NoSQL und ODM

Ausarbeitung mündliche INSY Matura

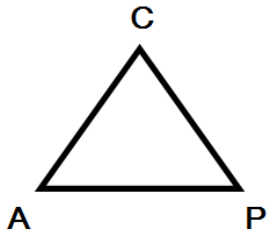
von René Hollander 5BHIT

25.04.16

Inhaltsverzeichnis

Allgemein NoSQL und ODM

CAP-Theorem



Das CAP-Theorem besagt, dass es in einem verteilten System unmöglich ist, dass gleichzeitig die drei Eigenschaften Konsistenz, Verfügbarkeit und Partitionstoleranz garantiert werden können. Nur zwei dieser Eigenschaften können gleichzeitig Garantiert werden. [1]

Konsistenz (Consistency)

Die Konsistenz der gespeicherten Daten. In verteilten Systemen mit replizierten Daten muss sichergestellt sein, dass nach Abschluss einer Transaktion auch alle Replikate des manipulierten Datensatzes aktualisiert werden. Diese Konsistenz sollte nicht verwechselt werden mit der Konsistenz aus der ACID-Transaktionen, die nur die innere Konsistenz eines Datenbestandes betrifft. [1]

Verfügbarkeit (Availability)

Die Verfügbarkeit im Sinne akzeptabler Antwortzeiten. Alle Anfragen an das System werden stets beantwortet. [1]

Partitionstoleranz (Partition tolerance)

Die Ausfalltoleranz der Rechner-/Servernetze. Das System arbeitet auch bei Verlust von Nachrichten, einzelner Netzknoten oder Partition des Netzes weiter. [1]

Transaktionen

Eine Transaktion fasst eine Folge an Operationen zusammen. Erst wenn alle Operationen fehlerfrei in dieser Transaktion ausgeführt wurden, ist der Datenbestand wieder konsistent.[2]

ACID

Eine Transaktion nach dem ACID (Atomic, Consistent, Isolated, Durable) Prinzip müssen folgende Eigenschaften garantiert werden: [2]

- Atomarität:
  Eine Transaktion wird entweder komplett oder gar nicht ausgeführt.
- Konsistenz:
  Nach Ausführung der Transaktion muss der Datenbestand wieder konsistent sein
- Isolation:
  Wenn mehrere Transaktionen gleichzeitig laufen, dürfen sie sich nicht gegenseitig beeinflussen

- Dauerhaftigkeit:
  Die Auswirkungen einer Transaktionen müssen im Datenbestand dauerhaft bestehen bleiben

BASE

In verteilten Systemen gibt es Probleme, alle ACID Eigenschaften zu erfüllen. Diese Probleme werden durch das CAP-Theorem beschrieben. Deshalb wird in solchen Systemen das BASE-Prinzip (Basically Available, Soft state, Eventual consistency) verfolgt. [2]

Redundanz

Damit ein System fehlertolerant arbeitet, kann es bestenfalls versuchen, das Auftreten von Fehlern vor anderen Prozessen zu verbergen. Dies kann durch Redundanz erreicht werden.

Es gibt verschiedene Ansätze um Redundanz bereitzustellen: Informationsredundanz, zeitliche Redundanz und technische Redundanz. [2]

Um Datenbankcluster vor ausfällen zu schützen wird mithilfe von technischer Redundanz Ersatzsysteme bereitgestellt auf die im Katastrophenfall zugegriffen werden können. Im Falle eines Ausfalles wird dann einfach ein anderer Server genutzt, der die selben Daten bereitstellt. [2]

Backups

Damit im Katastrophenfall beschädigte oder inkonsistente Daten wiederhergestellt werden können, werden in regelmäßigen Zeiträumen automatische Backups erstellt.

Datenformate

Um die gespeicherten Daten einfach verarbeiten zu können wurden verschiedene Datenformate spezifiziert. Mithilfe dieser Datenformate können heterogene Systeme einfach miteinander kommunizieren.

Grundsätzlich werden die Formate in Binäre und Textuelle Formate eingeteilt. Binäre Formate können einfacher und schneller von Maschinen verarbeitet werden. Oft sind Daten die in einem binären Format gespeichert oder übertragen werden kleiner und benötigen somit weniger Speicher und können schneller übertragen werden. Ein großer Nachteil ist, dass ein Binäres Format nicht „Human Readable“ sind, das heißt, dass sie nicht ohne größeren Aufwand vom Menschen gelesen werden können. Text basierte Formate haben den großen Vorteil, dass sie leicht von einem Menschen gelesen werden können. Computer benötigen für die Verarbeitung von Text basierten Formaten oft länger und sind dabei nicht so effizient wie bei binären Formaten.

JSON

Die Javascript Object Notation ist mittlerweile das am weitesten verbreitet Format zum Austausch von Informationen im Internet. Viele NoSQL Systeme erlauben es, die Daten als JSON Dokumente

direkt in der Datenbank abzulegen. Ein Vorteil und auch Nachteil ist es, dass dabei kein Schema zum Einsatz kommt. [3]

JSON Dokumente sind nicht in ihrer Größe beschränkt und können unendlich tief sein.

Datentypen

- Object:
  Ein Objekt hält Informationen als Key-Value Paare. Ein Schlüssel ist immer ein String und der Wert kann ein beliebiger anderer Datentyp sein. So lassen sich Objekte in einem Objekt speichern.
- Array:
  Ein Array speichert die Daten als geordnete Liste. Die Daten können wieder ein beliebiger Datentyp sein.
- String:
  Eine Unicode Zeichenfolge
- Number:
  Eine Gleitkommazahl mit doppelter Genauigkeit
- true, false:
  Boolscher Wahrheitswert
- null:
  Ein Wert ohne Wert

Binärdateien können in JSON nur kodiert als zum Beispiel Base64 String gespeichert werden.

