Masterarbeit

Speicherung von amtlichen Geodaten in der verteilten NoSQL-Datenbank MongoDB

Hochschule Anhalt

Anhalt University of Applied Sciences

Studiengang: Vermessung und Geoinformatik

vorgelegt von

Guohua Qi

Matr.-Nr.: 406544

Erstprüfer: Prof. Dr.-Ing. Holger Baumann

Zweitprüfer: Dr. Marcel Ziems

Dessau, den 18.06.2019

Ehrenwörtliche Erklärung

„Hiermit versichere ich, Guohua Qi, ehrenwörtlich, dass ich die vorliegende Masterarbeit mit dem Titel: „Speicherung von amtlichen Geodaten in der verteilten NoSQL-Datenbank MongoDB“ selbstständig und ohne fremde Hilfe verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die dem Wortlaut oder dem Sinn nach anderen Werken entnommen wurden, sind in jedem Fall unter Angabe der Quelle kenntlich gemacht. Die Arbeit ist noch nicht veröffentlicht oder in anderer Form als Prüfungsleistung vorgelegt worden.

Dessau, den 18.06.2019

.....

Guohua Qi

Abstrakt

Bis heute bilden relationale Datenbanken die wichtigste Grundlage für die Speicherung der Geodaten. Die fortschreitende Verbreitung von Internetanwendungen stellt immer neue Anforderung für die Speicherung der Daten. So fordern neue Technologien wie IoT, soziale Netzwerke und Cloud Computing von Datenspeichern wachsende Performance, Informationssicherheit, Durchsuchbarkeit sowie konsistentes Antwortverhalten. Hinzu kommen exponentiell wachsende Datenmengen von immer mehr verfügbaren Sensoren wie von Satelliten oder Smartphones. Deshalb entsteht ein Bedarf und eine Herausforderung zum Speichern des Big Geo Data. Dann wurden sogenannte NoSQL-Datenbanken, für ähnliche Problemstellungen entwickelt. Diese eigneten sich insbesondere für unstrukturierte Daten und hohe Schreibfrequenzen. Da NoSQL den Aspekt räumlicher Informationen, also den der Geodaten, nicht explizit behandelt, soll im Rahmen der vorliegenden Arbeit untersucht werden, inwieweit NoSQL-Datenbanken im Bereich Geodaten eingesetzt werden können. Im Besonderen wird hierbei auf MongoDB im Zusammenspiel mit amtlichen Geodaten eingegangen.

Zuerst wird ein hochverfügbares verteiltes Cluster-System entworfen und mit implementiert. Die aus PostGIS exportierten Daten werden dann verarbeitet und anschließend in MongoDB eingespeichert. Als Basis hierfür wurde ein „Virtual Machine Cluster“ implementiert. Im Anschluss an den ersten Teil werden vier Benchmark-tests durchgeführt, und sounterschiedliche Lösungen basierend auf MongoDB und PostGIS miteinander vergleichen. Die verwendeten Daten umfassen Vektordaten und Rasterdaten. Die verschiedenen Benchmark-tests werden unter denselben Hardware- und Software-Bedingungen durchgeführt, um möglichst objektive Ergebnisse zu erzielen.

Durch Experimente wird ein MongoDB-Cluster mit hoher Verfügbarkeit und leichter horizontaler Skalierung realisiert. Mithilfe der grafischen Aufbereitung der Testergebnisse lässt sich nicht nur nachweisen, dass die Reaktionszeit, sondern auch Speicherplatzverbrauch Durch den Einsatz von MongoDB gegenüber PostGIS verbessern werden kann. Außerdem kann gezeigt werden, dass sich mittels MongoDB Funktionen wie die Abfrage des Nächsten Nachbarn erheblich schneller realisieren lassen. In Bezug auf die „Bounding Box Query“ erweist sich PostGIS wegen des R-Tree als überlegen. Es bleibt anzumerken, dass die Analyseergebnisse noch keine allgemeinen Aussagen ermöglichen. Hierfür sind weitere gezielte Experimente für unterschiedliche Szenarien mit mehr Daten und mehr Rechenleistung notwendig. In praktischen Anwendungen müssen auch die Produktionsumgebung und -bedürfnisse berücksichtigt werden, z.B. unterstützt PostGIS mehr räumliche Abfragefunktionen und Indexe.

Schlussendlich enthalten diese Arbeitsanweisungen zur Erstellung eines MongoDB-Clusters. Zudem wird eine Referenz zum Auswählen einer effizienteren Datenbank bereitgestellt.

Schlagwörter: NoSQL, MongoDB, Geodaten, MongoDB-Cluster, Docker, PostGIS, hohe Verfügbarkeit, horizontale Skalierung, Benchmark-tests

Abstract

To date, relational databases are the most important foundation for geospatial storage. The prevalence of Internet applications has submitted new requirements to data storage. For example, new technologies such as IoT, social networking, and cloud computing of data storage require growing performance, information security, searchability, and consistent responsiveness. In addition, there are exponentially growing amounts of data from more and more available sensors such as satellites or smartphones. Therefore, there is a demand and a challenge to save the Big Geo Data. Then, so-called NoSQL databases were developed for similar problems. These were particularly suitable for unstructured data and high writing frequencies. Since NoSQL does not explicitly deal with the aspect of spatial information, the aim of this work is to investigate to what extent NoSQL databases can be used in the field of geodata. In particular, MongoDB will be discussed in conjunction with official geodata.

First, a high-availability distributed cluster system was designed and implemented. The data exported from PostGIS is then processed and then stored in MongoDB. The foundations for this was a virtual machine cluster implemented. Following the first part, four benchmark tests were performed, and different solutions were compared based on MongoDB and PostGIS. The data used includes vector data and raster data. The various benchmark tests are performed under the same hardware and software conditions to achieve the most objective results possible.

Experiments will realize a MongoDB cluster with high availability and easy horizontal scaling. Through the graphical results, it can be proved not only response time can be improved, but also storage space consumption by using MongoDB over PostGIS. Further, it can be shown that functions such as nearest neighbor queries can be realized much faster with using MongoDB. With regard to the "Bounding Box Query", PostGIS proves to be superior due to the R-Tree. It should be noted that the analysis results are not universal. This requires further targeted experiments for different scenarios with more data and more computing power. In practical applications, the production environment and demands must also be taken into account, e.g. PostGIS supports more spatial query functions and indexes.

In summary, this thesis contains instructions for creating a MongoDB cluster. It also provides a reference for choosing a more efficient database.

Keywords: NoSQL, MongoDB, Geodata, MongoDB-Cluster, Docker, PostGIS, high availability, horizontal scaling, Benchmark-tests

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

1NF	1. Normalform
2NF	2. Normalform
3NF	3. Normalform
4NF	4. Normalform
5NF	5. Normalform
ACID	Atomicity, Consistency, Isolation, Durability
API	Application Programming Interface
AWS	Amazon Web Services
BaaS	Backend as a Service
BASE	Basically Available, Soft State und Eventically Consistent
BCNF	Boyce-Codd-Normalform
BI	Business Intelligence
BSON	Binary JSON
BRW	Bodenrichtwert
CAP	Consistency Availability Partition Tolerance
CPU	Central Processing Unit
CRUD	Create-, Read/Retrieve-, Update-, Delete/Destroy-Operationen
CSV	Comma-separated values
DBMS	Database Management System
GB	Gigabyte
GUI	Graphical User Interface, Grafische Benutzeroberfläche
HDFS	Hadoop Distributed File System
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IFUR	INSERT FIND UPDATE REMOVE
I/O	Input/Output (Eingabe/Ausgabe)
IOPS	Input/Output operations Per Second
IoT	Internet of Things
IP	Internetprotokoll

JSON	JavaScript Object Notation
LGLN	Landesamt für Geoinformation und Landesvermessung Niedersachsen
MB	Megabyte
NoSQL	Not only SQL
OLAP	Online Analytical Processing
OLTP	Online Transaction Processing
Oplog	Operations-Log
OSM	OpenStreetMap
PODC	Principles of Distributed Computing
QPS	Queries per second
RDF	Resource Description Framework
SQL	Structured Query Language
SSH	Secure Shell
SSL	Secure Sockets Layer
REST	Representational State Transfer
WGS84	World Geodetic System 1984
SSD	solid-state drive
SUT	System Under Test
TB	Terabyte
TSV	Tab Separated Values
XML	Extensible Markup Language
YAML	Yet Another Markup Language

Vorwort

Die Zeit verging im Fluge, ich will bald Masterstudium absolvieren. Beim Masterstudiengang habe ich viel praktisches Wissen gelernt und viele fortgeschrittene Dinge und Ideen berührt. Ich bedanke sich sehr bei Herrn Prof. Dr. Holger Baumann für die Betreuung und Hilfe, sowie anderen Hochschullehrern und Kommilitonen, die mir geholfen haben.

Die vorliegende Masterarbeit mit dem Titel " Speicherung von amtlichen Geodaten in der verteilten NoSQL-Datenbank MongoDB " habe ich Während des Praktikums bei LGLN (Landesamt für Geoinformation und Landesvermessung Niedersachsen) geschafft. Das Thema wurde gemeinsam mit Herrn Dr. Marcel Ziems definiert und die dazugehörige Aufgabenstellung formuliert. Nach einer langen Einarbeitungszeit habe ich die Experimentierphase begonnen. Im Lauf dessen war ich auf viele Probleme gestoßen. Ich möchte recht herzlich Herr Dr. Ziems und Herr Bischof für die guten Vorschläge und Hilfe danken. Gleichzeitig danke ich auch den anderen LGLN-Kollegen, die mir geholfen haben.

Darüber hinaus geht ein Dank an meine Eltern und Ehefrau. Ihre Unterstützung und Verständnis haben mein Studium ermöglicht. Abschließend danke ich meine Freunde für ihre Unterstützung und Begleitung.

1 Einleitung

1.1 Hintergrund und Motivation

Mit dem weit verbreiteten Einsatz neuer Technologien wie IoT (Internet of Things), Smart City, Web 2.0 und sozialen Netzwerken wächst die zu bewältigende Datenmenge nicht zuletzt aufgrund neuer Sensoren wie Satelliten und Smartphones rasant. So können die daraus resultierenden Geodaten als Big Data bezeichnet werden. Bei Big Data bezieht sich das „Big“ auf die vier Dimensionen (Lesniak, 2017):

- Volume (Umfang und Datenmenge)
- Variety (Verschiedene Arten von Daten und Datenquellen.)
- Velocity (Geschwindigkeit, mit der die Datenmengen generiert und verarbeitet werden)
- Veracity (Datenqualität und Datenwert)

Big Data besteht zu 10% aus strukturierten Daten und zu 90% aus unstrukturierten Daten mit geringer Wertdichte, z.B.: Überwachungsvideos werden nur bei einem Unfall wertvoll. 80% davon sind zugleich bereits georeferenziert oder können werden, und werden auch als Big Geo Data bezeichnet (Baumann, 2014). Angesichts von Big Data wird der Mangel relationaler Datenbanken in Bezug auf horizontale Skalierbarkeit und hohe Verfügbarkeit, effiziente Abfragen und -Zugriff von Big Data sowie die Unterstützung für unstrukturierte Daten immer deutlicher. Eine neue Art von Datenbank, sogenannte NoSQL(Not only SQL) bieten hierbei interessante Lösungsmöglichkeiten. Es existieren keine ACID-Garantien in NoSQL Datenbanken, sondern folgt CAP Theorie und BASE Prinzip. BASE steht für Basically Available, Soft State und Eventically Consistent. Nach Datenmodell werden NoSQL Datenbanken in Key-Value Datenbanken, Spaltenorientierte Datenbanken, Dokumentenorientierte Datenbanken und Graphdatenbanken unterschieden, NoSQL-Vertreter wie Redis, Cassandra, HBase, CouchDB, MongoDB, Neo4j und so weiter.

Im Kontext von Big Geo Data werden Geodaten des öffentlichen und privaten Lenbens immer enger verknüpft. Zum Beispiel Plattformen wie GoogleMaps oder OpenstreetMap haben entsprechende Anwendungen für Jedermann verfügbar und damit eindrucksvoll demonstriert wie leistungsfähig entsprechende Internetanwendungen sein können. So ist es kein Wunder, dass Menschen derartige Ansprüche auch an amtliche Geodaten des öffentlichen Dienstes stellen. Um die Geodaten öffentlich im großen Umfang zu nutzen, müssen sie folgenden Bedingung erfüllen: gute Durchsuchbarkeit, hohe Übertragungsraten, die Anwendung von im Web gebräuchlichen Standards, Zweiwegekommunikation,

Integrität sowie Zugänglichkeit über mobile Endgeräte beinhalten. Die bisher übliche Datenhaltung mittels objektrelationaler geographischer Datenbanken, z.B. mittels PostgreSQL mit PostGIS Erweiterung erfüllt die Anforderungen nur zum Teil. Private Plattformen haben bereits gezeigt, dass NoSQL-Datenbanken wie MongoDB in Teilaspekte überlegen sind. Daher besteht hohes Interesse bei den öffentlichen Landesvermessungen der Bundesländer die heute verwendeten amtlichen Datenhaltungskomponenten entsprechend zu erweitern.

1.2 Struktur der Arbeit

Laut dem Datenbank-Ranking von DB-Engines[1] ist MongoDB derzeit eine der beliebtesten NoSQL-Datenbanken der Welt. Sie besitzt relativ umfassende Funktionen und hohe Leistung. Ein weiterer wichtiger Grund ist natürlich, dass sie Abfrage und Verwaltung der geografischen Daten unterstützt. Deshalb wählt diese Arbeit MongoDB als Forschungsobjekt und konzentriert sich auf die Durchführbarkeit und die Leistung von MongoDB im Bereich Geodaten.

Kapitel 1 erläutert den Hintergrund dieser Arbeit, die aufgetreten Probleme und die Motivation.

Im zweiten Kapitel werden die Begriffe der relationalen Datenbanken und NoSQL-Datenbanken und deren grundsätzlichen Prinzipien beleuchtet, z.B. Klassifizierung der NoSQL-Datenbanken, CAP Theorem und BASE Prinzip.

MongoDB wird hauptsächlich ausführlich im dritten Kapitel dargelegt. Zuerst werden die am häufigsten verwendeten Komponenten und Grundoperationen vollgestellt. Nachfolgend werden zwei wichtige Merkmale eingeführt, die die hohe Verfügbarkeit und horizontale Skalierung unterstützen. Zudem werden wichtige Features beschrieben.

Im vierten Kapitel wird eine Lösung für amtlichen Geodaten auf Basis eines hochverfügbaren leicht skalierbaren verteilten Cluster-Systems vorgeschlagen. Hierfür wird Cluster auf Basis zweier Methoden implementiert. Nach der ersten wird mittels VirtualBox Schritt für Schritt manuell vorgegangen. Die zweite Methode basiert auf Docker und ‚stack file'. Die folgende Abbildung 1 zeigt den Ablauf:

[1] https://db-engines.com/de/ranking

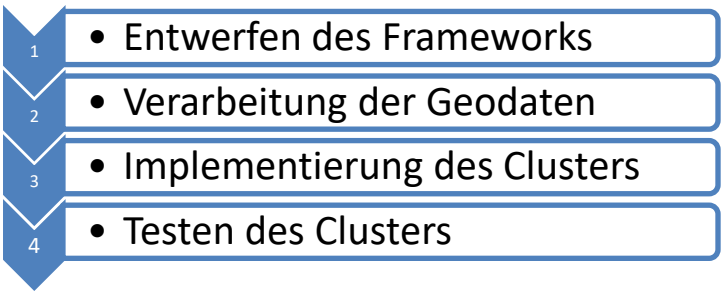


Abbildung 1 : Ablauf des Experiments

Kapitel 5 beschreibt einen Vergleichstest zwischen MongoDB und PostgreSQL aus. Um möglichst gleichartige Bedingungen zu erreichen, wird ein identischer Computer beim Test verwendet. Die Testdaten stammen von OSM (OpenStreetMap) und schließen Vektordaten und Rasterdaten ein. Nach dem Testen werden die Ergebnisse in Form von Tabelle und Diagrammen aufgearbeitet.

Anschließend werden im sechsten Kapitel einige Vorschläge unterbreitet, um die Performance zu steigern. Diese beziehen sich auf die Aspekte von Soft- und Hardware.

Im siebten Kapitel werden eine Zusammenfassung und einen Ausblick beschrieben.

2 Datenbanken

Eine Datenbank ist eine organisierte Sammlung von Informationen, die im Allgemeinen elektronisch von einem Computersystem ausgespeichert und abgerufen werden. Nach der Organisierung können die Informationen leicht zugegriffen, verwaltet und aktualisiert werden. Die Komplexität der Datenbank besteht darin, dass häufig formales Design zur Entwicklung verwendet wird. Das Database Management System (DBMS) ist das Softwaresystem, das mit Endbenutzer, Anwendungen und der Datenbank interagiert, um Daten zuzugreifen oder zu analysieren. Die DBMS-Software umfasst zusätzlich die Kernfunktionen zur Verwaltung der Datenbank. Eine Sammlung der Datenbank, des DBMS und der zugehörigen Anwendungen kann als „Datenbanksystem“ bezeichnet werden. (Mata-Toledo & Cushman, 2003)Oft bezieht sich der Begriff „Datenbank“ auch auf ein DBMS, ein Datenbanksystem oder eine Anwendung, die einer Datenbank zugeordnet ist.

Seit ihrer Gründung in den 1960er Jahren haben sich Datenbanken entwickelt, beginnend mit hierarchischen und Netzwerkdatenbanken. Diese zeigten jedoch Mängel bezüglich der Abbildung unserer komplexen realen Welt. Dementsprechend wurde das relationale Modell erstmals 1970 von Edgar F. Codd vorgeschlagen. Bis Mitte der 1980er Jahre entwickelten sich die relationalen Datenbanken zum bevorzugten Verfahren für die Datenspeicherung. In den führen 1990er Jahren dominierten RDBMS jedoch alle großen Datenverarbeitungsanwendungen, und sind bis heute (2019) dominant. Oracle, MySQL, SQL Server und PostgreSQL sind die häufigsten verwendeten DBMS. (GitHub, 2019) Die vorherrschende Datenbanksprache, standardisiertes SQL (Structured Query Language) für das relationale Modell, hat daneben die Datenbanksprachen für andere Datenmodelle beeinflusst. In den 1980er Jahren wurden Objektdatenbanken entwickelt, um das Problem des Object-relational Impedance Mismatchs zu überwinden. Das führte zur Prägung des Begriffs „postrelational“ und auch zur Entwicklung hybrider objektrelationaler Datenbanken.

Die nächste Generation von postrelationalen Datenbanken ist die NoSQL-Datenbank. Nach der kontinuierlichen Entwicklung wurden Key-Value-Datenbanken, Spaltenorientierte Datenbanken, Dokumentenorientierte Datenbanken und Graph-Datenbanken hauptsächlich auf den Market gebracht. Die NewSQL markiert die „nächste Generation“, an Implementierungen. Diese hält das relationale Modell und SQL bei und sorgt gleichzeitig für Skalierbarkeit und hohe Leistung von NoSQL-Systemen für OLTP-Workload (Online Transaction Processing) bieten.

Basierend auf „The evolving database landscape“ von Matt Aslett (Aslett, 2012) wurde die folgende Abbildung 2 zu Beziehung von NoSQL und NewSQL erstellt:

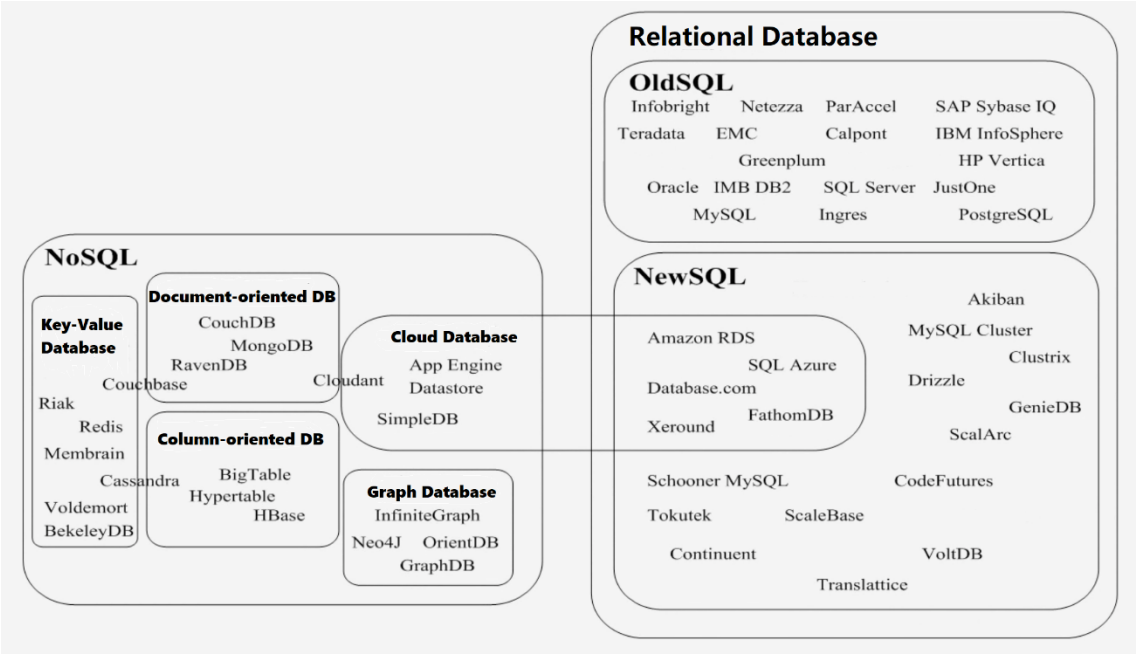


Abbildung 2: Klassifizierung der Datenbanken

Traditionelle Datenbanksysteme basieren auf dem Konzept, dass ein Datenmodell für alle Datentypen gilt. Diese Idee ist jedoch weit davon entfernt, sich an die aktuelle Situation anzupassen. Demzufolge ist Multi-Modell-Datenbanken entstanden, die mehrere Datenmodelle in einem einzigen integrierten Backend unterstützen. Einige unterstützen mehrere verschiedene Datenmodelle, z.B.:

- ArangoDB – Dokument (JSON), Key-Value, Graph
- Datastax – Dokument, Key-Value, Graph
- EnterpriseDB – Dokument (XML und JSON), Key-Value
- Redis – Dokument (JSON), Key-Value, Graph, Streaming, Time-Series

Weiterhin ist erwähnenswert, dass einige sowohl ein relationales Datenmodell als auch ein nicht-relationales Datenmodell unterstützen, z.B.:

- Azure Cosmos DB – Dokument (JSON), Key-Value, SQL
- OrientDB – Dokument (JSON), Key-Value, Graph, SQL
- MarkLogic - Dokument (XML und JSON), Graph, Binary, SQL
- Oracle Database – Dokument (XML und JSON), Key-Value, Graph, Objekte, relational

Heutzutage sind Datenbankprodukte sehr vielfältig. Häufig werden traditionale Datenbanken zur Verarbeitung der Transaktionsdaten verwendet, NoSQL wird hingegen zur Verarbeitung von Internetdaten und NewSQL zur Datenanalyse verwendet. Schlussendlich sind die jeweiligen Anforderungen entscheidend für die Auswahl des jeweils besten Ansatzes.

2.1 Relationale Datenbanken

Als das am weitesten verbreitete Datenbankmodell besitzt relationale Datenbanken eine vollständige theoretische Grundlage der relationalen Algebra. (Elmasri & Navathe, 2005) Eine Relationale Datenbank unterliegt einer Art Datenorganisation, die aus zweidimensionalen Tabellen und den Relationen zwischen Tabellen besteht. Jeder Spalt in einer Tabelle beinhaltet einen bestimmten Datentyp, eine Eigenschaft der Objekte darzustellen. Jede Zeile in einer Tabelle ist ein Datensatz, der die Informationen eines Objekts beschreibt. Weiterhin kann jede Zeile mit einer einzigen Kennung ausgestattet, die als primärer Schlüssel bezeichnet wird. Zeilen in mehreren Tabellen können vermittels der Fremdschlüssel miteinander verknüpft werden. (Heuer, Saake, & Sattler, 2003)Durch Verwendung des primären Schlüssels können unerwünschte Abhängigkeiten bei Datenmanipulationen vermieden werden und Datenredundanzen entfernt werden. Die so genannten fünf Normalformen in relationalen Datenbanken sind in Tabelle 1 (Mata-Toledo & Cushman, 2003, S. 145-187) dargestellt.

Tabelle 1: Normalformen in relationalen Datenbanken

1. Normalform (1NF)	Eine Relation ist in erster Normalform, wenn - sie nur atomare Attribute enthält, -Werte, die in einen Spalt gespeichert sind, derselben Domäne angehören sollen, - alle Spalten in einer Tabelle eindeutige Namen haben, - und die Reihenfolge, in der die Daten gespeichert werden, spielt keine Rolle.
2. Normalform (2NF)	Eine Relation ist in der zweiten Normalform, wenn - sie in der ersten Normalform ist und - jedes Nicht-Schlüssel-Attribut voll funktional abhängig vom ganzen Primärschlüssel ist. (keine partielle Abhängigkeit)
3. Normalform (3NF)	Eine Relation ist in der dritten Normalform, wenn - sie in der zweiten Normalform ist und - keine transitiven Abhängigkeiten besitz, Wenn jede Attributmenge ein Superschlüssel ist, also die Abhängigkeit ist trivial, erfüllt die Relation Boyce-Codd-Normalform. (BCNF)
4. Normalform (4NF)	Eine Relation ist in der vierten Normalform, wenn - sie in der dritten Normalform ist und - besitzt nur triviale mehrwertige Abhängigkeiten.
5. Normalform (5NF)	Eine Relation ist in der fünften Normalform, wenn - sie in der vierten Normalform und - es keine mehrwertigen Abhängigkeiten gibt. (keine 1: n oder m: n Beziehungen)

In relationalen Datenbanken müssen Datenbeziehungen während des Datenbankdesigns vordefiniert werden, und die Datenbank-Normalisierung ist eine unverzichtbare wichtige theoretische Basis für relationales Modell. Im Allgemeinen sind nur die ersten drei Normalformen erforderlich. Relationale Datenbank nutzt verschiedene Verfahren Zur Gewährleistung der Datenintegrität, die generell Vollständigkeit, Genauigkeit und Konsistenz von Daten darstellt. Beispielsweise gibt es primäre Schlüssel, Fremdschlüssel, „Not

NULL“ Einschränkung, „Unique“ Einschränkung, „Default“ Einschränkung, „Check“ Einschränkung und so weiter.

Darüber hinaus enthält relationale Datenbank ein fundamentales Merkmal – „Transaktionen“. Als Transaktion bezeichnet man einen oder mehrere SQL-Befehle, die als eine Folge von Operationen durchgeführt werden. Jede Transaktion repräsentiert eine logische Arbeitseinheit, die innerhalb eines Datenbankmanagementsystems gegen eine Datenbank ausgeführt wird, und auf eine kohärente und zuverlässige Weise unabhängig von anderen Transaktionen behandelt wird. Eine Transaktion steht im Allgemeinen für jede Änderung in einer Datenbank. Transaktionen in einer Datenbank haben zwei Hauptziele:

1. Wenn ein Systemfehlers auftritt, die Ausführung angehalten wird, oder viele Operationen der Datenbank aus unbekannten Gründen unvollendet bleiben, werden immer noch zuverlässige Arbeitseinheiten bereitgestellt, die eine korrekte Wiederherstellung nach Fehlern ermöglichen und eine konsistente Datenbank gewährleisten.
2. Isolation zwischen Programmen, die gleichzeitig auf die Datenbank zugreifen. Wenn diese Isolierung nicht angeboten wird, kann das Ergebnis des Programms falsch sein.

Eine Datenbanktransaktion muss definitionsgemäß atomar (atomic), konsistent (consistent), isoliert (isolated) und dauerhaft (durable) sein. Diese vier Eigenschaften von Datenbanktransaktionen[2] werden in Tabelle 2 dargestellt und als ACID[3] abgekürzt.

Tabelle 2 : ACID-Prinzip

A (Atomarität)	C (Konsistenz)	I (Isolation)	D (Dauerhaftigkeit)
Atomicity garantiert, dass jede Transaktion als eine "Einheit" behandelt wird, die entweder vollständig erfolgreich ist oder vollständig fehlschlägt. (all-or-nothing)	Alle in die Datenbank geschriebenen Daten müssen alle vordefinierten Regeln erfüllen, einschließlich Einschränkungen, Kaskaden, Trigger und Kombinationen davon.	Beim Mehrbenutzerbetrieb kann jede Transaktion unabhängig von den anderen manipuliert oder zurückgesetzt werden.	Durability erfordert, dass erfolgreiche beendete Transaktionen dauerhaft die Daten beeinflussen. Das bedeutet in der Regel, dass ihre Auswirkung auch bei einem Systemausfall nicht vernichtet werden.

Laut ACID können alle Standorte während der Transaktion nicht auf die Tabelle zugreifen, da ACID unter anderen sicherstellt, dass parallele laufende Transaktionen nicht gegenseitig bemerken bzw. anhand der Isolation bis zum Ende der Transaktion nicht zugegriffen werden kann. Das heißt, dass die Relation ist gesperrt (locked). Aufgrund dieser

[2] http://de.wikipedia.org/wiki/Transaktion_(Informatik)

[3] http://de.wikipedia.org/wiki/ACID

hervorragenden Merkmale können relationale Datenbanken mittels von SQL die Daten effizient und komplex abfragen, was bei NoSQL nicht möglich ist, z.B. SQL-Join. Aber auch wegen ACID wird der Overhead (L. Sun,2015, S. 4) sehr hoch, da es beständig geprüft wird, ob die Sperrung inzwischen abgeschafft ist, um weitere Transaktionen oder Operationen zu beginnen. Daher müssen in einigen Fällen Effizienz und Systemkonsistenz abgewogen werden. Es ist extra schwierig in verteilten Szenarien, Atomarität, Isolation und Konsistenz zu erreichen. Dazu kommen Probleme bei relationalen Datenbanken. Beispielsweise ist die Datenmodellierung so streng, dass viele unstrukturierte Daten schwerlich gespeichert und verwaltet werden können. Dabei besitzen relationale Datenbanken schlechte horizontale Skalierbarkeit. (Abadi, 2010)

2.2 NoSQL-Datenbanken

Der vollständige Name von NoSQL ist „Not only SQL“. In der Praxis bedeutet „NoSQL“ „non-relational database“, obwohl einige solcher Datenbanken SQL-Abfragen unterstützen (Tejada, Buck, Wilson, & Wasson, 2018). Bereits 1979 wurde Key/Hash-Datenbank DBM[4] von Ken Thompson als der erste Vertreter dateibasierter Datenbank entwickelt. Der Begriff NoSQL wurde erstmals 1998 von Carlo Strozzi benutzt, um seine Datenbank als Strozzi-NoSQL zu benennen. Strozzi-NoSQL war eine relationale Datenbank, aber unterstützte kein Structured Query Language (SQL). Zu diesem Zeitpunkt bedeutete NoSQL kein SQL. Zu Beginn des 21. Jahrhunderts löste die Nachfrage von Web2.0-Unternehmen eine schnelle Entwicklung der NoSQL-Datenbanken aus. Von 2006 bis 2009 wurden die meisten heute populären NoSQL-Datenbanken entwickelt. Bis zum 12. Mai 2009 trat der heutige Begriff NoSQL (Not only SQL) zum ersten Mal im Weblog von Eric Evans auf. (Evans, 2009) Die seither wachsende Verbreitung von NoSQL-Datenbanken ist besondere der rasanten Entwicklung von Internet of Things, Soziales Netzwerk und Cloud Computing Technologien begründet. Dadurch nehmen solche Datenhaltungskomponenten in beispiellosem Tempo zu, wie Abbildung 3 gezeigt. Die meisten Daten sind sehr aufwendig zu strukturieren, bzw. überhaupt nicht strukturiertbart werden. Man bezeichnet solche großen, komplexen und schnellebigen Datensätze auch als Big Data.

