Bachelorarbeit

Pawel Albrant

Eine vergleichende Untersuchung des Peer-to-Peer DBMS Cassandra mit Riak und PostgreSQL.

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Pawel Albrant

Eine vergleichende Untersuchung des Peer-to-Peer DBMS Cassandra mit Riak und PostgreSQL.

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. Dr. ing. Olaf Zukunft
Zweitgutachter: Prof. Dr. Stefan Sarstedt

Eingereicht am: 14. Juli 2015

Pawel Albrant

Thema der Arbeit
Eine vergleichende Untersuchung des Peer-to-Peer DBMS Cassandra mit Riak und PostgreSQL.

Stichworte
Datenbank, Cassandra DB, Riak, NoSQL, DBMS, verteilte Datenbank, Peer-to-Peer, Vergleich, PostgreSQL

Kurzzusammenfassung
Die folgende Bachelorarbeit bietet einen Vergleich von relationalen und nicht-relationalen Datenbanken. Dabei wird auf Kategorien der nicht relationalen Datenbanken eingegangen. Im weiteren Verlauf der Arbeit werden eine relationale und zwei nicht-relationale Datenbanken verglichen und wird auf Gemeinsamkeiten und Unterschiede eingegangen.

Pawel Albrant

Title of the paper
A comparative exploration of peer-to-peer DBMS Cassandra with Riak and PostgreSQL.

Keywords
Database, Cassandra DB, Riak, NoSQL, DBMS, Distributed Database, Peer-to-Peer, comparing, PostgreSQL

Abstract
In this bachelor thesis I compare the relational and non-relational Data Base Systems. The categories of non-relational databases are discussed. In the course of the work a relational and two of non-relational databases are compared, discussed similarities and differences.

Inhaltsverzeichnis

1. Einleitung

1.1. Motivation

„Future users of large data bases must be protected from having to know how the data is organized in the machine (the internal representation)...
Activities of users at terminals and most application programs should remain unaffected when the internal representation of data is changed and even when some aspects of the external representation are changed. Changes in data representation will often be needed as a result of changes in query, update, and report traffic and natural growth in the types of stored information."

(– E.F. Codd[1])

Seit Mitte der achtziger Jahre haben sich relationale Datenbankmanagementsysteme als die dominierenden DBMS etablieren können. Bereits in den Jahren zuvor haben sie viele der heute sogenannten vor-relationalen Systeme verdrängt, wie beispielsweise das Netzwerk-Datenbankmodell, das besonders in den siebziger Jahren eine große Marktbedeutung hatte. In allen erdenklichen Unternehmens- und Wirtschaftsbereichen, in denen Datenbanken eingesetzt wurden, war das relationale Datenbankmanagementsystem (im weiteren auch als RDBMS) das mit Abstand Häufigste.

In Anbetracht der Schnelligkeit der IT-Welt, die ständig mit neuen Innovationen, Veränderungen und technologischen Fortschritten aufwarten kann, stellen dreißig Jahre Dominanz eine beachtliche Leistung dar. Gegenwärtig jedoch befindet sich die IT in einem weiteren Wandel und dieser geht auch am Datenmanagement nicht vorbei. Das Cloud Computing ist der neue "Hype" der IT-Welt, dem immer mehr Unternehmen folgen[2]. Es birgt beträchtliche Potenziale, die jedoch nur abgerufen werden können, wenn alle Aspekte der Cloud-Plattform berücksichtigt werden. Einer dieser Aspekte ist die dynamische Skalierbarkeit oder die Fähigkeit, Server nach Bedarf hinzuschalten zu

[1] [Cod70]
[2] [Kur08]

können und Verarbeitungsprozesse oder Datenverwaltung auf diese zu verlagern bzw. auszuweiten.

Relationale Datenbankmanagementsysteme haben durch die Einhaltung des ACID-Transaktionskonzepts[3] stets ein strenges und solides Datenmanagement sichergestellt. Dies kann sich nun im Züge der Umstellung auf einen Cloud-Betrieb als unflexibel erweisen und wird in seiner bisherigen Form immer schwerer durchsetzbar sein. Das Cloud-Computing-Konzept baut auf verteilten Systemen auf, für welche die Begriffe Konsistenz", "Verfügbarkeit" und "Partitionstoleranz" eine zentrale Rolle spielen. Diese sind jedoch erwiesenermaßen nicht gleichzeitig realisierbar, was wiederum das Aufkommen eines neuen Transaktionskonzepts und einer neuen Art von Datenbanken begünstigt hat.

Diese Cloud-Datenbanken sind speziell für den Betrieb in der gleichnamigen Umgebung entwickelt und ausgerichtet worden. Ihre Datenmodelle sind vielfältig und weichen vom relationalen Tabellenmodell ab. Darüber hinaus erfolgt die Kommunikation nicht mehr über die, aus dem RDBMS bekannte Abfragesprache SQL (Structured Query Language), sondern über "gewöhnliche" Programmiersprachen. Viele Unternehmen, insbesondere jene, die stark vom Internet abhängig sind, haben bereits eine Cloud-Datenbank mit einem NoSQL-Datenmodell übernommen. Die überwiegende Anzahl der Datenbanken der NoSQL-Familie wird unter freier Lizenz vertrieben und kann üblicherweise direkt von der Seite des Herstellers heruntergeladen werden.

Ein weiterer Aspekt, auf welchen die rasche Verbreitung der NoSQL-Datenbanken[4] (siehe 2.1.2) zurück geführt werden kann, ist die relativ leichte Handhabung und der konzeptionell auf Einfachheit ausgelegte Aufbau der Datenbanken. Dieser erlaubt auch weniger versierten Datenbankanwendern, sich schnell zurecht finden zu können. In Anbetracht der schemafreien Struktur und der häufig sehr ausführlichen Dokumentation ist, unter Voraussetzung der Kenntnis der entsprechenden Programmiersprache, ein schneller Einstieg in die Materie möglich. Die Freiheiten, welche von den NoSQL-Datenbanken zur Verfügung gestellt werden, müssen jedoch mit einer größeren Sorgfalt seitens des Anwenders bei der Bearbeitung einhergehen, da viele absichernde Regulierungsmechanismen, wie sie das relationalen Datenmodell zur Verfügung stellt, nicht mehr vorhanden sind.

[3] Atomicity, Consistency, Isolation, Durability
[4] Not Only SQL

1.2. Zielsetzung

In letzter Zeit werden die NoSQL-Datenbanken immer beliebter und bekommen eine weite Verbreitung. Sie unterscheiden sich in sehr vielen Aspekten von den traditionellen relationalen Datenbanksystemen.

Diese Arbeit verfolgt daher das Ziel, ein theoretisches Fundament für NoSQL zu erstellen, eine vergleichende Untersuchung zwischen ausgewählten NoSQL-Datenbanken und einer relationalen Datenbank zu machen, die wesentlichen Unterschiede zu zeigen und die Vorteile und Nachteile der jeweiligen System am Ende aufzuzeigen.

1.3. Aufbau der Arbeit

Das erste Kapitel führt den Lesenden in die Thematik ein und stellt kurz den Aufbau der Arbeit vor.

Im zweiten Kapitel wird eine Beschreibung des Begriffes "Datenbank" gegeben und eine kurze Beschreibung der Eigenheiten solcher Systeme, wie zum Beispiel Verteilung, Replikation, Master- und Slave-Knoten, sowie die Grundlagen der Peer-to-Peer-Datenbanken.

Im dritten Kapitel werden die ausgewählten Datenbank-Systeme miteinander verglichen, auf Unterschiede sowie Gemeinsamkeiten eingegangen und akzentiert.
Zu Vergleichen sind Cassandra DB, Riak (beide sind NoSQL DBs) und PostgreSQL als relationale Datenbank.

Im vierte Kapitel werden die Ergebnisse zusammengefasst und diskutiert.

Das fünfte Kapitel beschäftigt sich mit Zusammenfassung und Ausblick

2. Grundlagen

2.1. Klassische und verteilte Datenbanksysteme

„Ein Datenbanksystem ist ein computergestütztes System zur persistenten Speicherung, Pflege und Wiedergewinnung umfangreicher Datenmengen, die im Multi-User-Betrieb unter Wahrung der Datenintegrität über definierte Kommunikationsschnittstellen gemeinsam genutzt werden können.“

(Vorlesung "Datenbanken"[1])

2.1.1. Klassische Datenbanksysteme

Die klassischen Datenbanksysteme zeichnen sich durch ein Client/Server basierendes Modell mit einem zentralisierten Server aus. Diese passen gut zu zentralisierten Unternehmen, da sie eine zentrale Administration des Servers ermöglichen. Normalerweise steuern klassische Datenbanksysteme relationale Datenbanken und benutzen SQL als Abfragesprache.

Eine Datenbank soll beliebige Daten verwalten, Informationen aus diesen Daten liefern und unberechtigten Personen den Zugriff auf die Daten verweigern können. Unter dem Verwalten der Daten versteht man das Eingeben von neuen Daten, das Löschen veralteter Daten sowie das Nachführen bestehender Daten.

Eine Datenbank hat folgende Aufgaben:

einen Zugriff auf die gespeicherten Daten ermöglichen, ohne dass der Benutzer wissen muss, wie diese Daten im System organisiert sind.

eine Zugriffskontrolle leisten, so dass die nicht von einem nicht autorisiertem Benutzer Daten sichten oder manipulieren können. Auch dürfen keine Fehlmanipulationen der autorisierten Benutzer passieren, durch die der Datenbestand unbrauchbar wird.

[1] [Ger14]

eine Änderung der Datenorganisation soll möglich sein, ohne dass der Benutzer seine Anwendungen anpassen muss.

Damit diese Anforderungen erfüllt werden können, muss eine Datenbank einige Werkzeuge und Komponenten bereitstellen:

Datenbankverwaltungssystem (DBMS) bildet den Kern der Datenbank und beinhaltet alle für die Datenverwaltung benötigten Funktionen wie Suchen, Lesen und Schreiben.

Datenbanksprache bildet die Schnittstelle zwischen dem Benutzer und dem DBMS und heißt Structured Query Language (im weiteren Verlauf SQL). SQL hat folgende Aufgabenbereiche:

Datendefinition wird benötigt, um die Datenstruktur, wie zum Beispiel Einrichtung der Tabellen oder Definition der Felder etc, aufzubauen.

Datenmanipulation erlaubt die Datensätze einzugeben, zu löschen und zu verändern.

Datenabfrage filtert die Daten nach frei wählbaren Kriterien.

Datenbanken lassen sich grundsätzlich in drei Hauptkategorien aufteilen[2]:

Hierarchische,

Relationale und objektrelationale,

Objektorientierte Datenbank

Die objektrelationalen Datenbanken bauen auf den relationalen auf, sind aber mit der Technik der Objektorientierung erweitert. Der Unterschied zu den relationalen Datenbanken liegt darin, dass beliebige benutzerdefinierte Datentypen gespeichert werden können, sogar solche, die eine Tabelle definieren, im Gegensatz zu den relationalen DBs, die nur Standarddatentypen verwenden[3].

Die objektorientierten Datenbanken sind aus der objektorientierten Programmierung entstanden[4]. Im Gegensatz zu den relationalen Datenbanken steht bei den objektorientierten Datenbanken nicht eine Tabelle, sondern das Objekt im Zentrum der Betrachtung.

Die hierarchische Datenbank besteht in der einfachsten Form aus einzelnen, sequentiell gespeicherten Daten, die in eine Datei geschrieben werden.

[2] [Ste09]
[3] [Ste09], S. 8
[4] [Kem06]

2.1.2. NoSQL Datenbanksysteme

Auch wenn der Begriff der NoSQL-Datenbank erst seit 2009 für viele Informatiker geläufig ist, gibt es ihn schon lange. Im Jahr 1998 prägte Carlo Strozzi (IBM) den Begriff "NoSQL" im Sinne von "no SQL" (kein SQL) für seine zwar relationale Datenbank, aber ohne SQL-API. Erst seit 2009 wird "NoSQL" als "Not only SQL" verstanden und wurde als Sammelbegriff für zum relationalen Modell "alternative" Datenbankentwicklung verwendet[5].

2.1.2.1. Stärken und Schwächen der NoSQL-Datenbanken

Das Angebot an NoSQL-Datenbanken hat sich rasant vervielfältigt und bietet mittlerweile eine Fülle von Datenbanklösungen mit unterschiedlichsten Datenmodellen, welche Daten auf verschiedene Weise handhaben. Von der Datenverwaltung in Form von Dokumenten, als Schlüssel-Wert-Paar bis hin zu einer grafischen Darstellungsweise steht dem Anwender ein breites Spektrum an Datenmodellen zur Auswahl. Die Entwicklung dieser Datenmodelle geht nicht zuletzt auch deshalb so schnell vonstatten, weil die NoSQL-Bewegung ein loser Zusammenschluss von Entwicklern und Interessenten ist, die keine festen Richtlinien oder bindende Normen vorgeben. Bisher sind keine Standardisierungsvorschriften für die NoSQL-Datenmodelle beschlossen oder herausgegeben worden. So kann praktisch Jeder, der das benötigte Wissen und die Kenntnisse besitzt, seine eigene NoSQL-Datenbank entwickeln, einsetzen oder weitergeben. Ein Beispiel für so einen Fall ist die Entwicklung von Redis. Der daraus resultierende Nachteil ist, das jede NoSQL-Datenbank für sich einzigartig ist und es somit zu Komplikationen bei der Kompatibilität von Applikationen kommen kann, die nur für eine spezielle Datenbank geschrieben wurden. In einem solchen Fall ist auch der Open-Source-Aspekt keine Hilfe. Analysten gehen deshalb davon aus, dass sich NoSQL-Entwickler in den kommenden Jahren unter anderem verstärkt auf die Ausarbeitung einer besseren Applikationskompatibilität zwischen den Datenmodellen konzentrieren werden[6,7]. Die fehlende Standardisierung macht sich zudem auch im Bereich der Datensicherheit bemerkbar. Es wurden bisher keine einheitlichen Richtlinien aufgestellt, die vorgeben, welche und wieviele Möglichkeiten ein NoSQL-Datenmodell zur Verfügung stellen oder mit welchen Methoden und Mechanismen dieser Schutz gewährleistet werden soll. Zwar unterstützt jede Datenbank Sicherungsmaßnahmen, wie beispielsweise das Backup für die Wiederherstellung

