0.1 ToDo

- CAP Theorem Kapitel vervollständigen
- etc.

0.2 Nodes

- Ein verteiltes System ist in der Regel auch besser skalierbar als ein einzelner Computer, da man auf einfache Art und Weise durch Hinzufügen weiterer Rechner die Leistungsfähigkeit erhöhen kann.
- Ein häufig anzutreffendes Szenario ist natürlich auch die Bereitstellung von entfernten Ressourcen, wie es bei der Wikipedia der Fall ist. Außerdem werden verteilte Systeme zur Erhöhung der Ausfallsicherheit benutzt, indem bestimmte Funktionalitäten von mehreren Rechnern angeboten werden (Redundanz), so dass beim Ausfall eines Rechners die gleiche Funktionalität von einem weiteren Rechner angeboten wird.
- etc.

  Weitere Gründe:

  - Fernzugriff auf bestimmte Ressourcen (Drucker, ...)
  - Kooperation (Computer Supported Cooperative Work)
  - Lastverteilung

- In verteilten Speichersystemen werden Daten mehrfach über verschiedene Server repliziert, um die Verfügbarkeit der Daten zu erhöhen und die Zugriffzeiten zu verringern.
- *George Coulouris:* „Ein verteiltes System (VS) ist ein System, in dem sich HW- und SW- Komponenten auf vernetzten Computern befinden und miteinander über den Austausch von Nachrichten kommunizieren“.

Vorwort

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Codeimplementierung

Sämtliche Codes liegen unter:
`https://github.com/vlxxxfa/vlxxxfa.github.io`

Inhaltsverzeichnis

Kapitel 1

NoSQL-Systemen

1.1 Definition

Im Vergleich zu den relationalen Datenbanken, die sich als eine strukturierte Sammlung von Tabellen (den Relationen) vorstellen, in welchen Datensätze abgespeichert sind, eignen sich NoSQL Datenbanken zur unstrukturierter Daten, die einen nicht-relationalen Ansatz verfolgen. Typische Beispiele für unstrukturierte Daten sind: Benutzer- und Sitzungsdaten, Chat-, Messaging- und Protokolldaten, Zeitreihendaten wie IoT- und Gerätedaten sowie große Objekte wie Videos und Bilder.[1]

Der Begriff NoSQL steht nicht für 'kein SQL', sondern für 'nicht nur SQL' (Not only SQL). Das Ziel von NoSQL ist, relationale Datenbanken sinnvoll zu ergänzen, wo sie Defizite aufzeigen. Entstanden ist dieses Konzept in erster Linie als Antwort zur Unflexibilität, sowie zur relativ schwierigen Skalierbarkeit von klassischen Datenbanksystemen, bei denen die Daten nach einem stark strukturierten Modell gespeichert werden müssen.[2] Dokumentdatenbanken gruppieren die Daten in einem strukturierten Dokument, typischerweise in einer JSON-Datenstruktur. Auch MongoDB, siehe dazu Kapitel 1.5 verfolgt diesen Ansatz und bietet darauf aufbauend eine reichhaltige Abfragesprache und Indexe auf einzelne Datenfelder. Die Möglichkeiten der Replikation und des Shardings zur stufenlosen und unkomplizierten Skalierung der Daten und Zugriffe macht MongoDB auch für stark frequentierte Websites äußerst interessant.([2], Kapitel 14, Seite 435)

[1] NSQL-Datenbanken: `http://de.basho.com/resources/nosql-databases/`, zugegriffen am 20. Dezember 2016

[2] MySQL vs. MongoDB: `http://www.computerwoche.de/a/datenbanksysteme-fuer-web-anwendungen-im-vergleich,2496589`, zugegriffen am 22. Dezember 2016

1.1.1 Kategorisierung von NoSQL-Systemen

Der Abschnitt stellt verschiedene NoSQL-Systemen mit ihren Vor- und Nachteilen vor.

Dokumentenorientierte Datenbanken

Siehe Artikel `http://pi.informatik.uni-siegen.de/Mitarbeiter/mrindt/Lehre/Seminare/NoSQL/einreichungen/NOSQLTEC-2015_paper_9.pdf`

siehe [1, S. 5-8]

Key-Value-Datenbanken

Spaltenorientierte Datenbanken

Objektdatenbanken

Graphdatenbanken

1.2 ACID

bla

1.3 Das CAP-Theorem

Im Jahr 2000 hielt Brewer[3] die Keynote auf dem ACM Symposium on Principles of Distributed Computing (PODC)[4], einer Konferenz über die Grundlagen der Datenverarbeitung in verteilten Systemen[5] (Principles of Distributed Computing). In seiner Keynote stellte Brewer sein **CAP**-Theorem vor, ein Ergebnis seiner Forschungen zu verteilten Systemen

[3] Eric A. Brewer ist ein Informatik-Professor an der University of California, Berkeley und einer der Erfinder der Suchmaschine Inktomi

[4] PODC2000: `http://www.podc.org/podc2000/`, zugegriffen am 02.01.2017

[5] In einem verteilten System im Bereich Datenverarbeitung werden gespeicherte Daten mehrfach über mindestens zwei verschiedene Server repliziert und miteinander synchronisiert, um die Verfügbarkeit der Daten zu erhöhen und die Zugriffszeiten der User zu verringern.