XML

Obwohl mittlerweile JSON das am meisten verwendete Repräsentationsformat ist, kann die Nutzung von XML dennoch einige Vorteile mit sich bringen. Ältere Systeme unterstützen eher XML als Format und es kann ein Schema definiert werden. XML Daten können zur Validierung gegen das Schema geprüft werden. So kann man leicht herausfinden, ob die Daten mit denen man arbeitet auch dem entspricht was man sich erwartet.

Andere Formate

BSON

Binary JSON ist ein Format, dass JSON Dokumente als Binärdaten darstellt um einfacher von Maschinen verarbeitet werden zu können. Das BSON Format wird vor allem zur Speicherung der Dokumente in MongoDB genutzt. [4]

Protobuf

Protocol Buffers ist ein Format, dass durch ein Schema definiert wird. Aus diesem Schema wird dann der Code generiert, um Objekte in einer der unterstützen Programmiersprachen, zum Beispiel Java, in Binärdaten zu serialisieren. Die serialisierten Binärdaten können dann in einer anderen Programmiersprache mit den generierten Serialisierern wieder gelesen und verwendet werden. Ziel dieses Dateiformates ist, dass es schnell und simpel sein soll. [5]

Normalformen

1. Normalform

2. Normalform

3. Normalform

Optimierung des Datenmodelles

Fetching

Eager

Lazy

Horizontale Skalierung

Sessions

Ebay Architektur

https://www.google.at/search?q=ebay+architecture+explained&oq=ebay+architecture+explained&aqs=chrome..69i57j69i64.5705j0j2&sourceid=chrome&ie=UTF-8

Map Reduce

NoSQL Datenbanken

Der Terminus NoSQL wird vor allem verwendet um Datenbanksysteme zu beschreiben die keine klassischen Relationalen DBMS sind. Bei NoSQL geht es in erster Linie darum, Alternativen für RDBMS zu entwickeln, wo eben solche klassischen Systeme ungeeignet sind. [6]

Motivation

Vor allem die ersten sogenannten Web 2.0 Startups setzten statt auf teure DBMS von Oracle auf eigens entwickelte Systeme die besser für das speichern von großen und speziellen Daten geeignet sind. Diese DBMS wurden mit der Zeit frei zur Verfügung gestellt. [6]

Es gibt einige Gründe warum NoSQL Systeme entwickelt wurden und auch werden. Klassische RDBMS sind sehr komplex und bieten viele Funktionen und implementieren meist das ACID-Prinzip. Für bestimmte Einsatzzwecke wird eine Vielzahl dieser Features nicht benötigt. [6]

NoSQL Systeme sind oft auch auf große Datenmengen und hohen Durchsatz ausgelegt. Um diese Eigenschaften umsetzen zu können wird auch auf horizontale Skalierbarkeit geachtet. Auch sind die Systeme darauf ausgelegt, auf vielen „schwächeren“ Maschinen zu arbeiten als auf einer „starken“. [6]

Auch die Vermeidung von rechenlastigem Objekt Relationalen Mapping durch die Speicherung der Daten in Dokumenten oder Key-Value Paaren ist ein Designkriterium für einige NoSQL DBMS. Vor allem Anwendungen mit einfachen Datenstrukturen die ohne Relationen auskommen profitieren von dieser Herangehensweise. [6]

Anwendungsgebiete

NoSQL Datenbanken eignen sich vor allem für Daten die nicht von den Vorteilen eines RDBMS profitieren. Daten die transient, zum Beispiel HTTP Sessions, können in schnellen In Memory Key-Value Stores wie Redis abgelegt werden. Informatinen die oft gelesen aber selten verändert werden, können in HBase, einem DBMS basierend auf Apache Hadoop, abgelegt werden und ganze JSON Dokumente können in MongoDB abgelegt werden. [6]

Jedes dieser DBMS hat seine Vor- und Nachteile und muss nach dem Einsatzzweck evaluiert und ausgewählt werden. Dabei sollten auch klassische RDBMS in Betracht bezogen werden.

Entwurf

http://www.umlzone.com/articles/nosql-data-modeling/

https://highlyscalable.wordpress.com/2012/03/01/nosql-data-modeling-techniques/

http://martinfowler.com/articles/schemaless/

http://www.ebaytechblog.com/2014/10/10/nosql-data-modeling/

ER, UML, …

Konzeptioneller und Logischer Entwurf

Architekturen

Nach der Klassifizierung von Yen ([7]) können NoSQL Systeme in die folgenden Kategorien eingeteilt werden:

- Key-Value-Cache
- Key-Value-Store
- Eventually-Consistent Key-Value-Store
- Ordered-Key-Value-Store
- Data-Structures Server
- Tuple Store
- Object Database
- Document Store
- Wide Columnar Store

Dokumentorientiert Datenbanken

Dokumentorientierte Datenbanken unterstützen die Grundlegenden CRUD (Create, Read, Update, Delete) Funktionen. Dabei werden schemalose Dokumente abgelegt und über einen eindeutigen Schlüssel referenziert. Schemalos heißt, dass jedes Dokument in einer Sammlung unterschiedlich sein kann. So können, sofern benötigt, einfach neue Informationen zu einem Dokument hinzugefügt werden, ohne das Schema der gesamten Datenbank zu verändern. Dokumente werden meist im JSON oder XML Format gespeichert.

Beispiel für ein JSON Dokument:

```
{
  "id": "550e8400-e29b-11d4-a716-446655440000",
  "name": {
    "first": "Max",
    "last": "Exemplaryman"
  },
  "dateofbirth": "01/01/1970"
}
```

Schlüssel

Viele dokumentorientierte Datenbanken generieren einen eindeutigen Schlüssel, sofern nicht ein eigener angegeben wurde, in der Art einer UUID (Universally Unique Identifier), um Dokumente zu identifizieren. So kann verhindert werden, dass es in einer verteilten Datenbank zu Kollisionen der Schlüssel kommt. Damit der Zugriff schneller ist, wird ein Index für diese ID erstellt.