[5] [Edl11]
[6] [Lea10]
[7] [Kel10]

nach einem Störfall, intern jedoch bestehen viele konzeptionelle Unterschiede. Die dokumentenorientierte Datenbank MongoDB zum Beispiel startet in einem so genannten Default-Modus, der keinerlei Einschränkungen für den Anwender beinhaltet. Das bedeutet es können nach belieben Datenbanken und Dokumente erstellt, verändert und wieder gelöscht werden. Selbiges gilt für das ebenfalls dokumentenorientierte System CouchDB. Beide bieten zwar die Möglichkeit Benutzergruppen mit unterschiedlichen Zugriffsrechten zu erstellen, allerdings erlauben diese keine Erstellung von Zugriffsbeschränkungen auf bestimmte Datenbankbereiche oder gar einzelne Dokumente. Diese Einstellungen muss der Anwender derzeit mit Hilfe von eigens programmierten Zugriffsroutinen verwirklichen[8,9]. Ein Beispiel für das Fehlen jeglicher interner Sicherheitsmechanismen ist die Graphendatenbank Neo4j, deren Betrieb von einem "trusted environment"(dt. vertraute Umgebung) ausgeht. Sämtliche Sicherheitsvorkehrungen, wie die auf der Internetpräsenz des Entwicklers vorgestellte access control list (ACL), müssen auf der Applikationsebene ausgeführt werden. Angemerkt sei an dieser Stelle, dass sich insbesondere die Open-Source-Datenbankprojekte nach wie vor in einer ständigen Entwicklungsphase befinden. Es ist somit nicht ausgeschlossen, dass zukünftig verbesserte Zugriffsbeschränkungen oder verfeinerte Rechtevergabesysteme installiert werden. Der gegenwärtigen Abwesenheit von Standards zum Trotz haben sämtliche NoSQL-Datenbanken prinzipiell eine Gemeinsamkeit. Sie wurden primär vor dem Hintergrund entwickelt, dass sie bei der Verwaltung von immensen Datenmengen und Größen eine hohe Performanz und Geschwindigkeit bereitstellen können. Auch die Tatsache, dass die stetig wachsenden Datenmengen langfristig Skalierungen der Datenbanken unumgänglich gemacht haben, wurde bei der Entwicklung der NoSQL-Datenbanken einbezogen

2.1.2.2. Datenbankskalierung

Prinzipiell gibt es zwei Strategien um Skalierungen zu verwirklichen. Die erste und einfachste ist die vertikale Skalierung, also die Übertragung von Applikationen und Daten auf einen größeren Rechner. Die vertikale Skalierung funktioniert für Daten einwandfrei, ist jedoch in vielerlei Hinsicht Einschränkungen unterworfen. Die offensichtlichste ist, dass über kurz oder lang die Kapazitäten selbst der größten Maschine dem Datenvolumen nicht mehr gewachsen sind. Die vertikale Skalierung ist zudem kostenintensiv, da sie Investitionen in größere Maschinen erfordert, um die steigende Zahl der Transaktionen bedienen zu können. Die horizontale Skalierung offeriert eine größere Flexibilität, ist

[8] [Mer10]
[9] [And10],S. 189-191

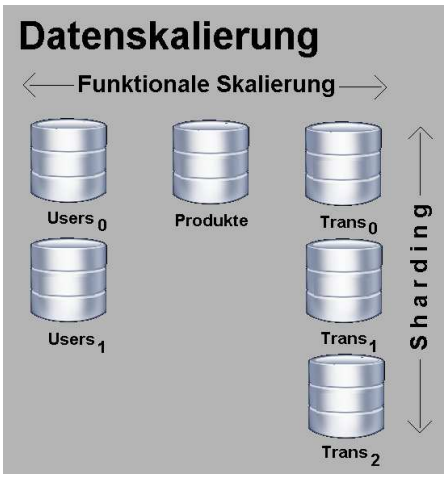

Abbildung 2.1.: Funktionale Skalierung und Sharding[10]

dafür aber wiederum komplexer, da das Hinzuziehen weiterer Server mit einem größeren Verwaltungsaufwand einhergeht. Horizontale Skalierung kann entlang zweier Richtungen erreicht werden. Zum Einen durch die funktionale Skalierung, die eine Gruppierung der Daten nach Funktionen beinhaltet und diese funktionalen Gruppen über Datenbanken verteilt. Zum Zweiten erfolgt eine Aufteilung von Daten innerhalb funktionaler Bereiche über mehrere Datenbanken – dies wird auch als "sharding" bezeichnet. Das Beispiel in Abbildung 2.1 zeigt, wie beide Ansätze zugleich in der horizontalen Skalierung verwirklicht werden. Die funktionalen Bereiche User, Produkte und Transaktionen können auf separate Datenbanken verteilt werden. Zusätzlich wird aber auch jeder Bereich auf mehrere Datenbanken verteilt, was dem Transaktionsvolumen dienlich ist. Wie die Abbildung zeigt, können die funktionalen Bereiche unabhängig voneinander skaliert werden. Die funktionale Partitionierung ist besonders dann empfehlenswert, wenn es darum geht, ein möglichst hohes Maß an Skalierbarkeit zu erreichen – eine grundlegende Anforderung für den Peer-to-Peer-Einsatz.

2.1.2.3. CAP-Theorem

Im Juli 2000 stellte der Computerwissenschaftler Eric A. Brewer die CAP-Theorem[11] auf, das behauptet, dass ein verteiltes System nicht über alle drei fundamentalen Charakteristika gleichzeitig verfügen kann, und zwar maximal zwei:

die vollständige Vereinbarkeit von Konsistenz (Consistency)

[11]CAP steht für Consistency, Availability, Partition Tolerance

Verfügbarkeit (Availability) und

Ausfalltoleranz (Partition Tolerance)

Konsistenz (Consistency): Ein System, welches konsistent ist, arbeitet entweder ganz oder gar nicht. Gilbert und Lynch verwenden den Begriff "atomic" (dt.: atomar, unteilbar) statt konsistent, was mehr Sinn macht, weil consistent für das "c" in ACID steht. Das bedeutet, dass die Daten niemals gegen bestimmte Beschränkungen verstoßen werden. Diese Daten müssen nach einer Transaktion in einen konsistenten Zustand gebracht werden.

Verfügbarkeit (Availability): bedeutet, dass ein Dienst oder System verfügbar ist. Gilbert und Lynch meinten, dass gerade die Verfügbarkeit einen genau dann im Stich lässt, wenn sie am meisten gebraucht wird.

Partitionstoleranz (Partition Tolerance) Wenn ein Programm oder eine Datenbank auf einem Rechner/Server betrieben wird, agiert dieser als eine Art atomarer Prozessor, im Sinne, dass es funktioniert oder nicht (bei einem eventuellen Absturz). Wenn man jetzt beginnt, die Dateien und Logik auf verschiedene Rechner bzw. Knoten zu verteilen, besteht das Risiko, dass sich Partitionen bilden können. Als Beispiel sei eine Datenbank gegeben, die auf 100 Knoten betrieben wird, welche wiederum auf zwei Schränke verteilt sind. Nun wird die Verbindung zwischen diesen Schränken unterbrochen – die Datenbank ist damit partitioniert. Ist das System partitionstolerant, dann wird die Datenbank auch weiterhin Lese- und Schreiboperationen durchführen können. In Anbetracht der Art der Verteilungsfähigkeiten, die das Internet zur Verfügung stellt, sind temporäre Partitionen eine relativ gewöhnliche und häufige Erscheinung. Selbiges gilt auch für globale Unternehmen mit mehreren Datenzentren[12,13]. Liegt eine Partition in einem Netzwerk vor, verliert es entweder die Konsistenz (da nun auf beiden Seiten der Partition Updates erlaubt werden) oder die Verfügbarkeit (weil der defekte Bereich bis zur Fehlerbeseitigung abgeschaltet wird). Partitionstoleranz bedeutet ganz einfach die Entwicklung einer Bewältigungsstrategie, um zu entscheiden auf welche der beiden anderen Systemeigenschaften verzichtet werden soll. Dies ist die Kernaussage des CAP-Theorems.

[12] [GL02]
[13] [Bro09]

Relationale Datenbanksystem sind also, bezogen auf die drei Eigenschaften des CAP-Theorems, primär auf strenge Konsistenz und Verfügbarkeit ausgelegt – "CA-Systeme". Die meisten NoSQL-Datenbanken kommen mit diesen Abstrichen besser zurecht, da sie, im Gegensatz zu den relationalen Datenbanken, nicht an die Umsetzung der ACID-Eigenschaften gebunden werden – für sie gelten andere "Regeln" (s. Kap. 2.1.2.4 "BASE-Systeme").

2.1.2.4. Alternatives Konsistenzmodell BASE

Aus dem CAP-Theorem lässt sich schließen, dass aufgrund des hohen Datenzugriffsbedarfs durch steigende Nutzerzahl das ursprünglich verwendete sogenannte "ACID-Paradigma" nicht mehr uneingeschränkt anwendbar ist. Unter dem Begriff "ACID" sind die vier Eigenschaften Atomicity, Consistency, Isolation und Durability zusammengefasst, welche durch Transaktionen in RDBMS durchgesetzt werden.

Viele NoSQL-Datenbanken setzen daher auf ein alternatives Modell, welches gewöhnlich als "BASE" bezeichnet wird und auf Transaktionierung verzichtet. BASE ist ein Akronym für:

Basically Available

Soft-State

Eventually Consistent

Der Grundgedanke beruht darauf, Hochverfügbarkeit für gelockerte Konsistenzbedingungen einzutauschen. Bei "Eventual Consistency" handelt sich also um einen optimistischen Ansatz, bei dem darauf verzichtet wird, alle im Cluster beteiligten Knoten durch Transaktionierung vor Inkonsistenzen zu sichern. Konsistenz der Daten wird so zu einem fließenden, sich über ein Zeitfenster ausbreitenden Zustand[14]. Zwar besteht so die Gefahr, dass für einen kurzen Zeitpunkt auf einigen Servern veraltete Daten zur Verfügung stehen, jedoch kann der hohe Bedarf an Zugriffen auf dieselben Datensätze eher gedeckt werden.

Es ist offensichtlich, dass dieses Vorgehen nicht für jeden Einsatzzweck geeignet ist. Eine finanzielle Transaktion ließe ein solches Konzept nicht zu, da der richtige Kontostand nicht zu jedem Zeitpunkt garantiert wäre. Da die Konfigurationsmöglichkeiten zwischen ACID und BASE mittlerweile fließend sind und manche Datenbanken sogar

[14] [Edl11]

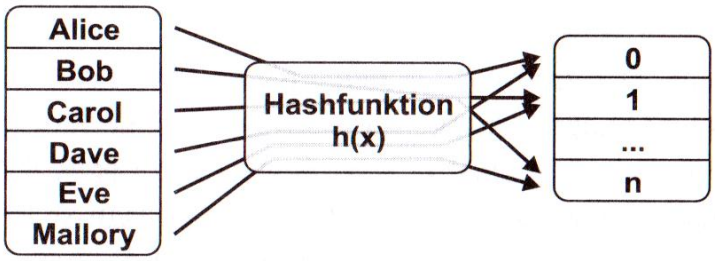

Abbildung 2.2.: Hashfunktion. Eingangswerte werden über eine bestimmte Funktion auf einen Werteraum abgebildet[16]

eine Kombination der Paradigmen unterstützen, ist es sinnvoll, Vor- und Nachteile für die jeweilige Anwendung im Voraus abzuwägen.

2.1.2.5. Consistent-Hashing-Strategie

Das Prinzip der Datenhaltung in einem Key/Value Store ähnelt einer Hash-Table, also einer Zuordnungstabelle der objektorientierten Programmierung bestehend aus zwei Spalten. Eine Spalte enthält dabei den über eine Hash-Funktion abgebildeten Schlüssel ("Key") über den der Zugriff erfolgt (s. Abbildung 2.2). Die zweite Spalte führt den assoziierten Wert, genannt "Value", welcher als einzige Bedingung den Datentypen entsprechen muss, die durch die Datenbank unterstützt werden. Die einzelnen Einträge in die Zeilen der Tabelle sind dabei ohne weitere Relation zueinander[15]. Die Verantwortung der Beziehungen zwischen den Einträgen obliegt damit voll der Clientanwendung.

2.1.2.6. Typen der NoSQL-Datenbanken

Alle NoSQL-Datenbanken werden in vier Kategorien unterteilt[17]:

Key-Value-Stores

Document-Stores

Wide-Column-Stores

Graph-Datenbanken.

[15] [Edl11]
[16] [Edl11]
[17] [Edl11]

Bei der Kategorisierung von NoSQL-Systemen werden die wichtigsten Merkmale der NoSQL-Datenbanksysteme betrachtet. Diese sind[18]:

kein relationales Datenmodell

Eignung für verteilte und horizontale Skalierung

schemafrei oder nur schwache Schemarestriktionen

einfache Datenreplikation zur Unterstützung der verteilten Architektur

einfache API

kein ACID sondern BASE als Konsistenzmodell

daneben gibt es auch Algorithmen und Protokolle, die allerdings nicht häufig verwendet werden

Key-Value-Store Key-Value-Store ist eines der ältesten Konzepte von NoSQL-Systemen, die seit den siebziger Jahren im Einsatz sind. Heute gehören sie zu den am schnellsten wachsenden NoSQL-Datenbanksystemen.

Der Aufbau ist einfach und ähnelt dem von relationalen Datenbanken. Ein Schlüssel (Key) zeigt auf einen Wert (Value), der eine strukturierte oder willkürliche Zeichenkette sein kann. Die Werte können dabei außer Zeichenketten auch Listen, Sets und Hashes sein[19,20]. Der Zugriff auf einen Wert erfolgt ausschließlich über einen Schlüssel, das bedeutet, dass hinter einem Schlüssel ein eindeutig identifizierbares Objekt hinterlegt ist.

Vorteile Durch das einfache Schema skalieren solche Systeme sehr gut und haben eine schnelle und effiziente Datenverwaltung. Ein weiterer Vorteil besteht darin, dass diese Systeme den Ansatz der Datenversionierung verfolgen, das heißt, dass die Daten sofort geschrieben werden können.

Nachteile Das einfache Schema der Key-Value-Stores schränkt die Abfragemöglichkeiten ein. Einfache Abfragen sind zwar schnell abgearbeitet, dafür gibt es aber oft keine Möglichkeit, eine komplexe Abfrage zu schreiben. Eine Darstellung von komplexen

[18] [Sad13]
[19] [Red12]
[20] [Sad13]

Objekt-Beziehungen ist aus dem Grund der Einfachheit der Schema auch nicht möglich, weshalb die nötigen Integritätsbedingungen in die Anwendung ausgelagert werden.