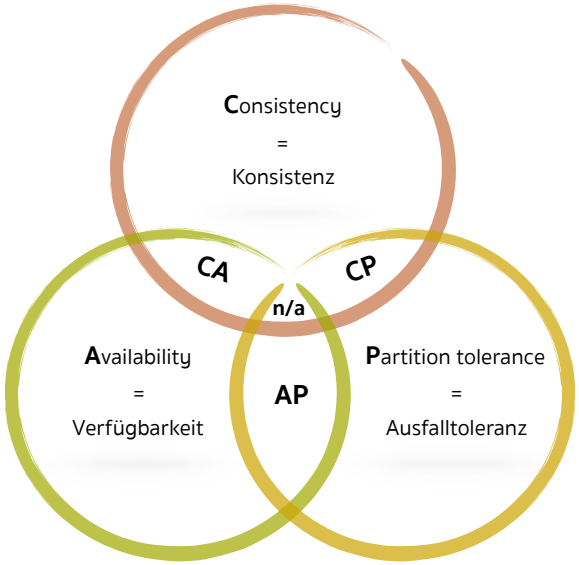

Abbildung 1.1: Anforderungen an verteilte Systeme gemäß dem **CAP**-Theorem

an der University of California [3, S. 13]. Brewer's Theorem wurde im Jahr 2002 von Seth Gilbert und Nancy Lynch formal bewiesen.

Das Akronym **CAP** steht für die englischsprachigen Begriffe **C**onsistency (Konsistenz), **A**vailability (Hochverfügbarkeit) und **P**artition Tolerance (Partitionstoleranz) und beschreiben die Anforderungen für die Skalierung an verteilte Systeme, die es zunächst näher zu erläutern gilt.

Was besagt eigentlich dieses **CAP**-Theorem? Das **CAP**-Theorem besagt, dass die verteilten Systeme, die mit großen Datenmengen zu tun haben, gleichzeitig die folgenden Anforderungen wie **C**onsistency (Konsistenz), **A**vailability (Hochverfügbarkeit) und **P**artition Tolerance (Partitionstoleranz) nicht erfüllen können.

- **C**onsistency (Konsistenz): Die *Konsistenz* der Daten in einem verteilten System bedeutet, dass alle replizierenden Knoten aus einem großen Cluster über die gleiche Datenstruktur verfügen. Falls ein Wert auf einem Knoten durch eine Transaktion per Schreiboperation geändert ist, muss der aktualisierte Wert auf Anfrage mit der Leseoperation von anderen Knoten zurückgeliefert werden können. Die Transaktion selbst ist eine atomare[6] Einheit in der Datenbank.

[6]Eine atomare Transaktion bedeutet, dass sie entweder ganz oder gar nicht ausgeführt wird. Falls eine atomare Transaktion abgebrochen wird, werden alle im Laufe der Transaktion schon durchgeführte Änderungen rückgängig gemacht.

- **A**vailability (Hochverfügbarkeit): Die *Hochverfügbarkeit* ist die weitere Anforderung, die besagt, dass immer alle gesendeten Anfragen durch User ans System beantwortet werden müssen und mit einer akzeptablen Reaktionszeit.

- **P**artition Tolerance (Partitionstoleranz): Die *Partitions- oder Ausfalltoleranz* bedeutet, dass der Ausfall eines Knoten bzw. eines Servers aus einem Cluster das verteilte System nicht beeinträchtigt und es fehlerfrei weiter funktioniert. Falls einzelne Knoten in so einem System ausfallen, wird deren Ausfall von den verbleibenden Knoten aus dem Cluster kompensiert, um die Funktionsfähigkeit des Gesamtsystems aufrecht zu halten.

Die graphische Darstellung für das Brewer's **CAP**-Theorem ist aus der Abbildung 1.1 zu entnehmen. Wie die Abbildung 1.1 erkennen lässt, können in einem verteilten System gleichzeitig und vollständig nur zwei dieser drei Anforderungen **C**onsistency (Konsistenz), **A**vailability (Hochverfügbarkeit), **P**artition Tolerance (Partitionstoleranz) erfüllt sein. Konkret aus der Praxis bedeutet das, dass es für eine hohe Verfügbarkeit und Partitions- oder Ausfalltoleranz notwendig ist, die Anforderungen an die Konsistenz zu lockern [1, S. 31].

Die Anforderungen in Paaren klassifizieren gemäß dem **CAP**-Theorem bestimmte Datenbanktechnologien. Für jede Anwendung muss daher individuell entschieden werden, ob sie als ein **CA-**, **CP-** oder **AP-**System zu realisieren ist.

- **CA** (**C**onsistency und **A**vailability): Die klassischen relationalen Datenbankmanagementsysteme (RDBMS) wie Oracle, DB2 etc. fallen in **CA**-Kategorie, die vor allem **C**onsistency (Konsistenz) und **A**vailability (Hochverfügbarkeit) aller Knoten in einem Cluster hinzielt. Hierbei werden die Daten nach dem **ACID**-Prinzip verwaltet. Die relationalen Datenbanken sind für Ein-Server-Hardware konzipiert und vertikal skalierbar. Das bedeutet, dass solche Systeme mit hochverfügbaren Servern betrieben werden und **P**artition Tolerance (Partitionstoleranz) nicht unbedingt in Frage kommt.

- **CP** (**C**onsistency und **P**artition tolerance): Ein gutes Beispiel für die Anwendungen, die zu der **CP-**Kategorie zu zuordnen sind, sind Banking-Anwendungen. Für solche Anwendungen ist es wichtig, dass die Transaktionen zuverlässig durchgeführt werden und der mögliche Ausfall eines Knotens sichergestellt wird.

- **AP** (**A**vailability und **P**artition tolerance): Für die Anwendungen, die in die **AP**-Kategorie fallen, rückt die Anforderung **C**onsistency (Konsistenz) in den Hintergrund. Beispiele für solche Anwendungen sind die Social-Media-Sites wie Twitter[7] oder Facebook[8], da die Hauptidee der Anwendung dadurch nicht verfällt, wenn zum gleichen Zeitpunkt die replizierten Knoten nicht über die gleiche Datenstruktur verfügen.