Organisation von Dokumenten

Dokumente werden meist in Collections (Sammlung) eingeteilt. Theoretisch ist es möglich, Dokumente mit unterschiedlichem Schema in einer Collection abzulegen, in der Praxis werden aber Dokumente für andere Informationen, zum Beispiel für Kunden und Waren, in verschiedenen Collections abgelegt um Abfragen einfacher und übersichtlicher gestalten zu können.

Referenzierung von Daten

Um Informationen miteinander zu verbinden, können Referenzen erstellt werden. Dabei ist zu beachten, dass das DBMS nicht überprüft ob diese Referenz möglich ist.

Beispiel:

```
{
  "id": "550e8400-e29b-11d4-a716-446655440000",
  "name": {
    "first": "Max",
    "last": "Exemplaryman"
  },
  "dateofbirth": "01/01/1970"
}
{
  "id": "9beefea2-df2a-47a8-ae99-07a61e8e9816",
  "user_id": "550e8400-e29b-11d4-a716-446655440000",
  "articles": [
    "Bread",
    "Butter",
    "Milk"
  ]
}
```

Die Auflösung der Referenzen wird dabei, zumindest bei MongoDB, nicht vom DBMS sondern vom Client gemacht. Ein gutes ODM System macht das alles automatisch.

Abfragen von Daten

Abfragen, Queries, von Daten sind bei den meisten dokumentorientierten Datenbanken möglich. Dabei können zum Beispiel alle Dokumente die einen bestimmten Wert für ein Attribut im Dokument haben abgefragt werden.

Beispiel für eine Abfrage: (Finde alle Benutzer mit dem Geburtsdatum 01/01/1970)

```
database.collections.users.find({dateofbirth: "01/01/1970"});
```

Eigenschaften

Dokumentorientierte Datenbanken haben im gesamten eine sehr hohe Performance und Skalierbarkeit. Durch die schemalosen Dokumente kann eine sehr hohe Flexibilität bei einer niedrigen Komplexität erreicht werden. Darunter leidet aber die Funktionalität dieser DBMS. [8]

Key-Value Datenbanken

Bei Key-Value Datenbanken kann man weiter zwischen Key-Value-Cache und Key-Value-Store unterscheiden. Bei einem Key-Value-Cache werden die Daten nur im RAM gespeichert und nicht persistiert. Vorteil dabei ist eine sehr hohe Zugriffsgeschwindigkeit. Nachteile sind der teure und begrenzte Speicher, RAM kostet pro GB um das 3-4 fache mehr als Enterprise SSDs, sowie der Verlust der Daten bei einem Ausfall des Servers.

Ein Key-Value-Store hingegen persistiert die Daten auf eine Festplatte und ist so vor Datenverlust geschützt.

Key-Value-Caches werden vor allem verwendet, um vergängliche Daten wie HTTP Session Informationen für mehrere Server zugänglich zu machen. Ein Key-Value-Cache kann aber auch verwendet werden, um Zugriffe auf das primäre DBMS oder den primären Datenspeicher zu verringern.

Datenmodell

Informationen werden in einer Key-Value Datenbank wie der Name schon sagt als Schlüssel mit einem dazugehörigen Wert gespeichert. Der Wert kann ein beliebiges Format haben, oder durch ein durch das DBMS spezifiziertes Format repräsentiert werden. So erlaubt es zum Beispiel Redis Listen zu verwalten, die mithilfe von Befehlen verändert werden können.

Eigenschaften

Key-Value Datenbanken haben eine sehr hohe Performance und Skalierbarkeit. Einige der Systeme können bis zu 100000 Operationen pro Sekunde auf nur einem Rechner ausführen. Da auch bei diesem Datenbanktyp Daten ohne Schema abgelegt werden können wird eine hohe Flexibilität bei niedriger Komplexität erreicht. Viele Systeme haben aber bis auf Speichern und Auslesen wenige Funktionen, eine Ausnahme ist dabei aber Redis. [8]

Spaltenorientierte Datenbanken

Graphdatenbanken

Graphendatenbanken werden genutzt, um stark vernetzte Informationen zu verwalten. Ein Graph besteht aus Knoten (Vertices) und Kanten (Edges). Ein Knoten kann Informationen halten und ist über Kanten mit anderen Knoten verbunden. Eine Kante kann wie ein Knoten Informationen enthalten.

Ein einfaches Beispiel für einen Graph sind Beziehungen zwischen Menschen:

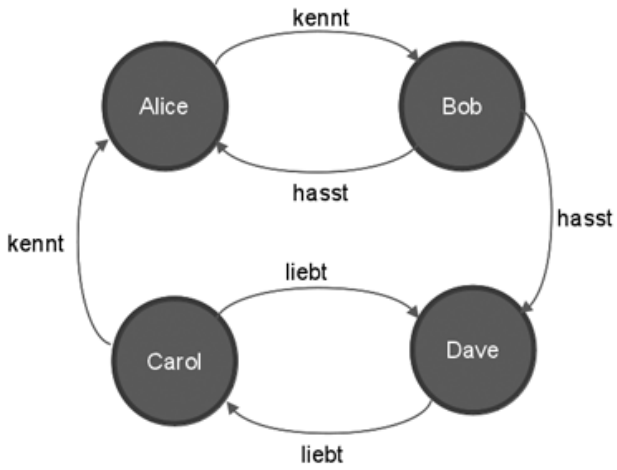


Abbildung 1: Graphendatenbank Beispiel

Der gezeigte Graph ist ein sogenannter gerichteter, benannter Multigraph. Gerichtet heißt, das die Kanten eine Richtung (Pfeil) haben. Benannt ist der Graph, da Kanten und Knoten eine Bezeichnung haben und ein Multigraph ist er, da Knoten mehrere Kanten haben.