Document-Store Der Aufbau von Document-Stores ist unterscheidet sich von dem der Key-Value-Stores. Die Daten sind hier nicht in Form von Tabellen gespeichert, sondern werden in Dokumenten abgelegt. Dabei ist ein Dokument anders zu verstehen als eine mit z. B. einem Wordprocessor bearbeitete Datei.

Ein Dokument ist eine Zusammenstellung von strukturierten Daten. Dabei kann so ein Dokument sowohl einen Tupel aus einem relationalen DBMS, eine Tabelle oder auch Objekte enthalten. Ein Schema, welches eine ganze Datenbank umfasst, existiert nicht. Solche Datenbanken werden auch oft "Schemafrei" genannt.Durch die Schemafreiheit existieren zwischen den Dokumenten keine Relationen. Auch bei der Struktur der Dokumente gibt es keine vorgeschriebene Einheit. Jedes einzelne Dokument kann eine Struktur haben, die von den anderen Dokumenten abweicht.

Die Dokumente werden durch einen diese kennzeichnenden Bezeichner angesprochen. In den Dokumenten werden die Daten in Form von Schlüssel-Wert-Paaren gespeichert. Jedem Schlüssel wird ein Wert zugewiesen. Der Wert kann ein Zeichenkette, Text, Liste oder auch Array sein, der wiederum geschachtelte Datentypen enthalten kann.

Vorteile Document-Stores speichern alle zusammenhängende Daten in einem Dokument. Dadurch eignen sich die Document-Stores zum Speichern der übertragenen Daten aus den HTML-Formularen. Außerdem ist eine hohe Skalierbarkeit der Document-Stores zu nennen.

Nachteile Dadurch, dass Document-Stores schemafrei sind, muss die Struktur der Dokumente vom Benutzer festgelegt und kontrolliert werden. Es führt zum Verzicht auf Komfortfunktonen, wie z.B. selbst definierte Constraints und Trigger. Außerdem muss ein Kontrollmechanismus zum Prüfen der eingegebenen Daten eingebaut werden.

Wide-Column-Store Die Idee eines Wide-Column-Stores wurde in achtziger Jahre vorgeschlagen, aber die erste spaltenorientierte Datenbank (sogenannte Wide-Column-Store) erst in den neunzigern entwickelt. Hinter Wide-Column-Stores verstecken sich gut kalibrierbare, mit sehr großen Datenmengen arbeitenden Datenbanksysteme. Als Gründer dieses Ansatzes kann Google mit BigTable angesehen werden[21].

[21] [CDG+06]

Wide-Column-Store gruppiert die Daten mehrerer Einträge nach ihren Spalten und nicht nach ihren Zeilen wie es bei den relationalen Datenbanken der Fall ist. Dabei besteht jeder Eintrag aus den Namen der Spalte, den Daten und einem Zeitstempel, auch Timestamp genannt, um die Aktualität der Daten zu prüfen.

Spalten, die zusammenhängen, bilden eine so genannte Column-Family, die äquivalent einer Tabelle in relationalen Datenbanken ist[22]. In einer Column-Family existiert keine logische Struktur und keine Einschränkungen. So kann eine Column-Family mehrere Tausend oder sogar Millionen von Spalten enthalten.

Die Column-Families werden in der Datenbank über Schlüssel identifiziert.

Vorteile Wide-Column-Stores sind superskalierbar und eignen sich daher sehr gut für Datenbanken mit großem Datenvolumen. Das liegt daran, dass durch die spaltenorientierte Struktur die Daten auf mehreren Servern verteilt werden und somit die Lasten eines einzelnen Servers gering gehalten werden können. Außerdem wird der Leseprozess bei einem Wide-Column-Stores beschleunigt, da keine unnötigen Daten gelesen werden, sondern nur die Daten, die ausgesucht worden sind. Auch der Schreibprozess wird beschleunigt, wenn es nur um eine einzelne Spalte geht.

Nachteile Der Nachteil eines Wide-Column-Stores ist der Aufwand bei Schreibprozessen, die über mehreren Spalten gehen. Zum Beispiel wenn zu einer bestehenden Person das Alter, die Adresse, das Gehalt hinzugefügt werden soll, dann muss man auf mehrere Columns (jeweils die Columns Alter, Adresse und Gehalt) zugreifen. Das verlangsamt den Schreibprozess.

Graph-Datenbanken Wie auch bei einem Graph bestehen Graph-Datenbanken aus Knoten und Kanten. Die Knoten repräsentieren die Tupel aus der relationalen Datenbank und die Kanten die Beziehungen zwischen den Knoten. In solchen Datenbanken können durch einfache Traversierung teure Datenbankabfragen, wie mehrere rekursiv geschachtelte Joins vermieden werden. Somit bieten Graph-Datenbanken eine bessere Performance als relationale DBMS.

Die Traversierung erfolgt wie beim Graphen als Breiten- und Tiefensuche, algorithmische Traversierung und Traversierung in zufälliger Reihenfolge. Wie es auch verschiedene Graphenarten gibt, z.B. gerichtet, ungerichtet, gewichtet usw. so gibt es auch verschiedene Modelle der Graph-Datenbanken. Hier muss nach Art der Anwendung ent-

[22] [Tiw11]

schieden werden, welches zugrunde liegende Modell sinnvoll ist. Wenn die Anzahl der Knoten und Kanten für einen Server zu groß wird, muss der Graph partitioniert werden. Die Partition ist eine Art der horizontalen Skalierung und wird auch bei anderen NoSQL-Datenbanken verwendet. Dabei wird versucht, den Gesamtgraph in Teilgraphen aufzuteilen. Dies bereitet manchmal eine Schwierigkeit, eine entsprechende Stelle für die Teilung zu finden. Es existiert hierfür auch keine mathematisch exakte Methode, sondern nur einen Paar heuristischen Algorithmen wie z.B. Clustering-Algorithmen. Manche Graphen können nicht aufgeteilt werden. Dann müssen manche Knoten in zwei oder mehreren Teilgraphen auftauchen. Das ist die so genannte überlappende Partitionierung.

Wenn es sich um eine Graph-Datenbank mit viel Lese- und relativ wenig Schreiblast handelt, bietet sich eine Replikation an, was auch von anderen Datenbankarten verwendet wird. Auch die Replikation ist eine horizontale Skalierung. Dabei wird der Graph vervielfacht und auf mehreren Servern gespiegelt. So ist die Last eines einzelnen Servers geringer und es wird eine bessere Performance erreicht.

Vorteile Der wichtigste Vorteil einer Graph-Datenbank ist einfache und effiziente Traversierung und dadurch bessere Performance als relationale Datenbanken. Durch die Struktur kann auf komplexe Anfragen wie rekursiv geschachtelte Joins verzichtet werden.

Nachteile Der große Nachteil einer Graph-Datenbank ist die Abfragesprache. Bisher existiert noch keine einheitliche Abfragesprache. Es gibt viele Abfragesprachen sowie Graphmodelle.

2.2. P2P-Systeme

Ein P2P-System ist ein Zusammenschluss von gleichberechtigten Arbeitsstationen in Netzwerken, die den Einsatz von verteilten Anwendungen und den Austausch von Dateien ermöglichen. Ein zentraler Server ist hierfür nicht notwendig. Populär wurden P2P-Netzwerke durch den Austausch von Musik- und Videodateien über sogenannte Tauschbörsen[23].

Essentielle Eigenschaft von Peer-to-Peer-Systemen ist die dezentrale Vernetzung, das heißt, dass keine zentrale Kontroll-, Daten- oder Dienstinstanz in Form eines Servers

[23] [Weh05]

existiert. Alle beteiligten Endsysteme stellen ihre Ressourcen und Dienste den anderen zur Verfügung, können diese teilen und gemeinsam nutzen. Dadurch, dass eine Serverinstanz fehlt, müssen sich alle Beteiligten eines P2P-Netzes selbst organisieren, was eine große Herausforderung darstellt. Dazu gehören die Verwaltung der Adressstrukturen und der Wegwahlverfahren. Durch solche Strukturierung unterscheidet man zwischen reinen und hybriden Systemen[24].

2.2.1. Reine vs. Hybride P2P-Netzwerke

Zentralisierte Systeme werden oft als P2P-Netze der ersten Generation bezeichnet. Hier wird ein zentraler Server eingesetzt, der die Anfragen der Nutzer koordiniert und globales Wissen über die Verteilung der Inhalte auf den Nodes besitzt. Alle Knoten bauen zuerst zu diesem Server eine Verbindung auf und nutzen dann Informationen von dort, um, je nach Verteilungsverfahren, zu anderen Knoten Verbindungen aufzubauen. Der bekannteste Vertreter dieser Gruppe ist Napster.

Ein **reines P2P-Netz** besteht nur aus gleichberechtigten Nodes, die zu jeder Zeit dem Netz beitreten können oder es verlassen können. Jeder Knoten übernimmt in diesem Fall gleichzeitig Server- und Client-Funktionen. Für diese Art der P2P-Netze muss ein Bootstrapping-Algorithmus existieren, damit ein neuer Knoten mindestens einen Teilnehmer des Netzes findet. Meistens sind dies vorgegebene IP-Adressen, die eine kurze Liste von Teilnehmern bereitstellen. In bestimmten Fällen sind auch Discovery-Mechanismen über Multicast oder in lokalen Netzen sogar Broadcast-Pakete möglich. Diese Form von P2P-Netzen ist einem mobilen Ad-Hoc-Netz sehr ähnlich und setzt oftmals ähnliche Algorithmen zum Aufbau des Netzes ein. Dies erfordert einen hohen Koordinationsaufwand zwischen den Knoten, da jeder Knoten einen Teil der Netztopologie kennen muss, um im Fall des Ausfalls eines Nachbarn reagieren zu können. Beispiele für diese Form der Organisation von P2P-Netzen sind Gnutella und Freenet.

Die **hybriden Systeme** besitzen keinen zentralen, vorgegebenen Server sondern bestehen nur aus Knoten, die auch jederzeit dem Netz beitreten können. Einige dieser Knoten bekommen allerdings dynamisch eine Sonderrolle zugewiesen und nehmen dann besondere Koordinationsfunktionen ein. Die Auswahl der Knoten kann nach verschiedenen Auswahlkriterien geschehen:

Topologie: In diesem Fall wird versucht Knoten zu wählen, die eine zentrale Position im Netz einnehmen. Ziel dieser Auswahl ist, die mittlere Distanz eines Knotens

[24] [Hos02]

zum nächsten Koordinationsknoten minimal zu halten und gleichzeitig die Zahl der Koordinationsknoten niedrig zu halten.

Verfügbare Bandbreite: Hier wird die verfügbare Bandbreite der Knoten gemessen und Knoten mit einer hohen verfügbaren Bandbreite werden zu einem Koordinationsknoten. So soll gewährleistet werden, dass Koordinationsknoten nicht durch den zusätzlichen Netzverkehr überlastet werden. Zusätzlich wird bei diesem Ansatz versucht, danach die Topologie anzupassen, um die Distanz der Knoten zu den Koordinationsknoten zu verringern.

Rechenleistung/Speicher: Einige P2P-Systeme benötigen in den Koordinationsknoten viel Speicher und wählen deshalb diese Knoten nach dem verfügbaren Speicher aus. Diese Auswahl hat den Vorteil der einfachen lokalen Entscheidbarkeit. Bei allen anderen Methoden ist viel Kommunikation mit den anderen Knoten nötig, bevor eine Einteilung möglich wird.

Diese Systeme vereinen viele Vorteile reiner P2P-Systeme und der zentralisierten Systeme, aber verlangen von einigen Teilnehmern einen größeren Beitrag. Bekannte Vertreter dieser Kategorie sind Gia und Kaazaa.

2.2.2. Synchronisation und Replikation

Unter **Replikation** oder **Replizierung** versteht man die mehrfache Speicherung derselben Daten an meist mehreren verschiedenen Standorten und die **Synchronisation** der Datenquellen.

Datenreplikation hat im Wesentlichen zwei Ziele:

zum Einen die Steigerung der Performance des Systems und zum Anderen die Erhöhung der Verfügbarkeit der Daten. In Filesharing-Systemen erfolgt üblicherweise eine implizite Replikation, d.h. Dateien, welche auf einen Peer heruntergeladen werden, dienen zeitgleich als Uploadquelle und somit als Kopie. Die Anzahl der Replikate im System ist also direkt abhängig vom Interesse der Nutzer. Daten, die nur selten angefordert werden, verschwinden schneller aus dem System als Daten, die für die Nutzer von großem Interesse sind[25].

Wenn die Daten unabhängig vom Grad des Interesses dauerhaft in einem P2P-System verfügbar gemacht werden sollen, so sind andere Techniken notwendig. Die einfachste Methode ist das Kopieren der Daten auf eine vorab festgelegte Anzahl

[25] [BMSV02]

von zufällig gewählten Knoten. Das größte Problem hierbei ist die Wartung der Replikate, weswegen üblicherweise auf die Möglichkeit einer Änderung der Daten verzichtet wird[26].

Synchronisationsart. Das Verteilen der Updates erfolgt bei den meisten Systemen asynchron (lazy replication), d.h. die ursprüngliche Transaktion löst eine Reihe von Subtransaktionen aus, welche die Änderungen zu einem geeigneten Zeitpunkt propagieren[27]. Durch die zeitversetzte Synchronisation kann es passieren, dass verschiedene Versionen eines Datensatzes im System existieren. Im Gegensatz dazu werden bei der eager replication alle Kopien im System als Teil der originalen Transaktion synchron aktualisiert. Zum Zeitpunkt der Änderung sind alle Replikate gesperrt, sodass es nicht zu ungewollter Versionierungs- oder Serialisierungsproblemen kommen kann. Allerdings steigt die Wahrscheinlichkeit von Deadlocks im System mit zunehmender Knoten- bzw. Transaktionsanzahl erheblich, konkret vertausendfacht sich die Deadlockrate bei einer Erhöhung der Knotenanzahl von einem auf zehn heben[28]. Außerdem wächst die Anzahl der Sperren mit der Anzahl der Knoten, sodass eager replication für P2P-Systeme mit mehreren tausend Teilnehmern ungeeignet ist. Beide Synchronisationsarten sind passive Methoden, da die Knoten im System ihre Updates zugewiesen bekommen. Zusätzlich gibt es noch die Möglichkeit, dass Knoten selbst aktiv werden und sich bei anderen Teilnehmern nach Änderungen erkundigen. Meist verwendet man diese Technik, um Knoten, welche das System verlassen haben, mit einem aktuellen Datenbestand wieder einzugliedern[29].