[4] https://de.wikipedia.org/wiki/DBM_(Datenbank)#cite_note-1

Abbildung 3 : What happens in an Internet Minute in 2019. (Desjardins, 2019)

Mit dem Aufkommen von Big Data gingen viele neue Probleme ein. Beispielsweise werden vergleichsweise kleine Stichprobenanalysen Grundgesamtheiten ersetzt. Bei der traditionellen Stichprobenanalyse ist es schwierig, die Genauigkeit und Repräsentativität der Stichprobe sicherzustellen. Für einige Datenanalysen ist es vollständig in der Lage, alle Daten mit den gegenwärtigen technischen Mitteln zu sammeln und zu verarbeiten, dann wird es keine Stichproben benötigt. Weiterhin wird mehr auf die Relevanz der Daten als auf die Kausalität der Daten bei der Big-Data-Analyse geachtet. Zu diesem Zeitpunkt ist die Effizienz beim Speichern und Analysieren von Big Data relativ wichtiger als Genauigkeit. Um die Effizienz zu erhöhen, kann man horizontale Skalierung bzw. vertikale Skalierung verwenden (Rahm, Saake, & Sattler, 2015). Die vertikale Skalierung erfolgt durch Hinzufügen von Ressourcen zu Systemknoten, um die Leistung zu verbessern, z.B.: Erweiterung des Speichers, Steigerung der CPUs oder Installierung einer leistungsstärkeren Grafikkarte. Im Vergleich zur vertikalen Skalierung ist die horizontale Skalierung aus Hardware-Sicht theoretisch unbegrenzt. Horizontale Skalierung bedeutet, die Systemleistung durch Hinzufügen zusätzlicher Computer zu erhöhen. Mehrere Unternehmen bevorzugen die horizontale Skalierung, da deren wirtschaftlicher Nutzen höher ist. Die relationale Datenbank ist jedoch in Bezug auf die horizontale Skalierung sehr schlecht. Gründe hierfür sind

Einerseits, dass wenn ein neuer Knoten hinzugefügt wird, müssen die Datensätze jedes Mal neu partitioniert werden, um den Lastdruck jedes Knotens auszugleichen. Andererseits besitzen relationale Datenbank vordefiniertes festes Schemata, deren die Partitionierung sehr kompliziert ist. Anderseits ist es fragwürdig, die Datenkonsistenz während eines Failovers oder eines gleichzeitigen Zugriffs aufrechtzuerhalten. Insbesondere bei der Globalisierung werden Daten in den vielen Fällen an verschiedenen geografischen Stand-

orten gespeichert. Bei verteilten leichtgewichtigen Transaktionen ist es jedoch wahrscheinlich, dass die Dauer der Sperrung viel länger ist als die von der granularen Transaktion benötigte Zeit, um eine starke Konsistenz von ACID Merkmalen beizubehalten (Rahm, Saake, & Sattler, 2015, S. 227-230). Wenn relationale Algebra[5] mit hoher Nebenläufigkeit ausgeführt werden, sind relationale Datenbanken für diese Aufgabe nicht geeignet.

Um diese neuen Herausforderungen zu lösen, werden die NoSQL-Datenbanken circa 2000 wieder bemerkt (Edlich, Friedland, Hampe, & Brauer, 2010, S. 1). Derzeit gibt es keinen klaren Geltungsbereich und keine klare Definition für NoSQL, da keine einheitliche Organisation oder Gremien steht. Im Allgemeinen weisen sie jedoch einige der folgenden Eigenschaften auf.

1. Ein nicht relationale Datenmodell wird als zugrundeliegendes Datenmodell benutzt.
2. Die Datenbanken sind für die verteilte und horizontale Skalierbarkeit ausgelegt.
3. Die Datenbank ist Open Source und kann große Datenmengen effizient verarbeiten.
4. Datenbanken sind schemafrei oder haben lediglich schwache Beschränkung des Schemas.
5. Einfache Datenreplikation wird im verteilten System unterstützt-
6. Man kann über einfache API auf die Datenbank zugreifen.
7. Die Datenbanken basieren normalerweise auf BASE und *Eventually Consistent* Prinzip

Die von Sourav Mazumder vorgeschlagene NoSQL-Systemarchitektur wird in Abbildung 4 dargestellt (Mazumder, 2010).

[5] https://en.wikipedia.org/wiki/Relational_algebra

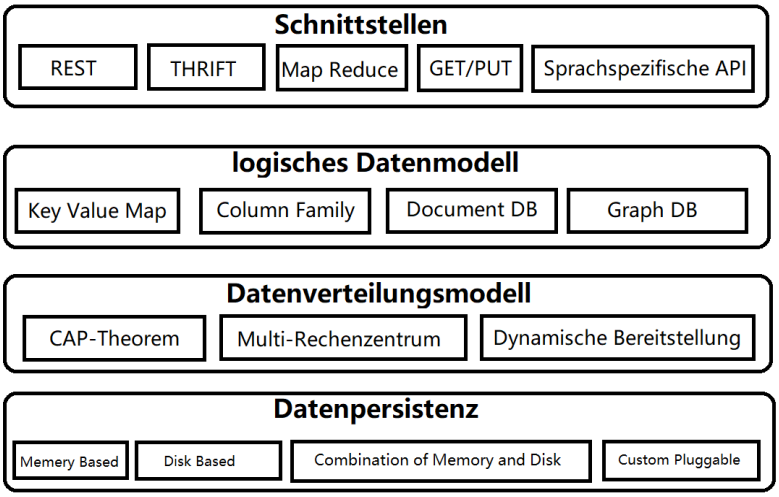


Abbildung 4 : NoSQL Framework (adapt nach)

Durch die obige Einführung können wir wissen, dass NoSQL-Datenbaken nicht relationale Datenbanken ersetzen, sondern die Schwierigkeiten lösen, die die relationalen Datenbanken im vorhandenen Hintergrund nicht lösen kann.

2.2.1 Klassifikation

Bisher gibt es über 200 nichtrelationale Datenbanken (DB-Engines, 2019) und verschiedene Möglichkeiten, NoSQL-Datenbanken zu klassifizieren. Gemäß dem Datenmodell kann es generell in vier Haupttypen unterteilt werden. Mit Abbildung 5 kann man diese verschiedenen Arten von Datenmodellen intuitiver verstehen.

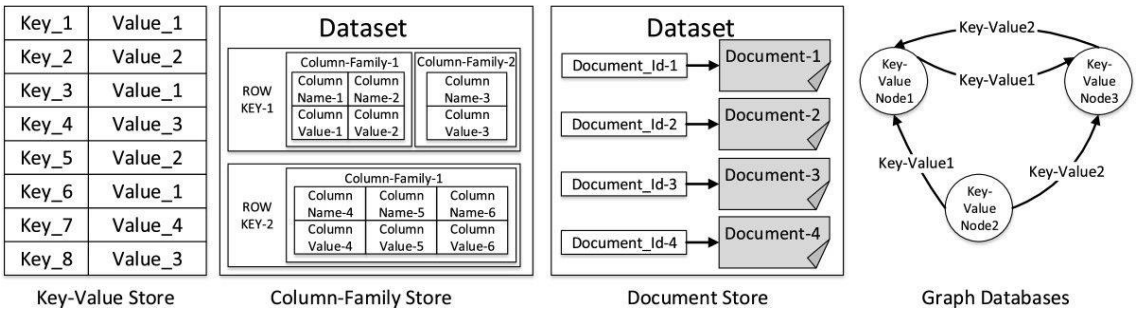


Abbildung 5: verschiedene Arten von NoSQL-Datenmodellen

Schlüssel-Werte Datenbanken (engl. Key-Value Database) verwenden das assoziative Map von Schlüsseln und Werten als ihr grundlegendes Datenmodell. In diesem Modell werden Daten als Sammlung von Schlüssel-Wert-Paaren dargestellt, sodass jeder mögliche Schlüssel in der Sammlung höchstens einmal vorkommt. Die Hauptidee stammt aus der Hash-Tabelle in der Datenstruktur. Dieses Datenmodell ist unkompliziert und kann Datenbankoperationen vereinfachen. Deswegen können Schlüssel-Werte Datenbanken schnellere und effizientere Datenabfragen erzielen (Edlich, Friedland, Hampe, & Brauer,

2010, S. 7). Das Schlüssel-Werte-Datenmodell kann auch Daten verschiedener komplexer Typen lesen und schreiben, z.B. Ganzzahlen, Boolesche Werte, Listen oder JSON-Dokumente. Die wichtigsten Funktionsmerkmale der Schlüssel-Werte Datenbank sind in Tabelle 3 dargestellt.

Tabelle 3 : Schlüssel-Werte Datenbank

Datenmodell	Key-Value-Paaren
Anwendungsszenario	-Warenkörbe bei Online-Shop - Speicherung von Session-Daten - als Zwischenspeicher
Vorteil	Schnelle und effiziente Datenverwaltung
Nachteil	Informationen im "Value" können nicht abgefragt werden. (nicht für komplexe Abfragen geeignet)
Beispiel	Redis, DynamoDB, Memcached

Im Gegensatz zu relationalen Datenbanken, in denen Daten für Zeilen gespeichert werden, orientieren sich **Spaltenorientierte Datenbanken** (engl. Column-oriented Database) an in Spalten gespeicherten Daten. Bei der aggregierten Berechnung einer einzelnen Spalte oder Abrufen einiger Spalten aus einer Tabelle mit vielen Spalten sind spaltenorientierte Datenbanken viel schneller als zeilenorientierte Datenbanken. Weiterhin hat sie höhere Komprimierung, da ähnliche Daten zusammen gespeichert werden. Aber wenn ein neuer Datensatz angelegt/aktualisiert wird oder ein einzelner Datensatz abgerufen wird, sind Spaltenorientierte Datenbanken langsamer als zeilenorientierte Datenbanken. Die wichtigen Merkmale der spaltenorientierten Datenbank werden in Tabelle 4 aufgeführt.

Tabelle 4 : Spaltenorientierte Datenbank

Datenmodell	Spaltenfamilien
Anwendungsszenario	- OLAP (Online Analytical Processing) - Data-Warehouse - verteilte Daten, die geografisch auf mehrere Rechenzentren gespeichert werden
Vorteil	- einfacher für horizontale Skalierung - bessere I/O Performance - starke Skalierbarkeit (leicht für dynamisches Hinzufügen neuer Felder)
Nachteil	- wenige Funktion - geeignet nicht für komplexe Abfragen mit Beziehungen
Beispiel	HBase, Cassandra, BigTable

In **Dokumentenorientierten Datenbanken** (engl. Document-oriented Database) werden alle Daten in den versionierten Dokumenten abgespeichert. Hierbei werden hauptsächlich Dokumenten in JSON oder XML semantisch gespeichert und ist semantisch. Jedes Dokument ist ein Datensatz, der prinzipiell aus einer geordneten Liste von Key-Value-Paaren besteht. Durch die flexible Datenstruktur ermöglichen die Dokumentdatenbanken, sich an die Anforderungen der Anwendungen anzupassen. Das bedeutet, dass keine Schemaaktualisierung und keine Ausfallzeit der Datenbank bei der Änderung des Datenmodells erforderlich sind. Gleichzeitig können die Dokumentenorientierte Datenbanken im Allgemeinen einen Sekundärindex für die Werte erstellen, um die Anwendung der oberen Schicht zu erleichtern. Dies wird von den normalen Schlüssel-Werte Datenbanken nicht unterstützt.

Als relationale Datenbanken sind Dokumentenorientierte Datenbanken für viele Datenabfragen effizienter, weil dort die Informationen, die in den relationalen Datenbanken zuerst aus mehrere Tabellen durch JOINs verknüpft werden müssen, in einem Dokument aggregiert werden können (Sadalage & Fowler, 2012, pp. 20-29). Tabelle 5 zeigt die Haupteigenschaften der Dokumentenorientierte Datenbanken.

Tabelle 5 : Dokumentenorientierte Datenbanken

Datenmodell	ähnlich wie Key-Value-Paaren, aber „Value“ ist versioniertes Dokument.
Anwendungsszenario	- Benutzerprofile - Inhaltsverwaltung (wie Blogs, Videoplattformen usw.) - Kataloge - Web Anwendungen
Vorteil	- hohe Skalierbarkeit - zusammenhängende Daten in einem Dokument - flexible Indexierung, - gute Leistung (hohe Nebenläufigkeit)
Nachteil	- keine einheitliche mächtige Abfragesprache - mehr Programmierung (z.B. Prüfungsmechanismen für die eingegebenen Werte) - keine Beziehung zwischen den Dokumenten
Beispiel	MongoDB, Couchbase, Cloudant

Graph-Datenbanken (engl. Graph Database) sind eine Art NoSQL-Datenbank, in der mithilfe von Graphen Beziehungsinformationen zwischen Entitäten gespeichert werden. In der Graph-Datenbank gibt es grundsätzlich zwei Datenmodelle: Property-Graph-Model und Resource Description Framework (RDF). Property-Graph-Model besteht aus Knoten (engl. Nodes), Kanten (engl. Edges oder Relationships) und ihren jeweiligen Eigenschaften (Properties). Die Knoten repräsentieren die Entitäten und können eine belie-

bige Anzahl von Eigenschaften (Schlüssel-Wert-Paare) enthalten. Die Kanten stellen gerichtete, benannte, semantisch relevante Verbindungen zwischen zwei Knoten bereit. Eine Kante hat immer eine Richtung, einen Typ, einen Startknoten und einen Endknoten. Kanten können wie Knoten auch Eigenschaften haben (Rouse, 2016). In den meisten Fällen besitzen Kanten quantitative Eigenschaften, z.B. Gewichte, Abstände, Kosten, Zeitintervalle oder Bewertungen. Aufgrund dieser effizienten Speichermethode können zwei Knoten eine beliebige Anzahl oder Art von Kanten aufweisen, ohne die Leistung zu beeinträchtigen. Eine Erweiterung des Property-Graph-Model ist Hypergraph. Eine Kante in Property-Graph-Model entspricht ein Paar Knoten. Stattdessen entspricht eine Hyperkante (engl. Hyperedge) in einem Hypergraph einer Mengen von Knoten.

Resource Description Framework (RDF) basiert auf gerichteten Graphen und besitzt eine klar definierte formale Semantik (Rahm, Saake, & Sattler, 2015). Alle Daten in RDF werden in Form von Subjekt-Prädikat-Objekt dargestellt. Subjekt-Prädikat-Objekt wird auch als Tripel bezeichnet. Das Subjekt und das Objekt bezeichnen die Ressource. Das Prädikat bezeichnet Merkmale der Ressource und drückt eine Beziehung zwischen dem Subjekt und dem Objekt aus.

Graph-Datenbanken werden entwickelt, um starke verknüpft bzw. komplex relationale Daten zu speichern und zu verwalten. Die Graph-Datenbank bietet in solchen Fällen eine bessere Leistung als relationale Datenbank und andere NoSQL-Datenbanken. Wenn mehrere Beziehungen zwischen zwei Entitäten im RDBMS bestehen, muss man zur Darstellung der Beziehungen mehrere Verknüpfungstabellen erstellen und die entsprechenden Informationen mit mehrere rekursiv verschachtelte JOINs abfragen. Zudem werden Darstellung und Abfragen mit wachsenden Datensätze immer zeitaufwendiger. Dem entgegen werden in Graph-Datenbanken Beziehungen bereits als Element erstellt, sodass sie direkt verwendet werden können. Daher besitzt eine Graph-Datenbank sehr gute Performance bei komplexen relationalen Abfragen. Weiterhin können vollständige ACID Transaktionen unterstützt werden (Neo4j, 2019). Tabelle 6 zeigt die wichtigen Merkmale der Graph-Datenbank.

Tabelle 6 : Graph-Datenbank

Datenmodell	Graph-Modell (Property Graph, Resource Description Framework)
Anwendungsszenario	- soziale Netzwerke - Navigationssystemen - Kaufempfehlungen (online) - personalisierte Werbung
Vorteil	- hohe Performance für komplexe relationale Daten - Unterstützung von ACID Transaktionen
Nachteil	- wenige Funktionen

	- keine einheitliche Abfragesprache - nicht leicht für horizontale Skalierung auf einer verteilten Architektur
Beispiel	Neo4j, Dgraph, JanusGraph

Neben der oben erwähnten vier Kategorien der NoSQL-Datenbanken gibt es noch einige andere, wie z.B. Time Series DB, Grid-DB, Objektorientierte DB, Navigational DB und so weiter.

2.2.2 CAP

Das CAP Theorem ist eine der Grundtheorien der NoSQL-Datenbanken. Es wurde erstmals im Jahr 2000 von dem amerikanischen Wissenschaftler Eric Brewer beim PODC - Symposium (Principles of Distributed Computing) vorgeschlagen (Brewer E., 2000). Folgend haben Seth Gilbert und Nancy Lynch 2002 die Richtigkeit der CAP-Theorem bewiesen (Gilbert & Lynch, 2002). Das CAP-Theorem gibt an, dass ein verteiltes System höchstens zwei Merkmale der Konsistenz (Consistency), Verfügbarkeit (Availability) und Partitionstoleranz (Partition tolerance) erfüllen kann, und es ist unmöglich, diese drei Merkmale gleichzeitig zu erfüllen.

1) Konsistenz (Consistency)

   Jede Leseoperation kann immer das Ergebnis der zuvor abgeschlossenen Schreiboperation lesen. Das heißt, nachdem die Daten eines bestimmten Knotens in einem verteilten Datenbanksystem geändert wurden, führen die entsprechenden Replikationsknoten bis zum Erreichen eines konsistenten Zustandes gleiche Operationen durch.

2) Verfügbarkeit (Availability)

   Verfügbarkeit bezeichnet eine akzeptable Antwortzeit. Unabhängig von Erfolg oder Misserfolg, muss jede Anfrage innerhalb einer akzeptablen Zeit beantwortet werden. Und der Ausfall eines Knotens hat keine Auswirkungen auf die Verfügbarkeit anderer Knoten.

3) Partitionstoleranz (Partition Tolerance)

   Partitionstoleranz bedeutet, dass das Gesamtsystem normal funktionieren kann, wenn einige Knoten nicht mit anderen Knoten kommunizieren können.

Im CAP Theorem können nur zwei Anforderungen gleichzeitig vollständig erfüllt werden, und die Partitionstoleranz in einem verteilten System implementiert werden muss. Deswegen kann nur eine von Datenkonsistenz und Systemverfügbarkeit mit Vorrang berücksichtigt werden. Ein Missverständnis über CAP-Theorem ist, dass eine der drei Eigenschaften jederzeit verworfen werden muss. Tatsächlich muss eine Entscheidung zwischen

Konsistenz und Verfügbarkeit getroffen werden, wenn eine Netzwerk-Partitionierung oder ein Ausfall auftritt. Zu allen anderen Zeiten sind keine Kompromisse erforderlich (Brewer, 2012).

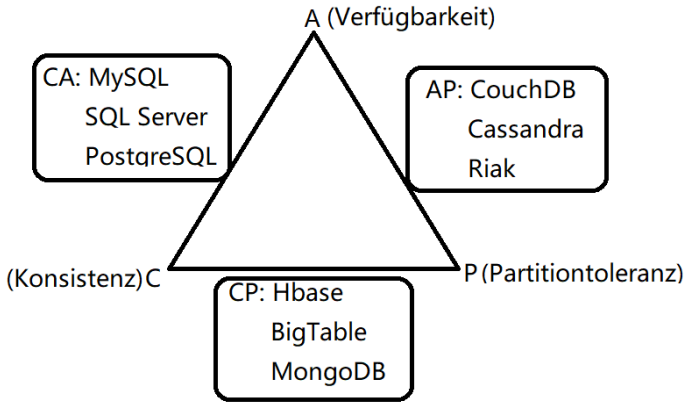


Abbildung 6 : CAP Theorem und Datenbanken

Laut dem CAP-Theorem können die Datenbanken generell in drei Kategorien untereilt werden: CA, CP, AP[6], sowie Abbildung 6 gezeigt. Zum Beispiel gehören die meisten relationalen Datenbankclusters zur Kategorie CA. Ihr Hauptziel ist die Datenkonsistenz und Verfügbarkeit aller Knoten. Die Datenbanken der CP-Kategorie können für viele Bank-Anwendungen verwendet werden, weil Datenkonsistenz im Finanzbereich höchste Priorität genießt. Daneben kann die Datenbanken der CP-Kategorie auf Cloud Computing angewendet werden.

2.2.3 BASE

Die Merkmale von Transaktionen in einer relationalen Datenbank werden normalerweise in Form von ACID, nämlich Atomizität, Konsistenz, Isolation und Persistenz ausgedrückt. Viele NoSQL-Datenbanken neigen jedoch dazu, sich an Partitionstoleranzen zu halten und die Verfügbarkeit oder Konsistenz aufzugeben. Davon wird das BASE-Modell abgeleitet. Der Zweck des BASE-Modells besteht darin, Konflikte zwischen Konsistenz und Verfügbarkeit in hoch skalierbaren verteilten Systemen zu lösen. Dieses Modell erreicht Verfügbarkeit und Skalierbarkeit, indem relationale Modelle und Transaktionen geschwächt werden (Edlich, Friedland, Hampe, & Brauer, 2010, S. 33-35). BASE bedeutet konkret:

1) Basically Available

[6] https://de.wikipedia.org/wiki/CAP-Theorem

Grundsätzlich verfügbar (engl. Basically Available): Wenn ein Teil eines verteilten Systems nicht verfügbar ist, können die anderen Teile weiterhin normal verwendet werden. Also ist Partitionsfehler zulässig.

2) Soft State

   Der „weiche" Zustand (engl. Soft State) bedeutet, dass der Zustand der Datenbank für einen bestimmten Zeitraum asynchron sein kann.

3) Eventual Consistency

   Letztendliche Konsistenz (engl. Eventual Consistency) ist eine spezifische Form von schwacher Konsistenz. Das Speichersystem garantiert, dass alle Zugriffe den zuletzt aktualisierten Wert zurückgeben, wenn keine neuen Aktualisierungen vorgenommen werden. Dies bedeutet, dass die Daten nicht stets konsistent sein müssen, sondern letztendlich konsistent.

Eric Brewer betont, dass BASE und ACID zwei Pole eines Spektrums repräsentieren (Edlich, Friedland, Hampe, & Brauer, 2010, S. 34). Alle Datenbanken sind über das Spektrum verteilt. Einige liegen näher an einem Pol, und Einige liegen näher an dem anderen Pol. Die relationalen Datenbanken befinden sich sehr nah am ACID-Pol. Die NoSQL-Datenbanken liegen in der Regel näher am BASE-Pol.

Werner Vogels, dem Chief Technology Officer von Amazon, fasste weiterhin fünf Varianten der letztendlichen Konsistenz auf seinem Weblog zusammen (Vogels, 2008).

- Causal Consistency

  Wenn der Prozess A dem Prozess B mitgeteilt hat, dass er ein Datenelement aktualisiert hat, gibt der nachfolgende Zugriff durch den Prozess B den aktualisierten Wert zurück und stellt sicher, dass der Schreibvorgang den vorherigen Schreibvorgang ersetzt. Der Zugriff auf Prozess C, der keinen kausalen Zusammenhang mit Prozess A hat, kann dennoch der normalen letztendlichen Konsistenz folgen.

- Read-your-write Consistency

  Nachdem Prozess A ein Datenelement aktualisiert hat, greift er immer auf den aktualisierten Wert zu, und der ältere Wert nie angezeigt wird. Dies ist ein Sonderfall des kausalen Konsistenzmodells.

- Session Consistency

  Session Consistency ist eine praktische Version von Read-your-write Consistency. Solange die Session besteht, garantiert das Speichersystem Read-your-write Consistency. Wenn die Session aufgrund eines bestimmten Fehlers beendet wird, muss eine neue Session erstellt werden.

- Monotonic Read Consistency

  Wenn ein Prozess einen bestimmten Wert für ein Objekt festgestellt hat, werden bei jedem nachfolgenden Zugriff keine vorherigen Werte zurückgegeben.

- Monotonic Write Consistency

  In Monotonic Write Consistency garantiert das System die Serialisierung der Schreibvorgänge über denselben Prozess. Wenn diese Konsistenz nicht gewährleistet ist, ist das System schwer zu programmieren.

Die oben beschriebenen Eigenschaften können miteinander kombiniert werden. Beispielsweise kann Monotonic Read Consistency mit Session Consistency kombiniert werden. Diese beiden Eigenschaften erleichtern Entwicklern das Erstellen von Anwendungen, während sich das Speichersystem die Konsistenz schwächen und eine hohe Verfügbarkeit bieten lässt.

3 MongoDB

MongoDB ist eine verteilte schemafreie Open-Source Dokumentendatenbank, die Hochverfügbarkeit, horizontale Skalierung und geografische Verteilung integriert, und seit ihrer Gründung sehr beliebt. Der Name „Mongo“ wird vom englischen Wort „humongous“ abgeleitet, was riesig bedeutet (MongoDB, 2013). Bereits im Oktober 2007 wurde MongoDB von dem Unternehmen 10gen entwickelt und wurde erstmals im Februar 2009 veröffentlicht (MongoDB, 2010). Um möglichst bessere Leistung zu besitzen, ist MongoDB in C++ geschrieben und unterstützt fast alle aktuellen Betriebssysteme. Die Datenbank erfüllt nicht nur die relevanten Funktionen der NoSQL-Datenbank, sondern bietet auch zahlreiche Vorteile, darunter automatisches Sharding, automatisches Failover, Unterstützung für die Cluster-Erweiterung, Unterstützung für räumliche Indizierung und vieles mehr.

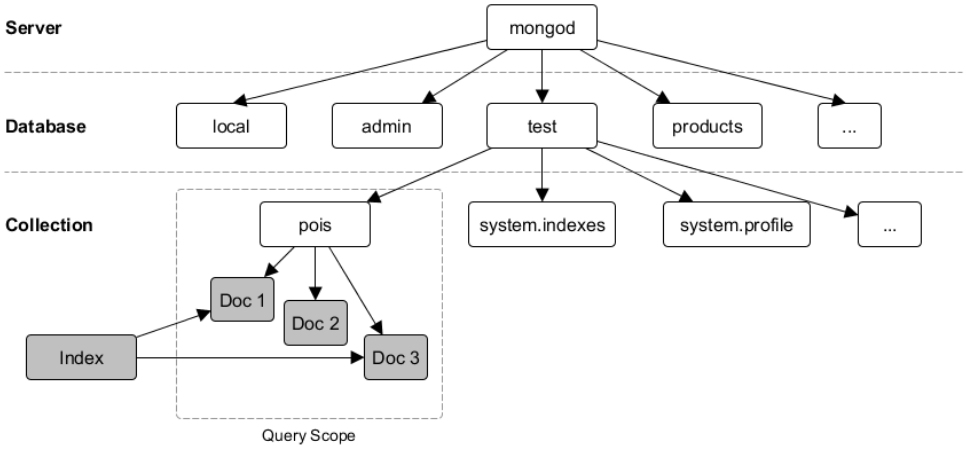


Abbildung 7 : Struktur einer MongoDB-Instanz (Trelle T., 2015)

Wie in Abbildung 7 gezeigt, kann ein Mongo-Server mehrere Datenbanken enthalten. Jede Datenbank kann wieder mehrere Colletions in MongoDB verwalten, die dem Begriff von Tabellen in den relationalen Datenbanken entsprechen. Die einzelnen Datensätze werden in einer *Collection* als einzelne Dokumente verwaltet. Dokumente sind die grundlegende Dateneinheit in MongoDB. In der folgenden Tabelle werden die verschiedenen SQL-Terminologien und -Konzepte sowie die entsprechenden MongoDB-Terminologien und -Konzepte aufgeführt.

Tabelle 7 : Terminologien und Konzepte (SQL und MongoDB)

SQL Terms/Konzepte	MongoDB Terms/Konzepte
Datenbank	Datenbank
Tabelle	Collection
Zeile	BSON Dokument

Spalte	Feld
Index	Index
Tabelle JOINs	$lookup[7], eingebettete Dokumente
Primärschlüssel (Benutzer geben eine eindeutige Spalte oder Spaltenkombination als Primärschlüssel an.)	Primärschlüssel (In MongoDB wird der Primärschlüssel automatisch auf das Feld „_id“ gesetzt.)
Aggregation (z.B. group by)	Aggregation Pipeline[8]
Transaktionen	Transaktionen Tip: Anstatt der Multi-Dokument-Transaktionen ist das denormalisierte Datenmodell (eingebettete Dokumente und Arrays) weiterhin optimal für die meisten Anwendungsfälle

Ein Datensatz wird als ein BSON-Dokument gespeichert. BSON ist eine binäre Darstellung von JSON-Dokumenten und steht für „Binary JSON“. Mit BSON besitzt MongoDB bessere Traversiebarkeit, kompaktere Speicherung (bei bestimmten Datentypen), effizientere Konvertierung und eine Möglichkeit der Speicherung von Binärdaten[9]. MongoDB-Dokumente bestehen aus „field-value pairs“ (siehe Abbildung 8). Als die grundlegende Dateneinheit in MongoDB muss jedes Dokument einen Primärschlüssel enthalten. In der Regel ist der Primärschlüssel ein Feld mit dem Namen „_id“ und vom Typ *ObjectId* (Trelle, 2014, S. 36) als eindeutige Kennung. Die Felder in Dokumenten ähneln den Spalten in einer relationalen Datenbank, und die Werte können verschiedene Datentypen unterstützen, einschließlich anderer Dokumente, Arrays und Arrays von Dokumenten, wie im MongoDB-Benutzerhandbuch angegeben[10]. Das heißt, dass Felder von Dokument zu Dokument unterschiedlich sein können und die Datenstruktur im Laufe der Zeit geändert werden kann.

[7] https://docs.mongodb.com/manual/reference/operator/aggregation/lookup/

[8] https://docs.mongodb.com/manual/reference/aggregation-quick-reference/

[9] bsonspec.org

[10] https://docs.mongodb.com/manual/reference/bson-types/

Abbildung 8: Dokumentenstruktur (MongoDB, 2019)

MongoDB unterstützt die meisten Abfragefunktionen in der aktuellen Datenbank, und verfügt über einen eigenen Mechanismus zur Indexerstellung wie eine relationale Datenbank. Die Abfrage von MongoDB unterstützt noch dynamische Abfragen und Implementierung des Map/Reduce-Algorithmus, um Berechnung und Analyse über riesige Daten parallel verteilt durchführen zu können.