1.4 BASE

- **B**asically **A**vailable: bla
- **S**oft State: bla
- **E**ventual consistency: bla

1.5 MongoDB

Die nicht-relationale Datenbank 'MongoDB' macht mit einem effizienten Dokument-orientierten Ansatz, einfacher Skalierbarkeit und hoher Flexibilität dem bewährten MySQL[9]-System zunehmend Konkurrenz.[10]

1.5.1 Einleitung

Das System 'MongoDB' wurde 2009 vom amerikanischen Startup 10gen nach rund zwei Jahren Entwicklung der Öffentlichkeit als Open-Source-Lösung vorgestellt. Der etwas gewöhnungsbedürftige Name stammt von dem englischen Begriff "humongous", der sich ins Deutsche mit 'gigantisch' beziehungsweise "riesig"übersetzen lässt. Die Lösung basiert auf der Programmiersprache C++ und ist für die Betriebssysteme Windows, Mac OS X und Linux erhältlich. Sowohl 32-Bit- als auch 64-Bit-Systeme werden unterstützt. Wie der

[7] Twitter: `https://twitter.com/`
[8] Facebook: `https://www.facebook.com/`
[9] MySQL: `https://www.mysql.com`
[10] MySQL vs. MongoDB: `http://www.computerwoche.de/a/datenbanksysteme-fuer-web-anwendungen-im-vergleich,2496589`, zugegriffen am 19. Dezember 2016

Hersteller erklärt, ist die Lösung auf starke Leistung, große Datenmengen, hohe Flexibilität sowie einfache Skalierbarkeit ausgelegt.[11]

Die MongoDB ist eine Open-Source Software, die unter einer Apache Lizenz veröffentlicht wird.

1.5.2 Tabellen haben ausgedient - Dokumente als Datensätze

Dokumentenorientierte Datenbanken speichern Daten nicht in Form von Tabellen, sondern in Form von Dokumenten. Im Fall der MongoDB handelt es sich um Dokumente im JSON-ähnlichen Format Binary JSON oder BSON. Die nicht-relationale Datenbank MongoDB speichert Datensätze in Form von Dokumenten im BJSON[12]-Format. Statt von Tabellen spricht man bei MongoDB von Kollektionen (Collections). Jede Kollektion kann Dokumente beinhalten analog zu Zeilen beziehungsweise Datensätzen in einer MySQL-Tabelle. Dokumente werden im so genannten BSON-Format gespeichert und ausgegeben. Sie sind dann Javascript-Objekten sehr ähnlich. Das Format stammt von dem kompakten JSON-Format (Javascript Object Notation) ab und ist, wie das Präfix 'Binary' (Binary JSON = BSON) andeutet, für eine Overhead-arme Darstellung von binären Datenobjekten ausgelegt. Jedes Dokument kann dabei eine beliebige Anzahl an Feldern besitzen, während eine verschachtelte Array-Struktur ebenfalls möglich ist. Zudem dürfen Dokumente auch innerhalb eines Dokuments gespeichert werden.[13]

Der entscheidende Unterschied zu relationalen Datenbanken besteht darin, dass MongoDB als NoSQL-Datenbank dokumentenorientiert arbeitet. Dokumentenbasierende Datenbanken sind auf eine schemafreie Struktur ausgelegt. Bei MongoDB gibt es also kein festes Tabellenschema und dadurch beispielsweise auch keine zwingenden Relationstabellen und "Joins", die mit der Weiterentwicklung und dem Ausbau der Datenbank immer komplexer werden. Stattdessen lassen sich Relationen entweder direkt im Datensatz speichern oder bei Bedarf individuell bei der Datenabfrage erstellen. Dadurch ist die Datenstruktur wesentlich flexibler als bei MySQL und lässt sich einfach horizontal skalieren.

[11] MySQL vs. MongoDB: `http://www.computerwoche.de/a/mysql-vs-mongodb-datenbanksysteme-im-vergleich,1233517`, zugegriffen am 22. Dezember 2016

[12] BJSON: `http://www.bjson.org`

[13] Siehe in Deutsch: `https://www.iks-gmbh.com/assets/downloads/Einfuehrung-in-MongoDB-iks.pdf`, zugegriffen am 19. Dezember 2016

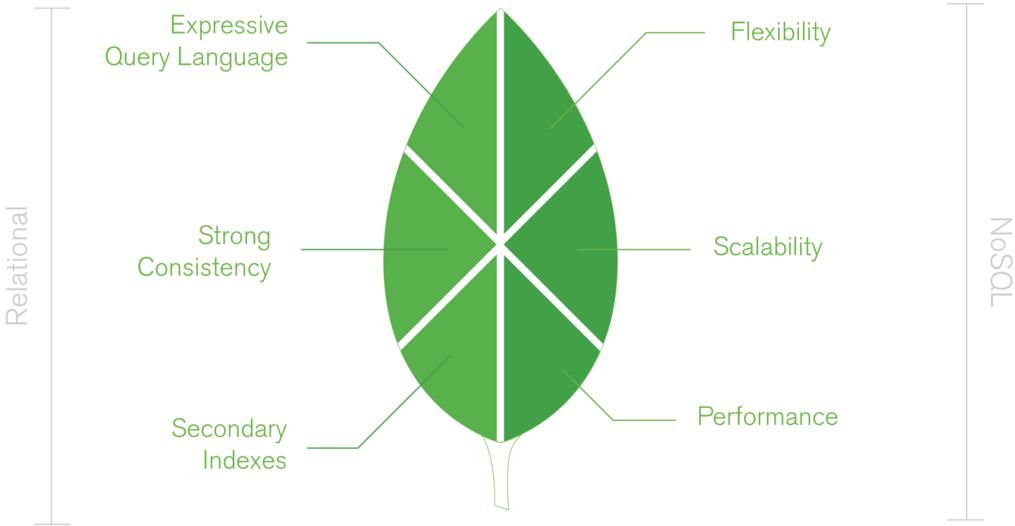

Abbildung 1.2: MongoDB Architektur[15]