Algorithmen

Ein Graph kann mit verschiedenen Algorithmen abgefragt werden:

- Breiten- und Tiefensuche
  Suche von Elementen im Graph
- Kürzester Pfad
  Suche des kürzesten Pfades zwischen 2 Knoten
- Eigenvektor

Abfragesprachen

Auch bei Graphdatenbanken gibt es keine standardisierte Sprache. Hier sind einige Sprachen die sich über die Zeit entwickelt haben aufgelistet:

- GraphQL: SQL Artige Sprache
- Gremlin: Für verschiedene Graphdatenbanken Systeme entwickelt (Neo4J, OrientDB, ...)
- Cypher: Speziell für Neo4J entwickelt

Andere

Objektdatenbank

Objektdatenbanken werden genutzt, um Objekte im Sinne der Objektorientierung abzuspeichern. Meist werden die Objekte einer Programmiersprache serialisiert und abgelegt. Auch hier gibt es je nach DBMS verschiedene Abfragesprachen und APIs.

Als standardisierte Sprache gibt es die Object Query Language die sehr stark an SQL angelehnt ist.

Multivalue

Tuple Space

Transaktionen und Konsistenz

Bei vielen nur BASE wenn überhaupt

Verwaltung mit Webinterface

MongoDB Cloud Manager

MongoDB Cloud Manager ist ein Tool, dass es erlaubt Statistiken zu einem MongoDB Cluster zu sammeln und auszuwerten. Auch können Probleme und Fehler im System an den zuständigen Administrator gemeldet werden. Auch kann der Cloud Manager verwendet werden um das Cluster zu automatisieren. [9]

Neben der Automatisierung gibt es eine Vielzahl an Werkzeugen zur Analyse von Queries und Verbesserung der Performance. Auch Backups können automatisiert werden, die einfach genutzt werden können um zum Beispiel Test Cluster mit Produktionsdaten zu erstellen. [9]

Die Cloud Manager Applikation wird von der MongoDB Inc. gehosted und kann 30 Tage gratis getestet werden. Danach kostet das Produkt 39$ pro Monat pro Server. [9]

Installation

Die erste Möglichkeit ist, jede MongoDB Instanz manuell zum Cloud Manager hinzuzufügen. Dies geschieht über die Command Line. In diesem Fall, ist dann keine Automatisierung des Systems möglich.[10]

Die zweite Möglichkeit ist, auf jedem Host einen Automation Client zu installieren, der mit dem Cloud Manager kommuniziert. Sobald eine MongoDB Instanz gestartet werden soll, startet der Automation Client eine Instanz und fügt den Server zum Replica Set hinzu. [10]

Interface

Einerseits gibt es ein Webinterface mit dem alle oben genannten Features abgedeckt werden, andererseits gibt es eine REST Schnittstelle um

Couchbase Web Console

Die Couchbase Web Console ist das zentrale Konfigurationswerkzeug für Couchbase Installationen. Es steht gratis und standardmäßig zur Verfügung. Sobald eine Couchbase Instanz gestartet wurde, ist die Web Console unter `http://<server_ip>:8091/` verfügbar. [11]

Einstellungen

Direkt nach der Installation von Couchbase müssen grundlegende Einstellungen vorgenommen werden, wie zum Beispiel RAM Nutzung und Buckets.

Zu den Einstellungsmöglichkeiten von der Web Console gehört auch das erstellen von Clustern und verwalten von Replikationen. Auch die Views und Daten in den Buckets können angesehen und bearbeitet werden.

Statistiken

Zu den Statistiken gehört die RAM und Speicherplatznutzung des Clusters und der einzelnen Nodes. Auch die Speichernutzung und Operationen auf die einzelnen Buckets können beobachtet werden.

Zugriff

Neben dem Webinterface gibt es eine REST API mit welcher die Einstellungen verändert werden können. Das Command Line Interface und das eigentliche Webinterface arbeiten mithilfe dieser API.

Datenzugriff

Anders als mit der Structured Query Language (SQL) gibt es für die NoSQL Systeme keine gemeinsame Sprache zur Abfrage von Daten. Jedes System hat entweder seine eigene Sprache oder eine API mit der man auf die Daten zugreifen kann.

N1QL

Mit N1QL hat Couchbase eine Abfragesprache die SQL um ein paar Funktionalitäten erweitert um mit JSON Dokumenten arbeiten zu können. Diese Sprache könnte theoretisch von anderen Systemen implementiert werden, wird aber momentan nur von Couchbase genutzt. [12]

Beispiel: [12]

Query Statement
```
SELECT * FROM databases WHERE category='NoSQL'
```

Results
```
{
  "name": "Couchbase Server",
  "version": "4.0",
  "category": "NoSQL",
  "features": [{
    "name": "N1QL",
    "capabilities": ["JOIN", "NEST", "UNNEST"]
  }]
}
```

Implementierungen

MongoDB

MongoDB ist eine schemalose, dokumentorientierte NoSQL Datenbank die großen Wert auf Skalierbarkeit, Replikation und große Datenmengen legt. MongoDB steht unter der AGPL frei zur Verwendung verfügbar und läuft auf nahezu allen Little-Endian Systemen. [13]

Aufbau

Datenbanken

Eine MongoDB Instanz kann mehrere Datenbanken verwalten. Eine Datenbank enthält wiederum mehrere Collections. [13]

Collections

Collections enthalten mehrere Dokumente. Diese Dokumente sind schemalos, das heißt, in einer Collection können Dokumente abgelegt werden, die verschieden aufgebaut sind. [13]