In MongoDB ist eine Operation an einem einzelnen Dokument atomar, weil man eingebettete Dokumente, Arrays und Datenbankreferenz[11] verwenden kann, um Datenbeziehungen in einer einzelnen Dokumentstruktur zu erfassen. Ab Version 4.0 bietet MongoDB die Möglichkeit, Transaktionen mit mehreren Dokumenten für Replica-Set durchzuführen. Außerdem sind verteilte Transaktionen für MongoDB 4.2 geplant[12]. Die neueste stabile Version und Entwicklerversion sind 4.0.10 und 4.1.13.

Heutzutage unterstützt MongoDB drei Speicher-Engines: WiredTiger-, MMAPv1- und In-Memory Speicher-Engine, und bietet steckbare Speicher-Engin-APIs, mit denen Dritte ihre eigenen Speicher-Eigines für MongoDB entwickeln können. Weiterhin stellen MongoDB Treiber für mehr als 10 Sprachen zur Verfügung[13]. Javascript ist die bevorzugte Sprache für die Interaktion mit MongoDB (Edlich, Friedland, Hampe, & Brauer, 2010, S. 116). Aufgrund der hohen Leistung und der umfangreichen Funktionen wird die MongoDB-Datenbank häufig in der Produktion verwendet.

Im Folgenden werden einige Grundkenntnisse ausführlich vorgestellt, die für diese Arbeit relevant oder wichtig sind.

3.1 Komponenten

MongoDB stellt sowohl einer Enterprise- als auch einer Community-Version zur Verfügung und unterstützt eine Vielzahl von Plattformen, z.B. Linux, Windows sowie MacOS X. MongoDB Community Version ist eine Open Source Version. MongoDB Enterprise

[11] https://docs.mongodb.com/manual/reference/database-references/#dbref-explanation

[12] https://docs.mongodb.com/manual/core/transactions/index.html

[13] https://www.mongodb.com/what-is-mongodb

Version bietet zusätzliche Sicherheitsfunktionen, eine In-Memory-Speicher-Engine, Verwaltungs- und Authentifizierungsfunktionen sowie Monitoring-Funktionen über Ops Manager. Dieser Arbeit bezieht sich nur auf die kostenlose 64bit-Community-Version für macOS. Die neueste Version des Installationspakets kann man von der offiziellen Website[14] herunterladen. Dann kann man den .tgz Tarball entpacken und ein Verzeichnis für die Datenspeicherung erstellen. Danach ist es möglich, einen lokalen Server zu starten.

```
# Extrahieren von Dateien aus dem .tgz Tarball
    $ tar -zxvf mongodb-osx-ssl-x86_64-4.0.6.tgz
# Erstellen eines Datenverzeichnis für Datenspeicherung
    $ mkdir -p ./data/db
# Öffnen des Installationsverzeichnisses
    $ cd ~/mongodb-4.0.6
# Starten Mongo-Server und Angeben des Pfads zum Datenspeicher-
verzeichnis
    $ ./bin/mongod –dbpah ./data/db
```

Listing 1 : Installation von MongoDB mit .tgz Tarball unter MacOS

Außerdem kann MongoDB einfach über Homebrew[15] installiert werden. Nach der Installation werden dreizehn Standardkomponenten generiert:

Kernprozesse:

- mongod (Datenbank-Server):

  Mongod ist der primäre Deamon-Prozess für das MongoDB-System. Es wird verwendet, um Datenanforderungen zu verarbeiten, den Datenzugriff zu verwalten und Verwaltungsoperationen im Hintergrund durchzuführen.

- mongos (Router für Sharding):

  Mongos ist ein Routing-Service für MongoDB-Shard-Konfigurationen, der Abfragen aus der Anwendungsebene verarbeitet und den Speicherort dieser Daten im Shard-Cluster ermttelt.

- mongo (Mongo-Shell):

  Mongo ist eine Befehlszeilen-Shell, damit Entwickler und Systemadministratoren mit Clients interagieren können. Mongo bietet auch eine voll funktionsfähige JavaScript-Umgebung für die Verwendung mit einer MongoDB.

Backup- und Wiederherstellung-Tools:

- mongodump:

[14] https://www.mongodb.com/download-center

[15] https://docs.mongodb.com/manual/tutorial/install-mongodb-on-os-x/

Mongodump ist ein Dienstprogramm zum Erstellen eines binären Exports von Datenbankinhalten, und kann Daten von mongod- oder mongos-Instanzen exportieren. mongodump kann auch als Teil der Backup-Strategie von mongorestore verwendet werden.

- mongorestore

  Das mongorestore-Programm lädt Daten entweder aus einem von mongodump erstellten binären Datenbank-Dump oder aus der Standardeingabe in eine mongod- oder mongos-Instanz.

- bsondump

  Bsondump konvertiert BSON-Dateien in für Menschen lesbare Formate, einschließlich JSON. Zum Beispiel ist basondump nützlich, um die von mongodup generierten Ausgabedateien zu lesen.

Import- und Export-Tools:

- mongoimport

  Mit mongoimport können Extended JSON[16]-, CSV- oder TSV-Dateien in MongoDB-Datenbanken importiert werden.

- mongoexport

  Mongoexport ist ein Dienstprogramm zum Exportieren von JSON- oder CSV-Daten, die in MongoDB-Instanzen gespeichert sind.

Diagnosewerkzeuge:

- mongostat

  Das Dienstprogramm mongostat bietet einen schnellen Überblick über den Status einer gegenwärtig laufenden mongod- oder mongos-Instanz.

- mongotop

  Mongotop stellt eine Methode zum Verfolgen der Zeit bereit, die eine MongoDB-Instanz für das Lesen und Schreiben von Daten benötigt. Die von mongotop bereitgestellten Statistiken werden basierend auf der einzelnen *Collection* generiert. Standardmäßig gibt mongotop einmal pro Sekunde Werte zurück.

[16] https://docs.mongodb.com/manual/reference/mongodb-extended-json/

- mongoreplay

  Mongoreplay ist ein Tool zum Erfassen und Wiedergeben von Datenverkehr, mit dem an MongoDB-Instanzen gesendete Befehle überprüft und protokolliert werden können, und diese Befehle zu einem späteren Zeitpunkt auf einem anderen Host wiedergegeben werden.

GridFS Tool:

- mongofiles

  Das Dienstprogramm mongofiles kann Dateien in GridFS (siehe Kapitel 3.6) über die Befehlszeile manipulieren. Dies ist besonders nützlich, da es eine Schnittstelle zwischen im Dateisystem und in GridFS gespeicherten Objekten bietet.

MongoDB Compass

- install_compass

  Install_compass ist ein plattformspezifisches Installationsskript für MongoDB Compass. MongoDB Commpass ist eine grafische Benutzeroberfläche und in der Lage, Dokumentenstruktur zu bearbeiten, Abfragen durchzuführen, Daten zu indizieren und vieles mehr.

Neben dem MongoDB Server hat MongoDB auch andere Datenbankprodukte für unterschiedliche Anwendungsszenarien entwickelt. In Anlehnung an andere NoSQL-Datenbankanbieter hat MongoDB Inc. 2016 eine Cloud-Datenbank mit dem Namen *MongoDB Atlas* als Dienst eingeführt. Atlas kann auf AWS, Microsoft Azure und Google Cloud Platform laufen. Kurz danach hat MongoDB eine Plattform namens Stitch für die Anwendungsentwicklung auf *MongoDB Atlas* veröffentlicht. *MongoDB Stitch* ist ein Backend as a Service (BaaS), mit dem Entwickler schnell Anwendungen erstellen können, ohne die Serverinfrastruktur einrichten zu müssen. Außerdem hat MongoDB auch *MongoDB Mobile* für mobile Geräte auf den Markt gebracht.

Darüber hinaus stellt MongoDB drei praktische Analysewerkzeuge, *MongoDB Charts*, *MongoDB Connector for BI* (Business Intelligence) und *MongoDB Connector for Spark*. *MongoDB Charts* kann eine Verbindung zu einer beliebigen MongoDB-Instanz als Datenquelle herstellen, Diagramme und Grafiken zu erstellen, Dashboards zu bilden und diese für die Zusammenarbeit mit anderen Benutzern freizugeben. Mittels *MongoDB Connector for BI* können Benutzer die NoSQL-Datenbank mit Business Intelligence-Tools verbinden, um Daten zu visualisieren und Berichte mithilfe von SQL-Abfragen zu erstellen. Apache Spark ist eine Leistungsstarke Verarbeitungs-Engine, die für Geschwindigkeit, Benutzerfreundlichkeit und anspruchsvolle Analyse ausgelegt ist. Die Vorteile von Spark und MongoDB können unter Einsatz von *MongoDB Connector for Spark* kombiniert werden. Beispielsweise kann MongoDB Verarbeitung von Datenströmen (Real-

Time Analytics), Machine-Learning-Algorithmen, Berechnung von Graphen (siehe Kapitel 2.2.1 über Graph-Datenbanken) und vielfältige Transformations-Funktionen (Latreider, 2018, S. 27). Die Integration dieser beiden Big-Data-Technologien erspart den Betriebsteams außerdem den mühsamen Datenaustausch zwischen getrennten Betriebs- und Analyseinfrastrukturen.

3.2 Grundoperationen

Im letzten Abschnitt sind die Standardkomponenten und entsprechenden Funktionen von MongoDB beschrieben. Wir wissen auch, wie man einen MongoDB-Server startet und mit der MongoDB-Shell eine Verbindung zum Server herstellt. CURD bedeutet die vier üblichen Operationen in relationalen Datenbanken: Create, Read, Update und Delete. Stattdessen wird die Abkürzung IFUR in MongoDB verwendet (Inden, 2016, S. 227). Im Folgenden wird es vorgestellt, wie man Dokumente einfügen, löschen, ändern und lesen kann. Davor lassen wir uns zunächst verstehen, wie man Datenbanken und *Collections* erzeugen, abfragen und löschen kann.

Die Syntax für MongoDB zum Erstellen einer Datenbank lautet wie folgt:

```
> use DATABASE_NAME
```

Die Syntax von MongoDB zum Löschen einer Datenbank lautet wie folgt:

```
> db.dropDatabase()
```

Alle Datenbanken Anzeigen:

```
>show dbs
```

Eine *Collection* wird mit der Methode *createCollection()* in MongoDB erstellt:

```
db.createCollection(name, options)
```

Davon gibt es zwei Parameter, *name* und *options*. *Name* steht für den zu erstellenden Name. *Options* sind optionale Parameter, die Optionen für die Speichergröße und den Index angeben. Das Optionsdokument enthält die folgenden Felder:

Tabelle 8 : Einige wichtige optionale Parameter von *Options*

Feld	Typ	Beschreibung
capped	boolean	(optional) Geben Sie true an, um eine begrenzte Sammlung zu erstellen. Wenn Sie true angeben, müssen Sie auch eine maximale Größe im Feld size festlegen.
autoIndexId	boolean	(optional) Geben Sie false an, um die automatische Erstellung eines Index für das Feld _id zu deaktivieren.

size	number	(optional) Geben Sie eine maximale Größe in Byte für eine Capped Collection (Trelle T. , 2014, S. 31) an. Sobald eine Capped Collection ihre maximale Größe erreicht hat, entfernt MongoDB die älteren Dokumente, um Platz für die neuen Dokumente zu schaffen. Das Größenfeld ist für Capped Collection erforderlich und wird für andere *Collections* ignoriert.
max	number	(optional) Die maximale Anzahl von Dokumenten, die in der Capped *Collection* zulässig sind.

Die *Collection* wird mit der Methode *drop()* in MongoDB entfernt:

```
db.collection.drop()
```

Praktisches Beispiel:

```
>use test
    Switched to db test
>db.createCollection("users")
    {"ok" : 1 }
>db.createCollection("users1")
    {"ok" : 1 }
>show collections
    users
    users1
>db.users1.drop()
    True
>show collections
    users
```

3.2.1 Insert

MongoDB bietet die folgenden Methoden zum Einfügen von Dokumenten in eine *Collection*:

Tabelle 9: Insert-Befehlsliste

db.collection.insertOne()	fügt eines einzelnen Dokuments in eine *Collection* ein.
db.collection.insertMany()	fügt merherer Dokumente in eine *Collection* ein.
db.collection.insert()	fügt eines einzelnen Dokuments oder mehrerer Dokumente in eine *Collection* ein.

Die maximale Größe eines einzelnen Dokuments beträgt 16 MB. Wenn die verwendete *Collection* noch nicht vorliegt, wird sie automatisch vor dem ersten Schreiben erstellt. Die Schreiboperationen in MongoDB folgt grundsätzlich dem *Fire-and-Forget*-Prinzip. Unter *Fire-and-Forget*-Prinzip versteht man, dass Anwendungen sich über den Fehlerstatus der letzten Schreiboperation aktiv erkundigen müssen. Und Der Server liefert den

Status nicht automatisch zurück. Wenn es erfolgreich ist, wird eine bessere Leistung erzielt.

Ein einfaches Beispiel wie folgt:

```
> db.users.insertMany( [ {id: "001", name: Jack, age: 24},
                       {id: "002", name: Jessy, age: 18},
                       {id: "003", name: Karl, age: 32} ] )
```

Tabelle 10: SQL to MongoDB-Zuordnungstabelle für Insert

SQL INSERT-Anweisung	MongoDB insertOne () -Anweisung

`INSERT INTO users(id, name, age)`
`VALUES ("001", Jack, 24)`
	`db.users.insertOne(`
	`{id: "001", name: "Jack", age: 24}`
	`)`

3.2.2 Remove

MongoDB stellt die folgenden Methoden zum Löschen von Dokumenten einer *Collection*:

Tabelle 11: Delete-Befehlslist

db.collection.deleteOne()	löscht höchstens ein einzelnes Dokument, das mit einem angegebenen Filter übereinstimmt, obwohl möglicherweise mehrere Dokumente mit dem angegebenen Filter übereinstimmen.
db.collection.deleteMany()	löscht alle Dokumente, die einem bestimmten Filter entsprechen.
db.collection.remove()	löscht ein einzelnes Dokument oder alle Dokumente, die einem bestimmten Filter entsprechen.

In MongoDB zielen Löschoperationen auf eine einzelne *Collection* ab. Alle Schreiboperationen in MongoDB sind atomar auf der Ebene eines einzelnen Dokuments. Sie können Kriterien oder Filter angeben, die die zu entfernenden Dokumente identifizieren. Diese Filter verwenden dieselbe Syntax wie Leseoperationen.

Ein einfaches Beispiel wie folgt:

```
# Löschen Sie Benutzerdaten, die älter als 20 sind.
    > db.users.deleteMany( age: { $gt: 20} )
```

Tabelle 12: SQL to MongoDB-Zuordnungstabelle für Delete

SQL Delete-Anweisung	MongoDB deleteMany() -Anweisung
`DELETE FROM users` `WHERE name = "Jack"`	`db.users.deleteMany( { name: "Jack" } )`
`DELETE FROM users`	`db.users.deleteMany({})`

3.2.3 Update

In MongoDB wird die Methode db.collection.update () verwendet, um vorhandene Dokumente in einer Sammlung zu ändern. Diese Methode definiert die Abfragekriterien zum Identifizieren der zu aktualisierenden Dokumente und aller Dokumentoptionen, die sich auf deren Verhalten auswirken. Operationen, die durch ein Update ausgeführt werden, sind in einem einzigen Dokument zusammengefasst. Im Folgenden sind die Befehle für allgemeine Aktualisierungsoperationen aufgeführt:

Tabelle 13: Update-Befehlslist

db.collection.updateOne()	aktualisiert Sie höchstens ein einzelnes Dokument, das einem angegebenen Filter entspricht, obwohl möglicherweise mehrere Dokumente dem angegebenen Filter entsprechen.
db.collection.updateMany()	aktualisiert alle Dokumente, die einem bestimmten Filter entsprechen.
db.collection.replaceOne()	ersetzt höchstens ein einzelnes Dokument, das mit einem angegebenen Filter übereinstimmt, obwohl möglicherweise mehrere Dokumente mit dem angegebenen Filter übereinstimmen.
db.collection.update()	aktualisiert oder ersetzt entweder ein einzelnes Dokument, das einem bestimmten Filter entspricht, oder aktualisiert alle Dokumente, die einem bestimmten Filter entsprechen.

Die Aktualisierung der Dokumentation muss auch vom Aktualisierungsoperator durchgeführt werden, z.B. *$set* legt den Wert eines Feldes in einem Dokument fest. $rename kann ein Feld umbenennen und viele andere mehr. Für Details zu bestimmten Operatoren, einschließlich Syntax und Beispielen, finden Sie im offiziellen Handbuch[17].

Ein einfaches Beispiel wie folgt:

```
# Ändern Sie das Alter von Xiao Ming auf 25.
  > db.users.updateMany( {name: "Jack"},
                         { $set: { age: 25}})
```

Tabelle 14: SQL to MongoDB-Zuordnungstabelle für Update

SQL Update-Anweisung	MongoDB updateMany() -Anweisung

`UPDATE people`
`SET age = 25`
`WHERE name = "Jack"`
	`db.people.updateMany(`
	`{ name: "Jack" },`
	`{ $set: { age: 25 } }`
	`)`

`UPDATE people` `SET age = age + 3`	`db.people.updateMany(` `{ age { $lt: 20 } } ,`

[17] https://docks.mongodb.com/manual/reference/operator/update

WHERE age < 20	{ $inc: { age: 3 } })

3.2.4 Find

Im Allgemeinen können innerhalb einer Abfrage nur Dokumente aus einer *Collection* durchsucht und zurückgegeben werden. Es gibt kein JOIN wie in relationalen Datenbanken. Die Abfrageergebnisse werden in Form eines *Cursors* bereitgestellt, und der Client kann durch den *Cursor* die Ergebnisse iterieren (Trelle T. , 2014, S. 93). Mit anderen Worten, wenn Sie eine Abfrage ausführen, liefert es einen *Cursor* auf das Abfrageergebnis zurück und nicht alle Ergebnisse in einem Durchgang, Im Wesentlichen ist der Cursor das Conduit, mit dem Sie alle Ergebnisse einer Abfrage aus der Datenbank extrahieren.

Diese Abfragen definieren die Kriterien oder Bedingungen, anhand derer MongoDB die Dokumente identifiziert, die an den Client zurückgeliefert werden müssen. Eine Abfrage kann eine Projektion enthalten, die die Felder definiert, die mit den zurückzugebenden Dokumentfeldern übereinstimmen. Sie können Abfragen mit vielfältigen Suchoperatoren modifizieren, z.B. *$limit*(Begrenzung), *$skip*(Überspringen), $*sort*(Sortierung) usw.

Ein einfaches Beispiel:

```
  db.<collection>.find ( condition, projection)
# Anzeigen aller Dokumente, die älter als 20 sind, ohne das Feld id
  db.users.find( { age: { $gt: 20} }, { id: 0, name: 1, age: 1 })
```

Darüber hinaus unterstützt MongoDB reguläre Ausdrücke, verschachtelte Dokumentabfragen und Arrayabfragen. Diese Arbeit wird nicht im Detail beschrieben. Eine Zuordnungsliste von repräsentativen SQL-Anweisungen und MongoDB-Abfrageanweisungen ist unten angegeben.

Tabelle 15: SQL to MongoDB-Zuordnungstabelle für Select

SQL Select-Anweisung	MongoDB find() -Anweisung
SELECT * FROM users	db.users.find()
SELECT name, age FROM users	db.users.find({ }, { id: 0, name: 1, age: 1 })
SELECT * FROM users WHERE age > 20 AND age <= 50 ORDER BY id DESC	Db.users.find({ age: { $gt: 20, $lte: 50 } }).sort({ user_id: -1 })
SELECT *	db.users.find(

FROM users WHERE name like "j%"	{ user_id: { $regex: /^j/ } }) or db.users.find({ user_id: /^j/ })
SELECT * FROM users LIMIT 5 SKIP 10	db.users.find().limit(5).skip(10)

Aggregation von Daten

Wenn die grundlegenden Abfragemethoden nicht ausreichen, bietet MongoDB auch viele Aggregationsfunktionen. Aggregationsoperationen fassen Werte aus mehreren Dokumenten zusammen und können verschiedene Operationen für die gruppierten Daten ausführen, um ein einzelnes Ergebnis zurückzugeben. MongoDB bietet drei Möglichkeiten, um eine Aggregation durchzuführen: Abfragemethoden zum Aggregieren, Aggregationsframework und Map/Reduce (vgl. Abbildung 9).

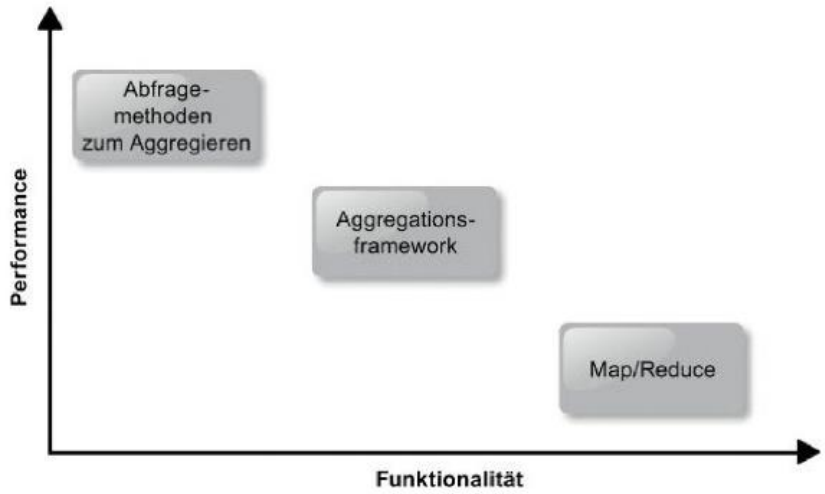


Abbildung 9: Übersicht der Aggregationsmöglichkeiten in MongoDB (Trelle T. , 2014, S. 195)

Das Aggregations-Framework basiert auf dem Konzept der Datenpipeline[18]. Dokumente werden in eine mehrstufige Pipeline eingegeben, die die Dokumente in ein aggregiertes Ergebnis umwandelt.

[18] https://en.wikipedia.org/wiki/Pipeline_(computing)

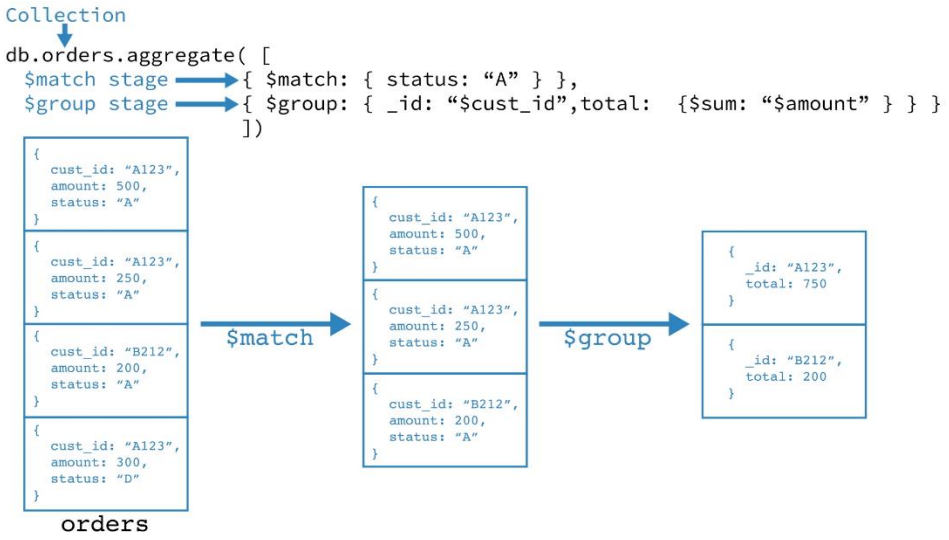


Abbildung 10: Ein Beispiel für Aggregation-Framework

Erste Stufe: Die Stufe "$ match" filtert die Dokumente nach dem Statusfeld und übergibt die Dokumente mit dem Status "A" an die nächste Stufe.

Zweite Stufe: In der $ group-Stufe werden die Dokumente nach dem Feld cust_id gruppiert, um die Summe des Betrags für jede eindeutige cust_id zu berechnen.

MapReduce[19] bestehen im Allgemeinen aus zwei Phasen: einer Map-Phase, in der jedes Dokument verarbeitet und ein oder mehrere Objekte für jedes Eingabedokument ausgegeben werden. Reduce-Phase kombiniert die Ausgabe der Map-Operation. Wie andere Aggregationsverfahren kann Map-Reduce eine Abfragebedingung angeben, um die Eingabedokumente auszuwählen sowie die Ergebnisse zu sortieren und einzuschränken.

Bei MapReduce werden benutzerdefinierte JavaScript-Funktionen zum Durchführen der Map- und Reduce-Operationen verwendet. Während das benutzerdefinierte JavaScript im Vergleich zur Aggregationspipeline eine große Flexibilität bietet, ist MapReduce generell weniger effizient und komplex als die Aggregationspipeline.

Index

In der Regel sind Indizes spezielle Datenstrukturen, in denen ein kleiner Teil des Datensatzes der *Collection* in leicht zu traversierender Form gespeichert wird. Abfragen werden effizient mit Hilfe von Indizes in MongoDB ausgeführt. Mithilfe von Indizes kann MongoDB Dokumente finden, die den Abfragekriterien entsprechen, ohne einen *Collection*-Scan durchzuführen. Wenn eine Abfrage einen geeigneten Index hat, verwendet MongoDB den Index und begrenzt die Anzahl der abgefragten Dokumente.

[19] https://en.wikipedia.org/wiki/MapReduce

In Indizes werden Feldwerte in der Reihenfolge des Werts gespeichert. MongoDB sortiert die Ergebnisse und gibt sie in der Reihenfolge der Indizes zurück. Die Indizes von MongoDB ähneln den Indizes in anderen Datenbanken. MongoDB definiert die Indizes auf *Collection*-Level[20] für die Verwendung in einem beliebigen Feld oder Unterfeld.

MongoDB unterstützt die folgenden Indextypen für die Abfrage:

Tabelle 16: Indextypen

Indextypen	Erklärung
Default _id	Jede *Collection* enthält einen Index für das Standardfeld *_id*. MongoDB erstellt während der Erstellung einer *Collection* einen eindeutigen Index für das Feld _id. Der Index _id verhindert, dass Clients zwei Dokumente mit demselben Wert für das Feld _id einfügen. Sie können diesen Index nicht im Feld _id ablegen.
Single Field	Zusätzlich zum von MongoDB definierten _id-Index unterstützt MongoDB die Erstellung benutzerdefinierter auf- und absteigender Indizes für ein einzelnes Feld eines Dokuments.
Compound Index	Für mehrere Felder unterstützt MongoDB benutzerdefinierte Indizes, z. B. Verbundindizes. Die Reihenfolge der Felder in einem Verbundindex ist in MongoDB von Bedeutung.
Multikey Index	MongoDB verwendet Multikey-Indizes, um Array-Daten zu indizieren. Beim Indizieren eines Feldes mit einem Array-Wert nimmt MongoDB für jedes Array-Element separate Indexeinträge vor.
Geospatial Index	Zum Abfragen von Geodaten verwendet MongoDB zwei Indextypen: 2D-Indizes (planare Geometrie) und 2D-Sphärenindizes (sphärische Geometrie).
Text Index	MongoDB bietet einen Textindextyp, der die Suche nach dem Inhalt der Zeichenfolgen in einer *Collection* unterstützt. Diese Textindizes speichern keine sprachspezifischen Stoppwörter[21] (z. B. "the", "a", "or") und verhindern die Wörter in einer *Collection*, um nur Stammwörter zu speichern.
Hashed Index	MongoDB unterstützt Hash-basiertes Sharding und stellt Hash-Indizes bereit. Diese indizieren die Hashes des Feldwerts.

Man kann einen Index in der Mongo-Shell mit *db.collecion.createIndex()* erstellen. Ein Beispiel ist wie folgt:

```
db.collection.createIndex(<key and index type specification>, <op-
tions>)
```

[20] https://docs.mongodb.com/manual/core/collection-level-access-control/index.html

[21] https://de.wikipedia.org/wiki/Stoppwort

```
# Erstellung eines absteigenden Single-Field-Indexes für das Namens-
feld
   db.users.createIndex( { name: -1 } )
```

Weitere Details finden Sie im offiziellen Handbuch[22].

3.3 Replica Set

MongoDB verwendet Replica Set, um die Ausfallsicherheit sicherzustellen. Das Replica Set ist ein Master-Slave-Cluster mit automatischem Failover[23]. Der größte Unterschied zum herkömmlichen Master-Slave-Cluster besteht darin, dass es keinen festen Master-Knoten gibt. Der Master-Knoten wird durch die Clusterwahl generiert. Wenn der primäre Knoten heruntergefahren oder angehalten wird, wird er daher automatisch auf andere Knoten umgeschaltet. Das besteht aus einer Reihe von Mongod-Instanzen, die Datenredundanz- und Hochverfügbarkeitsfunktionen bieten. Ein Replica Set enthält mehrere datentragende Knoten und optional einen Arbiter-Knoten. Von den datentragende Knoten wird nur ein Mitglied als Primärknoten (genannt *Primary*) betrachtet, während die anderen Knoten als Sekundärknoten (genannt *Secondaries*) betrachtet werden.

Der Primary ist der einzige Knoten im gesamten Replica Set, der schreibende Operationen ausführt. Während der primäre Knoten die Schreiboperationen schafft, zeichnet er alle Änderungen an seinen Datensätzen in seinem Bedienprotokoll auf, d. H. Oplog.

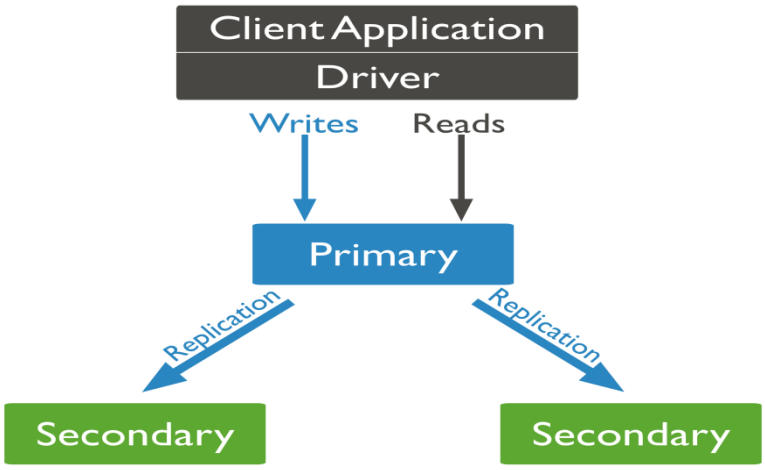


Abbildung 11: Replica Set Modell (MongoDB, 2019)

Das Oplog (Operations-Log) ist eine spezielle capped *Collection* (s. Abschnitt 3.2, S 34), die eine fortlaufend Aufzeichnung aller Operationen, mit denen die in der Datenbank gespeicherten Daten geändert werden. Die Sencondaries synchronisiert die Daten, indem

[22] https://docs.mongodb.com/manual/aggregation/

[23] https://en.wikipedia.org/wiki/Failover

sie das Oplog des Primary replizieren und die Operationen auf eigene Daten ausführen. Jedes sekundäre Mitglied kann Oplog von jedem anderen Mitglied importieren. Um die Replikation zu vereinfachen, senden alle Mitglieder des Replica Sets Heartbeats (Pings) an alle anderen Mitglieder. Die Knoten erkennen die Aktivität miteinander über den Heartbeat-Mechanismus. Wenn der Primary nicht verfügbar ist, halten ein wählbarer Secondary-Knoten eine Wahl ab, um neuen Primary auszuwählen.