1.5.3 Architektur

Zu den wichtigsten Eigenschaften, die für einen Einsatz von MongoDB sprechen, gehören:

- Hochverfügbarkeit: Auch bei Ausfall einer Datenbankinstanz soll die Applikation weiterhin verfügbar bleiben, d. h. nahtlos – ohne manuellen Eingriff – müssen redundante Instanzen bei einem Ausfall einspringen
- Skalierbarkeit: Mit transparentem Sharding, siehe Kapitel 1.5.8 – einem Verfahren zur horizontalen Skalierung – kann die Infrastruktur vergleichsweise einfach wachsenden Anforderungen angepasst werden
- Performanz: Vom Ansatz her haben dokumentenorientierte DBMS hier einen Vorteil, weil die Daten nicht erst aus mehreren Tabellen zusammengeführt werden[14]

Die Vorteile relationaler und NoSQL-Datenbanken sind aus der Abbildung 1.2 zu entnehmen.

[14]MongoDB Eigenschaften: `https://entwickler.de/online/datenbanken/mongodb-erfolgreich-ein-dokumentenorientiertes-datenbanksystem-einfuehren-115079.html`, zugegriffen am 12. Dezember 2016

[15]MongoDB Architektur: `http://www.moretechnology.de/mongodb-eine-dokumentenorientierte-datenbank/`, zugegriffen am 23. Dezember 2016

1.5.4 CRUD/IFUR

Create/Insert

Listing 1.5.1: Dokument speichern

```
> db.collection.insert(..)
```

Read/Find

Listing 1.5.2: Dokument finden

```
> db.collection.find(..)
> db.collection.findOne(..)
```

Update/Update

Listing 1.5.3: Dokument aktualisieren

```
> db.collection.update(..)
```

Delete/Remove

Listing 1.5.4: Dokument löschen

```
> db.collection.remove(..)
```

1.5.5 Schema Design

1.5.6 Performance

Siehe in Deutsch über Indexes: https://books.google.de/books?id=kRUbDAAAQBAJ&pg=PA53&lpg=PA53&dq=index+in+mongodb+was+ist+da&source=bl&ots=80Kgw664kZ&sig=rEhHo3g4JRVAVXwUr_In5xzWB8c&hl=en&sa=X&ved=0ahUKEwiBvd_VsbfQAhVDtBQKHX4_ASAQ6AEINjAC#onepage&q=index%20in%20mongodb%20was%20ist%20da&f=false

1.5.7 Aggregation Framework

1.5.8 Sharding

MongoDB bietet mit AutoSharding ein Feature, das es ermöglicht, einen Datenbankserver automatisch auf verschiedene physikalische Maschinen aufzuteilen und somit die Datenbank horizontal zu skalieren. Um das Sharding bei MongoDB zu konfigurieren, werden drei Komponenten benötigt...., siehe Sharding `https://www.iks-gmbh.com/assets/downloads/Einfuehrung-in-MongoDB-iks.pdf`

1.5.9 ODMs für MongoDB

ODM ist Object-Document Mapper für nichtrelationale Datenbanken wie MongoDB, Apache Cassandra etc.

Morphia

Morphia is the Java Object Document Mapper for MongoDB `http://mongodb.github.io/morphia/`

Doctrine

The Doctrine MongoDB ODM project is a library that provides a PHP object mapping functionality for MongoDB. `https://github.com/doctrine/mongodb-odm`

1.5.10 Beziehungen

Folgenden Relationen wie One-to-One **Teilabschnitt 1.5.10**, One-to-Many **Teilabschnitt 1.5.10** und Many-to-Many **Teilabschnitt 1.5.10** blabla

One-to-One

bla

One-to-Many

bla

Many-to-Many

bla

1.5.11 Storage Engines

MMAPv1

MMAPv1 is default a storage engine. MMAPv1 automatically allocates power-of-two-sized documents when new documents are inserted This is handled by the storage engine. MMAPv1 is built on top of the mmap system call that maps files into memory This is the basic idea behind why we call it MMAPv1.

WiredTiger

Listing 1.5.5: Alle mongod-Prozesse zwingend stoppen

```
> killall mongod
```

Für den Wahl des Storage-Engines **Wired Tiger** muss man im Terminal den Befehl aus Listing 1.5.6 ausführen.

Listing 1.5.6: Something else

```
> mongod -dbpath WT --storageEngine wiredTiger
```

Dann mongo starten mit 'mongo', dann:

Listing 1.5.7: Something else

```
> db.foo.insert({'name':'andrew'})
```

Listing 1.5.8: Something else

```
> db.foo.stats()
```

1.5.12 Indizes

Indizes in MongoDB werden als Binär-Baum[16] in einer vordefinierten Sortierreihenfolge abgelegt, siehe Abbildung 1.3

Der Index wird beim Erstellen, Updaten und Löschen eines Dokumentes auch aktualisiert. Zu viele Indizes machen schreib Operationen langsam und die Größe der Indizes steigt an, deswegen sollte ein Index nur auf Felder zeigen, auf die auch Query-Operationen angewendet werden. Beim Anlegen der Indizes ist die Sortierreihenfolge zu gunsten einer schnelleren Suche wichtig, da sie schon vorsortiert vorliegen und nicht erst bei der Liveabfrage sortiert werden müssen.[17]

Es gibt drei Möglichkeiten der Sortierung:

- Aufsteigend(1),
- Absteigend (-1),
- geospatial (2d).

