 6, June, S. 734 - 749. 25 April 2005. https://ieeexplore.ieee.org/document/1423975/author#authors, Abruf: 2018-10-23. (2005)

[ArangoDB 2018] ArangoDB: *Distributed team for a Distributed technology* https://www.arangodb.com/about-arangodb/, Abruf: 2018-10-07. (2018)

[Claypool u. a. 1999] Claypool, Mark ; Gokhale, Anuja ; Miranda, Tim ; Murnikov, Pavel ; Netes, Dmitry ; Sartin, Matthew: "Combining Content-Based and Collaborative Filters in an Online Newspaper", Computer Science Department, Worcester Polytechnic Institute, Worcester, USA, ACM SIGIR Workshop on Recommender Systems - Implementation and Evaluation, August 19, 1999 in Berkeley CA, USA. https://web.cs.wpi.edu/ claypool/papers/content-collab/content-collab.ps, Abruf: 2018-10-23. (1999)

[Diestel 2010] Diestel, Reinhard: *Graphentheorie.* 4. Aufl. Berlin ; Heidelberg : Springer, 2010. – ISBN 978-3-642-14911-5

[Edlich u. a. 2010] Edlich, Stefan ; Friedland, Achim ; Hampe, Jens ; Brauer, Benjamin: *NoSQL : Einstieg in die Welt nichtrelationaler Web-2.0-Datenbanken.* 1. Aufl. München : Hanser, 2010. – ISBN 978-3-446-42355-8

[Edlich u. a. 2011] Edlich, Stefan ; Friedland, Achim ; Hampe, Jens ; Brauer, Benjamin ; Brückner, Markus: *NoSQL : Einstieg in die Welt nichtrelationaler Web-2.0-Datenbanken.* 2. Aufl. München : Hanser, 2011. – ISBN 978-3-446-42753-2

[Gatling-Corp 2018] Gatling-Corp: *Homepage* https://gatling.io, Abruf: 2018-10-03. (2018)

[Grouplens 2018] Grouplens: *MovieLens* https://grouplens.org/datasets/movielens/, Abruf: 2018-11-06. (2018)

[Hunger 2014] Hunger, Michael: *Neo4j 2.0 : Eine Graphdatenbank für alle.* 1. Aufl. Frankfurt am Main : entwickler.press, 2014. – ISBN 978-3-86802-654-2

[Junghanns 2014] Junghanns, Martin: *Untersuchung zur Eignung von Graphdatenbanksystemen für die Analyse von Informationsnetzwerken.* Universität Leipzig, Institut für Informatik, Fakultät für Mathematik und Informatik Abteilung Datenbanken, Masterarbeit, http://lips.informatik.uni-leipzig.de/files/ma_junghanns.pdf Abruf: 2018-11-06. (2014)

[Kaltenmaier 2016] Kaltenmaier, Patrick: *Wikipedia in times of BigData – Einsatz von NoSQL-Systemen zur Verwaltung und Analyse von Wikipediainhalten.* Friedrich-Alexander-Universität Erlangen-Nürnberg Technische Fakultät, Lehrstuhl für Informatik 6 – Datenmanagement, Masterarbeit https://osr.cs.fau.de/wp-content/uploads/2016/07/kaltenmaier-2016-arbeit.pdf Abruf: 2018-11-06. (2016)

[Klahold 2009] Klahold, André: *Empfehlungssysteme, Recommender Systems — Grundlagen, Konzepte und Lösungen.* 1. Aufl. Wiesbaden : Vieweg+Teubner, 2009. – ISBN 978-3-8348-9558-5

[Kumar und Babu 2015] Kumar, A. V. ; Babu, G. A.: Domain Suitable Graph Database Selection: A Preliminary Report, In: *3rd International Conference on Advances in Engineering Sciences Applied Mathematics (ICAESAM'2015) 23. - 24. März, 2015 London (UK)*, IIENG, 2015. - S. 26-29, http://iieng.org/images/proceedings_pdf/1994E0315021.pdf Abruf: 2018-11-06. (2015)

[MacKenzie u. a. 2013] MacKenzie, Ian ; Meyer, Chris ; Noble, Steve: *How retailers can keep up with consumers*, https://www.mckinsey.com/industries/retail/our-insights/how-retailers-can-keep-up-with-consumers, Abruf: 2018-10-05. (2013)

[Molla 2018] Molla, Rani: *Netflix now has nearly 118 million streaming subscribers globally* https://www.recode.net/2018/1/22/16920150/netflix-q4-2017-earnings-subscribers, Abruf: 2018-11-06. (2018)

[Neo4j 2018] Neo4j, Inc.: *Homepage* https://neo4j.com, Abruf: 2018-10-07. (2018)

[Orientdb 2018a] Orientdb: *ETL* https://orientdb.com/docs/2.2.x/ETL-Introduction.html, Abruf: 2018-11-06. (2018)

[Orientdb 2018b] Orientdb: *Homepage* https://orientdb.com, Abruf: 2018-11-06. (2018)

[Predictive-Analytics-Today 2018] Predictive-Analytics-Today: *Top 27 Graph Databases* https://www.predictiveanalyticstoday.com/top-graph-databases/, Abruf: 2018-11-06. (2018)

[Schneeweiss 1985] Schneeweiss, Winfrid G.: *Grundbegriffe der Graphentheorie für praktische Anwendungen.* 1. Aufl. Heidelberg : Hüthig, 1985. – ISBN 3-7785-1231-5

[Stack-Exchange-Inc 2018] Stack-Exchange-Inc: *Stack Overflow Homepage* https://stackoverflow.com, Abruf: 2018-11-06. (2018)

[Temme 2015] Temme, Holger: *e-commerce: Graphdatenbanken für Online-Empfehlungen in Echtzeit*, https://digital-magazin.de/graphdatenbanken-online-empfehlungen-echtzeit/, Abruf: 2018-10-18. (2015)

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 14. November 2018 Ditmar Lange