Das Replica Set kann keine Schreiboperationen verarbeiten, bis die Wahl erfolgreich abgeschlossen wurde. Das Replica Set kann weiterhin Leseabfragen durchführen, wenn solche Abfragen so konfiguriert sind, dass sie auf Secondary-Knoten ausgeführt werden.

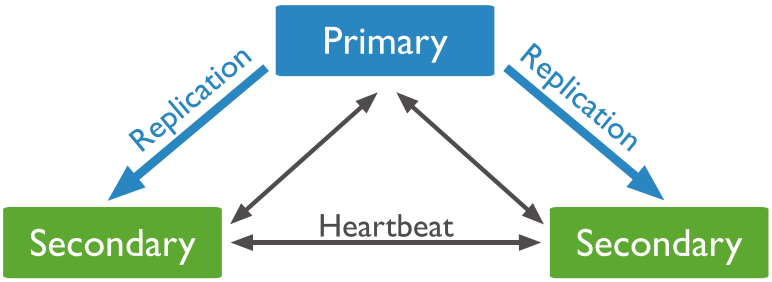


Abbildung 12: Funktionsprinzip der Secondaries (MongoDB, 2019)

Obwohl Clients keine Daten in Secondary-Knoten schreiben können, können sie Daten aus den Secondary-Knoten lesen. Der Cluster kann die Belastung des Primary-Knotens verringern, indem Lese- und Schreibzugriff getrennt werden.

Der Arbiter-Knoten enthält keinen Replikationsdatensatz und kann nicht der primäre Knoten sein. Das Replica Set verwaltet die Auswahl des Primary-Knotens unter Verwendung des Arbiter-Knotens und stellt sicher, dass die Anzahl der Knoten, die an der Abstimmung im Cluster teilnehmen, den Anforderungen der Wahl entspricht. Wenn die Anzahl der Mitglieder des Replica Sets gerade ist, muss beim Prozess der Wahl ein Arbiter-Knoten hinzugefügt werden.

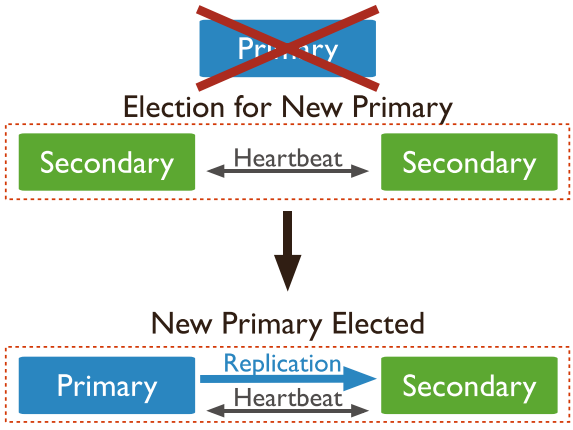


Abbildung 13: Automatisches Failover-Prozess (MongoDB, 2019)

Wenn die Verbindung zwischen Knoten unterbrochen wird, leitet das Replica Set ein automatisches Failover ein, d.h. ein neuer Primärknoten muss ausgewählt werden. Jeder teilnehmende Knoten (nicht-arbitrierter Knoten) hat eine Priorität: Der Knoten mit der Priorität 0 kann nur an der Wahl teilnehmen, kann jedoch nicht der Primary-Knoten sein. Die Knoten, deren Priorität nicht 0 ist, werden entsprechend der Priorität gewählt. Und der Knoten mit der höchsten Priorität wird zum Primary-Knoten. Wenn dieselbe Priorität vorhanden ist, wird der Primärknoten basierend auf dem Alter des Datensatzes ausgewählt.

3.4 Sharding

Sharding ist eine Methode zum Verteilen von Daten auf mehrere Computer. Das Sharding wird in MongoDB eingesetzt, um sehr große Datenmengen und hoher Durchsatz zu unterstützen. Mithilfe eines Shard Keys[24] kann man die Dokumente disjunkt auf mehrere Shards verteilen. Jede Teilmenge der sharded Daten ist ein *Chunk*. Jeder Chunk hat einen inklusive unteren und einen exklusiven oberen Bereich basierend auf dem Shard Key. Weiterhin ist ein Shard vorzugsweise ein vollständiges Replica Set oder kann auch eine einzelne mongod-Instanz sein.

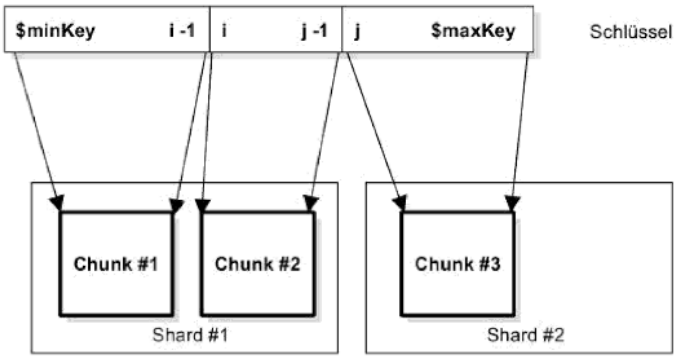


Abbildung 14: Sharding anhand des Shard Key (Trelle T. , 2014, S. 71)

Ein MongoDB-Sharded-Cluster besteht hauptsächlich aus drei Servertypen: *Mongos*, *Shard* und *Config Server*.

[24] https://docs.mongodb.com/manual/core/sharding-shard-key/index.html

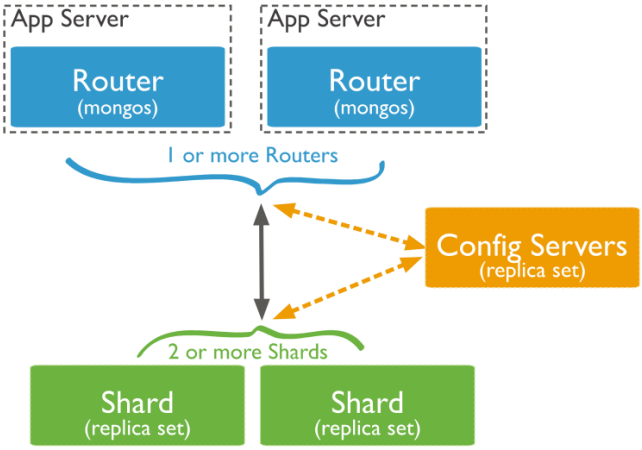


Abbildung 15: Architektur des sharded Clusters

Mongos

Mongos-Instanzen leiten Abfragen und Schreiboperationen an Shards in einem sharded Cluster weiter. Mongos bieten die einzige Schnittstelle zu einem sharded Cluster aus der Sicht von Anwendungen. Anwendungen verbinden oder kommunizieren niemals direkt mit den Shards. Der Mongos verfolgt, welche Daten sich auf welchem Shard befinden, indem er die Metadaten von *Config Servers* zwischenspeichert. Die Mongos verwenden die Metadaten, um Operationen von Anwendungen und Clients an die Mongod-Instanzen weiterzuleiten. Ein Mongos hat keinen dauerhaften Zustand und verbraucht nur minimale Systemressourcen (Trelle T. , 2014, S. 74).

Eine Mongos-Instanz leitet eine Abfrage an einem Cluster weiter, indem:

- die Liste der Shards bestimmt wird, die die Abfrage zu erhalten sind.
- ein Cursor auf alle ausgerichteten Shards erstellt wird.

Der Mongos führt dann die Daten von jedem der ausgerichteten Shards zusammen und gibt das Ergebnisdokument zurück. Bestimmte Abfragemodifikatoren, z. B. Sortieren, werden auf einem Shard wie dem primären Shard ausgeführt, bevor Mongos die Ergebnisse abruft.

Shards

Ein Shard enthält eine Teilmenge der sharded-Daten für einen sharded-Cluster. Die Shards des Clusters enthalten den gesamten Datensatz für den Cluster. Ab MongoDB 3.6 müssen Shards als Replica Set eingesetzt werden, um Redundanz und hohe Verfügbarkeit zu gewährleisten. Das Ausführen von Abfragen auf einem einzelnen Shard gibt nur eine Teilmenge von Daten zurück. Zur Erhaltung eines vollständigen Datensatzes muss man eine Verbindung zu Mongos herstellen, um Operationen auf Cluster-Level auszuführen, einschließlich Lese- oder Schreibvorgängen.

Jede Datenbank in einem sharded Cluster verfügt über einen primären Shard, der alle nicht sharded *Collections* für diese Datenbank enthält. Jede Datenbank hat einen eigenen primären Shard. Der primäre Shard hat keine Beziehung zum Primary in einem Replica Set. Wenn *mongos* eine neue Datenbank erstellt, wählt er den primären Shard aus, indem er den Shard in dem Cluster auswählt, der die geringste Datenmenge aufweist. Mongos verwendet das vom Befehl *listDatabase* zurückgegebene Feld *totalSize* als Teil der Auswahlkriterien.

Man kann den Befehl *movePrimary* verwenden, um das primäre Shard für eine Datenbank zu ändern. Das Migrieren des primären Shards kann viele Zeit in Anspruch nehmen. Abhängig von der zu migrierenden Datenmenge kann sich die Migration auf die gesamten Clustervorgänge auswirken. Die Auswirkungen auf den Clusterbetrieb und die Netzwerklast muss berücksichtigt werden, bevor der primären Shard ändert wird.

Config Server

Config Server speichern die Metadaten für einen sharded-Cluster. Die Metadaten spiegeln den Status und die Organisation aller Daten und Komponenten im Sharded-Cluster wider (Trelle T. , 2015, S. 75). Die Metadaten enthalten die Liste der *Chunks* auf jedem Shard und die Bereiche, die die *Chunks* definieren.

Die mongos-Instanzen speichern diese Daten im Cache und verwenden sie, um Lese- und Schreiboperationen an die richtigen Shards weiterzuleiten. Mongos aktualisiert den Cache, wenn sich Metadaten für den Cluster ändern, z. B. *Chunk Splits* oder das Hinzufügen eines Shards. Shards lesen auch *Chunk*-Metadaten von den Konfigurationsservern.

Die Konfigurationsserver speichern auch Informationen zur Authentifizierungskonfiguration, z. B. rollenbasierte Zugriffssteuerung oder interne Authentifizierungseinstellungen für den Cluster. MongoDB verwendet auch die Konfigurationsserver, um verteilte Sperren zu verwalten. Jedes sharded-Cluster muss über eigene Konfigurationsserver verfügen.

Ab MongoDB 3.2 können Konfigurationsserver für sharded-Cluster als Replica Set anstelle von drei gespiegelten *Config Servers* eingesetzt werden. Die Verwendung eines Replica Sets für die Konfigurationsserver verbessert die Konsistenz zwischen den Konfigurationsservern. Durch die Verwendung eines Replica Sets für Konfigurationsserver kann ein sharded-Cluster mehr als 3 Konfigurationsserver haben, da ein Replica Set bis zu 50 Mitglieder haben kann. Um Konfigurationsserver als Replica Set bereitzustellen, müssen die Konfigurationsserver die WiredTiger-Speicher-Engine[25] ausführen.

Wenn das Replica Set auf das folgende Szenario stößt, muss man die Verwendung eines Sharded-Clusters in Betracht ziehen.

[25] https://docs.mongodb.com/manual/core/wiredtiger/index.html

- Die Anforderungen an die Speicherkapazität überschreiten die Festplattenkapazität eines einzelnen Computers
- Die Datenmenge, die häufig verwendet wird, überschreitet die Speicherkapazität eines einzelnen Computers. Dadurch werden viele Daten von der Festplatte gelesen, was die Leistung beeinträchtigt.
- IOPS (Input/Output operations Per Second) übersteigt die Servicefähigkeit eines einzelnen MongoDB-Knotens.

3.5 GridFS

MongoDB enthält auch ein eigenes Dateisystem namens GridFS, ähnlich dem Hadoop distributed file system (HDFS)[26], das hauptsächlich zum Speichern von Dateien verwendet wird, die größer sind als die von BSON festgelegte Größe von 16 MB pro Dokument. Aufgrund dieser Ähnlichkeiten kann MongoDB anstelle von Hadoop verwendet werden.

Anstatt eine Datei in einem einzelnen Dokument zu speichern, unterteilt GridFS die Datei in *Chunks* und speichert jedes *Chunk* als separates Dokument. Standardmäßig verwendet GridFS die Standardgröße von 255 KB für *Chunks*. Das heißt, GridFS unterteilt eine Datei mit Ausnahme des letzten *Chunks* in *Chunks* von 255 KB. Das letzte *Chunk* ist nur so groß wie nötig.

Wenn Sie GridFS nach einer Datei abfragen, setzt der Treiber die *Chunks* nach Bedarf wieder zusammen. Sie können Bereichsabfragen für die über GridFS gespeicherten Dateien durchführen. Sie können auch auf Informationen aus beliebigen Abschnitten von Dateien zugreifen, um beispielsweise zur Mitte einer Video- oder Audiodatei zu springen. GridFS eignet sich nicht nur zum Speichern von Dateien mit größer als 16 MB, sondern auch zum Speichern aller Dateien, auf die Sie zugreifen möchten. Und es ist nicht erforderlich, die gesamte Datei in den Speicher zu laden.

In einigen Situationen kann das Speichern großer Dateien in einer MongoDB-Datenbank effizienter sein als in einem System-Level Dateisystem.

- Wenn Ihr Dateisystem die Anzahl der Dateien in einem Verzeichnis begrenzt, können Sie mit GridFS so viele Dateien speichern, wie erforderlich sind.
- Wenn Sie auf Informationen aus Teilen großer Dateien zugreifen möchten, ohne ganze Dateien in den Speicher laden zu müssen, können Sie mithilfe von GridFS Abschnitte von Dateien abrufen.

[26] En.wikipedia.org/wiki/Apache_Hadoop_distributed_file_system

- Wenn Sie möchten, dass Ihre Dateien und Metadaten auf einer Reihe von Systemen und Einrichtungen automatisch synchronisiert und bereitgestellt werden, können Sie GridFS verwenden.
- Wenn Sie geografisch verteilte Replica Set verwenden, kann MongoDB Dateien und deren Metadaten automatisch an eine Reihe von mongod-Instanzen und -Einrichtungen verteilen.

3.6 Geoquery

MongoDB unterstützt Abfrageoperationen für Geodaten. In MongoDB kann man Geodaten als GeoJSON-Objekte[27] oder als *Legacy Coordinate Pairs* abspeichern.

GeoJSON-Objekte

Um die Geometrie über einer erdähnlichen Kugel zu berechnen, werden Positionsdaten zuerst als GeoJSON-Objekt gespeichert. Ein eingebettetes Dokument wird verwendet, um GeoJSON-Daten anzugeben. Darin gibt es ein Feld namens "Typ", das den GeoJSON-Objekttyp festlegt, und ein Feld namens "coordinates", das die Koordinaten des Objekts bestimmt.

Wenn Breiten- und Längengradkoordinaten angegeben werden, muss zuerst den Längengrad und dann den Breitengrad angegeben werden:

Gültige Längengrade liegen zwischen -180 und 180 (beide inklusive).

Gültige Breitengrade liegen zwischen -90 und 90 (beide inklusive).

Außerdem verwendet MongoDB das WGS84-Referenzsystem für räumliche Abfragen von GeoJSON-Objekten. Ein Beispiel ist unten gezeigt:

```
location: {
       type: "Point",
       coordinates: [-73.856077, 40.848447]
}
```

Legacy Coordinate Pairs

Um Entfernungen auf einer euklidischen Ebene zu berechnen, speichern Sie Ihre Positionsdaten als Legacy Coordinate Pairs und verwenden Sie einen 2d-Index. MongoDB unterstützt Berechnung der sphärischen Oberfläche für Legacy Coordinate Pairs über einen 2dsphere-Index, indem die Daten in den GeoJSON-Punkttyp konvertiert werden.

Um Daten als Legacy-Koordinatenpaare anzugeben, können Sie entweder ein Array (bevorzugt) oder ein eingebettetes Dokument verwenden.

[27] https://docs.mongodb.com/manual/reference/geojson/

```
# Angeben über ein Array (bevorzugt):
  <field>: [<longitude>, <latitude> ]
# Angeben über ein eingebettetes Dokument:
  <field>: { <field1>: <longitude>, <field2>: <latitude> }
```

Geospatial Indexes

MongoDB bietet die folgenden geografischen Indextypen zur Unterstützung der geografischen Abfragen.

2dsphere

2dsphere-Indizes unterstützen Abfragen, mit denen Geometrien auf einer erdähnlichen Kugel berechnet werden.

Zum Erstellen eines 2dsphere-Index wird die Methode *db.collection.createIndex ()* benutzt, und wird das Zeichenfolgenliteral "2dsphere" als Indextyp angegeben:

```
db.collection.createIndex( { <location field> : "2dsphere" } )
```

Dabei ist das `<location field>` ein Feld, dessen Wert entweder ein GeoJSON-Objekt oder ein Legacy Coordinate Pairs ist.

2d

2D-Indizes unterstützen Abfragen, die Geometrien auf einer zweidimensionalen Ebene berechnen. Obwohl der Index *$nearSphere*-Abfragen unterstützen kann, die für eine Kugel berechnet werden, kann die Verwendung von 2D-Indizes für diese sphärischen Abfragen zu Fehlern führen (MongoDB, 2019). Wenn es möglich ist, wird 2dsphere-Index für sphärische Abfragen eingesetzt.

Zum Erstellen eines 2D-Index wird die Methode *db.collection.createIndex () angewendet*, und geben Sie das Feld *location* als Schlüssel und das Zeichenfolgenliteral "2d" als Indextyp an:

```
db.collection.createIndex( { <location field> : "2d" } )
```

Dabei ist das `<location field>` ein Feld, dessen Wert ein Legacy Coordinate Pairs ist.

Ein Geodatenindex können nicht als Shard-Schlüssel verwendet werden, wenn eine Collection verteilt wird. Ein Geodatenindex können jedoch für eine sharded-*Collection* erstellt werden, indem ein anderes Feld als Shard-Schlüssel benutzt wird.

Die folgenden geografischen Operationen werden für Sharded Collections unterstützt:

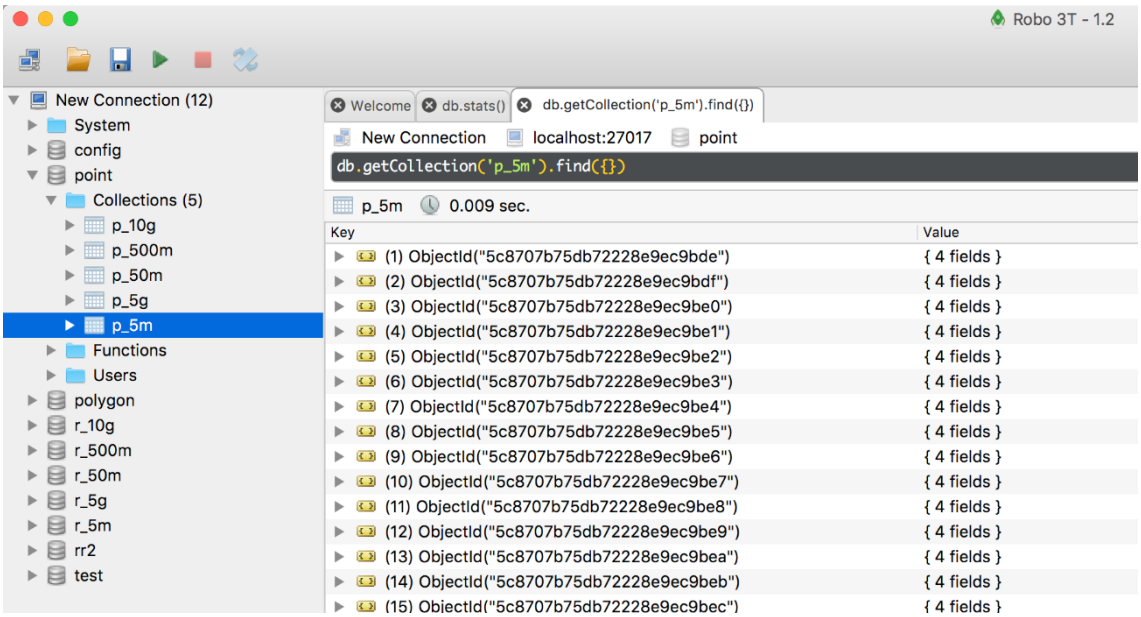
Tabelle 17: Operationen der Geodatenabfragen

Name	Erklärung
$geoIntersects	Wählt Geometrien aus, die sich mit einer GeoJSON-Geometrie überschneiden. Der 2dsphere-Index unterstützt *$geoIntersects*.
$geoWithin	Wählt Geometrien innerhalb einer begrenzenden GeoJSON-Geometrie aus. Die Indizes 2dsphere und 2d unterstützen *$geoWithin*.
$neare	Gibt räumliche Objekte zurück, die sich in der Nähe eines Punkts befinden. Benötigt einen Geodatenindex. Die Indizes 2dsphere und 2d unterstützen *$near*.
$nearSphere	Gibt räumliche Objekte zurück, die sich in der Nähe eines Punkts auf einer Kugel befinden. Ein Geodatenindex ist notwendig. Die Indizes 2dsphere und 2d unterstützen *$nearSphere*.
$box	Gibt ein rechteckiges Feld an, das *legacy coordinate pairs* für *$geoWithin*-Abfragen verwendet. Der 2D-Index unterstützt *$box*.
$center	Gibt einen Kreis an, der bei Verwendung von planarer Geometrie *Legacy Coordinate Pairs* für *$geoWithin*-Abfragen verwendet. Der 2D-Index unterstützt *$center*.
$centerSphere	Gibt einen Kreis an, der entweder *legacy coordinate pairs* oder das GeoJSON-Format für *$geoWithin*-Abfragen verwendet, wenn sphärische Geometrie verwendet wird. Die Indizes 2dsphere und 2d unterstützen *$centerSphere.*
$geometry	Gibt eine Geometrie im GeoJSON-Format für Geodaten-Abfrageoperatoren an.
$maxDistance	Gibt eine maximale Entfernung an, um die Ergebnisse von *$ near*- und *$ nearSphere*-Abfragen zu begrenzen. Die Indizes 2dsphere und 2d unterstützen *$ maxDistance*.
$minDistance	Gibt eine Mindestentfernung an, um die Ergebnisse von *$ near*- und *$ nearSphere*-Abfragen zu begrenzen. Nur zur Verwendung mit 2dsphere index.
$polygon	Gibt ein Polygon für die Verwendung der *legacy coordinate pairs* für *$ geoWithin*-Abfragen an. Der 2D-Index unterstützt *$ center*.
$uniqueDocs	Veraltet Ändert eine *$ geoWithin*- und *$ near*-Abfrage, um sicherzustellen, dass die Abfrage das Dokument einmal zurückgibt, auch wenn ein Dokument mehrmals mit der Abfrage übereinstimmt.

3.7 GUI und APIs

Robo3T ist eine Shell-zentrierte MongoDB-Benutzeroberfläche, die die Plattformen Windows, MacOS und Linux unterstützt. Es ist auch eine der wenigen Benutzeroberflächen, die SSL[28]-Verbindungen zu Ihrem MongoDB-Server unterstützt. Es gibt auch Unterstützung für die Verbindung über einen SSH-Tunnel[29]. Die Abfrageoberfläche zeigt Daten in der Baumansicht, Tabellenansicht und Textansicht an. Sie können Ihre Abfragen auch zur späteren Verwendung speichern. Eines der coolsten Features ist, dass es auch Unterstützung für die Shell bietet, sodass Sie weiterhin die Shell-Befehle verwenden können, mit denen Sie vertraut sind.

Tabelle 18: Benutzeroberfläche von Robo 3T



Es gibt normalerweise drei Möglichkeiten, MongoDB zu verbinden: MongoDB Compass, Mongo Shell und Treiber.

MongoDB Compass ist offizielle GUI für MongoDB. Man kann die Daten visuell verarbeiten und Ad-hoc-Abfragen[30] in Sekunden ausführen. Es unterstützt volle CRUD-Funktionen und ist verfügbar für Linux, Mac oder Windows.

Mongo Shell ist eine Kommandozeilenanwendung, die sich über TCP/IP an die MongoDB anschließt. Sie ist eine Komponente der MongoDB-Distributionen

Treiber bereitstehen für sehr viele Programmiersprachen und kann relativ leicht in der Anwendung eingebunden werden. In der folgenden Liste sind die unterstützten Programmiersprachen aufgeführt.

[28] https://info.ssl.com/article.aspx?id=10241

[29] https://en.wikipedia.org/wiki/Secure_Shell

[30] https://www.techopedia.com/definition/30581/ad-hoc-query-sql-programming

C	C++	C#	Go	Java	Node.js	Perl
PHP	Python	Ruby	Mongoid	Scala	Motor	

Darüber hinaus bietet MongoDB HTTP Interface und REST Interfaces.

REST Interfaces	Eve (Python), RESTHeart (Java), DrowsyDromedary (Ruby), Crest (Node.js), AMID, Kule und DreamFactory
HTTP Interface	Sleepy Mongoose (Python)

4 Ansatz zum Erstellen einer verteilten Datenbank mit MongoDB

In diesem Kapitel wird der Mechanismus zum Erstellen eines hochverfügbaren verteilten MongoDB-Clusters untersucht. Die Details zum Erstellen eines MongoDB-Clusters werden hervorgehoben. Weiterhin wurden die amtlichen Geodaten erfolgreich aus der PostGIS in den neuen Cluster migriert.

4.1 Konzept und Framework

Um einen hochverfügbaren, horizontal skalierbaren Cluster zu erstellen, müssen die Replikatset- und Sharding-Kernfunktionen von MongoDB kombiniert werden. Der Cluster besteht aus drei Servern, und jeder Server hat drei Shards, einen Mongos und einen Konfigurationsserver. Zusätzlich ist jeder Shard mit einem Replica-Set ausgestattet, der einen Primary-Knoten, einen Secondary-Knoten und einen Arbiter-Knoten enthält. Die IP-Adresse und Ports werden in der Abbildung 16 dargestellt.

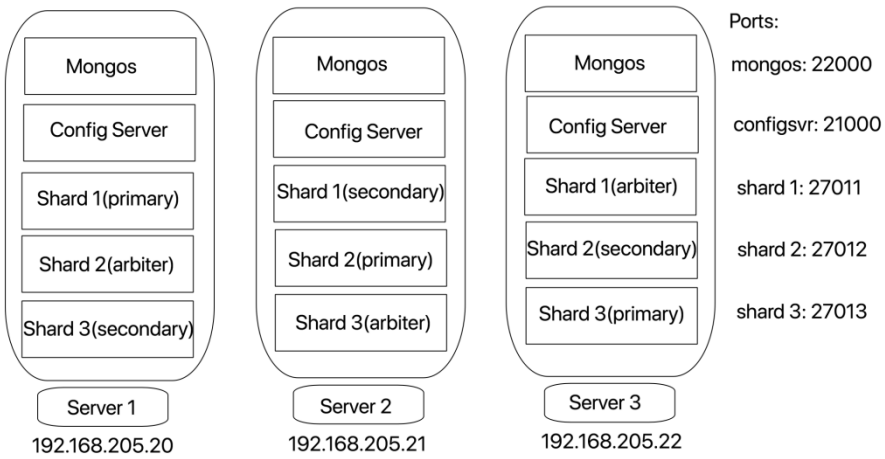


Abbildung 16: Framework des replicated sharded Clusters

Zum Erstellen eines MongoDB-Clusters wurde zunächst eine entsprechende Arbeitsumgebung aufgebaut. Das Folgende ist die experimentelle Umgebung und Konfiguration:

Virtual Machine: VirtualBox + Vagrant

Host OS: MacOS 10.13.6

VM OS: Centos7

Software: Vim + MongoDB 4.0.6

Plugin tools: scp

4.2 Verarbeitung der BRW-Daten

Die “BRW-Daten” steht für Bodenrichtwert-Daten, die von LGLN (Landesamt für Geoinformation und Landesvermessung Niedersachsen) geliefert werden. In diesem Experiment gibt es nur vier Arten der BRW-Daten: Bauland, Landwirtschaft, Punktgeometrien und Verfahren (vgl. Abbildung 17).

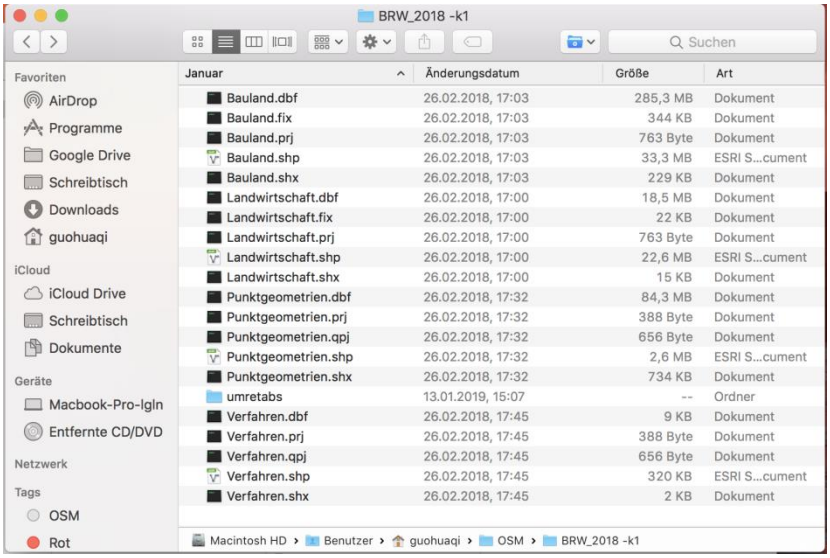


Abbildung 17: Bodenrichtwerte-Daten

In der .shp-Datei werden Geometriedaten gespeichert. Zur Speicherung der Sachdaten dient .dbf-Datei im dBSAE-Format[31]. .shx Datei wird als Index für die Geometrie verwendet, um eine schnelle Vorwärts- und Rückwärtssuche zu ermöglichen und die Attributdaten zu verknüpfen. Die Informationen zum Koordinatensystem werden im WKT-Format (Well-known Text) in der .prj-Datei gespeichert. Eine spezielle Projektionsdatei für QGIS ist .qpj-Datei.

Da *mongoimport*-Tool unterstützt vorerst nur JSON-, GeoJSON-, CSV- und TSV-Dateien, müssen die shape-Dateien in ein für MongoDB verfügbares Datenformat konvertiert werden. In dieser Arbeit wird QGIS verwendet, um die Dateien vom shape-Format in das GeoJSON-Format zu konvertieren.

