



**Hochschule
Bonn-Rhein-Sieg**
University of Applied Sciences

Schemalose Datenbank

MongoDB als Datenbank für Document Stores

**Falls erforderlich: Zur Erlangung des akademischen Grades eines
Master of Science
- Studiengang MCS -**

Fachbereich Informatik
Referent: Prof. Dr. Knolle
Falls erforderlich: Korreferent: B. Mager

eingereicht von:
Franz-Dominik Dahmann
Matr.-Nr. 9019745
Münchhausenweg 30
53340 Meckenheim

Sankt Augustin, den 06.11.2016

Zusammenfassung

Moderne Datenbanken sehen sich mit diversen Problemen konfrontiert. Es sollen extrem große Datenmengen verarbeitet werden und die Antwortzeiten sollen niedrig gehalten werden. Ferner soll es für sehr viele Nutzer gleichzeitig möglich sein Abfragen zu stellen. Aus diesen Gründen ist es in modernen Datenbanken notwendig ist, Datenstrukturen möglichst flexibel und performant zu speichern. Diese Probleme sollen mit verschiedenen Ansätzen aus dem Umfeld der NoSQL Datenbanksysteme gelöst werden. Die dokumentenbasierte Speicherung versucht für Probleme, welche sich meist auf große Mengen mit heterogenen Daten beziehen, einen Lösungsansatz zu finden. In diesem Umfeld ist die Funktionsweise der MongoDB äußerst populär, da sie auf hohe Leistung, große Datenmengen, hohe Flexibilität und einfache Skalierbarkeit ausgelegt ist. Die Grundlage für die der MongoDB zugrunde liegenden dokumentenbasierte Speicherung bieten die Key-Value Stores, welche unformatierte Werte unter einem Key persistieren. Document Stores vereinen den Vorteil der Schemafreiheit der Schlüssel-Wert Zuordnungen mit der Möglichkeit zur Strukturierung von Daten. Dadurch sind Datenbanken wie MongoDB in der Lage komplexere JSON ähnliche Datentypen zu verarbeiten und basierend auf ihren Attributen zu indexieren. Die hier umrissenen theoretischen Grundlagen werden detaillierter beschrieben und in dem NoSQL Kontext bewertet.

Um die theoretischen Grundlagen zu verdeutlichen werden mithilfe eines Anwendungsszenarios die Vorteile von MongoDB verdeutlicht. Grundlage für das Anwendungsszenario ist das logische Schema einer relationalen Datenbank, in der sowohl Abschlussarbeitsthemen, als auch alle beteiligten Personen, wie beispielsweise Studenten oder Prüfer, sowie Organisationen, gespeichert sind.

Ziel des Anwendungsszenarios ist es, dieses Schema mit Hilfe von MongoDB umzusetzen. Ferner soll die Eignung von MongoDB in diesem Kontext analysiert und bewertet werden. Damit MongoDB in diesem Kontext bewertet werden kann, werden unter anderem Performancetests an der Datenbank durchgeführt. Weiter soll eine Web-Anwendung implementiert werden, welche die grundsätzlichen Datenbankoperationen an der MongoDB veranschaulicht.

[Edl11] [Sad13] [Red12] [Han11] [Tud11] [And16b] [Dan16b] [And16a] [Dan16a] [Boi12]
[Par13] [Abr13] [Far14] [Sha15] [How14] [?]

Inhaltsverzeichnis

Zusammenfassung	II
1 Ausgewählte Persistenzmodelle	1
1.1 Key-Value Stores	1
1.1.1 Einleitung	1
1.1.2 Eigenschaften	1
1.1.3 Datenmodell	1
1.1.4 Abgrenzung zu Document-Stores	1
1.1.5 Anwendungsfälle	2
1.1.6 Bewertung	2
1.2 Document Stores	2
1.2.1 Einleitung	2
1.2.2 Eigenschaften	2
1.2.3 Datenmodell	3
1.2.4 Anwendungsfälle	4
1.2.5 Bewertung	4
2 Persistenzsysteme und Big Data Frameworks	5
2.1 MongoDB	5
2.1.1 Einleitung	5
2.1.2 Geschichte	5
2.1.3 Anwendungsfälle	6
2.1.4 Einrichtung	6
2.1.5 Verwendung	8
2.1.6 Grenzen	8
2.1.7 Zukunft	8
3 Ausgewählte Anwendungsszenarien	9
3.1 Web 2.0 - Abschlussarbeiten-DB	9
3.2 MongoDB	9
Literaturverzeichnis	10