Neben den benutzerdefinierten Collections gibt es die System Collections die diverse Informationen zu der Datenbank enthalten und automatisch erstellt werden. [13]

Dokument

Die Daten in einem Dokument werden im weit verbreiteten JSON Format abgelegt. Dabei kommt im DBMS aber ein binäres Format namens BSON zum einsatz, um die Performance zu maximieren und Speichernutzung zu minimieren. [13]

Auch werden Eingebettete Dokumente unterstützt. Anstatt Referenzen auf andere Dokumente zu erstellen, können wie in JSON üblich weitere Dokumente abgelegt werden. [13]

Abfragen

Es können jederzeit Dokumente anhand verschiedener Kriterien abgefragt werden, zum Beispiel: nach dem Wert eines Feldes, ob ein Zahlenwert größer oder kleiner als ein anderer ist, ob sich ein Element in einem Array befindet, und viele mehr. Dabei wird keine standardisierte Sprache wie SQL benutzt, sondern eine eigene genutzt, die eher einer Programmiersprache wie Javascript ähnelt. Auch können benutzerdefinierte Javascript Funktionen zur Datenabfrage an den Server übermittelt werden. [13]

Die Ergebnisse einer Abfrage werden mithilfe eines Cursor zur Verfügung gestellt. [13]

GridFS

GridFS ist ein Filestore der auf MongoDB aufbaut und es erlaubt das Limit von 16MB pro Dokument zu umgehen. GridFS ermöglicht es auch nur bestimmte Teile einer Datei einzusehen, ohne die ganze Datei an den Client übermitteln zu müssen. [13]

GridFS teilt dazu Dateien in 255KB Chunks auf und speichert diese mitsamt Metadaten in Dokumenten in einer Collection. [13]

Replikation

Damit Ausfälle verhindert werden können und die Last verteilt werden kann, bietet MongoDB zwei verschiedene Arten der Replikation an.

Die Master-Slave-Replikation ist veraltet und sollte nicht mehr genutzt werden. Es wird ein Master der Lese- und Schreibzugriffe verarbeitet definiert, sowie mehrere Slaves die nur Lesezugriffe bearbeiten. Slaves können auch genutzt werden um Backups der Daten anzufertigen. Jeder Node hat den kompletten Datenbestand. [13]

Die Replica-Sets sind ähnlich der Master-Slave-Replikation, sind aber in der Lage den Ausfall des Primary (Master) Nodes zu kompensieren. Auch bei der Methode gibt es nur einen Node, den Primary Node, der Schreibzugriffe ausführt. Secondary (Slave) Nodes sind nur in der Lage Lesezugriffe zu bearbeiten. Sollte aber der Primary Node ausfallen, stimmten die Secondary Nodes ab wer der neue Primary Node ist. Er über nimmt dann alle Schreibzugriffe. [13]

Sharding

Horizontale Skalierung wird durch Sharding erreicht. So kann mehr CPU Leistung sowie Speicher zur Verfügung gestellt werden. [14]

Dabei wird vom Entwickler ein sogennater Shard Key erstellt, der festlegt wie die Dokumente auf die einzelnen Shards aufgeteilt werden. Um Sharding zu konfigurieren sind verschiedene zusätzliche Dienste nötig. [14]

Ein Shard speichert die Daten und ist teil eines Replica-Sets. Die Query Routers (`mongos`) verbinden die Shards mit dem Clients und emulieren dabei eine MongoDB Instanz. Der Query

Router kümmert sich um die Abfragen und baut sich mithilfe der Daten von den einzelnen Shards die Antwort für den Client. Es können mehrere Query Router pro Cluster in Betrieb sein, um keinen Single Point of Failure darzustellen. Auch wird ein Config Server benötigt, der die Metadaten zum Cluster beinhaltet. Mithilfe dieser Metadaten werden die Anfragen beantwortet. [14]

Es gibt verschiedene Algorithmen mit denen die Daten verteilt werden können. Es gibt das Range Based Sharding bei dem die Dokumente anhand Numerischer Keys in verschiedene Chunks aufgeteilt werden. [14]

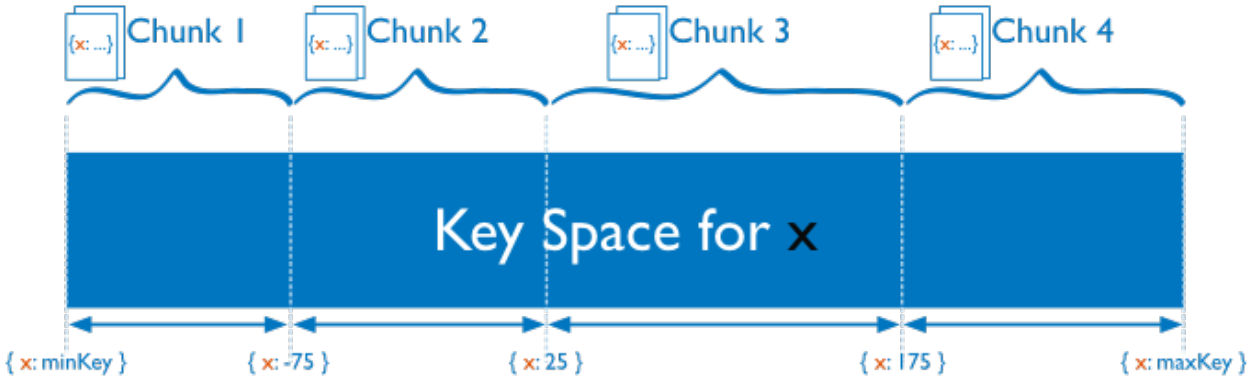


Abbildung 2: Range Based Sharding [14]