1) Importieren der Daten in QGIS

Man kann einfach die shape-Dateien in das QGIS-Bedienfeld ziehen.(vgl. Abbildung 18)

[31] https://de.wikipedia.org/wiki/DBASE

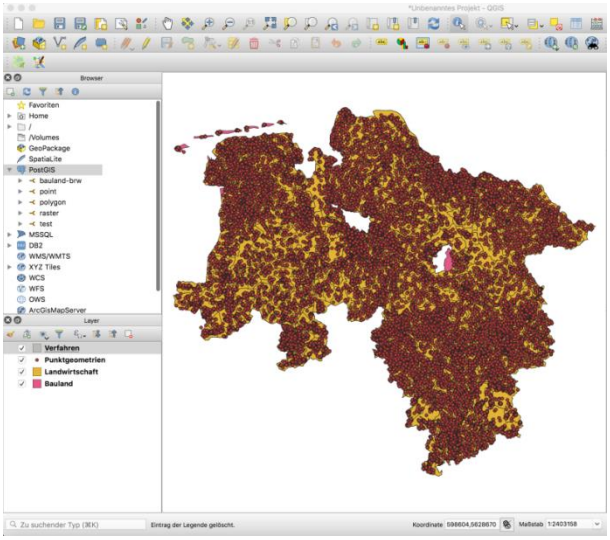


Abbildung 18: Überblick der experimentellen Daten

2) Exportieren der Daten im GeoJSON-Format

Zuerst soll man mit der rechten Maustaste auf den ausgewählten Layer klicken und dann auf die Option “Exportieren” klicken. Danach wird “Objekt speichern als ...” ausgewählt. Anschließend erscheint ein neues Arbeitsfenster (vgl. Abbildung 19). Als Nächstes wird GeoJSON-Format ausgewählt und der Dateiname festgelegt.. Und da MongoDB derzeit nur das WGS 84 Koordinatensystem unterstützt, muss man das ursprüngliche Koordinatensystem anpassen, bevor die Daten exportiert werden. Man kann hiernach zum Schluss auf OK klicken.

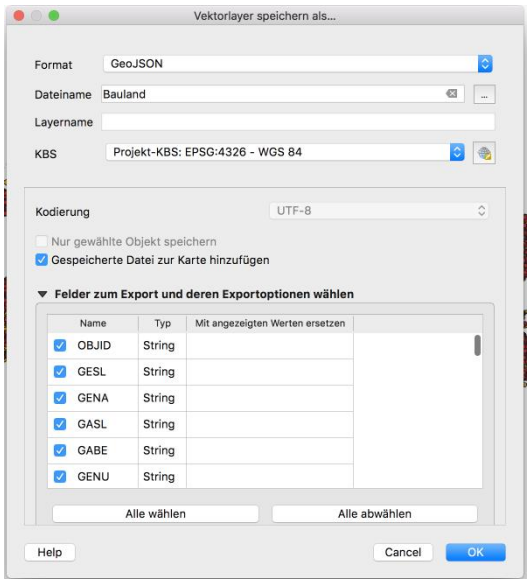


Abbildung 19: Export der Daten im GeoJSON-Format

3) Anpassung der Datenstruktur

Nach dem zweiten Schritt hat man vier GeoJSON-Dateien erhalten (vgl. Abbildung 20). Damit alle geografischen Objekte mittels Batchimport automatisiert in MongoDB abgelegt werden, ist eine geringfügige Änderung der Datenstruktur erforderlich. Entsprechend den Anforderungen von mongoimport-Tool muss die importierte Datei in Form eines reinen Arrays vorliegen. Mit anderen Worten müssen die Beschreibung am Anfang und die geschweiften Klammern gelöscht werden, wie in Abbildung 21 gezeigt.

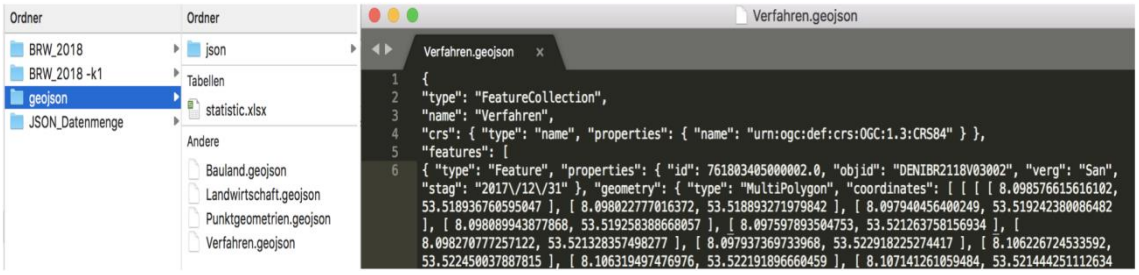


Abbildung 20: Überblick der GeoJSON-Dateien

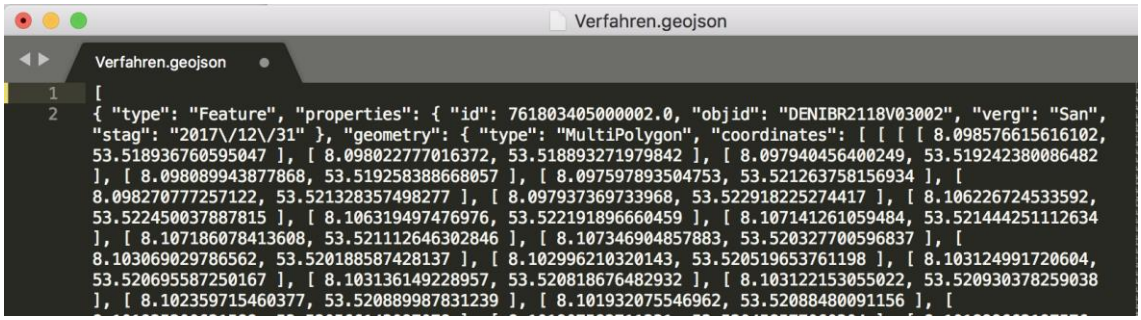


Abbildung 21: verarbeitete GeoJSON-Datei

4.3 Bereitstellung eines replicated sharded Clusters

Im Abschnitt 4.1 wird der Aufbau eines MongoDB-Clusters angegeben. Und er kann generell in fünf schritte unterteilt werden.

- Erstellung der drei virtuellen Maschinen als Clusterknoten
- Bereitstellung eines Replica Set der Konfigurationssever
- Aufsetzung der Replica Sets für Shards
- Konfigurieren der Routing-Server
- Aktivierung des Shardings auf der Datenbank

Vagrant ist ein Tool zum Erstellen und Verwalten von Umgebungen virtueller Maschinen in einem Workflow. Es nutzt eine deklarative Konfigurationsdatei, die alle Softwareanforderungen, Pakete, Betriebssystemkonfigurationen, Benutzer und mehr beschreibt.

Gemäß dem Aufbau des Clusters wurden drei virtuellen Maschinen zunächst mit Vagrant eingerichtet werden. Die konkreten Details werden unten angegeben.

I. Erstellung der drei virtuellen Maschinen als Clusterknoten (server1, server2, server3):

1) Installation von VirtualBox und Vagrant:

```
https://www.virtualbox.org/wiki/Downloads
https://www.vagrantup.com/downloads.html
```

2) Erstellen von drei virtuellen Maschinen mit Vagrantfile:

Herstellung der Dateiverzeichnisse:

```
$ mkdir mongocluster/server1
$ mkdir mongocluster/server2
$ mkdir mongocluster/server3
```

Generierung von Vagrantfile:

```
$ cd ~mongocluster/server1
$ vagrant init
```

Einstellung der Parameter:

```
$ vim vagrantfile
```

Vagrantfile beinhalt Informationen zum Konfigurieren der virtuellen Maschine, z.B. Betriebssystem, Hostname, IP-Adresse, Hauptspeicher und Anzahl der CPUs. Unten ist die Vagrantfile von Server 1:

```
Vagrant.configure("2") do |config|
    config.vm.box = "centos/7"
    config.vm.hostname = "server1"
    config.vm.network "private_network", ip: "192.168.205.20"
    config.vm.provider "virtualbox" do |v|
          v.memory = 4096
          v.cpus = 1
    end
end
```

 # Parameterliste:

```
server1_ip: 192.168.205.20,
server2_ip: 192.168.205.21,
server3_ip: 192.168.205.22,
Betriebssystem: centos7,
Haupspeicher: 4gb,
Anzahl der CPUs: 1
```

Starten der Virtual Machine

```
$ vagrant up
```

Einloggen mit ssh

```
$ vagrant ssh
```

3) Installation von MongoDB

Herunterladen der binären Installationsdatei:

```
https://fastdl.mongodb.org/linux/mongodb-linux-x86_64-
4.0.6.tgz
```

Übertragung der Installationsdatei vom Host auf die virtuelle Maschine mit dem Plug-in-Tool scp

```
# Installation von scp
  $ vagrant plugin install vagrant-scp
# Abfragen der ID der virtuellen Maschine (vgl. Abbildung 22)
  $ vagrant global-status
```

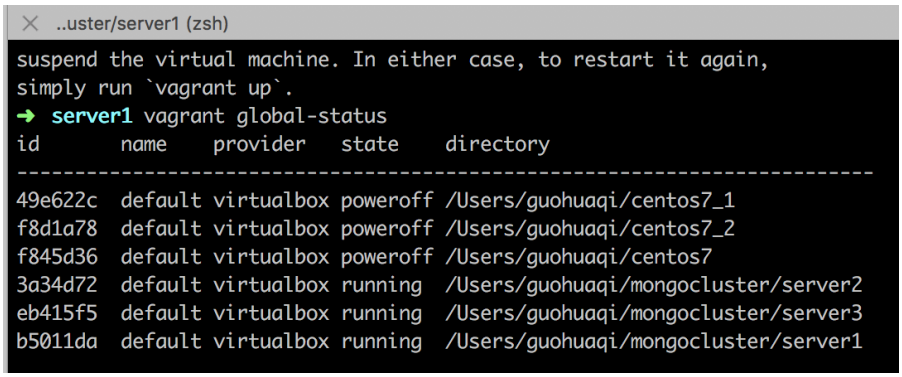


Abbildung 22: Status der virtuellen Maschine

Für den Speicherpfad der Installationsdatei steht „./download“. “id” wird als eindeutige Kennung der virtuellen Maschine bezeichnet. „/home/vagrant/local“ ist der Empfangspfad der virtuellen Maschine.

```
$ vagrant scp ./download id:/home/vagrant/local
# Beispiel (Server 1)
$ vagrant scp ~/mongocluster/download b5011da:/home/vagrant
```

Anschließend wird die Archivdatei entpackt, und der Name der dekomprimierten Datei wird umbenannt:

```
$ tar -xzvf mongodb-linux-x86_64-4.0.6.tgz -C /home/vagrant
$ mv mongodb-linux-x86_64-4.0.6.tgz mongodb
```

4) Erzeugung der Speicherverzeichnisse von Daten und Protokollen nach dem Entwurfsplan (siehe Abbildung 16 **)**

```
    # Config Sever
      mkdir -p /home/vagrant/mongodb/conf
      mkdir -p /home/vagrant/mongodb/config/log
      mkdir -p /home/vagrant/mongodb/config/data
    # mongos
      mkdir -p /home/vagrant/mongodb/mongos/log
    # Shard 1
      mkdir -p /home/vagrant/mongodb/shard1/log
      mkdir -p /home/vagrant/mongodb/shard1/data
    # Shard 2
      mkdir -p /home/vagrant/mongodb/shard2/log
      mkdir -p /home/vagrant/mongodb/shard2/data
    # Shard 3
      mkdir -p /home/vagrant/mongodb/shard3/log
      mkdir -p /home/vagrant/mongodb/shard3/data
```

Abbildung 23 Anzeigen des Speicherverzeichnisses von Daten und Protokollen.

Abbildung 23: Speicherverzeichnisse von Daten und Protokollen

5) Einrichtung der MongoDB-Umgebungsvariablen

```
    $ vim /etc/profile       # Hinzufügung des folgenden Inhalts
      export MONGODB_HOME=/home/vagrant/mongodb
      export PATH=$MONGODB_HOME/bin:/$PATH
  #Die Änderungen werden sofort wirksam.
    $ source /etc/profile
```

6) Schließung von selinux, firewall und iptables (Sonst kann es zu einem unbekannten Fehler kommen)

```
# close selinux
   $ vim /etc/selinux/config    #Öffnung der Konfigurationsdatei
    SELINUX = disabled            # Änderung des Parameters
# close firewall
  $ systemctl stop firewalld.service
# close iptables
  $ systemctl stop iptables.service
```

II. Einsetzung eines Replica Set für den Konfigurationsserver:

Dies dient zur Verbesserung der Verfügbarkeit des Clusters.

1) Herstellung einer Konfigurationsdatei nacheinander für drei Server:

```
$ vim /home/vagrant/mongodb/conf/config.conf
    # Konfigurationsinhalt
      pidfilepath = /home/vagrant/mongodb/config/log/configsvr.pid
      dbpath = /home/vagrant/mongodb/config/data
      logpath = /home/vagrant/mongodb/config/log/configsvr.log
      logappend = true
      bind_ip = 0.0.0.0
      port = 21000
      fork = true
    # Deklaration: Dies ist ein Konfigurationsserver.
      configsvr = true
    # Benennung des ReplicaSet
      replSet = configs
    # Maximale Anzahl von Verbindungen
      maxConns = 20000
```

2) Starten Sie den Konfigurationsserver der drei Server separat

```
$ ./mongod -f /home/vagrant/mongodb/conf/config.conf
```

```
[vagrant@server1 bin]$ ./mongod -f /home/vagrant/mongodb/c
onf/config.conf
about to fork child process, waiting until server is ready
 for connections.
forked process: 5280
child process started successfully, parent exiting
[vagrant@server1 bin]$
```

Abbildung 24: Starten der Konfigurationsserver

3) Melden Sie sich beim Konfigurationsserver an, von dem erwartet wird, dass er der primäre Knoten des Replica Set ist.

```
$ mongo --port 21000
```

4) Bekanntmachen der Mitglieder des Replica Set

```
> config = {
     _id : "configs",
    members : [
    {_id : 0, host : "192.168.205.20:21000" },
    {_id : 1, host : "192.168.205.21:21000" },
    {_id : 2, host : "192.168.205.22:21000" }
    ]
}
```

"_id" sollte mit dem in der Konfigurationsdatei konfigurierten replSetName übereinstimmen, und "host" ist die IP und der Port der drei Knoten.

5) Initialisierung des Replica Sets

```
> rs.initiate(config)
{
  "ok" : 1,
  "operationTime" : Timestamp(1550543026, 1),
  "$gleStats" : {
    "lastOpTime" : Timestamp(1550543026, 1),
    "electionId" : ObjectId("000000000000000000000000")
  },
  "lastCommittedOpTime" : Timestamp(0, 0),
  "$clusterTime" : {
    "clusterTime" : Timestamp(1550543026, 1),
    "signature" : {
      "hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
      "keyId" : NumberLong(0)
    }
  }
}
```

6) Abfrage-Status

```
> rs.status()
```

Der Status des Replica Set kann mit rs.status() abgefragt werden.

III. Aufsetzung der Replica Sets für Shards

A. Anwendung eines Replica Set für Shard 1

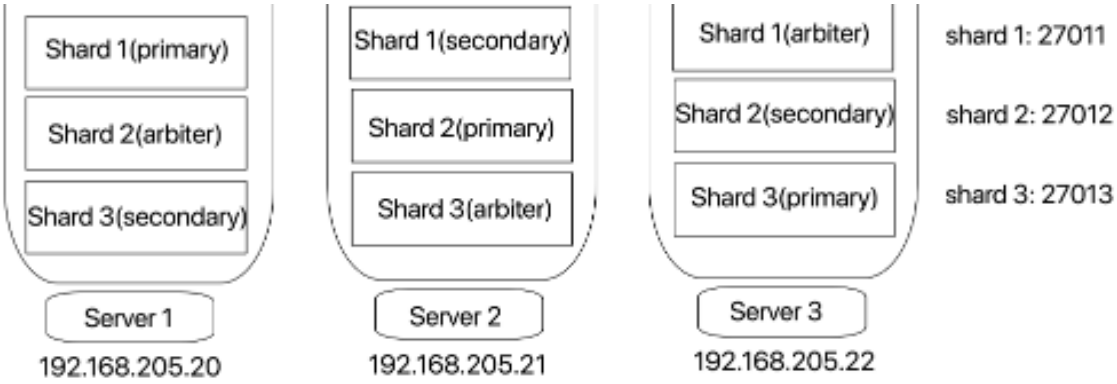


Abbildung 25: Parameter von Shards und Servern

1) Schreiben einer Konfigurationsdateie für jeden Server：

```
$ vim /home/vagrant/mongodb/conf/shard1.conf
#configuration concent
pidfilepath = /home/vagrant/mongodb/shard1/log/shard1.pid
```

```
dbpath = /home/vagrant/mongodb/shard1/data
logpath = /home/vagrant/mongodb/shard1/log/shard1.log
logappend = true
bind_ip = 0.0.0.0
port = 27011
fork = true
# Name des Replica Set
replSet = shard1
# Deklaration: Dies ist ein Shard Server
shardsvr = true
# Maximale Anzahl von Verbindungen
maxConns = 20000
```

2) Starten Sie den shard1-Server der drei Server separat

```
$ ./mongod -f /home/vagrant/mongodb/conf/shard1.conf
```

3) Melden Sie sich gemäß dem Entwurf in Abbildung 5 bei Server 1 an und initialisieren Sie das Replica Set.

```
$ ./mongo --port 27011
```

4) Wechseln Sie in die Admin-Datenbank:

```
> use admin
```

5) Identifizieren Sie die Mitglieder des Replica Set von Shard 1.

```
> config = {
    _id : "shard1",
     members : [
         {_id : 0, host : "192.168.205.20:27011" , priority:
3 },
         {_id : 1, host : "192.168.205.21:27011" , priority:
2},
         {_id : 2, host : "192.168.205.22:27011" , arbi-
terOnly: true }
     ]
 }
```

6) Initialisierung

```
> rs.initiate(config)
```

7) Wenn Folgendes zurückgegeben wird, bedeutet dies Erfolg.

```
> rs.initiate(config)
{ "ok" : 1 }
```

B. Anwendung eines Replica Set für Shard 2

Dieser Prozess ähnelt dem von Shard 1.

C. Anwendung eines Replica Set für Shard 1

Dieser Prozess ähnelt dem von Shard 1.

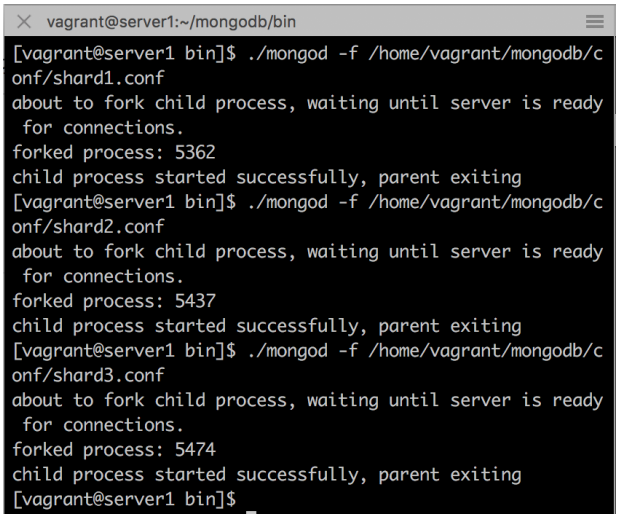


Abbildung 26: Starten der Shard-Servern

Abbildung 26 zeigt, dass die Shard-Servern reibungslos beginnt.

IV. Konfigurieren der Routing-Server (mongos):

1. Schreiben einer Konfigurationsdateie für jeden Server:

```
$ vim /home/vagrant/mongodb/conf/mongos.conf
  # Konfigurationsinhalt
  pidfilepath = /home/vagrant/mongodb/mongos/log/mongos.pid
  logpath = /home/vagrant/mongodb/mongos/log/mongos.log
  logappend = true
  bind_ip = 0.0.0.0
  port = 22000
  fork = true
  # Verbindung mit Konfigurationsservern
  configdb =      con-
  figs/192.168.205.20:21000,192.168.205.21:21000,192.168.205.22
  :21000
  # Maximale Anzahl von Verbindungen
  maxConns = 2000
```

2. Starten der Mongos von drei Servern

```
> ./mongos -f /home/vagrant/mongodb/conf/mongos.conf
```

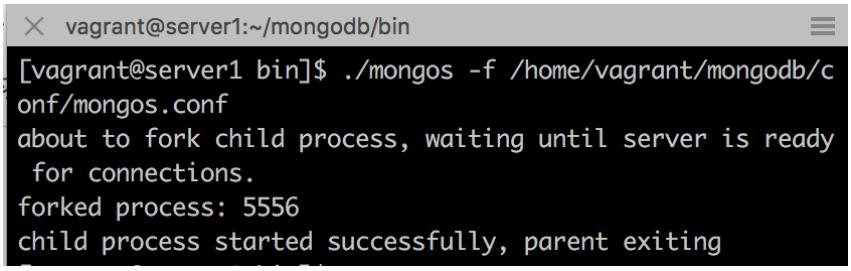


Abbildung 27: Starten des Mongos-Diensts (Server 1)

V. Aktivierung des Shardings auf der Datenbank:

Derzeit sind die Konfigurationsserver, die Routingserver und verschiedene Shard-Server eingerichtet. Die Anwendung kann das Sharding jedoch nicht verwenden, und Sie müssen die Konfiguration im Cluster festlegen, damit das Sharding wirksam wird.

1. **Anmeldung bei einem der Mongos:**

```
$ ./mongo --port 22000
```

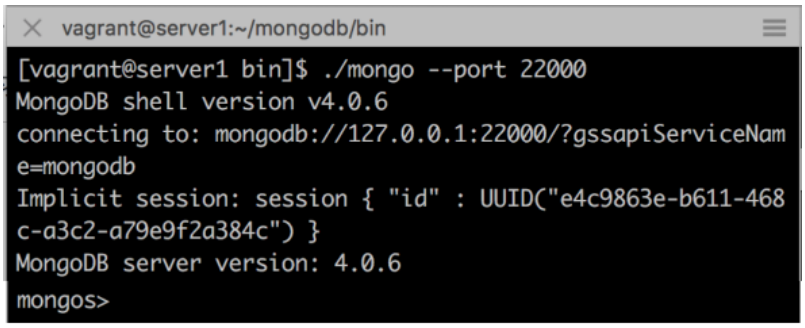


Abbildung 28: Anmeldung bei Router (Server 1)

2. **Wechseln Sie in die Admin-Datenbank**

```
> use admin
```

3. **Hinzufügung der Shard-Knoten dem Cluster:**

```
> sh.addShard ( "shard1/192.168.205.20:27011,
192.168.205.21:27011, 192.168.205.22:27011");
> sh.addShard ( "shard2/192.168.205.20:27012,
192.168.205.21:27012, 192.168.205.22:27012");
> sh.addShard ( "shard3/192.168.205.20:27013,
192.168.205.21:27013, 192.168.205.22:27013");
```

4. **Überblick des Clusters:**

```
> sh.status()
--- Sharding Status ---
  sharding version: {
    "_id" : 1,
    "minCompatibleVersion" : 5,
    "currentVersion" : 6,
    "clusterId" : ObjectId("5c6b68be0d36f556210f4a9b")
```

```
  }
  shards:
        {  "_id" : "shard1",  "host" :
"shard1/192.168.205.20:27011,192.168.205.21:27011",
"state" : 1 }
        {  "_id" : "shard2",  "host" :
"shard2/192.168.205.21:27012,192.168.205.22:27012",
"state" : 1 }
        {  "_id" : "shard3",  "host" :
"shard3/192.168.205.20:27013,192.168.205.22:27013",
"state" : 1 }
  active mongoses:
        "4.0.6" : 3
  autosplit:
        Currently enabled: yes
  balancer:
        Currently enabled:  yes
        Currently running:  no
        Failed balancer rounds in last 5 attempts:  0
        Migration Results for the last 24 hours:
              No recent migrations
  databases:
        {  "_id" : "config",  "primary" : "config",  "parti-
tioned" : true}
```

4.4 Einspeicherung und Sharding der BRW-Daten

Zuerst wird die BRW-Daten vom lokalen Mac auf virtuelle Maschine kopieren.

```
$ vagrant scp ~/OSM/geojson b5011da:/home/vagrant
```

Stellen Sie sicher, dass der Sharded Cluster funktioniert:

1) Anmeldung bei einem der Mongos:

```
$ ./mongo --port 22000
```

2) Importieren der BRW-Daten und Test-Daten:

```
$ ./mongoimport --db ** -c ** -file " ** " --jsonArray
#Importieren der  Test-Daten (BRW-Daten sind zu klein, um das
Sharding auszulösen)
$ ./mongo --port 22000
> for(var i=0;i<100000;i++){ db.users.insert({"name" :
"user"+i , "created_at" : new Date()}); }
```

3) Wechseln Sie in die Admin-Datenbank:

```
> use admin
```

4) Aktivierung des Sharding auf der Datenbank *test*:

```
> db.runCommand( { enablesharding : "test"});
```

5) Indizieren des Shard-Schlüssels:

```
> db.users.ensureIndex({"name" : 1})
```

6) Angeben der Collection und der Schlüsselkombination (sogenannte Sharding-Keys) für das Sharding :

```
> db.runCommand( { shardcollection : "test.users"
                        ,key : {"name": 1} } );
```

7) Überblick der Chunks:

```
# Anmeldung bei der admin Datenbank
> use admin
> sh.status( )
```

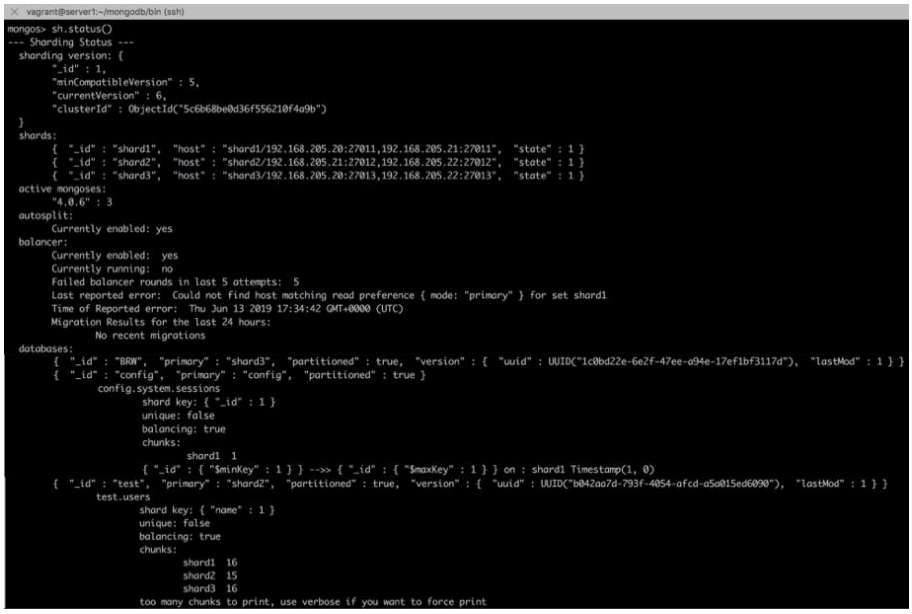


Abbildung 29: Status des Clusters nach dem Sharding

Nach dem Sharding werden Shard1 und Shard3 jeweils 16 *Chunks* verteilt, und Shard2 bekamm 15 *Chunks*.

4.5 Implementierung mit Docker

Eine andere Möglichkeit zum Erstellen eines Clusters ist die Verwendung des Dockers. Docker ist ein Tool, das das Erstellen, Bereitstellen und Ausführen von Anwendungen mithilfe von Containern vereinfacht. Damit kann ein Entwickler eine Anwendung mit allen benötigten Teilen wie Bibliotheken und anderen Abhängigkeiten packen und als Paket ausliefern. Auf diese Weise können Entwicklern dank des Containers sicher sein,

dass die Anwendung auf jedem anderen Linux-Computer ausgeführt werden kann und unabhängig von den benutzerdefinierten Computereinstellungen ist.

In gewisser Weise ist Docker ein bisschen wie eine virtuelle Maschine. Im Gegensatz zu virtuellen Maschinen erstellt Docker jedoch nicht das ganze virtuelle Betriebssystem, sondern ermöglicht es Anwendungen, denselben Linux-Kernel wie das System zu verwenden, auf dem sie ausgeführt werden (vgl.Abbildung 30). Und die Anwendungen lediglich etwas bereitstellen, das nicht bereits auf dem Host-Computer ausgeführt wird. Dies kann die Leistung erheblich verbessern und die Größe der Anwendung verringern. Darüber hinaus ist Docker Open Source. Dies bedeutet, dass jeder einen Beitrag zu Docker leisten und es erweitern kann, um seine Anforderungen zu erfüllen.

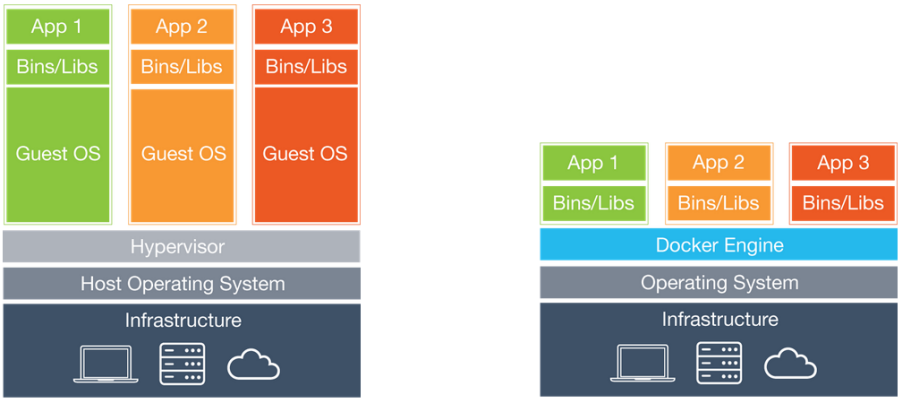


Abbildung 30: Virtuelle Maschine vs. Docker[32]

Da die Ergebnisse dieser Methode noch nicht perfekt ist, ist sie keine wichtige Einführung, kann aber als Referenz für nachfolgende Forscher verwendet werden. Der Erstellungsprozess besteht im Wesentlichen aus drei Schritten:

- Erzeugung 3 Virtual Machines mit integriertem Docker
- Definieren der Dienste in einem Compose File
- Bereitstellung des Clusters durch Ausführung des Compose Files

Im Vergleich zum Abschnitt 4.1 hat sich die Gesamtarchitektur des Mongo-Clusters kaum geändert, außer dass die Servernamen, die Ports und die IP-Adressen geändert wurden (vgl. Abbildung 31).