1 Ausgewählte Persistenzmodelle

1.1 Key-Value Stores

1.1.1 Einleitung

Key-Value Stores oder Key-Value Datenbanken sind eine der einfachsten Formen der NoSQL Datenbanken. Sie können beliebige Werte (englisch: Values) unter einem definierbaren Key persistieren bzw. laden. Dabei können sie mit einem herkömmlichen relationalen Datenbanksystem (RDBS) verglichen werden in welchem die Tabellen nur zwei Attribute besitzen: eine ID- und eine Value-Spalte [Sad13]. Im Unterschied zu dieser relationalen Datenbank erwarten Key-Value Stores keinen Datentyp für den Value. Hier können beliebige Werte als Blob gespeichert werden. Auch können Key-Value Datenbanken weder Relationen oder Beziehungen zwischen den Einträgen herstellen. Für die Semantik, Korrektheit der Werte, Beziehungen o.ä., ihre Verwendung, welcher Datentyp vorliegt und etwaiges Fehlerhandling ist allein die Anwendung, welche die Datenbank nutzt verantwortlich. Dieses sehr grundlegende Verhalten soll eine möglichst hohe Performance und Skalierbarkeit ermöglichen.

1.1.2 Eigenschaften

Konsistenz

Transaktionen

1.1.3 Datenmodell

1.1.4 Abgrenzung zu Document-Stores

Sehr ähnlich, Unterschied: Das Aggregat muss eine bestimmte Struktur haben, von bestimmtem Typ sein Mit Document sind weiter Abfragen möglich.. Hier nur per Key alle Daten dazu In der Praxis verschwimmt die klare Trennung

1.1.5 Anwendungsfälle

1.1.6 Bewertung

1.2 Document Stores

1.2.1 Einleitung

Document-Stores speichern strukturierte Daten in Datensätzen, die Dokumente genannt werden. Grundsätzlich vereinigen sie die Schemafreiheit von Key-Value-Stores mit der Möglichkeit zur Strukturierung der gespeicherten Daten. Somit lassen sich Document-Stores als Erweiterung von Key-Value Stores betrachten. Vorteil hierbei ist, dass es bei Document-Stores im Gegensatz zu Key-Value Stores möglich ist komplexere auf den Inhalt (die Attribute) der Dokumente gerichtete Abfragen zu stellen. Dies ist bei Key-Value-Stores nicht möglich.

1.2.2 Eigenschaften

Bei der Betrachtung von Document-Stores bezieht sich das Wort "document" auf die hierarchische Struktur der abgespeicherten Daten. Die hier betrachteten Dokumente bestehen jedoch lediglich aus binären Daten oder einfachem Text. Ferner kann es sich um Semi-strukturierte Daten wie JSON oder XML handeln. Während ältere Document-Stores auf XML setzten, ist bei neueren Document-Stores das JSON ähnliche BSON üblich.

In einer Document-Store basierten NoSQL-Datenbank ist es möglich beliebigen Daten zu speichern, ohne dass die Datenbank im Vorfeld Wissen über die Struktur der Daten hat oder weiß was die Daten bedeuten. Gegenüber relationalen Datenbanken bietet dies den Vorteil, dass das Datenbankschema bei kleinen Änderungen nicht großartig verändert werden muss.

In einer Document-Store Datenbank ist folgende Hierarchie typischerweise anzutreffen:

- Ein Dokument ist eine Grundeinheit für das speichern von Daten. Sie ist zu vergleichen mit einer Zeile einer Relationalen Datenbank.
- Ein Dokument besteht aus einem oder mehreren Key-Value Paaren, und kann weitere Dokumente und Arrays enthalten. die Arrays können wiederum ebenfalls Dokumente enthalten, was eine komplexere Strukturierung ermöglicht [?].

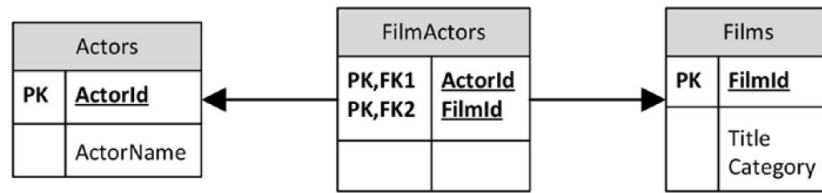


Abbildung 1.1: Beispiel eines JSON-Dokuments

- eine Sammlung oder ein Data Bucket ist ein Set von Dokumenten, die einen Zweck teilen. Dies ist äquivalent zu einer relationalen Tabelle. Hier ist jedoch relevant zu erwähnen, dass die Dokumente hier nicht vom gleichen Typ sein müssen, auch wenn es üblich ist, dass solche Dokumente die gleiche Art von Information darstellen.