Beim Hash Based Sharding wird von einem bestimmten Feld im Dokument ein Hash erzeugt und nach diesem Hash das Dokument einem bestimmten Chunk zugeteilt. [14]

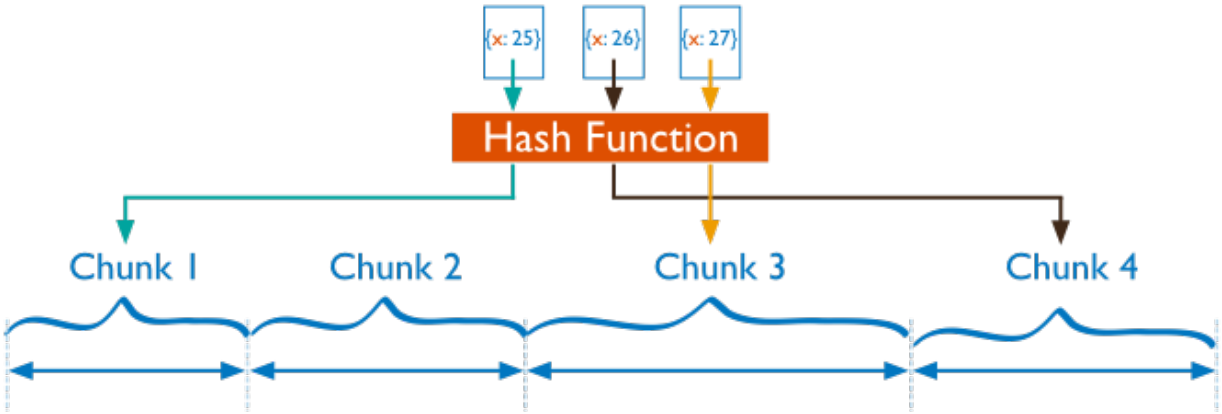


Abbildung 3: Hash Based Sharding [14]

Damit die Daten im Cluster sich nicht auf nur einen Shard konzentrieren, gibt es die Splitter und Balancer Dienste. Der Splitter teilt zu große Chunks in einem Shard auf kleinere. Der Balancer verteilt dann Chunks gleichmäßig auf die Shards damit alle Server gleich stark belastet werden. [14]

Das hinzufügen und entfernen von Shards ist möglich. Chunks werden automatisch auf neue Shards migriert. Bevor ein Shard entfernt werden kann, müssen aber erst alle Daten auf andere Shards migriert werden. Dies geschieht automatisch. [14]

Konsistenz

Die Daten auf dem Primary Node eines Replica Sets sind immer konsistent. Die Daten eines Secondary Nodes sind nur „eventually“ konsistent. Es kann also passieren, dass Daten auf einem Secondary Node veraltet sind. [15]

Transaktionen

MongoDB unterstützt keine Transaktionen, jedoch ist eine Schreiboperation auf ein Dokument atomar, selbst wenn eingebettete Dokumente vorhanden sind. Um Transaktionen möglich zu machen, empfiehlt die Dokumentation einen Two Phase Commit in der Applikation zu implementieren. [16]

Technische Aspekte

Damit der Zugriff auf die Daten möglichst schnell geschieht, werden die Daten auf die momentan zugegriffen wird im RAM behalten und alle 60 Sekunden auf die Festplatte persistiert. Sollte es in diesen 60 Sekunden zu einem Absturz kommen wären alle Daten verloren. MongoDB verwendet aber ein Journal um Datenverlust zu verhindern und trotzdem eine hohe Performance zu liefern.

Werkzeuge

Um mit einer MongoDB Instanz zu arbeiten, gibt es eine REST Schnittstelle, ein Command Line Tool und verschiedene Treiber für alle möglichen Programmiersprachen. Das Command Line Tool erlaubt es auch Lese- und Schreiboperationen auf die Daten durchzuführen. [13]

Example

Nach der Installation [17] kann man mit dem `mongo` Command Line Tool auf eine Datenbank zugreifen: `mongo <dbname>`

Sollte die Datenbank noch nicht bestehen, wird automatisch eine erstellt.

Das Command Line Tool verhält sich ähnlich einer Javascript REPL:

```
// Einfügen eines neuen Dokument in die Collection "cat"
> db.cat.insert({name: "Klaus"});
// Ausgabe aller Dokumente in der Collection "cat"
> db.cat.find();
{ "_id" : ObjectId("574f2124ade58646294c12bb"), "name" : "Klaus" }
> db.cat.insert({name: "Ottwin"});
> db.cat.find();
{ "_id" : ObjectId("574f2124ade58646294c12bb"), "name" : "Klaus" }
{ "_id" : ObjectId("574f212eade58646294c12bc"), "name" : "Ottwin" }
// Suche nach einem bestimmten Dokument nach dem Namen
> db.cat.find({name: "Klaus"});
{ "_id" : ObjectId("574f2124ade58646294c12bb"), "name" : "Klaus" }
// Bestehendes Dokument per Feld suchen und verändern
> db.cat.update({name: "Klaus"}, {name: "Klaus", birthday: new Date()});
> db.cat.find({name: "Klaus"});
{ "_id" : ObjectId("574f2124ade58646294c12bb"), "name" : "Klaus", "birthday" : ISODate("2016-06-01T17:54:47.064Z")
}
> db.cat.find();
{ "_id" : ObjectId("574f212eade58646294c12bc"), "name" : "Ottwin" }
{ "_id" : ObjectId("574f2124ade58646294c12bb"), "name" : "Klaus", "birthday" : ISODate("2016-06-01T17:54:47.064Z")
}
// Feld in allen Dokumenten verändern
> db.cat.update({}, {$set: {lives_left: 7}}, {multi: true});
```

```
> db.cat.find();
{ "_id" : ObjectId("574f212eade58646294c12bc"), "lives_left" : 7, "name" : "Ottwin" }
{ "_id" : ObjectId("574f2124ade58646294c12bb"), "birthday" : ISODate("2016-06-01T17:54:47.064Z"), "lives_left" :
7, "name" : "Klaus" }
// Wert eines bestimmten Dokumentes inkrementieren (In diesem Fall um eins dekrementieren)
> db.cat.update({name: "Klaus"}, {$inc: {lives_left: -1}});
> db.cat.update({name: "Klaus"}, {$inc: {lives_left: -1}});
> db.cat.update({name: "Klaus"}, {$inc: {lives_left: -1}});
> db.cat.find();
{ "_id" : ObjectId("574f212eade58646294c12bc"), "lives_left" : 7, "name" : "Ottwin" }
{ "_id" : ObjectId("574f2124ade58646294c12bb"), "birthday" : ISODate("2016-06-01T17:54:47.064Z"), "lives_left" :
4, "name" : "Klaus" }
// Alle Dokumente mit dem Feld lives_left größer als 5
> db.cat.find({lives_left: {$gt: 5}});
{ "_id" : ObjectId("574f212eade58646294c12bc"), "lives_left" : 7, "name" : "Ottwin" }
```