[32] https://jaxenter.de/docker-tools-vergleich-39670

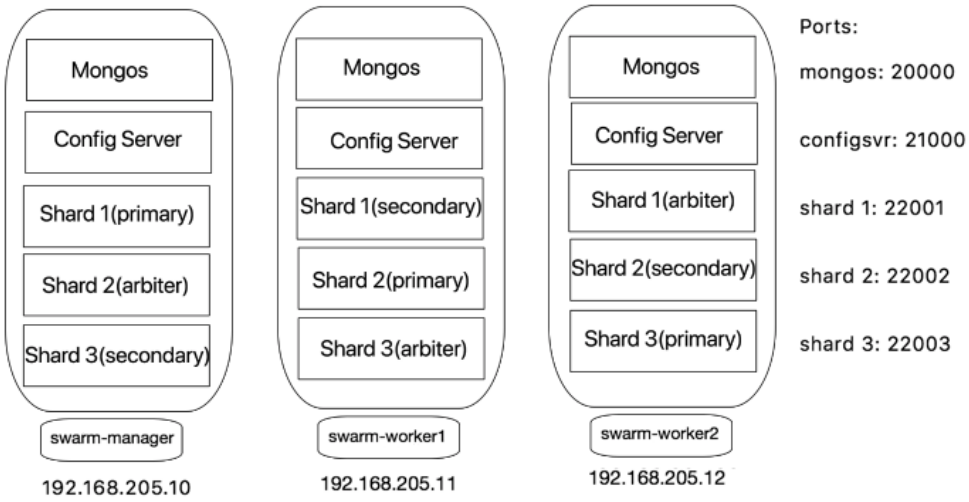


Abbildung 31: Framework des MongoDB-Clusters (Docker)

Arbeitsumgebung und Ausrüstung:

Virtual Machine: VirtualBox + Vagrant

Host OS: MacOS 10.13.6

VM OS: Centos7

Software: Vim + Docker

Plugin tools: scp

Erzeugung 3 Virtual Machines mit integriertem Docker

Der erste Schritt besteht darin, mithilfe von Vagrant drei virtuelle Maschinen mit integriertem Docker herzustellen. Und wurden sie als swarm-manager, swarm-worker1, swarm-worker2 benennt. Im Folgenden wird Swarm-Manager-Knoten als Beispiel verwendet.

```
$ mkdir centos7
$ cd centos7
$ vagrant init         # initialisieren
$ vim vagrantfile      # Bearbeitung des Vagrantfile mit vim
```

Vagrantfile beinhalt Informationen zum Konfigurieren der virtuellen Maschine, z.B. Betriebssystem, Hostname, IP-Adresse, Hauptspeicher, Anzahl der CPUs und Installationsbefehlszeile von Docker. Unten ist die Vagrantfile von Swarm Manager:

```
Vagrant.configure("2") do |config|
config.vm.box = "centos/7"
config.vm.hostname = "swarm-manager"
config.vm.network "private_network", ip: "192.168.205.10"
config.vm.provider "virtualbox" do |v|
v.memory = 6144
v.cpus = 1
```

```
End
# Installationsbefehlszeile von Docker
config.vm.provision "shell", inline: <<-SHELL
 # Löschung der alten Version
    sudo yum remove docker docker-client docker-client-latest
docker-common docker-latest docker-latest-logrotate docker-
logrotate docker-engine
 # Installierung der erforderlichen Pakete
    sudo yum install -y yum-utils device-mapper-persistent-data
lvm2
 # Konfiguration eines stabilen Repositorys
    sudo yum-config-manager --add-repo https://download.do-
cker.com/linux/centos/docker-ce.repo
 # Installierung und Starten von Docker
    sudo yum install -y docker-ce docker-ce-cli containerd.io
    sudo systemctl start docker
   SHELL
End
```

Danach kann man die virtuellen Maschinen starten und sich mit ssh einloggen. Weiterhin wird der Docker-Service gestartet.

```
    $ vagrant up    # Starten der virtuellen Maschine
    $ vagrant ssh    # Einloggen
  # users: root; default password: vagrant (Authentifikation)
    $ service docker start
```

Erzeugung der Speicherverzeichnisse von Daten und Protokollen

```
  # Config Sever
    mkdir -p /home/vagrant/local/mongodb/config/log
    mkdir -p /home/vagrant/local/mongodb/config/data
  # mongos
    mkdir -p /home/vagrant/local/mongodb/mongos/log
  # Shard 1
    mkdir -p /home/vagrant/local/mongodb/shard1/log
    mkdir -p /home/vagrant/local/mongodb/shard1/data
  # Shard 2
    mkdir -p /home/vagrant/local/mongodb/shard2/log
    mkdir -p /home/vagrant/local/mongodb/shard2/data
  # Shard 3
    mkdir -p /home/vagrant/local/mongodb/shard3/log
    mkdir -p /home/vagrant/local/mongodb/shard3/data
```

Danach muss ein Swarm Cluster hergestellt werden und ein Compose-File geschrieben werden. Docker Swarm ist ein Clustering- und Planungstool für Docker-Container. Mit

Swarm können IT-Administratoren und -Entwickler einen Cluster von Docker-Knoten als ein einheitliches virtuelles System einrichten und verwalten.

Compose ist ein Tool zum Definieren und Ausführen von Docker-Anwendungen mit mehreren Containern (Docker, 2019). Mit Compose wird eine YAML-Datei verwendet, um die Dienste einer Anwendung zu konfigurieren. Anschließend kann man mit einem einzelnen Befehl alle Dienste aus Ihrer Konfiguration erstellen und starten.

Ein Stack ist eine Sammlung von Diensten, aus denen eine Anwendung in einer bestimmten Umgebung besteht (Docker, 2019). Ein Stack File ist eine Datei im YAML-Format, die einen oder mehrere Dienste definiert, ähnlich einer Datei docker-compose.yml für Docker Compose, jedoch mit einigen Erweiterungen.

Wenn Docker Engine im Swarm-Mode ausgeführt wird, kann man mit d*ocker stack deploy* eine vollständige Anwendung für den Swarm bereitstellen.

```
# Herstellung eines Swarm Clusters
  $ docker swarm init --advertise-addr 192.168.205.10
```

```
[vagrant@swarm-manager centos7]$ docker swarm init --advertise-addr 192.168.205.10
Swarm initialized: current node (ntyvpbtc293fmt58uf2yzovew) is now a manager.

To add a worker to this swarm, run the following command:

    docker swarm join --token SWMTKN-1-1dtnocln846r8jtfdcf5w19ypa3mm98zl7qsruicv26h6a7mav-aie7f2q
f1bvqek2wktys3x4i5 192.168.205.10:2377

To add a manager to this swarm, run 'docker swarm join-token manager' and follow the instructions
.
```

Abbildung 32: Initialisierung eines Swarm Clusters

```
# Hinzufügen der Arbeitsknoten für den Swarm-Cluster mit denm
oben gezeigten Befehl
  $ docker swarm join --token SWMTKN-1-1dtno-
  cln846r8jtfdcf5w19ypa3mm98zl7qsruicv26h6a7mav-
  aie7f2qf1bvqek2wktys3x4i5 192.168.205.10:2377
```

```
[vagrant@swarm-manager centos7]$ docker node ls
ID                            HOSTNAME            STATUS              AVAILABILITY        MANAGER STATUS      ENGINE VERSION
ntyvpbtc293fmt58uf2yzovew *   swarm-manager       Ready               Active              Leader              18.09.2
bhieurysb38d4n2i4dh9jd8dq     swarm-worker1       Ready               Active                                  18.09.2
rrsg61kia982vflg8wgvg84uf     swarm-worker2       Ready               Active                                  18.09.2
```

Abbildung 33: Status des Swarm Clusters

Außerdem wird ein Overlay-Netzwerk erzeugt und als mongo benannt. Der Overlay-Netzwerktreiber kann ein verteiltes Netzwerk zwischen mehreren Docker-Daemon-Hosts erstellen. Dieses Netzwerk befindet sich über (Overlays) den host-spezifischen Netzwerken, sodass die mit ihm verbundenen Container (einschließlich Swarm-Service-Container) sicher kommunizieren können.

```
$ docker network create -d overlay mongo
```

```
[vagrant@swarm-manager centos7]$ docker network ls
NETWORK ID          NAME                DRIVER              SCOPE
124c5ec54a5f        bridge              bridge              local
172f1bf6b9de        docker_gwbridge     bridge              local
9949affed5ba        host                host                local
5r4vff5zu3ls        ingress             overlay             swarm
omr0gj8c8tnj        mongo               overlay             swarm
```

Abbildung 34: Überblick der Netzwerke

Die Syntaxregeln für das Verfassen von Compose File kann man im offiziellen Leitfaden[33] finden. Es gibt ein Stack-File für jeden Swarm-Knoten, oder können drei Dateien in einer Datei kombiniert werden. Das Folgende ist das Stack-File des Swarm-Manager-Knotens (stack1.yml):

```
  # Use root/example as user/password credentials
  version: '3.7'
  services:
    # Konfiguration von config server
    configsvr1:
      image: mongo
      ports:
        - 27019:21000
      volumes:
        - config_data:/data/configdb
      networks:
        - mongo
      command: mongod --noauth --configsvr --replSet csrs --dbpath
/data/db
    # Konfiguration von mongos
    mongos1:
      image: mongo
      ports:
        - 27015:20000
      networks:
        - mongo
      command: mongos --noauth --configdb csrs/configsvr1:21000,con-
figsvr2:21000,configsvr3:21000
      deploy:
```

[33] https://docs.docker.com/compose/compose-file/

```
        restart_policy:
          condition: on-failure
          delay: 3s
    # Konfiguration von shard1
    shard1_primary:
      image: mongo
      environment:
        MONGO_INITDB_ROOT_USERNAME: root
        MONGO_INITDB_ROOT_PASSWORD: pwd123
      ports:
        - 27022:22001
      networks:
        - mongo
      volumes:
        - mongo_data:/data/db
      command: mongod --dbpath /data/db --shardsvr --replSet shard1
      deploy:
        restart_policy:
          condition: on-failure
          delay: 3s
      depends_on:
        - configsvr1
   # Konfiguration von shard2
    shard2_arbiter:
      image: mongo
      networks:
        - mongo
      command: mongod --noauth --dbpath /data/db --shardsvr --replSet
shard2
      depends_on:
        - configsvr1
    # Konfiguration von shard3
    shard3_secondary:
      image: mongo
```

```
    environment:
      MONGO_INITDB_ROOT_USERNAME: root
      MONGO_INITDB_ROOT_PASSWORD: pwd123
    ports:
      - 27028:22003
    networks:
      - mongo
    volumes:
      - mongo_data:/data/db
    command: mongod --dbpath /data/db --shardsvr --replSet shard3
    deploy:
      restart_policy:
        condition: on-failure
        delay: 3s
    depends_on:
      - configsvr1
# Erstellung eines Visualisierungstools
  visualizer:
    image: dockersamples/visualizer:stable
    ports:
      - "8080:8080"
    stop_grace_period: 1m30s
    volumes:
      - "/var/run/docker.sock:/var/run/docker.sock"
    deploy:
      placement:
        constraints: [node.role == manager]
# Ladepfad für Datenvolumen
volumes:
  mongo_data:
  config_data:
# Kommunikationsnetzwerk
networks:
  mongo:
```

```
    driver: overlay
```

Bevor das erste Stack-File ausgeführt wird, wird das MongoDB-Image am besten im Voraus aus dem Docker-Repository geholt, um den Prozesse zu beschleunigen. Anschließend werden die anderen zwei Stack-Files ausgeführt. Und kann man den Clusterstatus mit dem Visualisierungstool *visualizer* überwachen.

```
# Ziehen des mongo Images
  $ docker pull mongo
# Ausführen der Stack-Datei
  $ docker stack deploy -c stack1.yml mongo
  $ docker stack deploy -c stack2.yml mongo
  $ docker stack deploy -c stack3.yml mongo
# Anzeigen der Serviceliste
  $ docker service ls
```

```
[vagrant@swarm-manager centos7]$ docker service ls
ID                  NAME                    MODE                REPLICAS            IMAGE                             PORTS
qtbqpjx9pwba        mongo_configsvr1        replicated          1/1                 mongo:latest                      *:27019->21000/tcp
cqcmevowo7qr        mongo_configsvr2        replicated          1/1                 mongo:latest                      *:27020->21000/tcp
v173epy9r010        mongo_configsvr3        replicated          1/1                 mongo:latest                      *:27021->21000/tcp
251om5u4wm88        mongo_mongos1           replicated          1/1                 mongo:latest                      *:27015->20000/tcp
mvj0f90f1ybf        mongo_mongos2           replicated          1/1                 mongo:latest                      *:27016->20000/tcp
ufd0nwugz57r        mongo_mongos3           replicated          1/1                 mongo:latest                      *:27018->20000/tcp
lowj57yoy7d0        mongo_shard1_arbiter    replicated          1/1                 mongo:latest
irctreicr9r5        mongo_shard1_primary    replicated          1/1                 mongo:latest                      *:27022->22001/tcp
zmmsc5jij6ba        mongo_shard1_secondary  replicated          0/1                 mongo:latest                      *:27023->22001/tcp
kd6q73cvqql4        mongo_shard2_arbiter    replicated          1/1                 mongo:latest
nlktfohkf0ir        mongo_shard2_primary    replicated          0/1                 mongo:latest                      *:27026->22002/tcp
ibbyeaio45x4        mongo_shard2_secondary  replicated          0/1                 mongo:latest                      *:27027->22002/tcp
e79jaiby8ih6        mongo_shard3_arbiter    replicated          1/1                 mongo:latest
4ml8bupzkc0i        mongo_shard3_primary    replicated          1/1                 mongo:latest                      *:27030->22003/tcp
mf3ceqs1qvuS        mongo_shard3_secondary  replicated          1/1                 mongo:latest                      *:27028->22003/tcp
v89wuks6cq98        mongo_visualizer        replicated          1/1                 dockersamples/visualizer:stable   *:8080->8080/tcp
```

Abbildung 35: Serviceliste

Nach der Bereitstellung des Clusters kann man zum Browser gehen und die IP-Adresse des Manager-Knotens mit Port 8080 in die Adressleiste des Browsers eingeben, um das Cluster zu überwachen. (http://192.168.205.10:8080/)

Jeder Knoten im Swarm zeigt alle auf ihm ausgeführten Containers an. Wenn ein Dienst ausfällt, wird er entfernt. Wenn ein Knoten ausfällt, wird er nicht ausfallen. Stattdessen wird der Kreis oben rot, um anzuzeigen, dass er ausgefallen ist.

4.6 Fazit

Durch dieses Experiment habe ich ein tieferes Verständnis für den Funktionsmechanismus von hochverfügbaren, verteilten und horizontal erweiterbaren Clustern und gleichzeitig gelernt, wie man Docker verwendet und die Bereitstellung eines Clusters mit dem Swarmmodus automatisiert.

Wie andere NoSQL-Datenbanken benötigt MongoDB keine vordefinierten Schemas und speichert alle Arten von Daten. Dies gibt Benutzern die Flexibilität, eine beliebige Anzahl von Feldern in einem Dokument zu erstellen, wodurch die Skalierung von MongoDB-Datenbanken im Vergleich zu relationalen Datenbanken vereinfacht wird. Einer der Vorteile der Verwendung von Dokumenten besteht darin, dass diese Objekte native Datentypen in verschiedenen Programmiersprachen zugeordnet werden. Durch eingebettete Dokumente werden außerdem weniger Datenbankverknüpfungen erforderlich, wodurch die Kosten gesenkt werden können.

Eine Kernfunktion von MongoDB ist die horizontale Skalierbarkeit, die es zu einer nützlichen Datenbank für Unternehmen macht, die Big-Data-Anwendungen ausführen. Darüber hinaus ermöglicht das Sharding der Datenbank die Verteilung von Daten über ein Cluster von Computern. Neuere Versionen von MongoDB unterstützen auch die Erstellung von Datenzonen basierend auf einem Shard-Schlüssel.

MongoDB unterstützt eine Reihe von Speicher-Engines und bietet steckbare Speicher-Engine-APIs, mit denen Dritte ihre eigenen Speicher-Engines für MongoDB entwickeln können (MongoDB, MongoDB Manual, 2019). Das DBMS verfügt auch über integrierte Aggregationsfunktionen, mit denen Benutzer MapReduce-Code direkt in der Datenbank ausführen können, anstatt MapReduce unter Hadoop auszuführen. MongoDB enthält auch ein eigenes Dateisystem namens GridFS, ähnlich dem Hadoop Distributed File System (HDFS), das hauptsächlich zum Speichern von Dateien verwendet wird, die größer sind als die von BSON festgelegte Größe von 16 MB pro Dokument. Aufgrund dieser Ähnlichkeiten kann MongoDB anstelle von Hadoop verwendet werden, obwohl die Datenbanksoftware in Hadoop, Spark und andere Datenverarbeitungs-Frameworks integriert ist.

Obwohl es viele Vorteile gibt, hat MongoDB einige Nachteile. Mit seiner automatischen Failover-Strategie richtet ein Benutzer nur einen Masterknoten in einem MongoDB-Cluster ein. Wenn der Master ausfällt, wird ein Slave-Knoten automatisch in den neuen Master konvertiert. Dies garantiert Kontinuität, ist jedoch nicht sofort verfügbar - es kann eine Minute dauern. Zum Vergleich: Die Cassandra NoSQL-Datenbank unterstützt mehrere Masterknoten. Wenn ein Master ausfällt, steht ein anderer für eine hochverfügbare Datenbankinfrastruktur bereit.

Der einzelne Masterknoten von MongoDB begrenzt die Schreibgeschwindigkeit der Daten, um die reibunslose Synchronisation des Secondary-Knotens zu gewährleisten. Datenschreibvorgänge müssen auf dem Master aufgezeichnet werden, und das Schreiben neuer Informationen in die Datenbank ist durch die Kapazität dieses Master-Knotens begrenzt. Ein weiteres mögliches Problem besteht darin, dass MongoDB durch die Verwendung von Fremdschlüsseleinschränkungen keine vollständige referenzielle Integrität bietet, was die Datenkonsistenz beeinträchtigen könnte.

5 Vergleich von MongoDB und PostGIS

In diesem Kapitel werden die Leistungsunterschiede zwischen MongoDB und relationalen Datenbanken im Bereich der geografischen Informationen verglichen. Aufgrund der experimentellen Bedingungen ist es nicht möglich, Vergleichstests für mehrere relationale Datenbanken durchzuführen. Daher wurde PostGIS als Vertreter der relationalen Geodatenbank ausgewählt, um das Experiment fortzusetzen.

PostGIS ist eine räumliche Datenbank. Räumliche Datenbanken speichern und bearbeiten räumliche Objekte wie jedes andere Objekt in der Datenbank. Im Folgenden werden drei Aspekte erläutert, die Geodaten mit einer Datenbank verknüpfen: Datentypen, Indizes und Funktionen. Räumliche Datentypen beziehen sich auf Formen wie Punkte, Linien und Polygone. Die mehrdimensionale räumliche Indizierung wird zur effizienten Verarbeitung räumlicher Operationen verwendet. In SQL bereitgestellte räumliche Funktionen dienen zum Abfragen von räumlichen Eigenschaften und Beziehungen. Die Kombination von räumlichen Datentypen, Indizes und Funktionen bieten eine flexible Struktur für eine optimierte Leistung und Analyse an.

PostGIS verwandelt das PostgreSQL-Datenbankmanagementsystem in eine räumliche Datenbank, indem die folgenden drei Eigenschaften unterstützt werden: räumliche Datentypen, Indizes und Funktionen (PostGIS, 2019). Da PostGIS auf PostgreSQL basiert, übernimmt PostGIS automatisch wichtige „Enterprise“-Funktionen sowie offene Standards für die Implementierung.

5.1 Testmethode

In dieser Arbeit wurde ein Vergleichsexperiment gemäß dem Konzept von Benchmark entworfen. Vor der Durchführung des Vergleichsexperiments sollte zunächst ein Indikatorsystem zur Quantifizierung der Vergleichsdaten entwickelt werden. Benchmark ist eine genormte Mess- und Bewertungsmethode, die normalerweise für Leistungstests im Computerbereich verwendet wird und hauptsächlich die Reaktionszeit, die Übertragungsrate und den Durchsatz testet (Wang, 2006).

Unter Benchmark-Test versteht man einen Softwaretest, der eine wiederholbare quantifizierbare Ergebnisse liefert. Auf dieser Basis können Sie aktuelle und zukünftige Softwareversionen einer bestimmte Funktion vergleichen. Hierbei handelt es sich um ein Verfahren zum Vergleichen der Leistung eines Software- oder Hardwaresystems, das auch als SUT (System Under Test) bezeichnet wird. Ein Benchmark muss wiederholbar und quantifizierbar sein. Zum Beispiel kann die Benutzererfahrung nicht in Zahlen quantifiziert werden, aber die Zeit, die ein Benutzer aufgrund einer guten Benutzeroberfläche auf einer Webseite verbringt, kann quantifiziert werden.

Nach dem Verständnis des Benchmarks kann ein Benchmark-Test in der Regel in sechs Punkte unterteilt werden: Zweck, Indikatoren, Belastung, Begrenzungen, Testmethode, Testergebnis.

Tabelle 19: Sechs Fragen für einen Benchmark-Test

Zweck	Vergleichen der Leistungsunterschiede zwischen MongoDB und PostGIS
Indikatoren	Reaktionszeit, Speicherplatzverbrauch,
Belastung	Bereitstellung der ausreichenden Belastung (Verarbeitung der ausreichend großen Datenmenge)
Begrenzungen	So viel wie möglich in derselben Betriebsumgebung und Konfiguration
Testmethode	Schritte, Testdauer, Indikator-Messmethode
Testergebnis	Ergebnisanzeige, Bewertung

Zu Beginn dieses Kapitels bestand der Zweck des Experiments darin, die Leistungsunterschiede zwischen MongoDB und PostGIS zu vergleichen. Reaktionszeit für verschiedene spezielle Operationen und Speicherplatzverbrauch wurden zu diesem Zweck als Indikatoren ausgewählt (siehe Tabelle 20).

Tabelle 20 : Indikatoren, Datenquelle und Datenmenge

Indikatoren	Point-Daten	Polygon-Daten	Raster-Daten
Reaktionszeit			
Speicherplatzverbrauch			
Quelle	Geofabrik	Geofabrik	Geoportal-Th.de
Insg. Datenmenge	10 GB	10 GB	10 GB

Da der Leistungsunterschied im Bereich der geografischen Informationen verglichen werden soll, werden nicht nur Vektordaten, sondern auch Rasterdaten als experimentelle Daten ausgewählt. Die Vektordaten enthalten auch Punkdaten und Polygondaten. Es gibt drei Arten von Daten, von denen jede 10 GB Daten besitzt. Vor dem Experiment wurden die 10GB Daten in fünf Größenordnungen aufteilen: 5MB, 50MB, 500MB, 5GB, 10GB. Auf diese Weise beobachtet man, wie sich die Leistung der Datenbank mit zunehmender Datenmenge ändert.

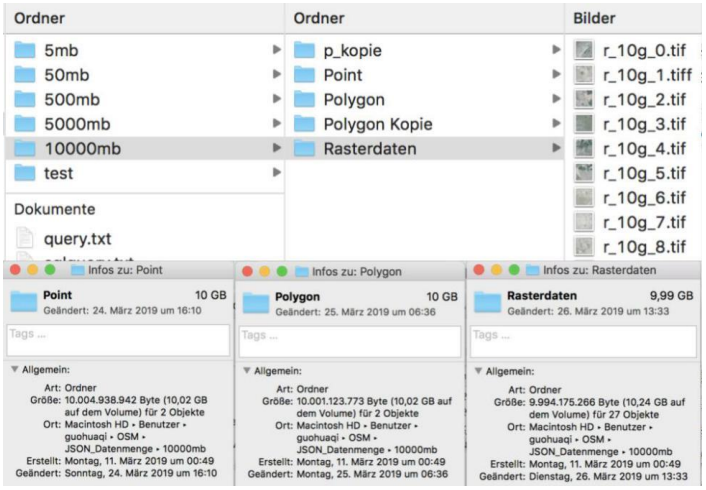


Abbildung 36: Datenliste inklusive der Dateiinformationen

Zudem werden Vektordaten von der Geofabrik-Website[34] heruntergeladen. In der Geofabrik-Website können OSM (Open Street Map) Daten schnell und einfach heruntergeladen. Die Rasterdaten stammen von der Geoportal-Th.de-Website[35], die nur historische Luftbilder und Orthofotos im Bundesland Thüringen liefert.

Tabelle 21: spezifische Operationen

Operation	Point-Daten	Polygon-Daten	Raster-Daten
Einspeicherung			
Select All			
Nächste Nachbarn Anfrage			
BoundingBox Query			

Die speziellen Operationen enthalten Einspeicherung der Daten, allgemeine Abfrage und räumliche Abfragen, einschließlich Abfragen von nächsten Nachbaren und Bounding Box-Abfrage (vgl. Tabelle 21). Dann wird die Reaktionszeit dieser Operationen mittels des Überwachungstools aufgenommen.

Ein weiterer wichtiger Punkt ist die Kontrolle über die Anzahl der auf Ihrem Computer ausgeführten Prozesse. Achten Sie immer auf die CPU- und Arbeitsspeicherauslastung, um sicherzustellen, dass bei allen Operationen die Systemleistung voll ausgeschöpft wird.

Die konkreten Schritte der Implementierung der quantitativen Bewertung sind wie folgt:

[34] http://download.geofabrik.de/index.html

[35] https://www.geoportal-th.de/de-de

1. Zuerst werden die Punktdaten, Polygondaten und Rasterdaten in der Datenbank gespeichert und wird die Zeit aufgezeichnet, die zum Speichern der Daten aufgewendet wurde. Um die Auswirkung von Datentypen auf die Datenspeicherung zu überprüfen, stehen drei Datentypen zur Verfügung: Punkt, Polygon und Raster. Um die Anpassungsfähigkeit der Datenbank an die Datenspeicherung zu bewerten, wurden für jeden Datentyp fünf Größenordnung der Daten hergestellt. Änderungen in der Speicherzeit werden protokolliert , wenn die Menge der gespeicherten Daten zunimmt. Danach werden der Speicherplatzverbrauch der beiden Datenbanken ausgewertet, indem die Größe jeder Datensätze abgefragt.

2. Um die Fähigkeit der Datentraversierung von Mongodb und Postgis zu testen, werden „Select all"-Operationen separat an den Daten durchgeführt, und wird die Antwortzeit der Operation aufgezeichnet.

3. Um die Fähigkeit der räumlichen Abfrage von Mongodb und Postgis als räumliche Datenbank zu bewerten, wird eine räumliche Abfrage der gespeicherten Vektordaten durchgeführt. Und die Antwortzeit der Abfrage ist zu protokollieren. Um die Anpassungsfähigkeit von räumlichen Abfragen zu bewerten, wurden fünf Datenabfragebereiche festgelegt. Auf diese Weise beobachten wir, wie sich die Antwortzeit für Datenbankabfragen mit zunehmendem Datenbereich ändert.

5.2 Vorbreitung der Geodaten

Vektordaten

Wie in Abbildung 37 gezeigt, kann die Shape-Datei nicht direkt abgerufen werden, sodass die komprimierte Datei im .bz2-Format heruntergeladen wird. Nach dem Dekomprimieren wird die .osm-Datei gewonnen. Der erste Schritt besteht darin, die .osm-Datei in einen Datentyp zu konvertieren, der von beiden Datenbanken unterstützt wird.

Faroe Islands	[.osm.pbf] (2.0 MB)	[.shp.zip]	[.osm.bz2]
Finland	[.osm.pbf] (425 MB)	[.shp.zip]	[.osm.bz2]
France	[.osm.pbf] (3.4 GB)	✖	[.osm.bz2]
Georgia (Eastern Europe)	[.osm.pbf] (45.4 MB)	[.shp.zip]	[.osm.bz2]
Germany	[.osm.pbf] (2.9 GB)	✖	[.osm.bz2]
Great Britain	[.osm.pbf] (1007 MB)	✖	[.osm.bz2]
Greece	[.osm.pbf] (171 MB)	[.shp.zip]	[.osm.bz2]
Hungary	[.osm.pbf] (153 MB)	[.shp.zip]	[.osm.bz2]

Abbildung 37: OSM-Daten von Geofabrik

Um das Datenformat zu konvertieren, werden OSM-Daten direkt in QGIS importiert. Als nächstes kann man die erforderlichen Layers, Punkt-Layer und Polygon-Layer, auswählen (siehe Abbildung 38).

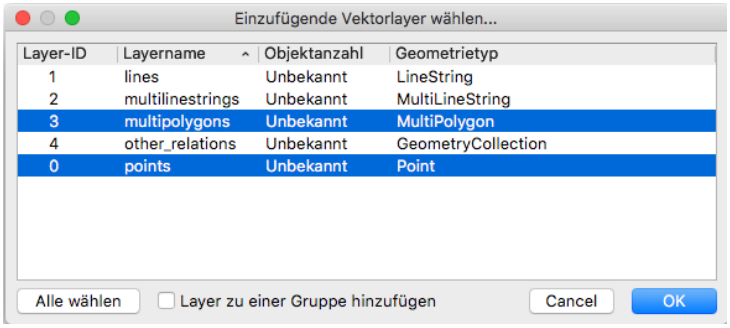


Abbildung 38: Auswahl der einzufügenden Vektorlayer

Klicken Sie dann mit der rechten Maustaste auf die zu exportierendem Layer und wählen Sie "exportieren" -> "Objekt speichern als ...". Wählen Sie dann das Ausgabeformat, den Namen der Ausgabedatei, das Koordinatensystem und andere Parameter. Klicken Sie abschließend auf OK (vgl. Abbildung 39).

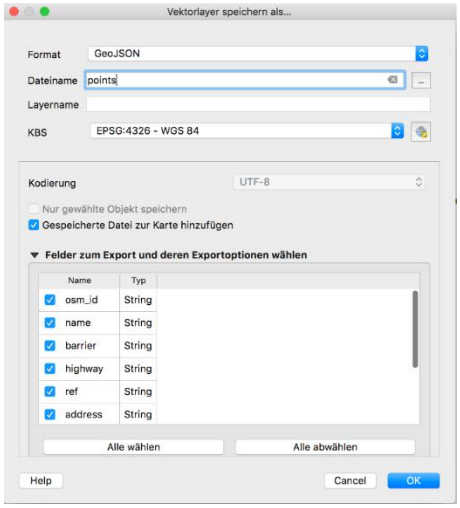


Abbildung 39: Datenexport-Fenster

Nach dem Export aller Daten wurden die Dateien entsprechend der Datengröße mit Sublime Text aufgeteilt oder zusammengefügt.

Rasterdaten

Rasterdaten werden von Geoportal-Th.de erhalten. Fast alle Bilder haben eine Größe von 409,7 MB. Deshalb wurde die Software „Vorschau“ verwendet, um die Bilder auf die richtige Größe zuzuschneiden.