1.2.3 Datenmodell

Im Zentrum des Datenmodells der Document-Stores stehen die Documents, die eine hierarchische Struktur aufweisen. So kann ein Dokument auf viele weitere Dokumente verweisen. Dies ist möglich, da ein Dokument Listen enthalten kann.

Wie bereits erwähnt ist es möglich mithilfe von Document-Stores ein relationales Schema nachzubilden. Dies ist jedoch häufig nicht sinnvoll, da es größere Vorteile bringt die Dokumente in kleineren Sammlungen zusammenzufassen und in eingebetteten Dokumente die größte Detailstufe zu bieten.

Sei 1.1 die Darstellung einer relationalen Datenbank mit Filmen und Schauspielern. In einer relationalen Datenbank würde es eine Tabelle mit Filmen und eine Tabelle mit Schauspielern geben, die zu einer gemeinsamen Tabelle gejoint werden würden um anzudeuten welcher Schauspieler bei welchem Film mitgewirkt hat. Mithilfe von Document-Stores wäre dieses Schema so auch möglich, jedoch gibt es hierfür einen besseren Ansatz. Da Document-Stores grundsätzlich keine Join Operationen ermöglichen und Entwickler es bevorzugen wenn die JSON Strukturen sich nah an dem objektorientierten Programmmentwurf orientieren, bietet sich ein Entwurf nach 1.2 eher an. Die Dokumente liegen üblicherweise in JSON bzw. BSON vor.

JSON

Die Java-Script-Object-Notation ist ein vom Menschen und Maschinen lesbarer Standard zur Beschreibung von Daten. JSON ist wie XML ein Standard zum Datenaustausch im Web. In 1.2 ist ein Beispiel für ein JSON-Dokument zu sehen. Vorteil

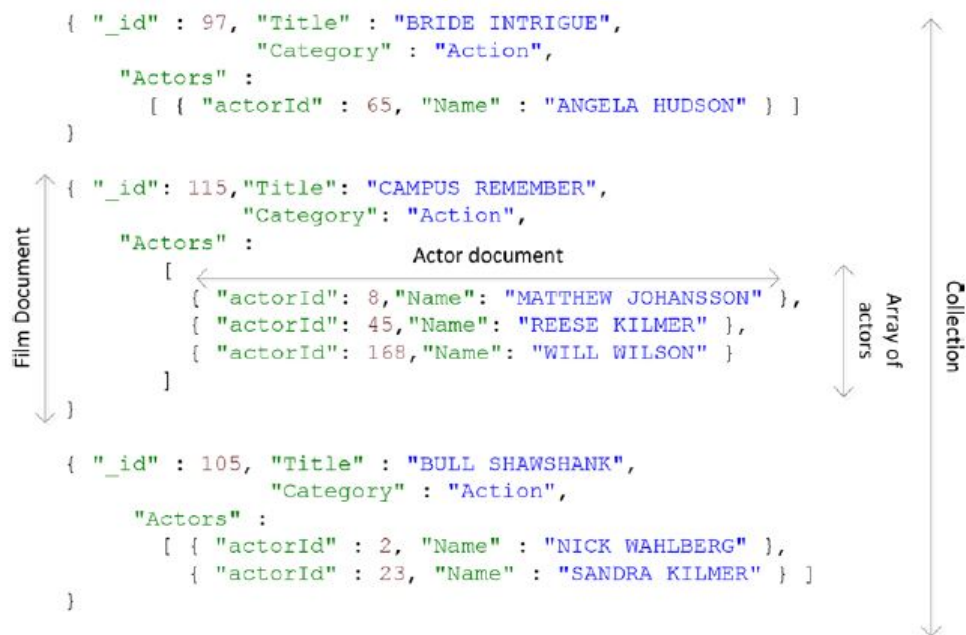


Abbildung 1.2: Beispiel eines JSON-Dokuments

von JSON ist, dass JSON-Dokumente einfach zu zerlegen und umzuwandeln sind. Dies verringert den Aufwand, der in der Anwendungsschicht betrieben werden muss.