Index hilft, Datenbanken zu optimieren. Die nachfolgende Abbildung 1.4 demonstriert

blabla

Neben dem obligatorischen Primär-Index auf dem Feld _id, das in jedem Dokument existieren und pro Collection eindeutig sein muss, können Sie in MongoDB bis zu 63 weitere Sekundär-Indizes pro Collection anlegen, um Suchanfragen zu beschleunigen. Ein Sekundär-Index kann auf einem einzelnen Feld oder einer Gruppe von Feldern angelegt werden.

Which optimization will typically have the greatest impact on the performance of a database?
Adding appropriate indexes on large collections so that only a small percentage of queries need to scan the collection.

---

[16] Binär-Baum: `http://www.hs-augsburg.de/mebib/emiel/entw_inf/lernprogramme/baeume/gdi_kap_4_1.html`, zugegriffen am 23. Dezember 2016

[17] Indizes: `http://wikis.gm.fh-koeln.de/wiki_db/MongoDB/Indizes`, aufgerufen am 24. Dezember 2016

[18] Indizes in MongoDB als Binä r-Baum: `http://www.slideshare.net/mongodb/indexing-strategies-to-help-you-scale`, zugegriffen am 23. Dezember 2016

[19] Efficiency of Index Use: `https://docs.mongodb.com/manual/indexes/`, zugegriffen am 12. Dezember 2016

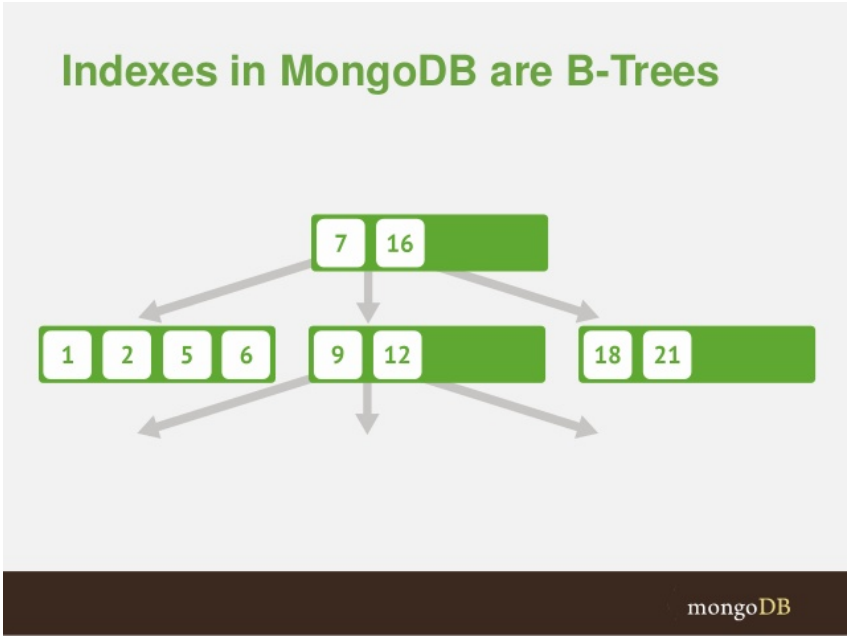

Abbildung 1.3: Indizes in MongoDB als Binär-Baum[18]

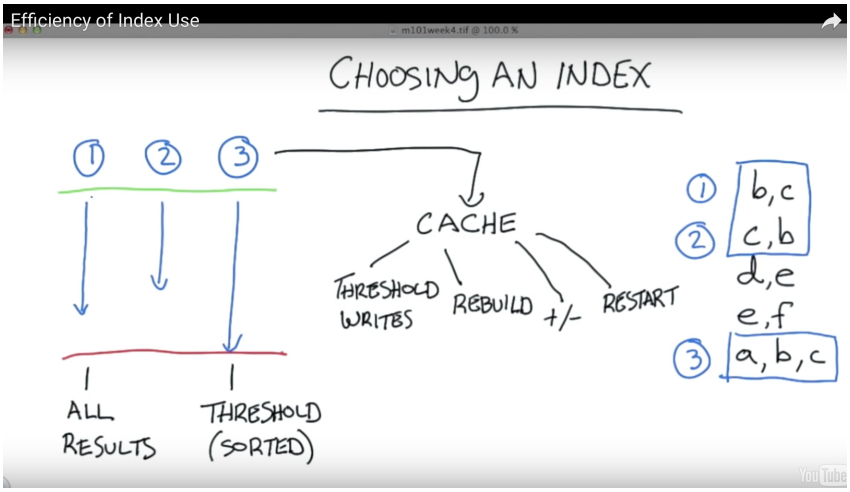

Abbildung 1.4: Efficiency of Index Use[19]

1.5.13 Creating Indexes

blabla

Listing 1.5.9: Mongo-Shell: Something else

```
> db.students.createIndex();
```

blabla

Listing 1.5.10: Mongo-Shell: Something else

```
> db.students.explain().find();
```

Quiz: Please provide the mongo shell command to add an index to a collection named students, having the index key be class, student_name. Neither will go in the 1"direction..

Listing 1.5.11: Something else

```
> db.students.createIndex({student\_name:1, class:1});
```

Listing 1.5.12: Mongo-Shell: Something else

```
> db.students.dropIndex({student_name:1});
```

Multikey Indexes

blabla

Index Creation Option, Unique

für jedes attribut kann man Unique definieren, d.h. doppelte Werte dürfen nicht vorkommen

Listing 1.5.13: Mongo-Shell: Something else

```
> db.students.createIndex({student_id : test}, {unique:true});
```

Please provide the mongo shell command to create a unique index on student_id, class_id, ascending for the collection students.