[26] [SS05]
[27] [SS05]
[28] [GHOS96]
[29] [SMK+01]

3. Vergleich der Datenbanken

Es existieren sehr viele Quellen in der Literatur, die sich mit dem Vergleichen und Testen von Software beschäftigen. Einige Methoden sind sogar ISO-normiert[1,2]. Allerdings gibt es nur wenige Ressourcen, die sich mit einem Vergleich von Datenbanken auseinander setzen. Im Internet sind von mir einige Vergleiche von NoSQL-Systemen gefunden, die nur wenige Eigenschaften in einer Tabelle zusammenführen. Diese waren für meine Zwecke allerdings kaum zu gebrauchen.

Durchführung der Evaluierung

Für diese Arbeit wurden aus der Literatur die Grundsätze des Vergleiches ausgewählt und auf die Datenbanken angewandt. Dabei sind die drei genannten Datenbanksysteme auf verschiedene Kriterien geprüft worden.

3.1. Auswahl der Datenbanksysteme

Seit Anfang an ist Cassandra-Datenbanksystem genommen worden, da es sich um eine bekannte Wide-Column-Store Datenbank handelt. Im Gegensatz zu dieser Datenbank wurde noch ein NoSQL- und ein relationales Datenbanksystem ausgewählt, so dass es die beiden NoSQL Datenbanksysteme von unterschiedlichen Typen sind. Da CassandraDB zu dem Typen "Wide-Column-Store" angehört, ist zur Wahl einer der drei im Unterkapitel 2.1.2.6 beschriebenen anderen Typen. Die Wahl fiel auf einen Key-Value-Store, welches der ältesten Typen unter den NoSQL-Datenbanksystemen ist. Aus dem breiten Fach wurde Riak ausgewählt – einen nicht ganz typischen Mitglied der Familie. Als relationale DBMS wurde PostgreSQL genommen, weil das eine bekannte, sehr weit verbreitete und stabile Open Source Datenbank ist.

[1] [Iso13]
[2] [Gra14]

Cassandra DB

Das Projekt Cassandra wurde im Juli 2008 auf Google Codes veröffentlicht, einer Internetseite? auf der Google Entwicklerwerkzeuge bereitstellt. Seit Februar diesen Jahres wird das Projekt von der Apache Software Foundation geleitet, die sich u. a. auch für CouchDB verantwortlich zeichnen[3]. Cassandra gilt als Open-Source-Imitat von Googles hausinternem Datenbanksystem BigTable und war ursprünglich für den Einsatz im sozialen Netzwerk Facebook gedacht.
In Anbetracht von Facebooks über 400 Mio. Nutzern war Cassandra primär darauf ausgelegt? große Datenmengen effizient verwalten zu können. Wenn die Nutzeranzahl schnell anstieg, so sanken die Reaktionszeiten. Vor allem die Suchläufe auf den auf MySQL basierenden Posteingängen, auf welchen täglich mehrere Mrd. Schreibzugriffe erfolgten, nahmen immer mehr Zeit in Anspruch[4].

Cassandra wurde in der Programmiersprache Java geschrieben und nutzt eine Synthese aus bekannten und etablierten Techniken, wie Partitionierung und Replikation, um die benötigte hohe Skalierbarkeit und Verfügbarkeit sicherzustellen[5]. Die Replikation kann in ihrem Umfang beschränkt werden, beispielsweise auf Datenzentren oder Serverschränke.

Die Knoten des Systems wissen stets? wo sie sich physisch befinden, was sehr vorteilhaft sein kann, um Latenzen gering zu halten. Der Eintritt eines neuen Knotens beginnt mit dem Kopiervorgang von Daten eines am stärksten ausgelasteten Knotens, woraus resultiert, dass sich beide den Schlüsselbereich und damit einhergehend die Arbeit teilen. Eine Lese- oder Schreiboperation kann auf einem beliebigen Knoten stattfinden, da dieser die Anfrage anschließend an den oder die betreffenden Knoten weiterleitet. Gewährleistet wird dies durch die Ermittlung des betroffenen Schlüssels. Die Speicherung erfolgt sowohl auf der Festplatte als auch im Hauptspeicher, in welchem zumeist die aktuellen Daten oder Indizes abgelegt werden. Jede Spalte (Column) erhält ihre eigene ID, über welche sie angesprochen werden kann.

Als Abfragesprache wird Cassandra Query Language[6] (weiter auch als CQL) benutzt, welche eine Syntax sehr ähnlich der von SQL hat. CQL hat im Vergleich zu SQL einige Einschränkungen, bedingt durch die Skalierbarkeit der Datenbank kommt. Das Tool erlaubt aber sogar einem unerfahrenen Benutzer, mit der Datenbank zu arbeiten, ohne dabei die Konzepte der Column-Families und Supercolumns zu verstehen. Eine weitere

[3] [Res10]
[4] [Bin10]
[5] [LM09]
[6] [CQL15]

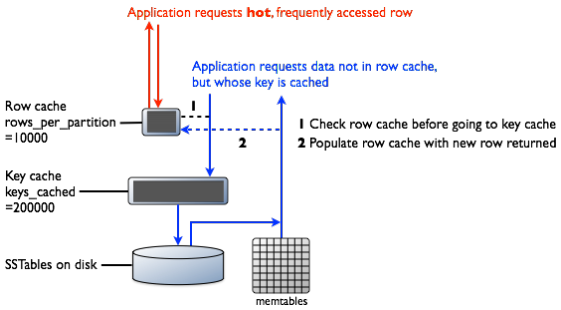

Abbildung 3.1.: Funktionsweise Caching in Cassandra

nützliche Eigenschaft von CQL ist die Möglichkeit von Abfragen über Keys und sogar Key-Ranges[7].

Für eine komfortablere Nutzung von CassandraDB können von Benutzern Trigger in Java geschrieben werden.

Schlüsselkomponenten der Struktur von CassandraDB

Jede Installation des Datenbanksystems wird **Knoten** genannt. Auf jedem Knoten im Systen werden die Nutzdaten gespeichert. Eine Menge dieser Knoten ist in einen so genannten **Datenzentrum** aufgebaut. Jeder Data Center bedient einen eigenen Workload, was kleinere Latenzzeiten für die Requests erlaubt. Ein und derselbe Knoten kann für unterschiedliche Workloads zuständig sein und dadurch in verschiedenen Data Center auftauchen. Die Datenzentren sind ihrerseits in Clustern angeordnet, die mindestens einen Datenzentrum enthalten.

Alle Daten werden zuerst in ein Log geschrieben, um sie persistent zu speichern. Danach können diese Daten in die so genannte Sorted String Table (SSTable) geschrieben werden. Erst nach diesem Prozess können die Daten im Log bearbeitet, gelesen oder entfernt werden.

Solche SSTables sind unveränderbare Daten-Dateien, in die Cassandra periodisch den Inhalt aus den so genannten Memtables hineinschreibt. Da die SSTables unveränderbar sind, können Informationen in die SSTable nur angehängt werden. Cassandra legt so eine für jede Tabelle eine vorläufige Kopie der neuen oder veränderten Daten an und wenn das ohne Fehler passiert, dann werden diese Änderungen an die SSTable angehängt. Siehe dazu Abbildung 3.1.

[7] [Kov11]

Riak DB

Riak gehört zu den bekanntesten Key-Value-Datenbanken und wurde 2009 von Basho Technologies vorgestellt[8] und gehört zu den bekanntesten Key-Value-Datenbanken. Obwohl die Entwickler Riak als eine Key-Value-Datenbank positionieren, speichert sie alle Informationen als JSON-Dateien ab, so wie es üblicherweise Document Store DBs machen.

Riak wurde nach dem Vorbild von Amazon's Dynamo entwickelt, was bedeutet, dass nicht einfach ein hochverfügbares, sondern möglichst immer verfügbares Datenbanksystem entwickelt wurde. Diese Eigenschaft wurde durch so genannte **shared-nothing-Architektur** erreicht, bei der alle Anfragen an eine andere Instanz automatisch weitergeleitet werden. Riak verwendet dabei, wie auch CouchDB eine Consistent-Hashing-Strategie, bei der es keinen Masterknoten gibt, sondern alle Knoten gleichgestellt sind.

Eine Erweiterung des Systems ist im laufenden Betrieb leicht möglich, da sich der Inhalt dynamisch verteilt. Riak ist ein BASE-System, das auf Verfügberkeit und Partitionierungsfähigkeit setzt, was bedeutet, dass Riak *eventually consistent* ist. Eventually Consistent heißt, dass das System in einen bestimmten Moment nicht konsistente Daten in sich enthält. Das Verhalten läßt sich an einem Beispiel leicht erklären. Das Online-Versandhaus Amazon möchte möglichst viele Kunden weltweit bedienen und betreibt daher ein weit verstreutes Netz an Servern. Amazon strebt eine hohe Verfügbarkeit an – muss aber dafür Einbußen bei der Konsistenz hinnehmen. Kauft A ein Buch in New York und B eins in Singapur und geschieht dies auf zwei verschiedenen Servern, sind die beiden Datenbanken nicht mehr synchron bzw. konsistent.

Der Namespace in einem Riak-Dokument gliedert sich wie folgt:

Buckets: bezeichnen den Namespace unter dem abgelegt wird.

Keys: frei wählbarer Schlüssel für Dokument.

Document: Inhalt der JSON-Datei oder sonstige Daten.

Für den REST-Zugriff via HTTP werden dann diese Angaben hintereinander gestellt, wie z.B.: path/<bucket>/<key>, was die Verwendung des Dokumentes mit GET, POST, PUT und DELETE zulässt.

Dabei kann das JSON-Dokument wiederum Links enthalten und dadurch auf andere Dokumente verweisen.

[8] [inc15]

Riak unterstützt folgende Programmiersprachen: Java, Ruby-on-Rails, Python, PHP, JavaScript, Elang. Alle diese Sprachen haben eine eingene Schnittstelle zu der Datenbank, es kann aber auch REST-Schnittstelle für einen Zugriff benutzt werden.
Riak weist folgende Vorteile auf[9]:

Einfache Skalierung ohne einen Master-Knoten

Versionierung und leichter Zugriff auf die Versionen mittels **Hinted Handoff**

Unterstützung vielfältiger Programmiersprachen

Kein Datenverlust im Falle eines Stromausfalls, da alle Daten direkt auf die Festplatte geschrieben werden (Datenhaltung im RAM-Speicher)

Die Nachteile von Riak sind die folgenden:

Schwächere Performance im Vergleich zu MongoDB oder Cassandra[10], da Riak auf eine Datenhaltung im RAM-Speicher verzichtet hat

Es existiert immer noch keine offizielle Literatur zu Riak außer Manuals auf der Entwicklerseite.

PostgreSQL

Bei PostgreSQL handelt es sich nach Angaben der PostgreSQL Global Development Group (2007) um eines der ältesten und am weitesten fortgeschrittenen objektrelationalen Datenbanksysteme (Objektrelationale RDBMS). Es unterstützt neben einer Reihe von eigenen Erweiterungen den SQL92 und den SQL99 Standard. PostgreSQL ist eine freie Datenbank unter der BSD-Lizenz, womit ihr Quellcode für Erweiterungen oder Verbesserungen offen steht. Sie gilt unter den lizenzkostenfreien Datenbanken als eine der stabilsten und zuverlässigsten überhaupt, ist in ihrer Größe lediglich durch den zur Verfügung stehenden Speicher beschränkt und bietet eine hohe Transaktionssicherheit.

Eine interessante Erweiterung, die vielen relationalen Datenbank Systemen zur Verfügung steht, ist die Möglichkeit, Daten mit Hilfe einer grafischen Oberfläche zu verwalten und zu ändern. Speziell für PostgreSQL sind pgAdmin und phpPgAdmin die bekanntesten. Die erste ist eine crossplattform-fähige standalone Applikation, das zweite Programm ist eine browserbasierte Version und erfordert somit einen Webserver. Beide

[9] [Edl11]
[10] [Edl11]

Tools unterstützen Syntaxhighlighting der SQL-Befehle und Unicode-Kodierung. Es ist aber auch kein Problem, die Datenbank über die Konsolenwerkzeuge zu steuern[11].

Wie schon erwähnt ist die PostgreSQL eine Open-Source-Datenbank, was einem Entwickler erlaubt, den Quellcode zu ändern und mit neuen Funktionen zu ergänzen.

In PostgreSQL besteht die Möglichkeit, intern eingebaute und an die von Oracle entwickelte PL/SQL angelehnte Programmiersprache PL/pgSQL zu nutzen. Diese erlaubt dem Benutzer eigene komplexe Typen, sowie auch Trigger und Stored Functions zu definieren. Alternativ können in C geschriebene Trigger als Bibliothek in die Datenbank importiert werden. Ebenfalls unterstützt PostgreSQL viele gängige Programmiersprachen, wie C++, Python, Java, PHP, Ruby und weitere. Für alle diese Sprachen existieren die so genannten "Driver", die es einem Entwickler erlauben, eine Verbindung zu der jeweiligen Datenbank herzustellen.

Die allgemeinen Daten der ausgewählten Datenbanksysteme sind zum Vergleich in der Tabelle 3.1 zusammengefasst.

3.2. Vergleichsdurchführung

Nachdem die ausgewählten Produkte kurz vorgestellt wurden, hat dieses Kapitel das Ziel, die wichtigsten Aspekte dieser drei Datenbanken zu vergleichen und zu bewerten. Dazu werden von jeder Datenbank solche Aspekte wie Architektur, Zugriffsgeschwindigkeit in Form von Auswertung der gemessenen Zugriffszeit, Anfragemöglichkeit und anderen erläutert.

3.2.1. Aufbau für den Vergleich

Für die Untersuchung benötigt man mindestens zwei über das Netzwerk verbundene Rechner wie es in der Abbildung 3.2 gezeigt ist, die für die Verteilung der Peer-to-Peer Knoten sorgen. Der erste ist ein Notebook mit Intel Core 2 Duo Prozessor und 4 Gigabyte Ram ausgestattet, als Betriebssystem läuft Ubuntu Linux 14.04 LTS (Long Term Support). Auf dem zweiten PC ist ein Intel Core i5 mit 8 Gigabyte Ram und ebenfalls Ubuntu Linux 14.04. Beide Systeme sind frisch installiert, als zusätzliche Software sind nur die Datenbanken hinzugefügt. Während der Ausführung der Tests ist nur eine Datenbank aktiv, die beiden anderen sind deaktiviert. Das erlaubt, die Reinheit des Experiments zu wahren. Somit führt das Betriebssystem außer einer Datenbank keine unnötigen Prozesse aus.