BSON

Unter BSON sind binär kodierte JSON-Dokumente zu verstehen. BSON erweitert das Modell von JSON um weitere Datentypen und ermöglicht effizientes kodieren und dekodieren innerhalb verschiedener Sprachen.

1.2.4 Anwendungsfälle

JSON basierte Document-Stores haben vor allem in Web-basierten Anwendungen ihre Vorzüge. Hier wird häufig JSON als Datenschicht verwendet, weswegen Document-Stores hier bevorzugt werden.

XML-basierte Document-Stores finden vor allem in Content-Management-Systemen Anwendung, da sie hier ein Management Repository für XML-basierte Textdateien zur Verfügung stellen.

1.2.5 Bewertung

2 Persistenzsysteme und Big Data Frameworks

2.1 MongoDB

2.1.1 Einleitung

Schemalose NoSQL Datenbanksysteme werden in der heutigen Zeit immer öfters verwendet, dank ihrer Flexibilität, Geschwindigkeit und Verlässlichkeit. MongoDB ist ein schemaloses Datenbanksystem, die auf Dokumentenspeicherung basiert. Diese Dokumente von MongoDB speichern die Daten in einem JSON ähnlichen, nämlich BSON (Binary JSON), was speziell für MongoDB erstellt wurde, Format. Hierbei spielt es keine Rolle, wie die Daten aufgebaut sind, die in das Dokument eingetragen werden. Dokumente in MongoDB werden in Collections gespeichert. Jede Collection fügt einem Dokument eine eindeutige ID zu. Die Dateigröße für Dokumente beträgt maximal 16MB. Eine Ansammlung von Collections stellt die Datenbank dar. Der Aufbau von einer Datenbank in MongoDB sieht wie folgt aus:

Bild: Datenbankstruktur

Die Collection kann man wie eine Tabelle in einer relationalen Datenbank ansehen und die Dokumente sind die jeweiligen Reihen, also die konkrete Dateneintragen in die Tabelle.

2.1.2 Geschichte

MongoDB wurde von Dwight Merriman, Eliot Horowitz und seinem Team erstellt. Doch dies nicht auf direktem Wege. 2007 plante das MongoDB Team ein Online-service für Web-Anwendungen. Dieser Service sollte die Möglichkeit bieten, eine Web-Anwendung zu entwickeln, zu hosten und zu skalieren. Aufgrund von einem nicht geeigneten Datenbanksystem entschloss sich das Team ein eigenes Datenbanksystem zu erstellen und zu nutzen. Diese hatte noch keinen Namen, da dieses System speziell für den Service des Teams erstellt wurde. Im Jahre 2008 wurde dann das Datenbanksystem fertiggestellt. 2009 entschloss sich das Team, dieses Datenbanksystem, was sie erstellt haben, als Open Source Produkt freizugeben. Dieses System wurde als MongoDB veröffentlicht. März 2010 kam die erste Version von MongoDB heraus, die man in einem größeren Umfang verwenden kann. Über die Jahre wur-

de MongoDB von dem Team weiterentwickelt und ist zurzeit mit der Version 3.2 veröffentlicht.

Philosophie

Die Philosophie von MongoDB ist im Vergleich zu den meisten anderen etwas unterschiedlich. Denn das Team von MongoDB hat sich dazu entschieden, dass MongoDB nicht die Lösung für jedes Problem sein soll. MongoDB soll eine Lösung für Analytische Probleme und komplexe Datenstrukturen. Zudem liegt der Fokus auf die Nutzung von Dokumenten, nicht auf Zeilen. Das MongoDB Team wollte ein Datenbanksystem was sehr schnell, gut skalierbar und einfach anzuwenden ist.

2.1.3 Anwendungsfälle

MongoDB wird von verschiedenen Unternehmen zu verschiedensten Anwendungen und Aufgaben verwendet. MTV verwendet MongoDB als Haupt-repository für das MTV-Network. SourceForge verwendet MongoDB als Back-End Speicher. Bit.ly verwendet MongoDB, um den Verlauf von Nutzern zu speichern. New York Times verwendet MongoDB für eine Foto-Abgabe.