Listing 1.5.14: Mongo-Shell: Something else

```
> db.students.createIndex({student_id:1, class_id:1}, {unique:true});
```

Index Creation, Sparse

Im Fall, wenn ein Attribut nicht in allen Dokumenten vorkommt, aber für dieses ein Unique Index definiert werden soll, muss Folgendes verwendet werden:

Listing 1.5.15: Something else

```
> db.students.createIndex({cell:1}, {unique:true, sparse:true});
```

blabla, siehe den Shellbefehl, blabla

Listing 1.5.16: Something else

```
> db.students.createIndex({student_id:1, class_id:1}, {unique:true});
```

siehe Codeauszug

1.6 Prototyp

Kommentare zu den Fotos hinzufügen, Eingebettete Kommentare, siehe `http://ezproxy.bib.fh-muenchen.de:2125/doi/pdf/10.3139/9783446431225.014`

1.7 Aggregation Framework

How good is it? Mapping between SQL and Aggregation Um die Nutzung der Aggregation Framework in MongoDB zu ermöglichen, stellt MongoDB Java Driver zur Verfügung.

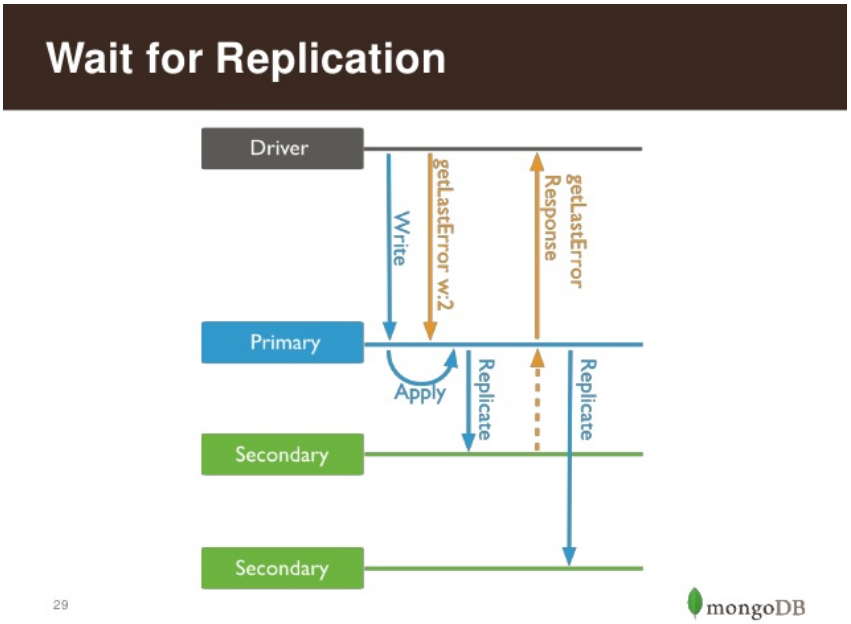

Abbildung 1.5: Replikation[20]

1.8 Replikation und Verfügbarkeit

MongoDB kann Server in Replikationsgruppen anordnen, damit bei Ausfall eines Servers die Verfügbarkeit der Datenbank trotzdem gewährleistet ist. In diesem Kapitel werden die Konfigurationsmöglichkeiten für Replikation und Verfügbarkeit der Daten veranschaulicht und zwar wie man Replikation und Verfügbarkeit in MongoDB konfiguriert.

1.8.1 Replikation (=Replication)

blabla

blabla

Typen von Replikationsgruppen:

- Regular
- Arbiter

[20]Replikation: `http://www.slideshare.net/mongodb/webinarserie-einfhrung-in-mongodb-back-to-basics-teil-3-interaktion-mit-der-datenbank`, zugegriffen am 18. Dezember 2016

[21]Master-slave replication in MongoDB: `http://spichale.blogspot.de/2013/08/replication-in-mongodb-concepts-and.html`, zugegriffen am 29. Dezember 2016

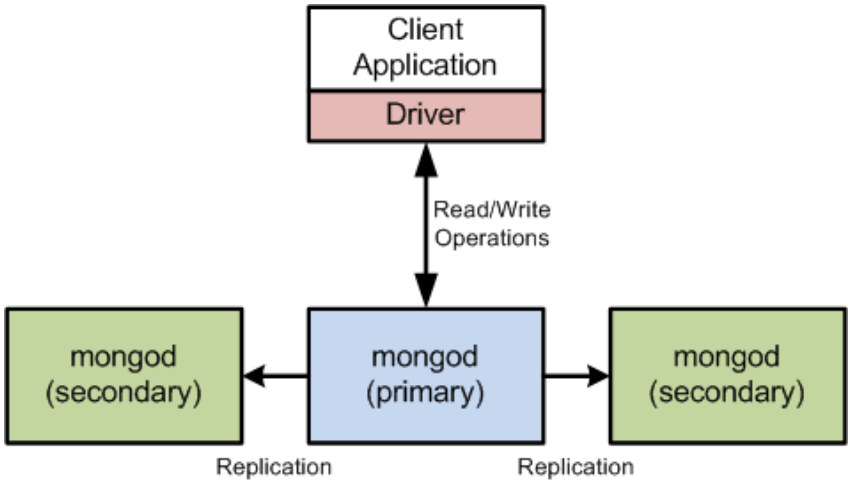

Abbildung 1.6: Master-slave replication in MongoDB[21]