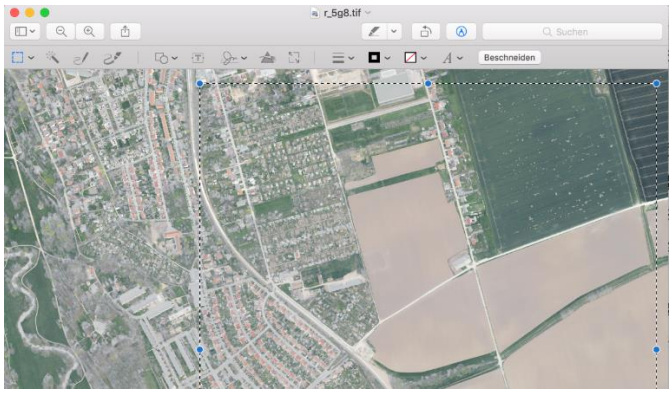


Abbildung 40: Bildverarbeitung

Nachdem die Daten geändert wurden, müssen noch zwei Datensätze vorbereitet werden. Eine Gruppe ist die Bezugspunkte für die Nächsten Nachbarn-Abfragen, und die andere ist der rechteckige Bereich für die *Bounding Box Query*.

```
# Abfragen der Punktkoordinaten
    $ select * from p_5m where name = 'Bitterfeld-Wolfen' and osm_id
    = '509212';
# Bezugspunkte:
5m:   0101000020E6100000FB3BDBA337642840311702CAF0D04940
50m:  0101000020E61000003EAA0606B5682140534ABEC8BA724840
500m: 0101000020E61000004A062571FB0229406F7143424AD34840
5g:   0101000020E6100000C3E457BD0DF726402409675CDD9B4740
10g:  0101000020E6100000C42D7A02BC682A40CA575DE223D24740
```

Koordinaten der Rechtecke:

```
        5m:
             Point: [ 11.300,51.400], [11.500,52.200]
             Polygon: [9.750,51.500], [9.950,52.300]
        50m:
             Point:  [8.200,47.700], [8.600,48.500]
             Polygon: [7.800,52.500], [8.200,53.300]
        500m:
             Point:  [7.600,46.600], [8.400,47.400]
             Polygon: [6.300,50.700], [7.100,51.500]
        5g:
             Point:  [13.600,46.870], [15.200,47.570]
             Polygon: [7.600,47.800], [9.20,48.600]
        10g:
             Point:  [12.500,46.600], [15.700,47.400]
             Polygon: [10.080,48.300], [13.280,49.100]
```

5.3 Testumgebung

Hardware:

Computer: MacBook Pro (15-inch,2017)

Prozessor: 2,9 GHz Intel Core i7

Speicher: 16GB 2133MHz LPDDR3

Grafikkarte: Intel HD Graphics 630 1536MB

Festplatte: 500GB SSD

Software:

Betriebssystem: MacOS High Sierra Version 10.13.6

Angewendete Software: PostgreSQL 9.6.11, PostGIS 2.5.1, MongoDB 4.0.6, QGIS 3.4.2, WPS Office for Mac Beta 5, Vorschau 10.0, Sublime Text 3.2, iTerm2 3.2.9, Robo 3T 1.2.1, pgAdmin 4 4.3

5.4 Durchführung des Benchmark-Tests

MongoDB

Starten des MongoDB-Servers:

```
$ ./mongod -f /data/etc/mongod.conf
```

Einfügen der Daten (ein Beispiel von 5MB-Punktdaten und 5MB-Rasterdaten):

```
$ ./mongoimport -d point -c p_5m --jsonArray "/Users/guo-
huaqi/OSM/JSON_Datenmenge/5mb/Point/sa_traffic.geojson"
```

```
➜ bin ./mongoimport -d point -c p_5m --jsonArray "/Users/guohuaqi/OSM/JSON_Datenmenge/5mb/Point/sa_traffic.geojson"
2019-06-16T19:34:07.973+0200    connected to: localhost
2019-06-16T19:34:08.421+0200    imported 25822 documents
```

```
$ ./mongofiles -d r_5m put /Users/guohuaqi/OSM/JSON_Daten-
menge/5mb/Rasterdaten/r_5mb.png
```

```
➜ bin ./mongofiles -d r_5m put '/Users/guohuaqi/OSM/JSON_Datenmenge/5mb/Rasterdaten/r_5mb.png'
2019-06-16T19:29:30.005+0200    connected to: localhost
2019-06-16T19:29:30.021+0200    added file: /Users/guohuaqi/OSM/JSON_Datenmenge/5mb/Rasterdaten/r_5mb.png
```

Abbildung 41 zeigt, wie alle Daten nach dem Import in die Datenbank aussehen.

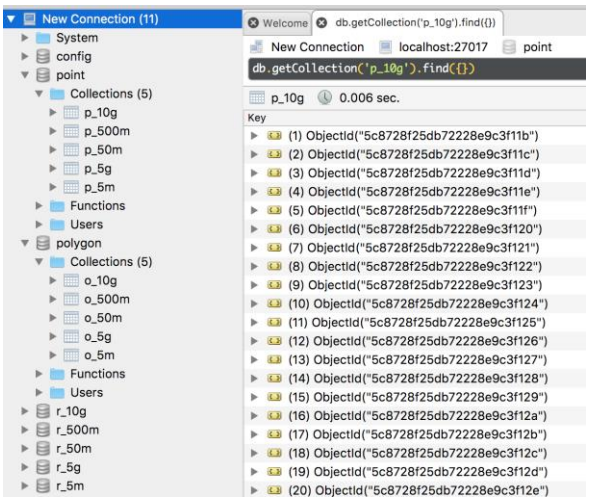


Abbildung 41: MongoDB-Datenbankübersicht

Erstellen des Index:

```
db.p_5m.createIndex({"geometry":"2dsphere"});
```

Abfrage speicherplatzverbrauch:

```
db.getCollection('p_5m').stats()
```

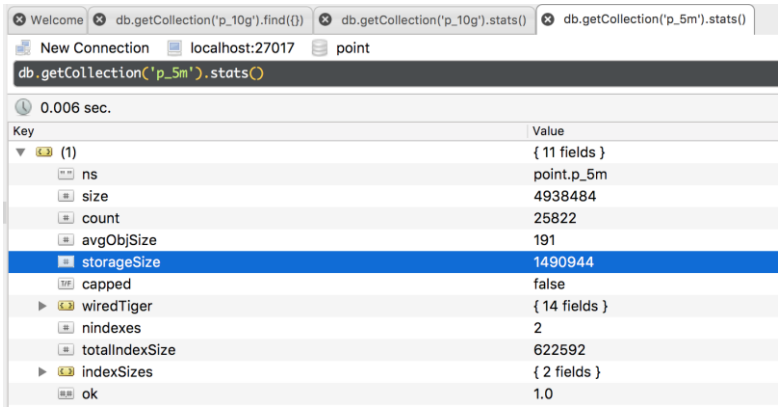


Abbildung 42: Abfrage der Größe von p_5m-Collection

Mongotop bietet eine Methode zum Verfolgen der Zeit, die ein MongoDB-Instanz mongod zum Lesen und Schreiben von Daten benötigt. Mongotop liefert Statistiken für jede *Collection*. Standardmäßig gibt mongotop jede Sekunde Werte zurück (vgl. Abbildung 43).

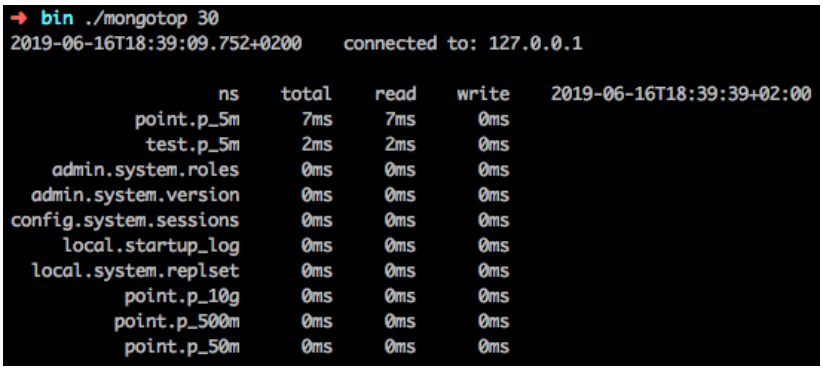


Abbildung 43: MongoTop

Nächste Nachbarn Abfrage:

```
db.p_5m.find({ geometry: { $near: { $geometry: { type: "Point" ,
                        coordinates: [ 12.195737, 51.6323483 ] },
                        $minDistance: 0 }}}).limit(10);
```

Bounding Box Query:

```
db.p_5m.find({ geometry: { $geoWithin: { $box:
                                    [ [ 11.300,51.400],
                                    [ 11.500,52.200] ]}}});
```

PostGIS:

Da QGIS Daten direkt in PostGIS importieren kann, wird die integrierte Funktionalität von QGIS direkt verwendet, um eine neue Verbindung zu PostGIS zu erzeugen (vgl.Abbildung 44).

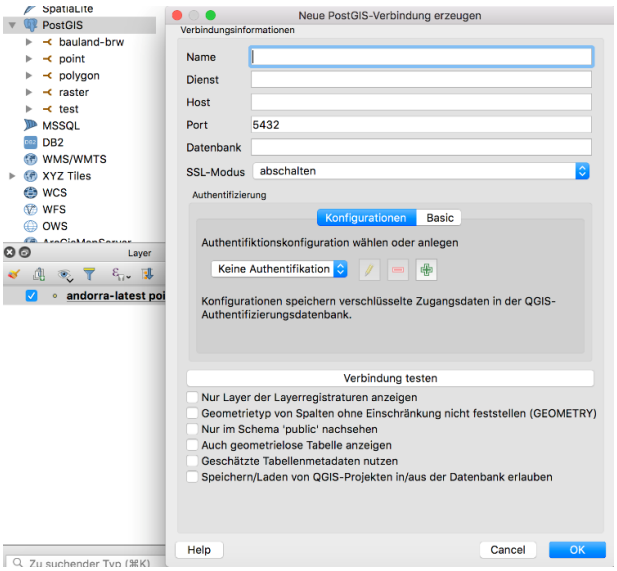


Abbildung 44: Erzeugung einer Verbindung zu PostGIS

Nach dem Verbindungsaufbau können Sie in der Symbolleiste auf "Datenbank" klicken und "DB-Verwaltung ..." auswählen. Dann erscheint das Fenster in Abbildung 45.

Abbildung 45: DB-Verwaltung

Anschließend können Sie mit der Option "Layer / Datei importieren" Daten nach POSTGIS importieren. Die Benutzer können Layer, Schemas, zu importierende Tabellen, Primärschlüssel, Koordinatensysteme und mehr einsetzen. Spezifische Parameter können in Abbildung 46 entnommen werden.

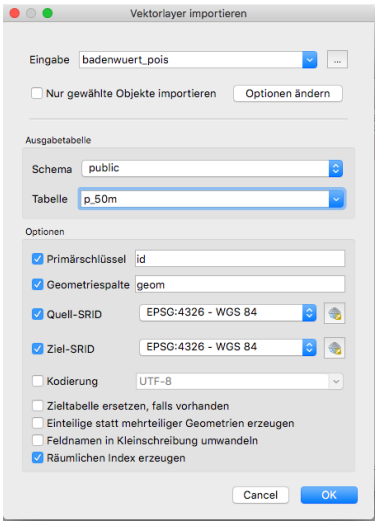


Abbildung 46: Vektorlayer Importieren

Einfügen der Rasterdaten:

```
./raster2pgsql -c r_5mb ~/osm/json_datenmenge/5mb/rasterda-
ten/5mb.tif -t 256x256 | psql -h localhost -p 5432 -U postgres -
d raster -W
```

Tabelle 22: Operatoren und Parameter des raster2pgsql Tools

-t TILE_SIZE:	Schneiden Sie das Raster in Kacheln, die jeweils in eine Tabellenzeile eingefügt werden sollen. TILE_SIZE wird als WIDTHxHEIGHT ausgedrückt oder auf den Wert "auto" gesetzt, damit der Loader mit dem ersten Raster eine geeignete Kachelgröße berechnen und auf alle Raster anwenden kann.
(c | a | d | p) Hierbei handelt es sich um sich gegenseitig ausschließende Optionen:	
-c:	Erstellen Sie eine neue Tabelle und füllen Sie sie mit Raster (n). Dies ist der Standardmodus
-a:	Fügen Sie Raster an eine vorhandene Tabelle an.
-d:	Tabelle löschen, neue erstellen und mit Raster füllen
-p:	Vorbereitungsmodus, nur die Tabelle erstellen.
-d dbname:	Geben Sie den Namen der Datenbank an, zu der eine Verbindung hergestellt werden soll.
-h hostname:	Gibt den Hostnamen des Computers an, auf dem der Server ausgeführt wird.
-p port:	Gibt den TCP-Port oder die lokale Unix-Domain-Socket-Dateierweiterung an, auf der der Server auf Verbindungen wartet.
-W Passwort	Erzwingen Sie, dass psql vor dem Herstellen einer Verbindung mit einer Datenbank zur Eingabe eines Kennworts auffordert.
-U username:	Stellen Sie mit dem Benutzernamen des Benutzers anstelle des Standardbenutzernamens eine Verbindung zur Datenbank her.

Abbildung 47 stellt eine Liste der PostGIS-Datenbanken nach dem Datenimport dar.

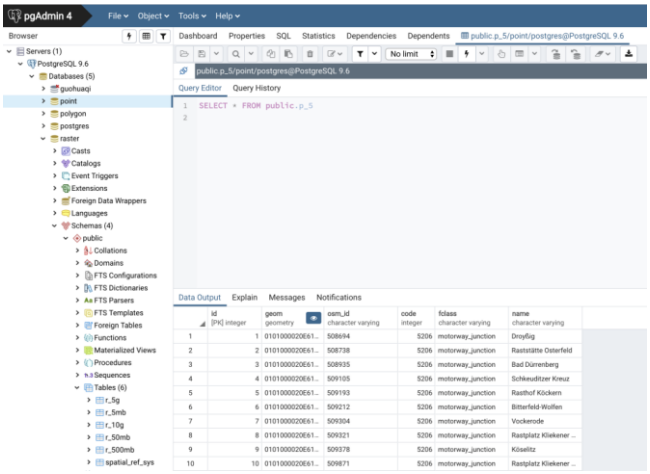


Abbildung 47: PostGIS Datenbankübersicht

Weiterhin werden SQL-Anweisungen für Nächste Nachbarn-Abfrage und Bounding Box Query angegeben.

Nächste Nachbarn-Abfrage:

```
select osm_id,name,fclass, ST_Dis-
tance('0101000020E6100000C42D7A02BC682A40CA575DE223D24740',geom) as
distance from p_5m
ORDER BY distance limit 10;
```

Bounding Box Query:

```
SELECT * FROM p_5m
WHERE p_5m.geom && ST_MakeEnvelope(11.300,51.400,11.500,52.20,
4326);
```

Monitoring:

pg_stat_statement installieren: Das Modul pg_stat_statements bietet eine Möglichkeit, die Ausführungsstatistik aller von einem Server ausgeführten SQL-Anweisungen zu verfolgen.

```
# Zurückgeben der Abfrageanweisung und entsprechende Reaktionszeit
# Query ist Text eines repräsentativen Statements
    select query,total_time from pg_stat_statements order by query;
```

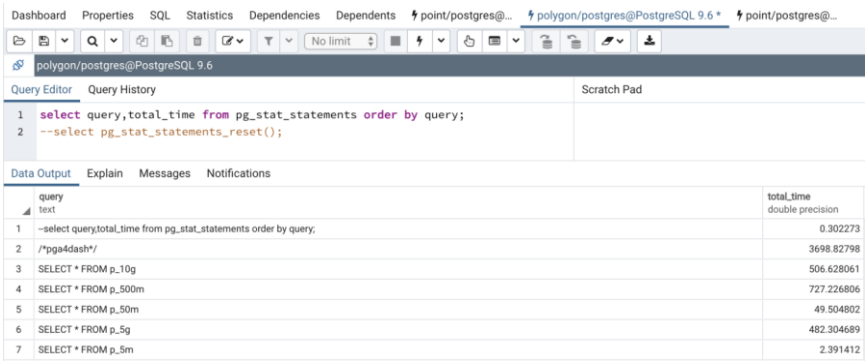


Abbildung 48: Abfragen von pg_stat_statements

Zurücksetzen aller Protokolle:

```
    --select pg_stat_statements_reset();
```

Abfragen der Datengröße:

```
    --select pg_relation_size('p_5m');
```

Abfragen der Rasterdatengröße :
Rasterdaten werden nicht in einer Datenbanktabelle gespeichert.

```
    --select pg_total_relation_size( ‘r_5mb’ );
```

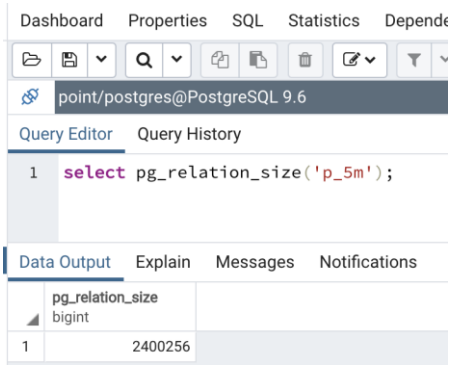


Abbildung 49: Abfragen der Datengröße

5.5 Bewertung der Ergebnisse

Dieser Vergleichstest legt drei Datentypen und fünf Größenordnungen der Daten für zwei räumliche Datenbanken fest. Um den Einfluss von zufälligen Fehlern so weit wie möglich auszuschließen, wird die Operation mindestens dreimal für jede Operation wiederholt und der Durchschnittswert als Testsergebnis verwendet.

Experimentelle Daten von MongoDB:

MongoDB					
Reaktionszeit-Punkte (mb)	5	50	500	5120	10240
Einspeicherung (ms)	1	1	4206	35822.33	118122.5
Select all (ms)	17	93.33	681.33	12059.33	31246.67
Nächste Nachbarn Anfrage (ms)	3	12.33	13.33	17.67	20.67
Bounding Box Query (ms)	50	360	2234.67	25014	73152.33

Reaktionszeit-Polygon (mb)	5	50	500	5120	10240
Einspeicherung (ms)	1	1	3688.33	64691.5	128186
Select all (ms)	17	71.33	1240	24802	39956
Bounding Box Query (ms)	110.33	1089.67	11629.33	117465.33	230617.67

Reaktionszeit-Raster-daten (mb)	5	50	500	5120	10240
Einspeicherung (ms)	1	1	269.33	10804.33	20854.33

Anmerkung: : Ergebnisse von Punkten : Ergebnisse von Polygon : Ergebnisse von Rasterdaten

Experimentelle Daten von PostGIS:

PostGIS					
Reaktionszeit-Punkte (mb)	5	50	500	5120	10240
Einspeicherung (ms)	198.92	1871.15	11837.04	92338.75	219489.76
Select all (ms)	25.99	296.35	1445.36	12301.36	28127.49
Nächste Nachbarn Anfrage (ms)	17.14	92.1	321.8	6339.45	10721.77
Bounding Box Query (ms)	8.74	107.54	914	539.26	753.78

Reaktionszeit-Polygon (mb)	5	50	500	5120	10240
Einspeicherung (ms)	86.05	751.3	7791.28	120187.17	238041.9
Select all (ms)	33.96	324.49	2117.85	21279.59	47276.12
Bounding Box Query (ms)	19.67	71.53	1216.5	4370.01	9208.96

Reaktionszeit-Raster-daten (mb)	5	50	500	5120	10240
Einspeicherung (ms)	355.66	2366.21	16179.57	211485.26	531710.07

Anmerkung: : Ergebnisse von Punkten : Ergebnisse von Polygon : Ergebnisse von Rasterdaten

Diagramm:

Die blaue Linie steht für Mongoinformationen und die orange Linie für Postgis.

Speicherungsverbrauch:

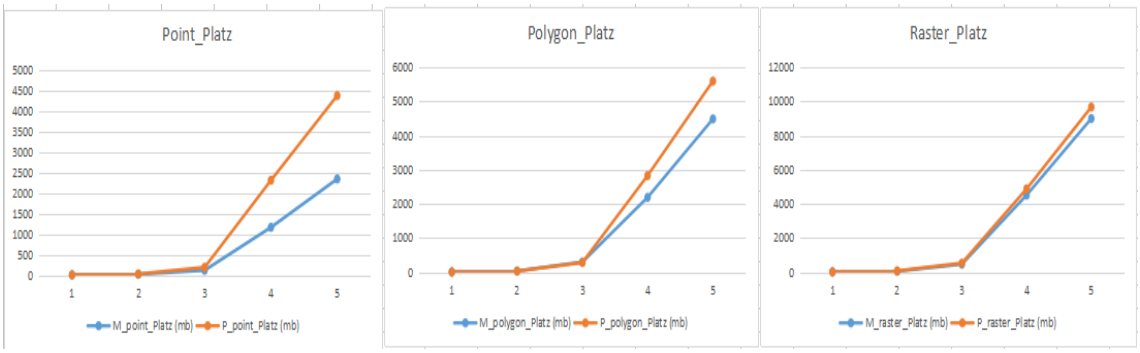


Abbildung 50: Speicherungsverbrauch

Wie aus Abbildung 50 ersichtlich, gibt es fast keinen Unterschied zwischen den beiden Datenbanken bei der Größenordnung von 1, 2 oder 3 und der Rasterdaten. Mit zunehmender Datenmenge werden die Vorteile von MongoDB immer deutlicher, insbesondere in Bezug auf Punktdaten. Einer der wichtigsten Gründe kann sein, dass PostGIS viele leere Spalten oder Zellen enthält.

Reaktionszeit der Einspeicherung:

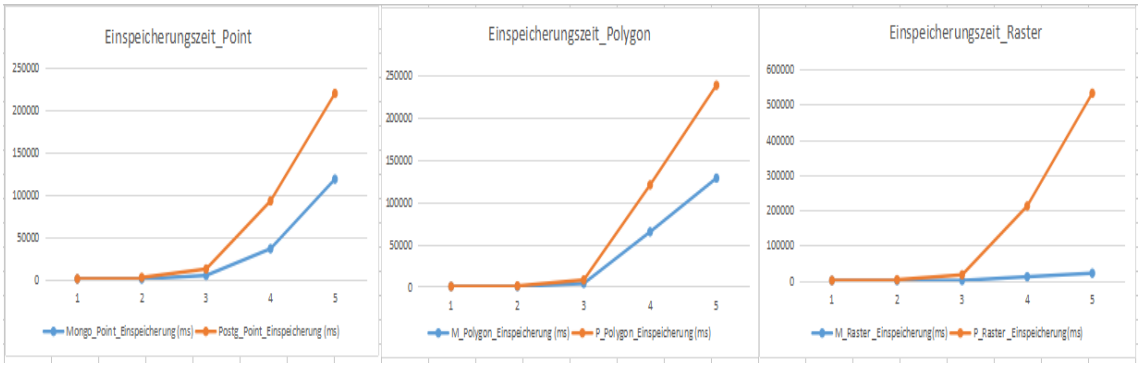


Abbildung 51: Reaktionszeit der Einspeicherung

In Bezug auf die Reaktionszeit des Datenimports ist MongoDB über PostGIS umfassend, ob es sich um Vektordaten oder Rasterdaten handelt. Ein wichtiger Grund ist, dass PostGIS ein strenges Paradigma hat und ACID beibehalten muss. Und MongoDB garantiert nur die eventuelle Konsistenz.

Reaktionszeit der Operation (select all):

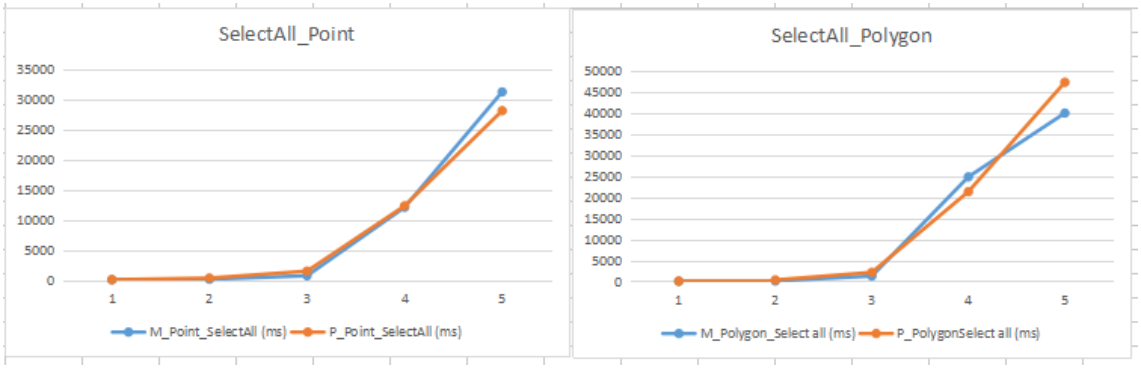


Abbildung 52: Reaktionszeit der Operation (select all)

In der Abfrage "Alles auswählen" gibt es nur eine geringe Schwankung zwischen den beiden Datenbanken. Um mehr über die Unterschiede zwischen den beiden in diesem Vorgang zu erfahren, benötigen Sie möglicherweise viel mehr Daten.

Reaktionszeit der Nächsten Nachbarn Abfrage:

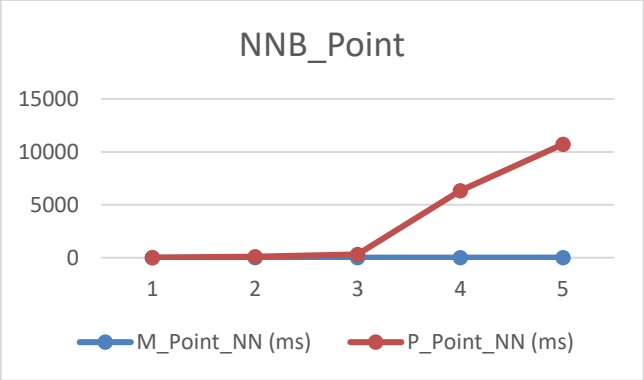


Abbildung 53: Reaktionszeit der Nächsten Nachbarn Abfrage

Die Reaktionszeit der PostGIS erhöht sich deutlich ab 500mb mit zunehmender Datenmenge. Im Vergleich zur PostGIS wird MongoDB fast nicht von insgesamten Datenmengen beeinflusst. Dies kann mit dem Index, dem Zeiger oder den Daten im Cache zusammenhängen.

Reaktionszeit (Bounding Box Query):

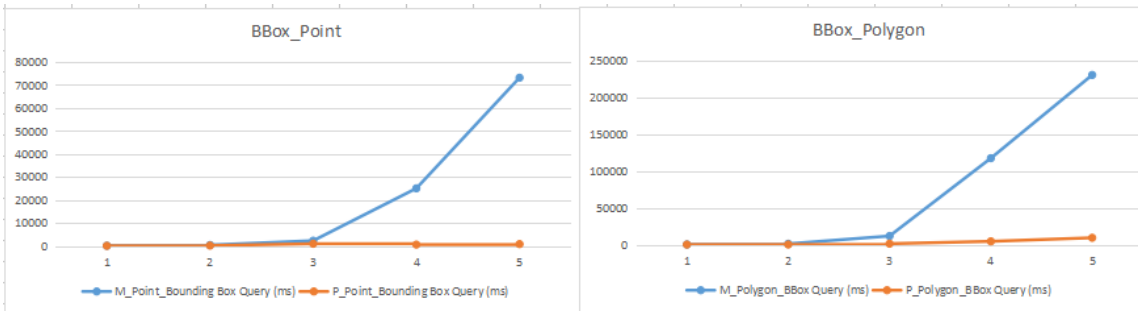


Abbildung 54: Reaktionszeit (Bounding Box Query)

Postgis ist in Bounding Box-Abfragen deutlich schneller als Mongodb. Der Hauptgrund für diesen Unterschied ist wahrscheinlich der räumliche Index. PostGIS unterstützt R Tree Index. Der R-Tree-Index im Index-Scan-Prozess einen Cache für die I/O erstellt, hat er in der Index-Scan-Phase einen Vorteil (Shao, 2017). Mongodb verwendet einen räumlichen 2dsphere-Index. Der auf der Grid-Technologie basierende 2dsphere-Index muss jedoch doppelte Indexeinträge im Raster überprüfen und beseitigen und komplexere sphärische Berechnungen verwenden, um einen gewissen Einfluss auf die Leistung zu haben.

Bewertungsformular:

Indikatoren	Spvb	R_(Es)	R_(SA)	R_(NNA)	R_(BBQ)
Point	M > P	M > P	M ≈ P	M >> P	P >> M
Polygon	M > P	M > P	M ≈ P	_	P >> M
Raster	M ≈ P	M >> P	_	_	_

Anmerkung: Spvb = Speicherplatzverbrauch, R = Reaktionszeit, Es = Einspeicherung, SA = Select ALL, NNA = Nächste Nachbarn Anfrage, BBQ = Bounding Box Query

> : besser	>> auffällig besser	M : MongoDB	P : PostGIS	R : Reaktionszeit

5.6 Fazit

Durch dieses Experiment habe ich ein tieferes Verständnis für diese beiden Datenbanken gewonnen. Nach der Idee von benchmark wurde das Vergleichsexperiment der Datenbank abgeschlossen. Es umfasst das Sammeln von Daten, das Verarbeiten von Daten, das Entwerfen experimenteller Methoden, das Konfigurieren der Arbeitsumgebung, das Abschließen von Experimenten und das Analysieren von Ergebnissen.

MongoDB besitzt bessere Leistung bei dem Schreiben und der nächsten Nachbarn Abfrage als PostGIS. Aber im Vergleich zur PostGIS sinkt die Abfrageleistung der MongoDB stark mit der Zunahme der Datenmenge in der Bounding Box Query.

Mit der hervorragenden Skalierbarkeit von Mongodb eignet es sich besser für Szenarien mit gleichzeitigem Zugriff, wie Web-GIS und Mobile-GIS, und ist auch für weniger leistungsfähige Computer geeignet (Agarwal & Rajan, 2015). Aber PostgreGIS bietet Ihnen umfassende räumliche Funktionen, obwohl Sie in Ihrer Anwendung möglicherweise nicht so viel benötigen. Zudem unterstützt es noch B-Tree Index, R-Tree Index und GiST Index.

Benutzer, die MongoDB als Geodatenbank verwenden möchten, sollten berücksichtigen, ob während der Verwendung häufig komplexe räumliche Abfragen verwendet werden. Da MongoDB jetzt nur zwei räumliche Indizes unterstützt, kann es sich nicht an komplexe Abfrageszenarien anpassen. Die hervorragende Speichergeschwindigkeit von MongoDB macht es jedoch zu einem Szenario mit hoher Datenkapazität und schnellen Speicherung, z. B. eine Datenbank zum Speichern von Satellitenbildern (Lopez, 2016). Im Folgenden finden Sie einige allgemeine Referenzstandards (AIGeorge, 2018). Wenn Ihr

Unternehmen eine oder mehrere Eigenschaften erfüllt, ist die Wahl von MongoDB die richtige Entscheidung:

1) Keine dokumenten- oder tabellenübergreifenden Transaktionen und Unterstützung für komplexe Join-Abfragen.
2) Das gespeicherte Datenformat ist flexibel, nicht fest oder gehört zu halbstrukturierten Daten.
3) Business Concurrent Access ist groß, brauchen Tausende von QPS (Queries per second).
4) Massive Datenspeicherung über TB-Niveau (Terabyte), und die Datenmenge steigt weiter an.
5) Die erforderlichen Speicherdaten sind persistent, nicht verloren..
6) Eine große Anzahl von geografischen Standortabfragen oder Textabfragen sind notwendig.