2.1.4 Einrichtung

MongoDB wird für alle Betriebssysteme angeboten und ist derzeit in der Version 3.2 erhältlich. Dadurch lässt sich MongoDB auf Ubuntu 12.04+, Windows Server 2008+ und Mac OS X 10.7+ installieren und verwenden. Um MongoDB auf einem Linux Ubuntu Server zu installieren, muss man folgende Schritte befolgen: 1. Importierung von dem public key durch das package management System. 2. Ein Listen-Datei für MongoDB erstellen. 3. Lokale Dateien aktualisieren. 4. Installation von MongoDB Paket. Um MongoDB auf einem Windows Server zu installieren, muss man wie folgt vorgehen: 1. Bestimmen, welchen Build von MongoDB benötigt wird. 2. MongoDB für Windows runterladen. 3. Installation von MongoDB via Asministrator Kommandozeile. Um MongoDB auf einem Mac OS System zu installieren, sind folgende Schritte notwendig: 1. Binäre Dateien von der benötigten MongoDB Version runterladen. 2. Heruntergeladene Dateien extrahieren. 3. Extrahierte Dateien in das Zielverzeichnis kopieren. 4. Pfad von MongoDB sicherstellen. Alle Vorgehensweisen für die Installation von MongoDB verwenden eine Kommandozeile.

Referenzierung

Die Referenzierung von MongoDB unterscheidet sich von der Referenzierung von den relationalen Datenbanken. Eine Möglichkeit der Referenzierung in MongoDB wäre

die Verschachtelung, also ein Dokument in einem Dokumenten schreiben. Damit kann man die Referenz auf ein anderes Objekt direkt erkennen, was auch für das System einfacher ist, an diese Daten zu gelangen. Das Problem hierbei kann aber eine zu starke Verschachtelungsstruktur sein, was auf Kosten der Übersichtlichkeit geht, zusätzlich wird für ein Dokument mehr Platz gebraucht, da in einem Dokument ein weiteres existiert. Die zweite Möglichkeit für eine Referenzierung, wäre der Verweis auf das jeweilige Dokument. Das verbessert die Übersicht und man kann erkennen, voraus die Daten herkommen. Ein Problem was MongoDB mit sich bringt, ist, dass es keine Direktreferenzierung gibt. In der Relationalen Datenbank kann man mit dem JOIN Befehl die Referenz zu einem anderen Datensatz in Bezug auf einen anderen Datensatz direkt ansprechen und anzeigen lassen. Dies kann MongoDB nicht. Um dies zu ermöglichen, muss man das in einer eigenen Applikation programmieren.

Replikation

Eine Replikation bei MongoDB ist die Kopie von Daten auf weiteren Servern. Dies nennt man replica set. Ein replica set umfasst Server mit derselben Funktionalität und dient zur Stabilisierung der Aufgabe des jeweiligen replica sets. Die MongoDB Website empfiehlt das Verwenden von drei Servern pro replica set, um somit die Ausfallrate gering zu halten. Die Daten in einem replica set sind auf allen Servern identisch. MongoDB verwendet das Master-Slave System für die Replikation von Daten innerhalb des replica sets. In dem replica set gibt es genau einen Primary und beliebig viele Secondary Server und ggf. einen Arbiter. Datenänderungen wie z.B. Schreiboperationen werden auf dem Primary Server in dem replica set durchgeführt. Nachdem die Operation erfolgreich beendet wurde, übernehmen die Secondary Server die Änderung von dem Primary Server. Sollte der Primary Server ausfallen, so wählen die Secondary Server einen neuen Primary Server aus. Man ist in der Lage, in dem replica set einen Arbiter hinzuzufügen. Ein Arbiter besitzt nicht die Daten, die in dem replica set verteilt werden. Der Arbiter dient lediglich dazu, nur bei einer Abstimmung eine Stimme abzugeben, damit eventuelle Probleme bei einer Abstimmung vermieden werden können. Ein Arbiter selber kann nicht zu einem Primary oder Secondary Server ernannt werden.

Bild: Replikationsaufbau

Cluster Sharding

Ein sharded Cluster bei MongoDB ist eine Methode, um Daten auf unterschiedlichen Servern zu speichern und dann diese schnell aufzurufen. Ein sharded cluster besteht aus drei Komponenten: Den Config Server, den Query Server und Shards. Um ein sharded cluster einzurichten, braucht man mindestens einen Config Server, einen Query Server und 2 Shards. Dies sieht dann wie folgt aus:

Bild: Aufbau vom Cluster Sharding in MongoDB
Config Server: Der Config Server in einem sharded Cluster beinhaltet die Metadaten für dieses. Diese Metadaten

beinhalten die Informationen zu der Organisation und die Zustände der jeweiligen Komponenten in dem sharded Cluster. Ein Config Server ist in einem replica set. Ein replica set verbessert die Konsistenz über die verschiedenen Config Server. Zusätzlich kann man dank dem replica set mehr als 3 Config Server nutzen. Query Server (Router): Mit dem Query Server ist man in der Lage, Daten auf den verschiedenen Shards zu schreiben und zu Lesen. Dies vollbringt er mit den Metadaten, die der Query Server von dem Config Server bekommt. Der Query Server ist die einzige Schnittstelle in dem sharded Cluster, womit man auf die Daten durch Anwendungen zugreifen kann.