[11] [Wor02]

Datenbanksystem	Cassandra	Riak	PostgreSQL
Hersteller	Apache Foundation	Basho Technologies	PostgreSQL Global Development Group
Erster Release	2008	2009	1989
Lizenzierung	Open Source	Open Source	Open Source
Sprache der Implementierung	Java	Erlang	C
Cross-Plattform Unterstützung	Windows, Linux, BSD, OS X	Windows, Linux, OS X	Windows, Linux, BSD, OS X, Unix, Solaris
Datenbank Modell	Wide-Column-Store	Key-Value-Store	Objektrelationale DBMS
Abfragesprache	CQL	keine Bestimmte	SQL
Unterstützte Programmiersprachen	C++, C#, Erlang, Haskel, Java, JavaScript, Perl, Python, Ruby	C++, C#, Erlang, Groovy, Haskel, Java, JavaScript, Lisp, Perl, PHP, Python, Ruby, Smalltalk	.NET-Sprachen, C/C++, Java, Perl, Python, PHP, Tcl
Partitionierung	Sharding	Sharding	nicht vorhanden
Replikation	keine	keine	Master-Slave-Replikation
Konsistenz	Eventual Consistency	Eventual Consistency	Immediate Consistency
Technische Dokumentation	www.datastax.com/docs	docs.basho.com	www.postgresql.org/docs/manuals

Tabelle 3.1.: Erster vergleichender Überblick über die Datenbanksysteme

Abbildung 3.2.: Verteilung der Datenbankknoten im Netzwerk

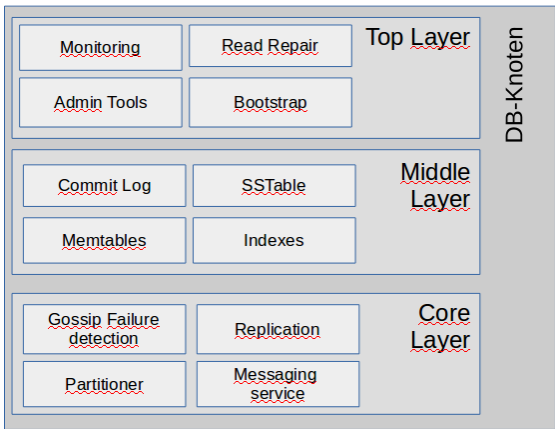

Abbildung 3.3.: Architektur der Cassandra DB[12]

Das Netzwerk besteht aus diesen zwei Rechnern, die über einen Router verbunden sind, so wie es auf der Abbildung 3.2 gezeigt ist. Das Notebook ist drahtlos mit dem Router verbunden und der andere Rechner ist mit dem Kabel verbunden. Die Ausführung der Anfragen passiert ausschließlich netzintern, d.h. der Router ist nicht an das Internet angeschlossen. Für diese Untersuchung werden alle drei Datenbanken installiert und für die Reinheit des Experiments immer zwei Datenbanken gestoppt. Nachdem alle Messungen durchgeführt sind, wird eine andere DB gestartet und die Messungen wiederholt werden.

In jeder Datenbank wurde jeweils ein Schema erstellt und als Vorbereitung mit fünftausend Einträgen gefüllt.

3.2.2. Architektur von Datenbanksystemen

In diesem Kapitel ist die Architektur von PostgreSQL, Riak und Cassandra kurz vorgestellt.

[12][Dat15]

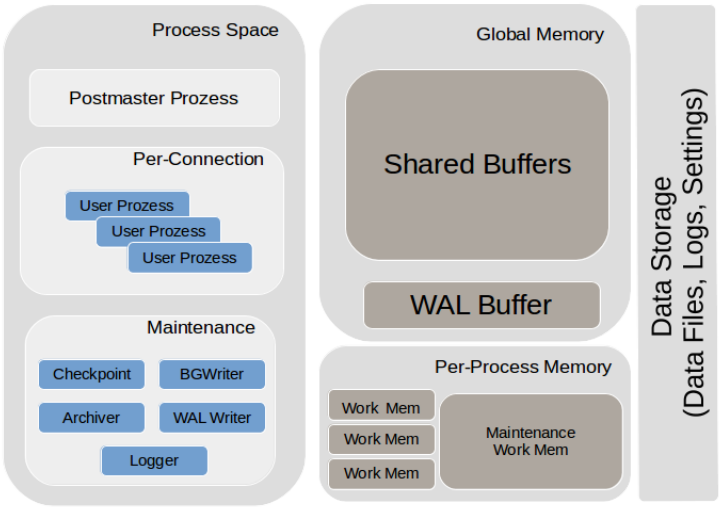

Abbildung 3.4.: Architektur der PostgreSQL-Datenbank[13]

Architektur von Cassandra DB Auf der Abbildung 3.4 sieht man die Architektur von CassandraDB. Da Cassandra eine Peer-to-Peer Datenbank ist, hat sie nicht die klassische Server-Client-Architektur, sondern besteht aus einer Anzahl gleichberechtigter Nodes, die in einem Data-Center gebunden sind. So ein Data-Center besteht dabei aus mindestens einem Node.

Diese Nodes bestehen aus drei Schichten, die auf der Abbildung 3.3 gezeigt sind:

Core Layer ist für die Kommunikation zwischen den verschiedenen Nodes und Clusters zuständig, aber auch Replikation und Partition der Daten. Die Kommunikation zwischen den Knoten passiert unter der Kontrolle der so genannten "Gossip Failure detection", welche ausgefallene bzw. abgetrennte Knoten identifiziert.

Middle Layer In dem Middle Layer werden die Daten selbst gehalten, der Zugriff von den Clients gesteuert, Daten werden überwacht und Log-Files erstellt.

Top Layer wurde für die Administration der Datenbank konzipiert und enthält somit auch alle dafür nötige Werkzeuge.

Architektur von PostgreSQL Auf der Abbildung 3.4 ist die Architektur des PostgreSQL Datenbank-Management-Systems abgebildet. Das System arbeitet mit einer

[13] [Cyb]

Client-Server-Architektur. Die Serverseite ist durch das Datenbank Management System als Postmaster-Prozess realisiert. PostgreSQL benutzt ein sogenanntes "Prozess pro Verbindung"-Modell. Wenn eine Anfrage von einem Client an den Postmaster gestellt wird, startet er einen Server-Prozess (auch "Backend" genannt), der die Anfrage abarbeitet. Backends agieren mehr oder weniger unabhängig, sie werden nur vom Postmaster überwacht.

Der Maintenance-Bereich wird wie folgt aufgeteilt:

Checkpoint überprüft die Anfragen an die Datenbank.

Background Writer(BGWriter) ist er Bestandteil des DBMS seit der Version 8.0 und hat die Aufgabe, Daten gesammelt auf die Festplatte zu schreiben. In den früheren Versionen haben parallel laufende Prozesse die Daten gleichzeitig auf die Festplatte geschrieben, was zum Nachteil hatte, dass die Festplatte sehr oft positioniert werden musste (lange Zugriffszeit). Nach der Einführung des BGWriters ist der Schreibprozess wesentlich effizienter geworden.

WAL-Writer(Write-Ahead Logging) hat als Aufgabe die zukünftige von einem Client angefragte Änderungen, wie z.B. eine Tabelle ändern oder Indizes einfügen oder entfernen, in einen Log schreiben, so dass die z.B. wegen eines Programmabsturzes fehlgeschlagenen Änderungen wieder zurückgenommen werden können.

Archiver ist ein optionaler Prozess. Dieser archiviert alle abgeschlossenen Aufgaben, die in dem Prozess-Ordner mit ".ready"-Dateierweiterung gekennzeichnet sind.

Logger schreibt in dem Log-Ordner alle Operationen, die ein Benutzer auf der Datenbank ausführt, Fehler und Warnungen, wenn solcher Zustand im Datenbank System eintritt. Die Log-Dateien erlauben dem Administrator, eine Analyse der Fehler durchzuführen und die Fehler im System zu korrigieren.

Bei der Speicherverwaltung werden zwei Bereiche reserviert:

Global Memory ist für alle Prozesse, die datenbankweit laufen, wie z.B. der Postmaster-Prozess, der Logger oder auch BGWriter, reserviert und dort werden in Shared Buffers alle Informationen zwischengespeichert, bevor sie auf die Festplatte geschrieben werden. Für den WAL-Buffer hat PostgreSQL einen kleinen Extra-Bereich der Global Memory reserviert.

Per-Process Memory wird auch reserviert, sobald eine Anfrage von dem Client kommt.

Architektur von Riak DB Es konnten keine Informationen über den internen Aufbau von Riak DB gefunden werden. Auch die Anfragen an Basho und im Riak-IRC-Channel wurden nicht beantwortet.

3.2.3. Installation und die Einrichtung

In diesem Abschnitt wird der Prozess der Installation behandelt. Da ich mit Ubuntu-Linux als Betriebssystem arbeite, wird genau dieser Aspekt behandelt. Somit wird die Installation unter Windows und anderen Distributionen von Linux, aber auch BSD und Mac OS X, nicht berücksichtigt.

Installation von Cassandra DB Auf der Seite des Herstellers[14] kann der Anwender die genaue Installationsanleitung finden. Nach der Zeitmessung war das der langwierigste Prozess aus allen drei Installationen. Die Installation mit Durchlesen der Instruktionen, Eintragen der nötigen Quelle in die Repositories und der Installation selbst betrug insgesamt knapp 7 Minuten.

Die Installationsanleitung ist nur in englischer Sprache vorhanden. Die Anleitung ist als Schritt-für-Schritt-Anleitung aufgebaut. Allerdings ist alles auf einen fortgeschrittenen Benutzer ausgerichtet, da es nicht erklärt wird, wo irgendwelche Zeilen eingefügt werden, sondern nur gesagt wird, dass es getan werden muss.

Nach der Installation muss die Datenbank einfach gestartet werden. Der Administrator-Zugang ist schon eingerichtet, ein Passwort wurde dabei nicht gesetzt. Wenn weitere Benutzerzugänge benötigt werden, sind diese schnell eingerichtet.

Installation von Riak Die Anweisungen sind auf der Seite von Basho[15] nicht so leicht zu finden, da sie in dem Hintergrund der Dokumentation etwas versteckt sind. Allerdings sind sie sogar auch für einen unerfahrenen Benutzer sehr verständlich geschrieben.

Die Installation ist schnell durchgelaufen, es ist mit über 5 Minuten etwas schneller als die Installation von Cassandra gewesen, konnte aber den ersten Platz nicht erreichen. Bei der Installation sind ebenfalls keine Probleme aufgetreten.

Zum Abschließen der Installation ist eine Portnummer für den Zugriff über das Netzwerk, ein Passwort für das Administrator-Konto einzugeben. Anschließend muss die Datenbank gestartet werden.

[14] [AT13]
[15] [inc15]

Installation von PostgreSQL In dieser Sektion hatte PostgreSQL die besten Werte gezeigt. Es war die schnellste und einfachste Installation. PostgreSQL ist eine weit verbreitete und sehr gut bekannte Datenbank. So hat sie es auch in die Repositories von vielen Linux Distributionen geschafft. Das führt dazu, dass der ganze Prozess vom einfachen Suchen in der Synaptic und anschließender Installation einschließlich aller Abhängigkeiten abgeschlossen ist. Für mit der Kommandozeile vertrauten Benutzer ist die Installation recht schnell durchgeführt.

Falls jemand eine andere Version als in die Repositories angebotene haben möchte, kann diese auch selbst von der Seite des Herstellers herunterladen und nach der Dokumentation installieren. Die Dokumentation ist ebenfalls in englischer Sprache geschrieben, allerdings sind auch auf deutsch verfasste Handbücher zu finden. Sie sind jedoch für etwas ältere Versionen der Datenbank, die immer noch gepflegt werden.

Zum Einrichten eines Benutzers muss der Superuser auf den "postgres"-Account wechseln und darin einen neuen Benutzer anlegen. Die Rechte für den neu angelegten Benutzer werden in psql – der PostgreSQL-Kommandozeile – oder alternativ in pgAdmin3 – dem grafischen Tool – vergeben.

Datenbanksystem / Kriterium	Cassandra	Riak	PostgreSQL
Anleitung zur Installation	✓	✓	✓
Sprachen der Installationsanleitung	englisch	englisch, japanisch	englisch, deutsch, französisch,spanisch,russisch, japanisch
Einrichtung nach der Installation	Anlegen der zusätzlichen Benutzer	Administrator-Account und Zugangsport, neuen Benutzer, wenn benötigt	neuen Benutzer Anlegen
gesamte benötigte Zeit	6min 48sec	5min 51sec	4min 23sec

Tabelle 3.2.: Vergleichstabelle: Installation

3.2.4. Vorstellung der implementierten Schemata

Wie im Kapitel 2.1.2.6 erwähnt wurde, besitzen die NoSQL-Datenbanken im Vergleich zu relationalen DBMS keine ausgeprägten Schemata. Das wurde so entwickelt, weil

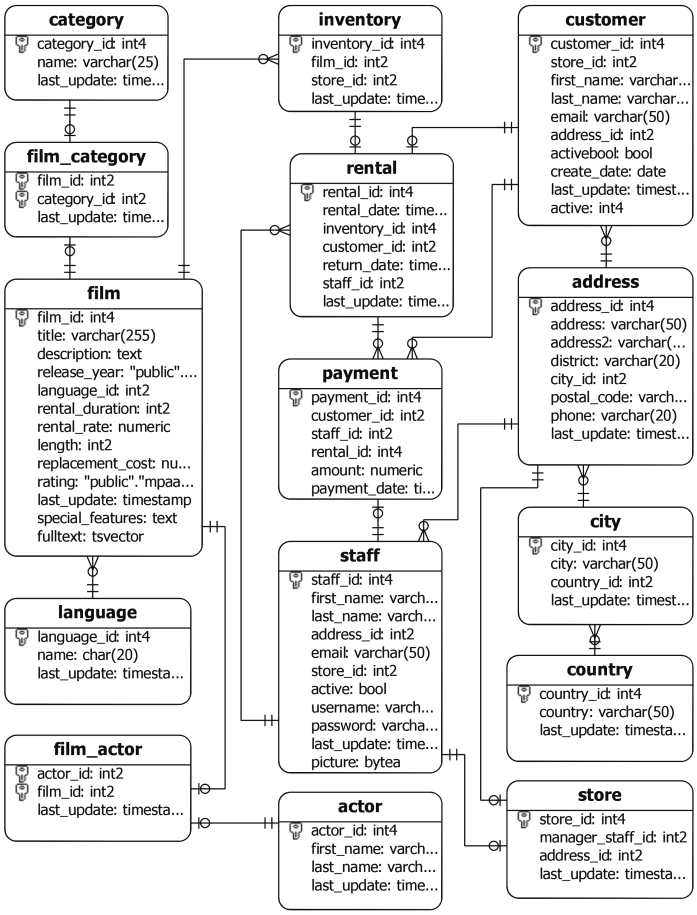

Abbildung 3.5.: Tabellenschema für PostgreSQL

die Joins auf den verteilten Tabellen ungeheuer viel Zeit kosten. Deswegen werden die Daten partitioniert. Daraus resultiert der Unterschied in den Schemas von relationalen SQL- und NoSQL-Datenbanken.