Couchbase

Was wird unterstützt? Replikation, Lastaufteilung, Transaktionen, Datenformate, ...

Cassandra

Was wird unterstützt? Replikation, Lastaufteilung, Transaktionen, Datenformate, ...

Redis

Redis ist eine Mischung aus einem Datenstrukturspeicher, Datenbank, Cache und Message Broker. Dabei wird eine Vielzahl an Datentypen unterstützt: Strings, Hashes, Lists, Sets, Sorted Sets, Bitmaps, Hyperloglogs und Raumbezogene Indizes. Auch unterstützt Redis Replikation, Lua Scripting, Transaktionen und verschiedene Level an Persistenz. [18]

Keys

Da Redis ein Key-Value Store ist, benötigt jeder zu speichernde Datensatz einen eindeutigen Schlüssel. Der Schlüssel hat als Format einen Binär Sicheren String, das heißt, dass beliebige Daten, über Text bis zu einem Bild, als Schlüssel genutzt werden können. Die maximale Länge für einen Schlüssel beträgt 512MB. Es wird aber geraten, die Schlüssel möglichst klein zu halten um Speicher zu sparen und aufwendige Vergleiche der Schlüssel zu verhindern. [19]

Die offizielle Dokumentation schlägt vor die Schlüssel im folgenden Schema zu erstellen: `object-type:id`. Beispiel: `user:1000` [19]

Mit dem EXISTS Kommando kann überprüft werden, ob ein Schlüssel bereits verwendet wird. Mit dem DEL Kommando kann der Schlüssel inklusive Wert entfernt werden. [19]

Einem Schlüssel kann auch eine Time to Live zugewiesen werden. Sobald die angegebene Zeit abgelaufen ist, wird der Schlüssel mitsamt Wert entfernt. Dafür wird das EXPIRE Kommand verwendet. Mit dem TTL Kommando kann man die Zeit ausgeben, die der Schlüssel noch zu Leben hat und mit dem PERSIST Kommando kann eine Time to Live entfernt werden, sofern der Schlüssel noch existiert. [19]

Datenformate

Es wird eine Vielzahl an Datenformaten mit spezialisierten Kommandos zur Modifizierung der Daten zur Verfügung gestellt. Im folgenden wird auf die wichtigsten Eingegangen:

Strings

Strings sind wie Keys Binär Sicher, es können wieder beliebige Daten gespeichert werden. Strings dürfen maximal 512MB groß sein. Mit dem SET und GET Kommando können diese Werte verändert werden. Mit dem APPEND Kommando kann ein weiterer Wert zum String hinzugefügt werden. Es gibt einige spezielle Kommandos, wie zum Beispiel INCR und DECR, die den String als Integer behandeln und um 1 inkrementieren oder dekrementieren. Das MSET beziehungsweise MGET Kommando kann genutzt werden um mehrere SET und GET Operationen auf verschiedenen Schlüsseln zusammenzufassen. [19]

Lists

Listen in Redis sind mithilfe einer Linked List Datenstruktur implementiert. Vorteil an einer Linked List, ist dass das Hinzufügen von Elementen am Anfang oder Ende der Liste ist konstanter Zeit verläuft. Zugriff auf einen bestimmten Index in dieser Liste ist aber aufgrund der gewählten Implementierung sehr langsam und von der Länge der Liste abhängig. Für den Fall, wo Zugriff auf Basis von einem Index nötig ist, kann ein Sorted Set verwendet werden. [19]

Mit dem LPUSH und RPUSH kann auf der linken Seite der Liste (Anfang) beziehungsweise auf der rechten Seite der Liste (Ende) ein Element hinzugefügt werden. Mit dem LRANGE Kommando können Teile der Liste abgerufen werden. Mit LLEN kann man die Länge der Liste abrufen und mit den LPOP und RPOP Befehlen können Elemente am Anfang oder Ende der Liste entfernt werden. [19]

Mithilfe des LTRIM Kommandos können Elemente die zu viel sind gelöscht werden. In Kombination mit LPUSH und LTRIM können so nur die Letzten n Elemente im speicher behalten werden. [19]

Mit dem BRPOP und PLPOP Kommando gibt es auch blockierende Varianten von POP. Wenn kein Element in der Liste enthalten ist, wird gewartet bis eines hinzugefügt wird. [19]

Hashes

Hashes können wie eine Map verwendet werden. So kann eine beliebige Zahl and Key-Value Paaren in einem Key gespeichert werden um zum Beispiel ganze Objekte abzubilden. Für Operationen auf Hashes werden die Kommandos HMSET und HGET/HMGET genutzt. Auch Spezialfälle für die Kommandos zum Inkrementieren bestehen.