- Delayed/Regular
- Hidden

1.8.2 Erstellen von Replikationsgruppen

Listing 1.8.1: Skript fürs Erstellen einer Replikationsgruppe

```
#!/usr/bin/env bash

mkdir -p /data/rs1 /data/rs2 /data/rs3

// Start von drei lokalen mongod-Instanzen als Replikationsgruppe

mongod --replSet m101 --logpath "1.log" --dbpath /data/rs1 --port 27017
--oplogSize 64 --fork --smallfiles
mongod --replSet m101 --logpath "2.log" --dbpath /data/rs2 --port 27018
--oplogSize 64 --smallfiles --fork
mongod --replSet m101 --logpath "3.log" --dbpath /data/rs3 --port 27019
--oplogSize 64 --smallfiles --fork
```

Das Skript mit dem Inhalt aus Listing 1.8.1 ist mit dem aus Listing 1.8.2 auszuführen:

Listing 1.8.2: Erstellen einer Replikationsgruppe anhand eines Skriptes

```
vlfa:scripts vlfa$ bash < create_replica_set.sh
```

Bei der Ausführung des Skriptes kann zu den Problemen führen. Um aktuelle Prozesse mit mongo anschauen und stoppen zu können, muss man folgenden Befehl angeben. The problem was that I have runned mongod without any parameters before I started launching the nodes. First kill all the mongo, mongod and mongos instances to guarantee the environment is clear.`http://stackoverflow.com/questions/25839559/mongodb-server-is-not-running-with-replset`

Listing 1.8.3: Auflistung aktueller mongo(s,d)-Prozesse

```
vlfa:scripts vlfa$ ps -ef | grep 'mongo'
```

Danach ist wichtig, Prozesse zu stoppen. Dafür muss man nach dem Befehl kill die ProzessID eingeben, siehe Listing 1.8.4. Dann wird die Möglichkeit fürs Erstellen eigener Replikationsgruppe ermöglicht, siehe dazu Listings 1.8.1 und 1.8.2.

Listing 1.8.4: mongo(s,d)-Server zwingend stoppen

```
// konkreten mongo(s,d)-Server zwingend stoppen
vlfa:scripts vlfa$ kill 'PID'

// alle mongo(s,d)-Server zwingend stoppen
vlfa:scripts vlfa$ killall mongo(s,d)
```

Damit ist die Konfigurationsgruppe mit 3 Servern angelegt. Zum Anschauen einer log-Datei;

Listing 1.8.5: 1.log-Inhalt

```
2016-12-19T14:58:11.637+0100 I CONTROL  [initandlisten] MongoDB starting :
pid=25626 port=27017 dbpath=/data/rs1 64-bit host=vlfa.fritz.box
// irrelevant
2016-12-19T14:58:11.639+0100 I CONTROL  [initandlisten] options:
{ net: { port: 27017 }, processManagement: { fork: true }, replication:
{ oplogSizeMB: 64, replSet: "m101" }, storage: { dbPath: "/data/rs1",
mmapv1: {smallFiles: true}}, systemLog: {destination: "file", path: "1.log"}}
// irrelevant
```

Die Replikationsgruppe starten.....blabla

Listing 1.8.1: Skript zum Start der Replikationsgruppe

```
config = { _id: "m101", members:[
          { _id : 0, host : "localhost:27017", priority:0, slaveDelay:5},
          { _id : 1, host : "localhost:27018"},
          { _id : 2, host : "localhost:27019"} ]
};

rs.initiate(config);
rs.status();
```

Die Server aus Listing 1.8.1 nehmen nun Kontakt miteinander auf, gründen die Gruppe und wählen den Primary-Server aus. Wie im Skript aus Listing 1.8.1 zu entnehmen ist, kann der Zustand der Replikationsgruppe mit `rs.status()` geprüft werden. Bei Ausfall des Primary-Servers wählen die Secondaries untereinander entsprechend einen neuen Primary-Server. Damit wird die Ausfallsicherheit des Servers erreicht. Die Mindestanzahl an Server in einer Replikationsgruppe liegt bei drei.

Listing 1.8.6: Skript ausführen

```
vlfa:scripts vlfa$ mongo --port 27018 < init_replica.js
```

Die Priorität '0' teilt mit, wer Primary Member in der Replikationsgruppe ist. Korrigieren, Stimmt nicht....`https://docs.mongodb.com/v3.2/core/replica-set-priority-0-member/`

Listing 1.8.1: Skript zur Initialisierung der Replikationsgruppe

```
1 public static void main (String[] args) throws InterruptedException {
2         MongoClient client = new MongoClient(asList(
3                 new ServerAddress("localhost", 27017),
4                 new ServerAddress("localhost", 27018),
5                 new ServerAddress("localhost", 27019)));
6
7                 // weitere Operationen
8 }
```

Listing 1.8.7: Simulation des Server-Ausfalls 'PRIMARY'

```
m101:PRIMARY> rs.stepDown()

Result:

2016-12-19T21:24:12.739+0100 I NETWORK  [thread1]
trying reconnect to 127.0.0.1:27018 (127.0.0.1) failed
2016-12-19T21:24:12.760+0100 I NETWORK  [thread1]
reconnect 127.0.0.1:27018 (127.0.0.1) ok
m101:SECONDARY>
```

Der aktuelle MongoDB Java Treiber ist in Version 3.4.0 verfügbar und kann bequem als Maven Dependency geladen werden, siehe Listing 1.8.2.

Listing 1.8.2: MongoDB Java Treiber in Version 3.4.0 als Maven Dependency

```
<dependency>
        <groupId>org.mongodb</groupId>
        <artifactId>mongo-java-driver</artifactId>
        <version>3.4.0</version>
</dependency>
```

Um die Sicherung der Zugehörigkeit der Mitglieder zu konkreter Replikationsgruppe festzustellen, siehe Listing 1.8.3, Zeilen 6-8...