In Abbildung 3.5 ist eine Datenbank zu sehen, die einen vollständigen DVD-Verleihgeschäft darstellt. Die Tabellen sind durch Fremdschlüßel miteinander verbunden. Dieses Konzept ist in den NoSQL-Datenbanken nicht bekannt. Deswegen ist es nötig, die Tabellen zu dernormalisieren, damit Abfragen dieselben Ergebnisse liefern, wie bei den relationalen Datenbank.

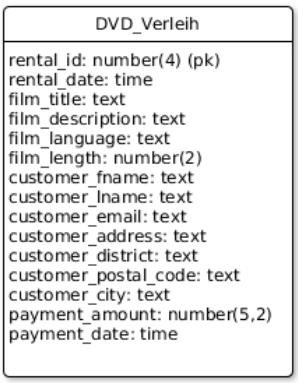

Abbildung 3.6.: Tabellenschema für Cassandra und Riak

Somit kommt es zu dem in Abbildung 3.6 abgebildeten Schema, welches in Cassandra und Riak verwendet werden. Es wurden die "last_update"- und Mitarbeiter-Spalten ausgelassen, weil es sonst die Tabelle noch größer werden würde.

3.2.5. Zugriffsmöglichkeiten auf die Datenbankinstanzen

In diesem Unterkapitel werden Möglichkeiten eines Zugriffes auf die laufende Instanzen der ausgewählten Datenbanken behandelt.

Zugriffsmöglichkeiten auf Cassandra DB Cassandra stellt eine Abfragesprache **"Cassandra Query Language (CQL)**, dessen Syntax dem von SQL sehr ähnlich gebaut ist. Die CQL erlaubt es, die Daten aus der Datenbank bequem abzufragen. Dabei simuliert die Sprache, dass die Daten in einer Tabelle enthalten sind. Die Daten lassen sich in selbst definierten komplexe Typen sowie auch in vordefinierten Listen, Sets und Maps ordnen. Dabei gibt es einige Nachteile von vordefinierten gegenüber den selbst definierten Typen. Die in die Cassandra integrierten komplexen Typen lassen beim Lesen keinen Zugriff auf einzelne Elemente zu. In einer Abfrage kann der Benutzer feststellen, ob ein Element in der Menge der Elemente vorhanden ist, unabhängig davon, ob es sich um eine Liste, ein Set oder eine Map handelt. Der Schreibvorgang lässt einen direkten Zugriff auf ein Element zu, wenn dieser schon vorhanden ist und der Benutzer den Inhalt ersetzen oder ergänzen will. Lese- oder Schreib-Abfragen eines Benutzers werden über das ganze Netzwerk geleitet, so dass die Daten, die auf den anderen Nodes gespeichert sind, auch gelesen oder sogar geändert werden können.

Neben dieser existiert noch die Möglichkeit eines Zugriffs über einen Web-Interface. Dafür stellt Apache eine Thrift API, die eine Schnittstelle zu vielen Programmiersprachen darstellt. Dadurch lassen sich flexible und sichere Zugriffe auf die Daten ausführen.

Zum Ändern der Daten einer Tabelle stellt Cassandra gleich mehrere Varianten zur Verfügung. Der Benutzer kann auswählen zwischen einem Einfügen neuer Daten mit UPDATE-Befehl oder mit Hilfe von einem INSERT-Befehl. Update-Befehl von CQL ist sehr ähnlich dem UPDATE von relationalen Datenbank Systemen aufgebaut, so dass ein Benutzer, der Erfahrungen mit relationalen Datenbanken machte, auch mit Cassandra keine Probleme mit dieser Aufgabenstellung haben sollte. Anders sieht es dagegen mit dem INSERT-Befehl aus. Da es in den relationalen Datenbanken Primary Keys vergeben werden, die auch noch nicht doppelt auftreten dürfen, ist es klar, dass man mit INSERT nicht weit kommt, weil das System sofort einen Versuch der Duplizierung entdeckt und meldet. Bei der Cassandra DB kann der Benutzer den INSERT-Befehl beliebig oft ausführen. Der Inhalt wird in diesem Fall einfach nur überschrieben. Noch eine Besonderheit des Updates ist, dass bei einer nicht existierenden Datenzeile und einem Versuch eine Update auszuführen, ein INSERT durchgeführt wird.

Die Daten können sehr leicht aus der Tabelle gelesen werden. Die Syntax ist wie im Fall mit Insert, Update und Delete ebenfalls dem Befehl von relationalen Datenbank Systemen ähnlich.

Zugriffsmöglichkeiten auf Riak Riak kann den Benutzer nicht mit einer cleveren Abfragesprache überraschen. Alle an die Datenbank gestellten Abfragen werden über die RESTfull-Schnittstelle geleitet werden, unabhängig, ob diese auf der Konsole oder über eine API einer Programmiersprache ausgeführt werden. Die Programmiersprachen bieten etwas mehr Komfort an und sind wiederverwendbar, die Kommandos auf der Konsole sind im Gegenteil schlicht und leichter verständlich.

Die Strategie von Insert und Update von Riak gleicht sich der von Cassandra DB an, d.h. dass man Insert für Updates und umgekehrt benutzen kann. Die Daten sind danach direkt in der Datenbank abgespeichert.

Da die Tabelle als Key-Value-Form aufgebaut ist, läßt sich die Value-Spalte mit neuen Daten leicht erweitern

Zugriffsmöglichkeiten auf PostgreSQL PostgreSQL bietet Werkzeuge, die standard für die relationale Datenbanksysteme sind. Die Möglichkeiten sind etwas breiter, was die

Abfragen angeht, d.h. dass die Daten aus unterschiedlichen Tabellen zusammengestellt werden können und so wird die bessere Datenfragmentierung gewährleistet.

Datenbank System / Kriterium	Cassandra	Riak	PostgreSQL
Daten in Tabellen verwalten	×	×	✓
Daten einfügen	Insert- und Alter-Befehl, RESTful-Schnittstelle	Über die Insert-Variante des Befehls von RESTful-Schnittstelle	Insert-Befehl
Daten abfragen	Select-Befehl	Über die Select-Variante des Befehls von RESTful-Schnittstelle	Select-Befehl
Daten ändern	Insert- und Alter-Befehl	Insert- und Alter-Varianten der Kommandos	Alter-Befehl
Daten löschen	Delete-Befehl	Über die Delete-Variante des Befehls von RESTful-Schnittstelle	Delete-Befehl

Tabelle 3.3.: Übersicht über die Zugriffs- und Änderungsmöglichkeiten auf die Daten

Datenbank System / Kriterium	Cassandra	Riak	PostgreSQL
Insert (5000 Einträge)	0,697±0,112 ms	10,607±3,029 ms	1,148±0,212 ms
Select (100 Mal alle Einträge)	1,085±0,182 ms	10,193±2,811 ms	1,912±0,300 ms
Select mit filtern der Daten	1,008±0,137 ms[16]	9,370±1,712 ms	1,437±0,231 ms
Update 100 Einträge	1,120±0,209 ms	9,929±1,009 ms	1,619±0,388 ms
Delete 100 Einträge	0,833±0,225 ms	11,391±1,091 ms	1,099±0,197 ms

Tabelle 3.4.: Durchschnittszeiten bei Datenzugriffen

Wie man aus der Tabelle 3.4 nicht schwer erkennen kann, hat ein direkter Zugriff einen enormen Geschwindigkeitsvorteil gegenüber einem Zugriff über das Netzwerk. So weist Cassandra die kleinsten Zugriffszeiten aus, welche nur einen kleinen Unterschied

[16]Spalte, nach der gefiltert wird muss zuerst extra indexiert werden

gegenüber denen von PostgreSQL ausmachen. In dieser Tabelle ist die Zeit mit absoluten Fehler angegeben, die für eine Ausführung eines Zugriffes auf die jeweilige Datenbank verbraucht wurde.

3.2.6. Backup und Wiederherstellung

Obwohl die Hardware und Software, auf denen die Datenbanken laufen, viel robuster geworden sind, die Server vor einem Stromausfall geschützt sind, Firewalls und Antiviren-Software uns vor Angriffen von Außen schützen, gibt es immer wieder Situationen, in denen eine Sicherungskopie, der so genannte Snapshot, sehr sinnvoll ist.

Die drei Datenbank-Installationen wurden auf diese Möglichkeit getestet, die gültige Datenbank "unabsichtlich" gelöscht und danach aus dem Snapshot wiederhergestellt.

Dabei muss man auch in Betracht ziehen, dass verteilte Datenbanksysteme die meisten Daten redundant halten, so dass es im Falle eines Ausfalls von einem Knoten nicht zu einem kritischen Verlust kommt. Es existieren trotzdem Techniken, wie der Datenbank Administrator eine Sicherung der Daten durchführen kann.

Riak: Backup und Wiederherstellung Auf der Website von Basho (der Hersteller von Riak DB) im Kapitel über die Backup-Möglichkeiten ist vorab darauf hingewiesen, dass ein Backup eigentlich nicht benötigt wird, aber einfach möglich ist. Dafür macht man eine Kopie des Ordners, in dem die Daten gespeichert sind. Ansonsten gibt es kein integriertes Tool.

Vor dem Backup muss der Knoten zuerst angehalten werden. Wenn es auf einem lokalen Laufwerk passiert, reicht ein einfaches Komprimierungsprogramm, welches aus den Daten ein kompaktes Archiv erstellt und auf der Festplatte ablegt. Für ein Remote-Backup macht man das mit Hilfe des rsync-Befehls, erst nachdem die Daten komprimiert worden sind.

Die umgekehrte Operation, die Wiederherstellung, besteht aus dem Kopieren von der Sicherungsplatte und Entpacken ins Zielverzeichnis. Bei der Wiederherstellung muss der Knoten ebenfalls angehalten sein.

Cassandra: Backup und Wiederherstellung Im Unterschied zu Riak besitzt Cassandra DB einen eingebauten Mechanismus zum Sichern und Wiederherstellen des Datenbestands. Dabei muss die Datenbank bzw. die Instanz, die auf dem Knoten installiert ist, nicht angehalten werden.

Das Backup-Tool wird nicht aus der Datenbank-Umgebung aufgerufen, sondern in dem Terminal. Nach dem Ausführen des Backups liegen im selben Verzeichnis auch andere Snapshots, die früher gemacht wurden.

Für die Wiederherstellung des Datenbestandes wird der Inhalt des Backups in das Wiederherstellungsverzeichnis kopiert und von dort aus mit dem eingebauten Werkzeug in die Datenbank geladen.

Die "nodetoolSoftware hatte keine Schwierigkeit, die Daten zu sichern und danach in die Datenbank wieder aufzuspielen. Es wurden bei dem Vorgang keine Probleme festgestellt. Die Benutzung ist im Vergleich zu den beiden anderen etwas komplizierter, aber mit der Beschreibung in der Dokumentation leicht nachvollziehbar.

PostgreSQL: Backup und Wiederherstellung Bei diesem so lange auf dem Markt agierenden und sehr weit verbreiteten Produkt ist die Technik des Backups und der Wiederherstellung schon seit langer Zeit ein Standard-Bestandteil des Software Paketes. Das "pg_dump"-Programm tut seinen Dienst zuverlässig und in diesem Fall sind ebenfalls keine Fehler aufgetreten.

In der Tabelle 3.5 sind die Erfahrungen zusammengefasst.

Datenbank System / Kriterium	Cassandra	Riak	PostgreSQL
Backup-Möglichkeit	nodetool	tarball + rsync	pg_dump
Backup ohne Fehler	✓	✓	✓
Wiederherstellung der Daten	nodetool	rsync	pg_dump
Wiederherstellung ohne Fehler	✓	✓	✓
Vergleich der gesicherten und hergestellten Daten	✓	✓	✓

Tabelle 3.5.: Backup und Wiederherstellung der Daten

3.2.7. Verbindung mit Programmiersprachen

Alle drei Datenbanksysteme können leicht durch Programmiersprachen angesprochen werden. Es wurden in allen drei Datenbanken durch kurze Programme die vorhandenen Datensätze erzeugt, ausgelesen, verändert und gelöscht. Es sind bei der Ausführung keine Fehler aufgetreten.

Damit die Anbindung an die NoSQL-Datenbanken klappt, müssen zuerst einige Bibliotheken eingebunden werden, die auf der Seite des Datenbankherstellers entweder zur Download-Seite verlinkt oder direkt zum Download angeboten sind.

Die Programmieranleitungen für die NoSQL-Datenbanken sind ebenfalls in der Dokumentation gegeben. Die Anleitungen sind detailliert beschrieben und haben auch für Anfänger verständlichen Aufbau. Es wurde nur ein Problem mit Cassandra festgestellt, weil ein TimeUUID als Primary Key in der Tabelle gewählt wurde und dafür gibt es keinen äquivalenten Datentyp in der Java und so musste der Typ durch einen einfachen UUID simuliert werden.

```
1 UUID personid = UUIDs.timeBased();
```

Listing 3.1: Schlüßelgenerierung (Cassandra)

Als Beispiel wurde eine Personentabelle erstellt. Die Nutzdaten werden als ein JSON-Datensatz eingefügt. Dass passiert im Code mittelse einer Map, wo der Schlüßel und Wert als String gespeichert sind.

```
1 mss.put("name", "Vasja␣Pedalkin");
2 mss.put("adresse", "125413␣Plazza␣Av.␣App.#14");
3 mss.put("email", "goodZone@trashmail.com");
4 mss.put("geschlecht", "male");
5 mss.put("ort", "Traumland");
6 mss.put("plz", "12AB56");
7
8 String persondata = mapToString(mss);
```

Listing 3.2: Erstellen von JSON-Datensatz für die Cassandra Datenbank

Die Operationen an der Datenbank sehen dann wie folgt aus:

```
1 String sql = "INSERT␣INTO␣person␣(personid,␣person_data)"
2         + "values␣("+ personid + "," + persondata + ")";
3 session.execute(sql);
4 session.execute("update␣person␣SET␣person_data['email']="
5         + "'mollis@ametrisus.org'␣where␣personid="+personid+";");
6 session.execute("DELETE␣FROM␣person␣WHERE␣personid␣=␣"+personid);
7 ResultSet results = session.execute("select␣*␣from␣person␣where"
8         + "␣person_data␣contains␣'mollis@ametrisus.org';");
```

Listing 3.3: Datenbankoperationen mit Cassandra

In dem ResultSet sind danach alle Datensätze mit der entsprechenden Email-Adresse enthalten.