Bild: Datenverteilung von dem Query Server

Shard: Bei einer Shard handelt es sich um einen Teil der Gesamtdatenbank. Die Daten, die in die Datenbank eingetragen werden, werden nach den Metadaten des Configservers analysiert und dann durch den Query Server auf das jeweilige Shard gespeichert. Zum Beispiel: Eine 3 Shard Datenbank beinhaltet Daten von Personen in einem Unternehmen. Auf Shard 1 werden alle Mitarbeiter gespeichert, die im Nachnamen mit dem Buchstaben A bis H anfangen. Shard 2 hat den Nachnamen von I bis Q und Shard 3 R bis Z. Dadurch, dass die Daten aufgeteilt und auf unterschiedlichen Shards gespeichert werden, ist man in der Lage schneller an die gewünschten Informationen zu kommen. Shards können sich in einem replica set befinden. Dieses dient für Redundanz und hohe Verfügbarkeit der Daten.

2.1.5 Verwendung

Um mit MongoDB Daten einzufügen oder auszulesen, sind verschiedene CRUD-Operationen notwendig, wie bei einer relationellen Datenbank. Diese Operationen ähneln wie JavaScript Anweisungen und sind daher leicht anzuwenden. Dies sind die wichtigsten Befehle zur Verwendung von MongoDB:

Tabelle: Befehlsliste für MongoDB

Beim Arbeiten von MongoDB sollte stets geachtet werden, dass man auf der richtigen Datenbank arbeitet und dass die Dokumente in die richtige Collections eingetragen werden.

2.1.6 Grenzen

2.1.7 Zukunft

3 Ausgewählte Anwendungsszenarien

3.1 Web 2.0 - Abschlussarbeiten-DB

3.2 MongoDB

Literaturverzeichnis

- [Abr13] ABRAMOVA, Veronika: *NoSQL Databases: MongoDB vs Cassandra*. ACL Digital Library, 2013
- [And16a] ANDREAS, Meier: *Datenmanagement*. Springer, 2016
- [And16b] ANDREAS, Meier: *Zur Nutzung von SQL- und NoSQL-Technologien*. Springer, 2016
- [Boi12] BOICEA, Alexandru: *MongoDB vs Oracle – Database Comparison*. Researchgate, 2012
- [Dan16a] DANIEL, Fasel: *Übersicht über NoSQL-Technologien und -Datenbanken*. Springer, 2016
- [Dan16b] DANIEL, Fasel: *Was versteht man unter Big Data und NoSQL*. Springer, 2016
- [Edl11] EDLICH, Stefansub: *NoSQL: Einstieg in die Welt nichtrelationaler Web 2.0 Datenbanken*. 2. München Hanser Verlag, 2011
- [Far14] FARAJ, Azhi: *COMPARATIVE STUDY OF RELATIONAL AND NON-RELATIONS DATABASE PERFORMANCES USING ORACLE AND MONGODB SYSTEMS*. IJCET, 2014
- [Han11] HAN, Jing: *Survey on NoSQL database*. IEEE, 2011
- [How14] HOWS, David: *MongoDB Basics a quick introduction to MongoDB*. APRESS, 2014
- [Par13] PARKER, Zachary: *Comparing NoSQL MongoDB to an SQL DB*. ACL Digital Library, 2013
- [Red12] REDMOND, Eric: *Sieben Wochen, sieben Datenbanken*. O'Reilly Verlag, 2012
- [Sad13] SADALAGE, Pramod J.: *NoSQL: A Brief Guide to the Emerging World of Polyglot Persistence*. Addison-Wesley, 2013
- [Sha15] SHAKUNTALA, Gupta E.: *Practical MongoDB Architecting, Developing, and Administering MongoDB*. APRESS, 2015
- [Tud11] TUDORICA, Bogdan G.: *A comparison between several NoSQL databases with comments and notes*. IEEE, 2011