Listing 1.8.3: Sicherung der Zugehörigkeit zu konkreter Replikationsgruppe

```
1  public static void main (String[] args) throws InterruptedException {
2          MongoClient client = new MongoClient(asList(
3                  new ServerAddress("localhost", 27017),
4                  new ServerAddress("localhost", 27018),
5                  new ServerAddress("localhost", 27019)),
6                  MongoClientOptions.builder()
7                          .requiredReplicaSetName("m101")
8                          .build());
```

1.9 Fazit

Siehe Listing 1.8.7

Doch wie der Begriff Not only SQL (NoSQL) andeutet, stehen beide Datenbanksysteme

nicht unbedingt in direkter Konkurrenz zueinander, sondern können sich gegenseitig ergänzen. Dennoch, wenn es um die persistente Datenspeicherung bei Web-Anwendungen geht, stellen relationale Datenbanken nicht mehr die einzige Alternative dar. Bei eigenen Projekten wären Entwickler heute also gut beraten, die Vor- und Nachteile der beiden Systeme gegenüberzustellen und entsprechend den eigenen Anforderungen und Prioritäten zu bewerten. Muss das System mit großen Datenmengen effizient umgehen können? Werden hohe Anforderungen an Skalierbarkeit und Flexibilität der Datenbank gestellt? Sollen sich die Daten über mehrere Server verteilen lassen? Sind häufige Änderungen an der Datenstruktur in Zukunft zu erwarten? Wenn Sie die meisten dieser Fragen mit "Ja"beantworten, dann sollten Sie sich MongoDB zumindest näher anschauen.

Daten in MongoDB verfügen über ein flexibles Schema. Kollektionen (=Collections) erzwingt keine Struktur.

Listing 1.9.1: Verbindungsaufbau

```
1 public static void main(String[] args) {
2
3   MongoClient mongoClient = new MongoClient("localhost", 27017);
4         MongoDatabase db = mongoClient.getDatabase("test");
5         MongoCollection<Document> collectionOfZips = db.getCollection("zips");
6
7         // weitere CRUD-Operationen mit der ausgewählten Kollektion
8 }
```

1.9.1 Sharding und Verfügbarkeit

Die Datenmengen in Form von Blöcken (=Chunks) wird auf `n`-Knoten verteilt. Jedes Dokument landet auf genau einem Knoten. Auf jedes Dokument wird über sog. ShardKey zugegriffen. ShardKey muss bei der Erstellung des Sharding-Systems angegeben werden. Bei der Angabe des Shard-Keys muss Folgendes ...???? berücksichtigt werden. Das Ziel des Ganzen ist die horizontale Skalierbarkeit an Datenmengen.

Test 1.9.1

1.10 Apache Cassandra

Cassandra[22] zählt, neben MongoDB[23], zu den derzeit populärsten NoSQL-Datenbanken. Cassandra war ursprünglich eine proprietäre Datenbank von Facebook und wurde 2008 als Open-Source-Datenbank veröffentlicht. Beispiele für weitere NoSQL-Datenbanken sind SimpleDB[24], Google Big Table[25], Apache Hadoop[26], MapReduce[27], MemcacheDB[28] und Voldemort[29]. Unternehmen, die auf NoSQL setzen, sind unter anderem NetFlix[30], LinkedIn[31] und Twitter[32].[33]

Cassandra ist als skalierbares, ausfallsicheres System für den Umgang mit großen Datenmengen auf verteilten Systemen (Clustern) konzipiert. Sie ist die beliebteste spaltenorientierte NoSQL-Datenbank und im Gegensatz zu MongoDB (C++) in Java geschrieben. Aufgrund ihrer architektonischen Eigenschaften wird Cassandra häufig in Big-Data-Projekten eingesetzt, kann in Zusammenarbeit mit einem Applikations-Server/Framework aber auch gut für komplexe Webanwendungen verwendet werden.

[22] Apache Cassandra: `http://cassandra.apache.org`, zugegriffen am 16. Dezember 2016
[23] MongoDB: `https://www.mongodb.com`, zugegriffen am 16. Dezember 2016
[24] SimpleDB: `https://aws.amazon.com/de/simpledb/`, zugegriffen am 16. Dezember 2016
[25] Google Big Table: `https://research.google.com/archive/bigtable.html`, zugegriffen am 16. Dezember 2016
[26] Apache Hadoop: `http://hadoop.apache.org`, zugegriffen am 16. Dezember 2016
[27] MapReduce: `http://hortonworks.com/apache/mapreduce/`, zugegriffen am 16. Dezember 2016
[28] MemcacheDB: `http://memcachedb.org`, zugegriffen am 16. Dezember 2016
[29] Voldemort: `http://www.project-voldemort.com/voldemort/`, zugegriffen am 16. Dezember 2016
[30] NetFlix: `https://www.netflix.com/de-en/`
[31] LinkedIn: `https://www.linkedin.com/feed/`
[32] Twitter: `https://twitter.com/?lang=en`
[33] NoSQL: `http://www.searchenterprisesoftware.de/definition/NoSQL`, zugegriffen am 16. Dezember 2016

Literaturverzeichnis

[1] S. Edlich. *NoSQL: Einstieg in die Welt nichtrelationaler Web 2.0 Datenbanken*. 2., aktualisierte und erw. Aufl. München: Hanser, 2011.

[2] A. Hollosi. *Von Geodaten bis NoSQL: leistungsstarke PHP-Anwendungen: Aktuelle Techniken und Methoden für Fortgeschrittene*. München: Hanser, 2012.

[3] O. Kurowski. *CouchDB mit PHP*. Frankfurt am Main: Entwickler.press, 2012.

Abbildungsverzeichnis

Quelltextverzeichnis