Im Fall von Riak muss zwangsläufig Maven-Technologie eingesetzt werden, damit die fehlenden Bibliotheken installiert und benutzt werden können.

Es gibt einen Aspekt, der nicht getestet werden konnte. Es handelt sich dabei um Hibernate für die NoSQL-Datenbanken. Das Projekt startete in 2011 und bis jetzt gibt es keine vernünftige Anleitung für einen Projekt mit Cassandra oder Riak. Die Beschreibung des Vorgangs auf der Hibernate Seite[17] konnte nicht richtig nachvollzogen werden, weil immer ein Fehler kam, dass die Datenbank nicht angesprochen werden konnte.

Mit der PosgreSQL gab es auch hier kein Problem, weil es eine schon länger erprobte Technik für die relationale Datenbanken ist.

```
//Connection settings
String url = "jdbc:postgresql://localhost/dvdrental";
String user = "myDB";
String password = "myPassword";

try {
        //connect with settings
        con = DriverManager.getConnection(url, user, password);
        st = con.createStatement();
        //test connection -> result is the version of data base
        rs = st.executeQuery("SELECT␣VERSION()");
```

Listing 3.4: PostgreSQL Verbingungsaufbau

Nachdem der Verbindungsaufbau erfolgreich stattgefunden hat, können Abfragen auf die Datenbank durchgeführt werden.

```
//Select all data from existing Table
rs = st.executeQuery("SELECT␣*␣FROM␣category;")
//Ausgabe der gefundenen Datens"atze
while (rs.next()){
        System.out.println( rs.getInt("category_id")
        + "\t" +rs.getString("name")
        + "\t" +rs.getDate("last_update"));
}
```

Listing 3.5: PostgreSQL Select-Abfrage

Für INSERT-, UPDATE- und DELETE-Operationen wird statt ëxecutQuery ein ëxecuteUpdateKommando verwendet, ansonsten bleibt es genau so einfach.

[17] [Hib15]

```
9  int res = st.executeUpdate("INSERT␣INTO␣category" +
10         "(name,␣last_update)␣VALUES␣('Mystics',␣now());");
```

Listing 3.6: PostgreSQL Select-Abfrage

In der Tabelle 3.6 sind die Ergebnisse dieses Kapitels in kürze zusammengeführt. Diese Tabelle zeigt, welche Datenbank welche Eigenschaften beim Programmieren realisiert und was noch fehlt.

Datenbanksystem / Kriterium	Cassandra	Riak	PostgreSQL
Verbindung zu der Datenbank aus IDE	✓	✓	✓
Insert durchführen	✓	✓	✓
Select durchführen	✓	✓	✓
Update durchführen	✓	✓	✓
Delete durchführen	✓	✓	✓
Hibernate anbindung	*times*	*times*	✓

Tabelle 3.6.: Zugriff und Verändern der Daten mittels eines Programms

3.2.8. Peer-to-Peer Eigenschaften der Datenbanken

In diesem Kapitel geht es um die Eigenschaften, die verteilte NoSQL-Datenbanken besitzen. Im Gegensatz zu NoSQL- haben die klassischen relationalen Datenbanken diese Eigenschaft nicht. Es liegt daran, dass die Server-Client-Architektur sich nicht so gut verteilen lässt, wie es bei den ohne priorisierten Knoten verteilten NoSQL-Datenbanken der Fall ist. Die Client-Server-Architektur erlaubt zwar, dass die Knoten auf unterschiedlichen und sogar entfernten Rechnern installiert sind, muss aber dafür garantieren, dass alle Knoten permanent erreichbar sind.

Die Architektur der NoSQL-Datenbanken erlaubt es, die Knoten beliebig zu verteilen und sorgt sich auch nicht darum, dass ein oder sogar mehrere Knoten aus dem Netz verschwinden oder hinzugefügt werden. Dieses Verhalten kann nur die Anzahl der gelieferten Daten beeinflussen, eventuell auch die Qualität der Daten. Es werden aber alle zu diesem Zeitpunkt verfügbare Knoten durchsucht und die für die gestellte Abfrage relevanten Daten geliefert.

Wie im Kapitel 3.2.4 beschrieben wurde, bestehen die Cassandra und Riak Datenbanken aus zwei Knoten auf zwei Rechnern. Für die Darstellung eines Ausfalls wurde der Knoten ausgeschaltet und danach eine Anfrage an die Datenbank gestellt. Die Anfrage

gelang sowohl über denselben Rechner, auf dem der funktionierende Knoten installiert ist, als auch über das Netz. Bei der Durchführung der Abfragen mit einem "ausgefallenen" Knoten wurden einige Ergebnisse nicht gefunden, die nur auf dem fehlenden Knoten gespeichert sind. Danach wurde der zuerst abgetrennte Knoten (Notebook) wieder verbunden und der Andere (PC) wurde entfernt und der Versuch wurde wiederholt.

Die Abfragezeit der über ein Netzwerk gestellten Anfragen an die Datenbank hat bis zehn Millisekunden Unterschied im Vergleich zu den direkten Anfragen. Dieser Unterschied kann dadurch erklärt werden, dass die Anfrage durch die Netzwerkverbindung etwas länger dauert und dazu noch die Suche auf dem anderen Knoten ausgeführt wird.

In der Dokumentation zu Cassandra ist beschrieben, dass es nicht empfehlenswert ist, alle Knoten eines Clusters zu den Replikationsknoten zu machen. Dies gilt allerdings erst ab einer Größe von 4 Knoten, weil dann die Datenbank den Ausfall von einem mit replizierten Daten befüllten Knoten gut verkraften kann. Somit sind beide in dem Testsystem verwendete Computer als replizierende Knoten für die Datenbank eingerichtet. Solche Einschränkung ist laut Dokumentation bei Riak nicht gegeben.

Der Ausfall eines Knotens wird dadurch kompensiert, dass die anderen vorhandenen Knoten die Daten zur Verfügung stellen. Der Benutzer bemerkt den Ausfall nicht, sondern erhält als Antwort eventuell weniger Treffer, wenn es sich bei dem Ausgefallenen Knoten um einen Replikationsknoten handelt.

Die aktiver Replikationsknoten bekommen und senden ihre Daten mit Hilfe von Peer-to-Peer Netzwerk. Das durch Torrent-Protokoll weit bekannte und verbreitete Methode hat eine gute Eigenschaft, nähmlich dass eine getrennte Verbindung keinen Einfluss auf die Konsistenz der Datenbank hat. Das führt dazu, dass die nicht synchronisierten Datensätze später nachgezogen werden können. Bei der Untersuchung war das auch der Fall, weil die Datensätze nicht alle auf Anhieb sondern Stück für Stück kopiert wurden.

Die neu eingefügten Datensätze werden zuerst in dem direkt angesprochenen Knoten gespeichert und erst bei der Synchronisierung auf die andere im Cluster verbundene Knoten übertragen.

Beide NoSQL-Systeme schafften die Datensynchronisation ohne Probleme, wenn ein Knoten ausgefallen ist oder hinzugefügt wurde. Die integrierten Peer-to-Peer-Mechanismen erlauben, dass die Knoten ohne Einfluss auf die Arbeit der ganzen Datenbank ein- und austreten können.

4. Ergebnisse

Die Installation der drei Datenbanken ist sehr gut in der Dokumentation beschrieben und läuft ohne Probleme durch. Die Datenbank-Instanzen lassen sich bequem per Kommandozeile ein- und ausschalten, so dass diese beim Testen nicht die Testergebnisse beeinflussen können.

Sicherheit ist bei allen Kandidaten groß geschrieben. So sind während der Untersuchung für jedes System kleinere Updates erschienen. Das zeigt, dass alle Systeme aktiv weiter entwickelt werden und der Benutzer Bugfixes und Patches erhält. Was die interne Sicherheit angeht, so steht Riak etwas hinter den beiden anderen Konkurrenten. Direkt nach bzw. am Ende der Installation legten Cassandra und PostgreSQL ein Administrator-Kennwort an. Bei Riak wurde dieser Schritt einfach übersprungen und als Option dem Anwender gelassen. Es existiert aber die Möglichkeit, das Passwort im Nachhinein anzulegen. Das Kontoverwaltungssystem ist bei Riak komplizierter gestaltet, als es im Fall von Cassandra und PostgreSQL ist. Die beiden letzteren haben ein ganz einfaches Rechteverwaltungssystem. Dabei ist das von Cassandra an das System von SQL angelehnt. Die Syntax ist bei den beiden sehr ähnlich aufgebaut und kann aus der cql-Shell von Cassandra bzw. psql-Shell von PostgreSQL kontrolliert und verändert werden. Die Rechtevergabe von Riak ist ausschließlich durch die Manipulation der Konfigurationsdatei möglich.

Während dieser Untersuchung wurde festgestellt, dass alle drei Datenbanken sehr stabil laufen, allerdings nicht immer für jeden Zweck geeignet sind. So ist PostgreSQL gut für die strukturierte Daten geeignet, die platzsparend gespeichert werden sollen. Dafür sind die Cassandra und Riak nicht geeignet, weil man beim Speichern der Daten nicht auf Redundanz verzichten kann. Das macht diese Datenbanken nicht schlechter, da sie in anderen Gebieten glänzen können, wie z. B. Zugriffsgeschwindigkeit und Datenverteilung.

Die Datenverteilung macht PostgreSQL zwar auch, aber dabei bleibt der Client von dem Server abhängig. Die Verteilung von Riak und Cassandra hat dieses Problem nicht, da die beiden Datenbanksysteme keine Server-Client-Architektur besitzen, alle Knoten

sind gleichwertig und das macht diese Datenbanken so ausfallsicher. Auch wenn mehrere Knoten ausfallen sollten, hat die Datenbank keine Probleme mit den Abfragen, weil dann die redundanten Daten aus den anderen Knoten geholt werden. Ein Ausfall von einem Teil des PostgreSQL-Systems stellt insofern ein Problem dar, dass es keine bzw. falsche Daten geliefert werden. Bei einem Ausfall des Servers kann der Benutzer überhaupt nicht auf die Datenbank zugreifen, wenn das Client-System ausfällt, dann erhält der Benutzer einfach eine leere Menge an Daten. Allerdings ist dieses Verhalten von PostgreSQL während der Untersuchung nicht simuliert worden, weil es nichts mit dem Peer-to-Peer-Verhalten zu tun hat.

Mit Blick auf die Antwortzeit auf die gestellten Abfragen steht Riak gegenüber Cassandra und PostgreSQL etwas nach. Die beste Antwortzeit im Vergleich hat die Cassandra Datenbank gezeigt, um einige Zehntelsekunden langsamer war PostgreSQL und Riak hat sogar einige Sekunden länger gebraucht. Es kann daran liegen, dass Riak keine eigene Oberfläche für Anfragen besitzt, sondern die Werkzeuge des Betriebssystems benutzt. Dies soll aber kein Nachteil sein, da die Abfragen trotzdem verständlich und übersichtlich aussehen und ein unerfahrener Benutzer schnell eingearbeitet werden kann. Die Abfragesprache von Cassandra ist sehr ähnlich der von relationalen Datenbanksystemen aufgebaut ist. Der größte Unterschied besteht nur darin, dass die Tabellen nicht einfach durch Fremdschlüssel verknüpft werden können. Abfragen, die mehrere Tabellen betreffen, können leicht durch Programmiersprachen gelöst werden.

Alle drei Datenbanksysteme unterstützen mehrere Programmiersprachen. Bei der Vergleichsuntersuchung wurden Testfälle mit Java gebaut, laut der Dokumentation stehen auch solche gängigen Sprachen wie z. B. C# und Python zur Verfügung. Die beiden NoSQL-Datenbanken sind stärker auf die Sprachen, die mit der Web-Entwicklung zu tun haben, wie z. B. PHP, Ruby, JavaScript, spezialisiert, weil sie viel in Umgebung von Internet verwendet werden. Das soll aber nicht heißen, dass sie nicht in den Standardapplikationen verwendet werden dürfen. Umgekehrt gilt auch, dass PostgreSQL in allen Bereichen inkl. Web-Entwicklung verwendet wird.

Noch ein weiterer Unterschied, den die Programmiersprachen ans Licht bringen ist, dass es bis jetzt nur die relationalen Datenbanken die Synchronisierung über Hybernate erlauben, die beiden NoSQL-DBMS dieses Verfahren aber nicht unterstützen. Es gab einen Versuch, Cassandra Hybernate beizubringen, die letzten Updates auf der Website waren von vor 3 Jahren.

Eine Vereinfachung der Datenverwaltung für den Benutzer stellt die grafische Oberfläche (GUI) dar. In diesem Punkt steht ebenfalls nur PostgreSQL GUI zur Verfügung, die

beiden anderen Datenbanksysteme haben keine solche Programme. Ein Programmierer kann für sich eine GUI selbst erstellen und mit dem eigenen Programm verwenden.

In der Tabelle 4.1 sind die Ergebnisse der Untersuchung zusammengeführt.