6 Verbesserungsvorschlägen zur Steigerung der Performance

Die Leistungsoptimierung für MongoDB kann im Allgemeinen in zwei Hauptaspekte unterteilt werden: Softwareoptimierung und Hardwareoptimierung (Henry, 2018).

6.1 Software-Optimierung

1. Verbindungsoptimierung:

   Wenn zu viele Serververbindungen bestehen, wird auch die Leistung beeinträchtigt. Sie können dies über den Serverstatus überprüfen. Die Anzahl der Verbindungen zu jedem Server wird aufgerufen. Die Verbindungen müssen in einer Stapelstruktur gespeichert werden und jede Verbindung wird vom System begrenzt.

2. Index-Optimierung:

   Indizes sind in der gesamten Datenbank weit verbreitet, und MongoDB ist keine Ausnahme. MongoDB bietet einen "explain" -Befehl, um zu erfahren, wie das System Abfrageanforderungen verarbeitet. Der Befehl "explain" wird verwendet, um das System zu beobachten. Gleichzeitig ist "MongoDB Database Profiler" eine langsame Abfrageprotokollfunktion, die als Grundlage für die Optimierung der Datenbank verwendet werden kann. Die Indizierung kann die Abfrageleistung erheblich verbessern, und der Index ist nicht je mehr desto besser. Immer wenn ein Index erstellt wird, wird eine Indextabelle hinzugefügt, um die angegebenen Spalten zu indizieren. Beim Einfügen oder Ändern einer indizierten Spalte muss die Datenbank die ursprüngliche Indextabelle neu anordnen. Der Vorgang des Neuordnens verbraucht viel Leistung.

   Daher müssen Sie beim Erstellen eines Indexes vorsichtig sein, um einen Index zu erstellen, und Sie müssen die Funktionen jedes Indexes im Extremfall verwenden. Das heißt, für den Fall, dass die Indexanforderung erfüllt werden kann, gilt: Je kleiner die Anzahl der Indizes, desto besser. Gleichzeitig wird die Anzahl der Wiederholungen aller Daten in der Indexspalte als Granulat bezeichnet, auch Kardinalität des Index genannt. Wenn die Datenpartikel zu groß sind, kann der Index die Leistung nicht erbringen.

3. Abfragen-Optimierung:

   Wenn Sie in der Abfrage einige Felder zurückgeben möchten und diese Felder im Index platziert werden können, kann MongoDB Indexabfragen durchführen. Diese Art der Abfrage greift nicht auf das Dokument zu, auf das der Zeiger zeigt, sondern gibt das Ergebnis direkt unter Verwendung der indizierten Daten zurück.

Wenn Sie für die $AND-Abfrage die Dokumente abfragen möchten, die die Bedingungen A, B und C erfüllen, sind 40000, 8000 bzw. 300 Dokumente erfüllt. Wenn Sie in dieser Reihenfolge vorgehen, wird die Effizienz erheblich verringert, Sie können die anspruchsvollsten Abfragen in die vordere Abfrage einfügen, und der Suchbereich wird erheblich verringert. In ähnlicher Weise können bei Abfragen vom Typ OR die bedingten Vorbedingungen, die die Ergebnisse maximieren, den Suchraum für nachfolgende Abfragen einschränken (Chodorow, 2011).

4. Journaling-Optimierung:

   Das heißt, die Protokolloptimierung. MongoDB Aufgrund der Art der speicherinternen Datenbank werden Schreiboperationen zuerst im Speicher gespeichert, wodurch die Persistenz von Datendateien verringert wird. Unterstützt die Protokollierung gleichzeitig mit dem Schreiboperation, sodass Daten, die verloren gehen, entsprechend dem Protokoll wiederhergestellt werden können. Auf diese Weise werden Ihre Datendateien in einem dauerhaften Speicher abgelegt, ohne dass sich dies auf die aktuellen Daten im Speicher auswirkt. Die Leistung wird bei Aufrechterhaltung der Datensicherheit nicht beeinträchtigt.

5. Aktiv- und Standby-Optimierung:

   Um die Effizienz beim Lesen und Schreiben zu erhöhen, wird der Cluster für die Trennung von Lesen und Schreiben geöffnet. Unter Verwendung des ReplicaSet-Clusters sind die Hauptschreiboperationen PRIMARY mit hoher Leistung, und das Lesen erfolgt vom allgemeinen SECONDARY, der verwendet wird, um den Lese- und Schreibdruck von PRIMARY mit hoher Intensität zu teilen (Zhao Y. , 2013). Der Nachteil ist jedoch, dass die Master-Slave-Synchronisation verzögert ist, da die Synchronisation asynchron ist und die Echtzeitleistung der Daten verringert wird. Im Protokollsystem ist die Echtzeitanforderung für die Daten nicht sehr hoch, so dass die Lese-Schreib-Trennstrategie auf das System anwendbar ist.

6.2 Hardware-Optimierung

1. Erweiterung der Speicher:

   Es ist möglich, den Speicher von Knoten mit mehr heißen Daten zu erweitern, um den häufigen Datenaustausch der Speicherplatten zu verringern. Aufgrund des Speichermechanismus von MongoDB ist die Cache-Neuerstellung vorteilhafter: Starten Sie das System neu, und erstellen Sie den Cache neu, indem Sie die Datendatei nach / dev / null kopieren. Wenn der Speicher kleiner ist als die Menge heißer Daten, treten häufig Seitenumbrüche auf, und der Datenaustausch zwischen dem Speicher und der Festplatte verlangsamt die Abfrage erheblich.

2. Verwendung von SSD:

   Bei Verwendung von SSD als Sharding-Lösung wird angesichts des hohen SSD-Preises nur ein Teil der häufig verwendeten Daten gespeichert (Nguyen & Lee, 2016) und die aktuellen Daten werden direkt ausgetauscht (In-Memory-Daten).

3. Lastausgleich:

   Die Mongod-Instanz im Cluster hat keinen sehr hohen CPU-Anforderungen. Die Mongos-Instanz verbraucht jedoch viel CPU-Ressourcen, sodass die CPU-Konfiguration des Mongos-Knotens hier höher ist.

7 Zusammenfassung und Ausblick

Im Zusammenhang mit Big Data werden immer mehr Daten mit Geoinformationsdaten kombiniert, wie z. B. Sensoren für das Internet der Dinge, das automatische Fahren von Autos, Positionierungsanforderungen für mobile Geräte usw., die zu großen Änderungen im Bereich der Geoinformationen geführt haben. Angesichts massiver unstrukturierter oder semistrukturierter Daten waren traditionelle relationale räumliche Datenbanken in einigen Szenarien nicht geeignet, und es war schwierig, die Anforderungen an Hohe Speicherauslastung und schnelle Antwort auf Anfragen für massive Daten zu erfüllen. Daher beginnt dieser Artikel mit der beliebten NoSQL-Datenbank-MongoDB, um ihre Verwendbarkeit im Bereich der Geoinformationsanwendungen zu untersuchen. Der Hauptinhalt dieses Artikels und die geschaffte Arbeit werden wie folgt zusammengefasst:

1. Zunächst werden hauptsächlich Hintergrund und Motivation des Themas vorgestellt. Indem Sie die aktuellen Herausforderungen der Datenbank erklären, ist es hilfreich, die NoSQL-Datenbank zu erlernen und zu recherchieren.

2. Dann werden eine allgemeine Einführung und Klassifizierung der Datenbank. Anschließend werden die wichtigen Theorien und Prinzip von der relationalen Datenbank sowie von NoSQL ausführlich vorgestellt.

3. Die beliebte MongoDB-Datenbank im NoSQL-System wird als Forschungsobjekt ausgewählt. Der Artikel konzentriert sich auf die Architektur, grundlegende Operationen und wichtige funktionale Merkmale von MongoDB. Schließlich werden einige wichtige Tools und Funktionen im Zusammenhang mit diesem Artikel vorgestellt.

4. Dieser Artikel verwendete zwei Experimente, um MongoDB zu untersuchen.

    1) Einen hoch verfügbaren, verteilten, horizontal erweiterbaren Cluster wurde entworfen und wurde die Clusterbereitstellung auf zwei Verfahren implementiert. Eine ist manuell und die andere ist die automatische Bereitstellung mit dem *Composefile* im Docker-Swarm-Modus. Mit dem manuellen Verfahren kann man die Details der konkreten Prinzipien besser verstehen. Über die automatische Bereitstellung kann man Zeit und Mühe sparen, und sie ist besser für die Produktionsumgebung geeignet. Es stellt sich heraus, dass MongoDB eine gute Benutzerfreundlichkeit, eine Hochverfügbarkeit und eine einfache horizontale Skalierbarkeit aufweist.

    2) Unter Verwendung der Benchmark-Idee wurde ein Vergleichsexperiment zwischen MongoDB und PostGIS entworfen. Mit den experimentellen Daten wurden die beiden Datenbanken analysiert und bewertet. Zuletzt wurden Referenzstandards für die Auswahl von MongoDB angegeben. MongoDB eignet sich besser für große Datenmengen, hohen gleichzeitigen Zugriff, hohe

Verfügbarkeit, flexibles Datenmodell und keine komplizierten Join-Abfrageszenarien. Für komplexe räumliche Abfragen ist PostGIS besser geeignet.

Überblick:

Web2.0 fördert die Entwicklung der NoSQL-Datenbank, und für die aufkommende 5G-Technologie wissen wir noch nicht, wie sie die Welt in Zukunft verändern und wie sie den Bereich der Informatik beeinflussen wird. Angesichts der rasanten Entwicklung von Wissenschaft und Technologie müssen wir weiterhin neues Wissen erlernen und neue Technologien in den Bereich der geografischen Informationen integrieren, damit sie uns dienen können.

Die Folgestudie zu MongoDB kann auch unter folgenden Gesichtspunkten beginnen:

1. Eine davon ist die Software-Optimierung, wie Abfragen-Optimierung, Dokumentenstruktur-Optimierung, Index-Optimierung, Sharding-Strategie usw. Jede kann als Thema studiert wird.
2. Ein weiterer Aspekt ist eine Sekundärentwicklung von MongoDB möglich, z. B. die Entwicklung eines R-Tree Index für MongoDB.
3. Es kann auch mit anderen Produkten kombiniert werden, um die Leistung zu verbessern, z.B. mit Spark[36], um die Rechengeschwindigkeit zu verbessern. Verwenden Sie Yarn[37], um den Datenaustausch des Clusters, den Lastenausgleich und mehr zu optimieren.

.Apropos Artikel kann auch unter den folgenden beiden Aspekten optimiert werden:

1) Die Größe der Testdaten sollte weiter zunehmen, und die Datenmenge in 10 GB ist weit von der Grenzleistung der Datenbank entfernt, da bei einigen Tests keine eindeutige Schlussfolgerung oder Ursache gefunden werden kann.
2) Erweitern Sie die Arten von räumlichen Abfragen weiter, um deren Leistungsunterschiede umfassend zu analysieren

[36] https://en.wikipedia.org/wiki/Apache_Spark

[37] https://code.fb.com/web/yarn-a-new-package-manager-for-javascript/

Anhang A:

A1. Composefile für swarm-worker1: stack1.yml

```
# Use root/example as user/password credentials
version: '3.7'
services:

  # Konfiguration von config server
  configsvr1:
    image: mongo
    ports:
      - 27019:21000
    volumes:
      - config_data:/data/configdb
    networks:
      - mongo
command: mongod --noauth --configsvr --replSet csrs --dbpath /data/db

  # Konfiguration von mongos
  mongos1:
    image: mongo
    ports:
      - 27015:20000
    networks:
      - mongo
    command: mongos --noauth --configdb csrs/configsvr1:21000,con-
figsvr2:21000,configsvr3:21000
    deploy:
      restart_policy:
        condition: on-failure
        delay: 3s

  # Konfiguration von shard1
```

```
  shard1_primary:
    image: mongo
    environment:
      MONGO_INITDB_ROOT_USERNAME: root
      MONGO_INITDB_ROOT_PASSWORD: pwd123
    ports:
      - 27022:22001
    networks:
      - mongo
    volumes:
      - mongo_data:/data/db
    command: mongod --dbpath /data/db --shardsvr --replSet shard1
    deploy:
      restart_policy:
        condition: on-failure
        delay: 3s
    depends_on:
      - configsvr1


 # Konfiguration von shard2
  shard2_arbiter:
    image: mongo
    networks:
      - mongo
    command: mongod --noauth --dbpath /data/db --shardsvr --replSet
shard2
    depends_on:
      - configsvr1


  # Konfiguration von shard3
  shard3_secondary:
    image: mongo
```

```
    environment:
      MONGO_INITDB_ROOT_USERNAME: root
      MONGO_INITDB_ROOT_PASSWORD: pwd123
    ports:
      - 27028:22003
    networks:
      - mongo
    volumes:
      - mongo_data:/data/db
    command: mongod --dbpath /data/db --shardsvr --replSet shard3
    deploy:
      restart_policy:
        condition: on-failure
        delay: 3s
    depends_on:
      - configsvr1


# Erstellung eines Visualisierungstools
  visualizer:
    image: dockersamples/visualizer:stable
    ports:
      - "8080:8080"
    stop_grace_period: 1m30s
    volumes:
      - "/var/run/docker.sock:/var/run/docker.sock"
    deploy:
      placement:
        constraints: [node.role == manager]


# Ladepfad für Datenvolumen
volumes:
  mongo_data:
```

```
  config_data:


# Kommunikationsnetzwerk
networks:
  mongo:
    driver: overlay
```

A2. Composefile für swarm-worker1: stack2.yml

```
# Use root/example as user/password credentials
version: '3.7'
services:


  configsvr2:
    image: mongo
    ports:
      - 27020:21000
    volumes:
      - config_data:/data/configdb
    networks:
      - mongo
    command: mongod --noauth --configsvr --replSet csrs --dbpath /data/db


  mongos2:
    image: mongo
    ports:
      - 27016:20000
    networks:
      - mongo
    command: mongos --noauth --configdb csrs/configsvr1:21000,con-
figsvr2:21000,configsvr3:21000
    deploy:
      restart_policy:
```

```
      condition: on-failure
      delay: 3s

shard1_secondary:
  image: mongo
  environment:
    MONGO_INITDB_ROOT_USERNAME: root
    MONGO_INITDB_ROOT_PASSWORD: pwd123
  ports:
    - 27023:22001
  networks:
    - mongo
  volumes:
    - mongo_data:/data/db
  command: mongod --dbpath /data/db --shardsvr --replSet shard1
  deploy:
    restart_policy:
      condition: on-failure
      delay: 3s
  depends_on:
    - configsvr2

shard2_primary:
  image: mongo
  environment:
    MONGO_INITDB_ROOT_USERNAME: root
    MONGO_INITDB_ROOT_PASSWORD: pwd123
  ports:
    - 27026:22002
  networks:
    - mongo
  volumes:
```

```
      - mongo_data:/data/db
    command: mongod --dbpath /data/db --shardsvr --replSet shard2
    deploy:
      restart_policy:
        condition: on-failure
        delay: 3s
    depends_on:
      - configsvr2

  shard3_arbiter:
    image: mongo
    networks:
      - mongo
    command: mongod --noauth --dbpath /data/db --shardsvr --replSet
shard3
    depends_on:
      - configsvr2

volumes:
  mongo_data:
  config_data:

networks:
  mongo:
driver: overlay
```

A3. Composefile für swarm-worker2: stack3.yml

```
# Use root/example as user/password credentials
version: '3.7'
services:
```

```
  configsvr3:
    image: mongo
    ports:
      - 27021:21000
    volumes:
      - config_data:/data/configdb
    networks:
      - mongo
command: mongod --noauth --configsvr --replSet csrs --dbpath /data/db


  mongos3:
    image: mongo
    ports:
      - 27018:20000
    networks:
      - mongo
    command: mongos --noauth --configdb csrs/configsvr1:21000,con-
figsvr2:21000,configsvr3:21000
    deploy:
      restart_policy:
        condition: on-failure
        delay: 3s


  shard1_arbiter:
    image: mongo
    networks:
      - mongo
    command: mongod --noauth --dbpath /data/db --shardsvr --replSet
shard1
    depends_on:
      - configsvr3


  shard2_secondary:
```

```
    image: mongo
    environment:
      MONGO_INITDB_ROOT_USERNAME: root
      MONGO_INITDB_ROOT_PASSWORD: pwd123
    ports:
      - 27027:22002
    networks:
      - mongo
    volumes:
      - mongo_data:/data/db
    command: mongod --dbpath /data/db --shardsvr --replSet shard2
    deploy:
      restart_policy:
        condition: on-failure
        delay: 3s
    depends_on:
      - configsvr3

  shard3_primary:
    image: mongo
    environment:
      MONGO_INITDB_ROOT_USERNAME: root
      MONGO_INITDB_ROOT_PASSWORD: pwd123
    ports:
      - 27030:22003
    networks:
      - mongo
    volumes:
      - mongo_data:/data/db
    command: mongod --dbpath /data/db --shardsvr --replSet shard3
    deploy:
      restart_policy:
```

```
        condition: on-failure
        delay: 3s
    depends_on:
      - configsvr3


volumes:
  mongo_data:
  config_data:


networks:
  mongo:
    driver: overlay
```

Anhang B:

Vergleichtest-Rohdaten

Point

MongoDB.

Reaktionszeit-Punkte (mb)	5	50	500	5120	10240
	<1	<1	4069	37995	118467
	<1	<1	4397	34422	117778
	<1	<1	4152	35050	118122.5
Einspeicherung (ms)	1	1	4206	35822.33333	118122.5
	21	131	1152	14987	36970
	13	71	409	11723	26236
	17	78	483	9468	30534
Select all (ms)	17	93.33333333	681.3333333	12059.33333	31246.66667
	3	13	12	19	24
explain	3	15	13	18	20
	3	9	15	16	18
Nächste Nachbarn Anfrage (ms) (10)	3	12.33333333	13.33333333	17.66666667	20.66666667
	52	369	2449	26515	75597
	49	345	2153	25170	72920
	49	366	2102	23357	70940
Bounding Box Query (ms)	50	360	2234.666667	25014	73152.33333
point (mb)	5	50	500	5120	10240
	1.41	15.15	108.41	1161.01	2350.81
	1.42	15.17	118.01	1161.15	2334.46
Speicherplatzverbrauch (mb)	1.415	15.16	113.21	1161.08	2342.635

PostGIS:

Reaktionszeit-Punkte (mb)	5	50	500	5120	10240
	197.47	1881.71	11969.56	92860.98	216163.45
	198.34	1862.08	11709.42	91816.52	222816.07
	200.94	1869.67	11832.14	92338.75	219489.76
Einspeicherung (ms)	198.9166667	1871.153333	11837.04	92338.75	219489.76
	23.47	258.87	1460.14	12407.82	29607.57
	18.5	304.2	1375.2	12184.49	27159.68
	35.99	325.98	1500.74	12311.76	27615.23
Select all (ms)	25.98666667	296.35	1445.36	12301.35667	28127.49333
	18.8	94.37	345.49	6156.49	10615.56
	16.17	91.1	328.76	6517.91	10806.61
	16.45	90.83	321.8	6343.94	10743.14

Nächste Nachbarn Anfrage (ms) (10)	17.14	92.1	332.0166667	6339.446667	10721.77
	1.04	107.79	1178.47	591.72	811.18
	9.6	108.72	814.76	518.83	698.92
	15.59	106.12	748.76	507.23	751.25
Bounding Box Query (ms)	8.743333333	107.5433333	913.9966667	539.26	753.7833333
point (mb)	5	50	500	5120	10240
	2.32	23.61	185.16	2308.42	4375.86
	2.32	23.61	185.16	2308.45	4375.86
Speicherplatzverbrauch (mb)	2.32	23.61	185.16	2308.435	4375.86

Polygon

MongoDB.

Reaktionszeit-Polygon (mb)	5	50	500	5120	10240
	<1	<1	3960	64645	127618
	<1	<1	3780	64738	134973
	<1	<1	3325	64691.5	121967
Einspeicherung (ms)	1	1	3688.333333	64691.5	128186
	19	96	1658	24292	36548
	16	60	1138	28735	39899
	16	58	924	21381	43421
Select all (ms)	17	71.33333333	1240	24802.66667	39956
	113	1099	11730	117598	234250
	116	1109	11179	116264	225875
	102	1061	11979	118534	231728
Bounding Box Query (ms)	110.3333333	1089.666667	11629.33333	117465.3333	230617.6667
polygon (mb)	5	50	500	5120	10240
	3.08	29.94	294.51	2189.11	4483.25
	3.07	29.82	294.37	2188.86	4483.72
Speicherplatzverbrauch (mb)	3.075	29.88	294.44	2188.985	4483.485

PostGIS:

Reaktionszeit-Polygon (mb)	5	50	500	5120	10240
	90.7	752.48	7785.09	120630.51	241152.99
	83.51	748.97	7872.7	119743.82	234930.8
	83.93	752.45	7716.04	120187.165	238041.895
Einspeicherung (ms)	86.04666667	751.3	7791.276667	120187.165	238041.895
	26.08	249.3	2007.64	23062.74	47336.95
	49.86	324.72	2171.71	20590.67	46454.5
	25.93	399.45	2174.19	20185.35	48036.92
Select all (ms)	33.95666667	324.49	2117.846667	21279.58667	47276.12333

	16.88	66.49	1161.55	4998.97	9367.12
	23.12	72.84	1281.03	4008.85	9614.99
	19.02	75.27	1206.93	4102.22	8644.77
Bounding Box Query (ms)	19.67333333	71.53333333	1216.503333	4370.013333	9208.96
polygon (mb)	5	50	500	5120	10240
	1.02	25.84	277.98	2826.91	5592.73
	1.02	25.84	277.98	2826.91	5592.73
Speicherplatzverbrauch (mb)	1.02	25.84	277.98	2826.91	5592.73

Rasterdaten

MongoDB.

Reaktionszeit-raster (mb)	5	50	500	5120	10240
	<1	<1	279	10465	20080
	<1	<1	258	10743	22989
	<1	<1	271	11205	19494
Einspeicherung (ms)	1	1	269.3333333	10804.33333	20854.33333
raster (mb)	5	50	500	5120	10240
	4.8	47.97	460.42	4504.77	8989.32
	4.81	47.97	460.3	4504.76	8989.44
Speicherplatzverbrauch (mb)	4.805	47.97	460.36	4504.765	8989.38

PostGIS:

Reaktionszeit-raster (mb)	5	50	500	5120	10240
	346.5	2604.49	16205.93	181469.53	573905.94
	361.19	2214.78	16423.44	206654.64	544075.55
	359.29	2279.35	15909.34	246361.61	477148.73
Einspeicherung (ms)	355.66	2366.206667	16179.57	211495.26	531710.0733
raster (mb)	5	50	500	5120	10240
	9.77	67.7	521.77	4871.3	9663.58
	9.8	67.7	521.74	4871.4	9663.58
Speicherplatzverbrauch (mb)	9.785	67.7	521.755	4871.35	9663.58

Quellenverzeichnis

Abadi, D. (2010, Aug. 31). *The problems with ACID, and how to fix them without going NoSQL*. Retrieved from DBMS MUSINGS: dbmsmusings.blogspot.com/2010/08/problems-with-acid-and-how-to-fix-them.html

Agarwal, S., & Rajan, K. (2015). Performance Analysis of MongoDB Vs. PostGIS. *Free and Open Source Software for Geospatial (FOSS4G) Conference Proceedings*, S. Article 50.

AIGeorge. (2018 年 4 月 21 日). Anleitung von MongoDB. 检索来源: Jianshu: https://www.jianshu.com/p/50f3f74336c8

Aslett, M. (2012, Nov. 2). *Updated database landscape graphic*. Retrieved from the451group: blogs.the451group.com/information_management/2012/11/02/updated-database-landscape-graphic/

Baumann, P. (2014, Aug.). *Big Geo Data: Standards and Best Practices.* Washington: IEEE.

Brewer, E. (2000, Jan.). *Towards Robust Distributed Systems.* Retrieved from http://www.eecs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf

Brewer, E. (2012, Jan. 17). CAP twelve years later: How the "rules" have changed. *Computer*. Retrieved from InfoQ.

Chauhan, D. (2017). Using the Advantages of NOSQL: A Case Study on MongoDB. *International Journal on Recent and Innovation Trends in Computing and Communication*, 90-93.

Chodorow, K. (2011). *50 Tips and Triccks for MongoDB Developers.* O'Reilly Media.

DB-Engines. (Mai 2019). *DB-Engines Ranking*. Von DB-Engines: db-engines.com/de/ranking abgerufen

Desjardins, J. (13. März 2019). *What Happens in an Internet Minute in 2019?* Von Visual Captialist: www.visualcapitalist.com/what-happens-in-an-internet-minute-in-2019/ abgerufen

Docker. (2019). *Product manuals*. Von Docker docs: https://docs.docker.com/ee/ abgerufen

Edlich, S., Friedland, A., Hampe, J., & Brauer, B. (2010). *NoSQL - Einstieg in die Welt nichtrelationaler Web 2.0 Datenbanken.* München: Carl Hanser Verlag.

Elmasri, R., & Navathe, S. (2005). *Grundlagen von Datenbankensystemen.* Pearson Studium.

Evans, E. (2009, May 12). *NOSQL 2009*. Retrieved from Eric Evans's Weblog: blog.sym-link.com/2009/05/12/nosql_2009.html

GeY. (2017). Researching on MongoDB query optimization. Nanjing: Thesis Submitted to Nanjing University of Posts .

Gilbert, S., & Lynch, N. (2002). *Brewer's conjecture and the feasibility of consistent, available, partition tolerant web services.* Retrieved from https://users.ece.cmu.edu/~adrian/731-sp04/readings/GL-cap.pdf

GitHub. (May 2019). *TOPDB Top Database index.* Von GitHub: pypl.github.io/DB.html abgerufen

Hahlen, J., Wernitz, J., & Beck, C. (05.2012). *Georeferenzierung von Daten - Situation und Zukunft der Geodatenlandschaft in Deutschland.* Berlin: Rat für Sozial- und Wirtschaftsdaten (RatSWD).

Henry, O. B. (15. 6 2018). *How to Optimize Performance of MongoDB.* Von Serverlnines: https://severalnines.com/blog/how-optimize-performance-mongodb abgerufen

Heuer, A., Saake, G., & Sattler, K. (2003). *Datenbanken kompakt.* Bonn: mitp-Verlag.

Holfmann, D. (2017).

Inden, M. (2016). *Der Java-Profi: Persistenzlösungen und REST-Servicees.* Heidelberg: dpunkt.verlag GmbH.

Kleuker, S. (2016). *Grundkurs Datenbankentwicklung.* Wiesbaden: Springer Vieweg.

Kumar, L. (2015). Comparative analysis of NoSQL (MongoDB) with MySQL Database. *Scientific Journal Impac Factor*, 120-127.

Latreider, H. (2018). *Konzeption und Entwicklung einer Plattform zur Echtzeitanalyse temporaler Graphen.* Hamburg: Hochschule für angewandte wissenschaften Hamburg.

Lesniak, M. (14. Dez. 2017). *5x V. Die großen fünf Merkmale von Big Data*. Von MICROMATA: www.micromata.de/blog/big-data/big-data-v5/ abgerufen

Lopez, M. (2016, 12). Integration of NoSQL Databases for Analyzing Spatial Information in Geographic. *ResearchGate*, p. 354.

Mata-Toledo, R., & Cushman, P. (2003). *Relationale Datenbanken.* Bonn: mitp-Verlag.

Mazumder, S. (2010, Apr. 21). *NoSQL in the Enterprise*. Retrieved from InfoQ: www.infoq.com/articles/nosql-in-the-enterprise

Meier, A. (2016). Berlin: Springer Vieweg.

MongoDB. (2010, März 8). *State of MongoDB March, 2010*. Retrieved from MongoDB: www.mongodb.com/blog/post/state-of-mongodb-march-2010

MongoDB. (2013, Aug. 27). *10gen Announces Company Name Change to MongoDB, Inc.* Retrieved from MongoDB: www.mongodb.com/press/10gen-announces-company-name-change-mongodb-inc

MongoDB. (2019). *MongoDB Manual*. Retrieved from Replica Set Members: https://docs.mongodb.com/manual/core/replica-set-members/

Neo4j. (2019). *Chapter 1. Introduction*. Von The Neo4j Operations Manual v3.5: neo4j.com/docs/operations-manual/3.5/introduction/ abgerufen

Nguyen, T.-D., & Lee, S.-W. (17. 10 2016). I/O characteristics of MongoDB and trim-based optimization in flash SSDs. *ACM New York*, S. 139-144.

PostGIS. (2019). *Introduction to PostGIS*. Von pOSTgis: https://postgis.net/workshops/postgis-intro/introduction.html abgerufen

Rahm, E., Saake, G., & Sattler, K.-U. (25. Feb. 2015). *Verteiltes und Paralleles Datenmanagement.* Berlin&Heidelberg: Springer Verlag. Von W3C: www.w3.org/RDF/ abgerufen

Rouse, M. (2016, Jun.). *graph database*. Retrieved from Tech Target: whatis.techtarget.com/definition/graph-database

Sadalage, J. P., & Fowler, M. (2012). *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence.* Addison-Wesley Professional. Retrieved from Wirtschaftsinformatik Wiki - Kewee.

Scheugenpflug, S. (2005). *Relationale und Objektrelationale Datenbankkonzeptein Geoinformationssystemen.* München: Technischen Universität München.

Schulz, M. (2016). *Untersuchungen zum Einsatz von.* Dessau.

Shao, X. (2017). *Design and Implementation R-Tree Spatial Index in MongoDB Database*. Wuhan: Wuhan University.

ShaoX. (2017). Design and Implementation R-Tree Spatial Index in MongoDB Database. Wuhan: Wuhan University.

Sun, L., Lu, Y., & Shu, J. (2015, May 18). DP²: Reducing Transaction Overhead with Differential and Dual Persistency in Persistent Memory. *ACM Digital Library*, p. 3.

Tejada, Z., Buck, A., Wilson, M., & Wasson, M. (02. Dez. 2018). *Non-relational data and NoSQL*. Von Microsoft Azure: docs.microsoft.com/en-us/azure/architecture/data-guide/big-data/non-relational-data abgerufen

Trelle, T. (2014). *MongoDB - Der praktische Einstieg.* Heidelberg: dpunkt.verlag GmbH.

Trelle, T. (28. Jul. 2015). *MongoDB für Software-Entwicckler*. Von Infomatik Aktuell: www.informatik-aktuell.de/betrieb/datenbanken/mongodb-fuer-software-entwickler.html abgerufen

Value, I. I. (kein Datum). *Analytics: The real-world use of big data.* University of oxford.

Vogels, W. (2008, Dez. 22). *Eventually Consistent - Revisited*. Retrieved from All Thins Distributed: www.allthingsdistributed.com/2008/12/eventually_consistent.html

WangL. (2006). Summa of Benchmark Performance Test. 计算机工程与应用, 页 45-48.

Zhao, Y. (2013). Research on MongoDB Design and Query Optimization in Vehicle Management Information System. *Applied Mechanics and Materials*, S. 246-247.

ZhaoL. (2016). The Research of High Availability and PerformanceOptimization of Distributed MongoDB Clusters. Chengdu: University of Electronic Science and Technology of China.