Kriterium \Datenbanksystem	Cassandra	Riak	PostgreSQL
Installation	einfach	einfach	sehr einfach
Dokumentation	vielfältig, vor allem Internet und Bücher	nur auf der Seite des Herstellers, Bücher in geringer Anzahl	umfangreiches Standard Handbuch, sehr viele Bücher und Quellen im Internet
Stabilität	keine Probleme aufgetreten	keine Probleme aufgetreten	keine Probleme aufgetreten
Programmierung	leicht verständliche Anleitung, mehrere kompatible Sprachen	relativ leicht verständliche Anleitung, mehrere kompatible Sprachen	sehr leicht verständliche Anleitung, sehr viele kompatible Sprachen
Peer-2-Peer-Eigenschaften	Ausgefallener Knoten wird schnell bemerkt und aus dem System entfernt. Ein neu hinzugekommener Knoten wird schnell entdeckt und leicht in das Netz integriert. Die Synchronisation passiert über sogennante Sync-Knoten, die dann die Synchronisation auf andere Knoten verteilen	Ausgefallener Knoten wird nach einiger Zeit bemerkt und aus dem System entfernt. Ein neu hinzugekommener Knoten wird schnell entdeckt und in das Netz integriert. Die Synchronisation passiert ebenflls über sogennante Sync-Knoten, die dann die Synchronisation auf andere Knoten verteilen	nicht vorhanden
Verwendung	Hauptsächlich für Internetanwendungen, die schnelle Zugriffszeiten benötigen	Hauptsächlich für Internetanwendungen, die schnelle Zugriffszeiten und schemalose Datenbank benötigen	sowohl Desktopanwendungen als auch für das Internet, u. z. wo komplexe Arbeitsabläufe eingesetzt werden

Tabelle 4.1.: Ergebnisse der vergleichender Untersuchung

5. Zusammenfassung und Ausblick

5.1. Zusammenfassung

In dieser Arbeit wurde eine Untersuchung durchgeführt, die Ähnlichkeiten und Unterschiede von relationalen (PostgreSQL) und Peer-to-Peer-Datenbanken (Cassandra und Riak) vergleichen. Diese vergleichende Untersuchung zeigt die Möglichkeiten, die jedes der Datenbanksysteme mit sich bringt und in welchem Kontext diese Möglichkeiten verwendet werden können.

Für die Untersuchungen der einzelnen Datenbanksysteme wurde eine Testumgebung erstellt. Die Architektur der Testumgebung basiert auf Abbildung 3.2. Mit Hilfe dieser Testumgebung wurden die kleinen Mängel in der Dokumentation festgestellt, die eine Einrichtung schwieriger machen. Weitere Tests haben die Funktionen von Peer-to-Peer Datenbanken erlaubt. Diese Tests waren erfolgreich und hatten keine Fehler in der Funktionsweise der Datenbanksysteme gezeigt.

Einen kleineren Aufwand stellte das Erstellen und Betreiben der Datenbank selbst dar. In dieser Hinsicht gab es keine nennenswerte Probleme. Die ungewohnte Syntax von Riak wird nach einiger Zeit schnell zur Normalität und man sie merkt nicht mehr. Ein großer Nachteil von Riak ist, dass es keine Befehlsvervollständigung existiert, wie es im Falle von Cassandra und PostgreSQL ist.

Auch die fehlenden GUI-Tools stellen ein Nachteil für die NoSQL-Datenbanken dar. Dadurch bietet PostgreSQL eine komfortablere Verwaltungs- und Arbeitsumgebung.

In der Praxis haben alle Datenbanksysteme sich gut geschlagen und die Stabilität und Reife der Produkte gezeigt.

5.2. Ausblick

An den hier vorgestellten Untersuchungen können an vielen Stellen weitere Untersuchungen durchgefürht werden. So besteht die Möglichket, Durchführung von Analyse der Peer-to-Peer-Eigenschaften der beiden NoSQL-Datenbanken zu vertiefen und aus-

zuweiten. Damit kann ein besserer Vergleich zwischen Riak und Cassandra weitere Ergebnisse für ein tieferes Verständnis mit sich bringen.

Eine Fortsetzung der Untersuchung könnte darin bestehen, ein Netzwerk aus mehr als zwei Knoten zu erstellen und dieses mit einem selbst programmierten GUI zu betreiben. Als weitere Vertiefung wären dann die Reaktion der Datenbank auf mehrere gleichzeitige Ausfälle der Knoten während einer Abfrage.

Diese Arbeit Untersucht nur die zwei Klassen der Datenbanksysteme. Andere Systemtypen könnten weitere Erkenntnisse einbringen.

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Literaturverzeichnis

[And10] J Anderson. *CouchDB the definitive guide.* O'Reilly Media, Inc, Sebastopol, Calif, 2010.

[AT13] Apache.org-Team. Homepage von cassandra-projekt, 2013.

[Bin10] Simon Bin. Key value stores: Dynamo und cassandra. Seminararbeit, Januar 2010.

[BMSV02] Ranjita Bhagwan, David Moore, Stefan Savage, and Geoffrey M. Voelker. Replication strategies for highly available peer-to-peer storage, 2002.

[Bro09] J. Browne. Brewer's cap theorem, Januar 2009.

[Buc] DATACOM Buchverlag. Das große online-lexikon für informationstechnologie.

[CDG+06] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: A distributed storage system for structured data. electronic, 2006.

[Cod70] E. F. Codd. A relational model of data for large shared data banks. *Magazine Communications of the ACM*, 13(6):377–387, June 1970.

[CQL15] CQL. *CQL3.* Apache Foundation, v3.1.7 edition, 2015.

[Cyb] CyberTec. Postgresql architektur. Website.

[Dat15] DataStax. Arbeitsweise von cache in cassandra. electronic documentation, 2015.

[Edl11] Stefan Edlich. *NoSQL : Einstieg in die Welt nichtrelationaler Web 2.0 Datenbanken.* Hanser, München, 2011.

[Ger14] Wolfgang Gerken. Vorlesung datenbanken, 2014.

[GHOS96] Jim Gray, Pat Helland, Patrick O'Neil, and Dennis Shasha. The dangers of replication and a solution. *SIGMOD Rec.*, 25(2):173–182, June 1996.

[GL02] Seth Gilbert and Nancy Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *ACM SIGACT News Vol. 33*, Juni 2002.

[Gra14] Nico Grau. *ISO 21500 Project Management Standard Characteristics, Comparison and Implementation.* Shaker, Aachen, 2014.

[Hib15] Hibernate.org. Getting started with hibernate ogm. electronical, 2015.

[Hos02] Wolfgang Hoschek. A unified peer-to-peer database protocol, 2002.

[inc15] Basho inc. offizielle seite von riak db, 2015.

[Iso13] Iso. Iso/iec/ieee 29119-4: Test techniques, 2013.

[Kel10] D. Kellogg. Six thoughts on the nosql movement, 2010.

[Kem06] Alfons Kemper. *Datenbanksysteme : eine Einführung.* Oldenbourg, München u.a, 2006.

[Kov11] Kristof Kovacs. *Gegenüberstellung der NoSQL Datenbanken*, 2011.

[Kur08] Johann Kurz. Cloud computing - it in der "wolke", November 2008.

[Lea10] N. Leavitt. Will nosql databases live up to their promise?, Januar 2010.

[LM09] A. Laksham and P. Malik. Cassandra - a decentralized structured storage system, 2009.

[Mer10] D. Merriman. *Security and Authentication - MongoDB.* MongoDB, 2010.

[Red12] Eric Redmond. *Sieben Wochen, sieben Datenbanken.* O'Reilly, Köln, 2012.

[Res10] Monash Research. Cassandra technical overview, 2010.

[Sad13] Pramod Sadalage. *NoSQL distilled : a brief guide to the emerging world of polyglot persistence.* Addison-Wesley, Upper Saddle River, NJ, 2013.

[SMK+01] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications, 2001.

[SS05] Yasushi Saito and Marc Shapiro. Optimistic replication. *ACM Comput. Surv.*, 37(1):42–81, March 2005.

[Ste09] Rene Steiner. *Grundkurs relationale Datenbanken : Einführung in die Praxis der Datenbankentwicklung für Ausbildung, Studium und IT-Beruf.* Vieweg + Teubner, Wiesbaden, 2009.

[Tiw11] Shashank Tiwari. *Professional NoSQL.* Wiley, Indianapolis, IN, 2011.

[Weh05] R. Steinmetz & K. Wehrle. *Peer-to-Peer Systems and Applications (Lecture Notes in Computer Science / Information Systems and Applications, incl. Internet/Web, and HCI).* Springer, 2005.

[Wor02] John Worsley. *Practical PostgreSQL.* O'Reilly & Associates, Sebastopol, CA, 2002.

Appendix

A. Abkürzungsverzeichnis

ACID	Atomicity, Consistancy, Isolation, Durability
ACL	Access control list
API	Application programming interface
BASE	Basically Available, Soft state, Eventual consistency
CAP	Consistancy, Availability, Partition Tolerance
DB	Datenbank
DBS	Datenbanksystem
DBMS	Datenbankmanagementsystem
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
ISO	International Organization for Standardization
IT	Information technoligy
JSON	JavaScript Object Notation
K/V-Store	Key/Value-Store
NoSQL	Not Only SQL
P2P	Peer-to-Peer
RDBMS	relationale Datenbank Management System
SQL	Structured Query Language

Tabelle A.1.: Im Text vorkommende Abkürzungen

B. Glossar

Begriff	Bedeutung
API	ein Programmteil, der von einem Softwaresystem anderen Programmen zur Anbindung an das System zur Verfügung gestellt wird[1].
Berkeley Software Distribution (BSD)	eine Unix-Betriebssystem-Variante, die von der University of California, Berkley, im Jahre 1997 entwickelt wurde.
Betriebssystem (BS)	eine Menge von systemnahen Programmen oder Modulen für die Verwaltung der physischen Ressourcen des Systems, das Laden und Ausführen der Programme und die Rechner-Benutzer-Schnittstelle.
Bibliothek	eine Sammlung von Routinen, Prozeduren, Programmteilen oder Programmen, die häufig gebraucht werden und die in einer Bibliotheksdatei zusammengefasst sind. Die einzelnen Routinen und Prozeduren können vom Programmierer aus der Library abgerufen und für die Programmerstellung benutzt werden.
Cassandra	eine 2008 von Facebook entwickelte Wide-Column-Stores Datenbank.
Datenbank (auch DB oder DBS)	ein System zur elektronischen Datenverwaltung. Die wesentliche Aufgabe eines DBS ist es, große Datenmengen effizient, widerspruchsfrei und dauerhaft zu speichern und benötigte Teilmengen in unterschiedlichen, bedarfsgerechten Darstellungsformen für Benutzer und Anwendungsprogramme bereitzustellen

Continued on next page

Tabelle B.1 – continued from previous page

DBMS	die Verwaltungssoftware zu der Datenbank. Sie verwaltet die Daten, die in der Datenbank gespeichert werden.
Document-Store	Siehe Unterkapitel-2.1.2.6
GNU-GPL-Lizenz	stellt sicher, dass man Quellcodes anfordern und sie für weitere Software und eigene Anwendungen benutzen kann.
GNU (GNU is not Unix)	Es handelt sich dabei um ein Projekt, das 1984 ins Leben gerufen wurde mit dem Ziel ein offenes, lizenzfreies Betriebssystem, das ähnlich Unix sein sollte, zu entwickeln.
Hibernate	ein Framework zur Abbildung von Objekten auf relationalen Datenbanken für die Programmiersprache Java - es wird auch als Object Relational Mapping Tool bezeichnet.
Join	dienen in SQL zur Verbindung von Tabellen und/oder Views.
JSON	ein kompaktes Format zum Austausch von Daten, die in verschiedenen Programmiersprachen implementiert wurden.
Key	Ein Schlüssel dient der eindeutigen Identifizierung eines Datensatzes (Tupels) in einer relationalen Datenbank - jede Relation hat mindestens einen Schlüssel
Key-Value-Store	Siehe Unterkapitel-2.1.2.6
Linux	ein GPL v.3-lizenziertes Betriebssystem, das sich durch eine hohe Portabilität und Plattformneutralität auszeichnet.
Maven	ein Projekt der Apache Software Foundation für das Konfigurationsmanagement von Software und bezeichnet mit Maven 2 konkret ein Build-Management-Tool.

Continued on next page

Tabelle B.1 – continued from previous page

NoSQL	Datenbanken, die einen nicht-relationalen Ansatz verfolgen.Diese Datenspeicher benötigen keine festgelegten Tabellenschemata und versuchen, Joins zu vermeiden, sie skalieren dabei horizontal
Open-Source-Software	Programme, dessen Quellcode frei zugänglich ist. Der Quellcode solcher Programme kann beliebig weiter entwickelt werden. Die Weiterentwicklungen und Modifikationen müssen unter dem Namen erfolgen unter Beibehaltung des originären Urheberrechts.
Peer-to-Peer-Netzwerk (P2P)	sind Rechnernetze bei denen alle Rechner im Netz gleichberechtigt zusammen arbeiten. Das bedeutet, dass jeder Rechner anderen Rechnern Funktionen und Dienstleistungen anbieten und andererseits von anderen Rechnern angebotene Funktionen, Ressourcen, Dienstleistungen und Dateien nutzen kann
Redis	ist eine Open Source lizenzierte, erweiterte Schlüßel-Wert-Datenbank.
Relation	beschreibt eine Menge von Tupeln - die Datensätze einer relationalen Datenbank.
Replikation	ein Verfahren der Datensicherung bei dem dieselben Daten von einem Speichermedium auf ein oder mehrere andere Speichermedien kopiert werden.
REpresentational State Transfer (REST)	ein Architekturstil mit dem Webservices realisiert werden können. Es benutzt Prinzipien, die in großen, verteilten Anwendungen wie im World Wide Web (WWW) eingesetzt werden.
Riak	eine der bekanntesten Key-Value-Datenbanken, die von Basho entwickelt wurde.
SQl	eine Datenbanksprache zur Definition von Datenstrukturen in relationalen Datenbanken sowie zum Bearbeiten (Einfügen, Verändern, Löschen) und Abfragen von darauf basierenden Datenbeständen.

Continued on next page

Tabelle B.1 – continued from previous page

Unix	ein von den Bell Laboratories in den 70er Jahren entwickeltes Betriebssystem für Minicomputer, inzwischen für einen weiten Bereich von Rechnern vom Personal Computer (PC) bis hin zum großen Mainframe verfügbar.
Wide-Column-Store	Siehe Unterkapitel-2.1.2.6
Windows	ein 1983 von Microsoft eingeführtes Betriebssystem für PCs, auf dem eine Reihe bekannter Windows-Varianten basiert. Windows wurde als grafische Benutzeroberfläche von dem Disk Operating System (DOS) von Microsoft konzipiert.

Tabelle B.1.: Glossar

[1] die nötigen Informationen wurden mit Hilfe von [Buc] zusammengestellt

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(5) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 14. Juli 2015 Pawel Albrant