Sets

Sets können für ungeordnete Daten genutzt werden. Jeder Datensatz kann nur ein Mal vorkommen. Die Befehle SADD, SMEMBERS, SISMEMBER und SPOP werden zum Hinzufügen, Auslesen,

Überprüfen und Löschen von Elementen genutzt. Mit dem SINTER Kommando können mehrere Sets miteinander verschnitten werden. Alle Werte die in jedem der angegebenen Sets vorzufinden sind, werden von diesem Kommando zurückgegeben. [19]

Sorted Set

Auch beim Sorted Set kann jeder Wert nur ein Mal vorkommen. Die Werte werden aufgrund eines Scores geordnet. Der Score ist eine Gleitkommazahl die angibt, wie die Werte geordnet sind.

Publish Subscribe

sdgfsdfgfdsfg

Persistenz

Redis speichert die Daten in erster Linie im RAM, die Persistierung der Daten ist aber möglich und kann in die folgenden Level unterteilt werden: [20]

- In Memory:
  Die Daten werden nicht persistiert und sind volatile. Wenn der Server gestoppt wird oder abstürzt, gehen alle Daten verloren.
- RDB:
  In regelmäßigen Intervallen wird ein Snapshot der Daten gemacht und persistiert. Dieses Intervall kann mit dem SAVE Kommando angepasst werden.
- AOF:
  Jedes Kommando das der Server erhält wird geloggt und persistiert. Bei einem Neustart wird dieser Log genutzt um die Daten zu rekonstruieren. Wird der Log zu groß, wird ein Rewrite Hintergrundprozess gestartet der diverse Kommandos zusammenfasst.
- AOF und RDB:
  Bei einem Neustart wird das Logfile zum Wiederherstellen der Daten genutzt.

Replikation

htdj

Partitioning

hdfj

Transaktionen

dfghj

Scripting

ghdh

LRU Cache

dfgh

Benchmarks

Performance der Systeme untereinander. Auch Einfachheit und Datenintegrität beachten!

Embedded NoSQL

Es wird immer stärker versucht, Last vom Server auf den Client zu verschieben, damit mehr Clients mit weniger Servern bedient werden können. Dies konnte vor allem durch Technologien wie Javascript und Mobile Apps erreicht werden. Um Daten am Client zu speichern gibt es verschiedene Embedded Datenbanksysteme. Solche Systeme sind auf kleinere Datenmengen ausgelegt und unterstützen weniger Funktionen als DBMS im Backendbereich.

Mit SQLite gibt es ein ein RDBMS das standardmäßig auf allen Android Systemen zu Verfügung steht. So können ohne viel Aufwand geringe Mengen an Informationen gespeichert und abgefragt werden. [21]

IndexedDB

IndexedDB erlaubt es, im Browser clientseitig große Datenmengen zu speichern. Diese Daten bleiben über Sessions hinweg persistent und eigenen sich vor allem für Webapplikation die keine Durchgehende Internetverbindung benötigen. [22]

Zum Speichern der Daten kommen Schlüssel-Wert paare zum Einsatz. Die Werte können komplexe Objekte sein und die Schlüssel Eigenschaften dieser Werte. [22]

Auch Transaktionen werden unterstützt, diese werden aber durch einen Auto-Commit beendet. Der Zugriff geschieht durch eine Synchrone oder Asynchrone Schnittstelle und verwendet keine Abfragesprache wie SQL. [22]

UnQLite

UnQLite ist eine in C geschriebene, eingebettete NoSQL Datenbank. Sie kann als Documentstore und Key-Value Store verwendet werden. Eine Datenbank wird als eine Datei auf der Festplatte gesehen und ist portabel um auf verschiedenen anderen Systeme verwendet zu werden. Um Daten nur temporär zu speichern, kann eine Memory Datenbank statt einer dateibasierten erstellt werden. Eine Datenbank kann mehrere Collections beinhalten die Dokumente im JSON Format oder Schlüssel/Wert Paare enthalten. [23]

Die Kommunikation mit dem Key-Value Store erfolgt ausschließlich über eine simple C Schnittstellt. Operationen auf den Documentstore werden mithilfe der Jx9 Skriptsprache ausgeführt. ACID konforme Transaktionen sind mit der Thread sicheren API möglich. [23]

Couchbase Mobile

Couchbase Mobile ist auf die Zusammenarbeit mit einem Couchbase Server optimiert. Es werden zwar Lokale Datenbanken unterstützt, diese können aber über den sogenannten Sync Gateway mit dem Server synchronisiert werden. So können Events ausgelöst am Client ausgelöst werden, wenn sich Daten am Server verändern. Die lokalen Datenbanken unterstützen die standardmäßigen CRUD Operationen, Abfragen und Indexing über eine native API. Diese Funktionalität ist in einem nur 500kB großen Softwarepacket, dass auf fast allen Plattformen verfügbar ist, enthalten. [24]

Couchbase Mobile enthält auch Couchbase Server. Dieser Server speichert seine Daten aber nicht lokal, sonder in der Cloud, und erlaubt es mehrere Milliarden Datensätze zu speichern. [24]

Die Daten zwischen Client und Server werden mit AES verschlüsselt übertragen. Zugriff wird durch verschiedene Authentifizierungssysteme gesichert. [24]

ODM

Ein ODM Framework wird genutzt, um Daten aus einem DBMS für den Entwickler einfach und sinnvoll zur Verfügung zu stellen. So kann der Aufwand den der Entwickler mit Validierung und Speicherung hat, durch ein automatisches System abgenommen werden. Auch Abfragen können dadurch vereinfacht werden. Ein weiterer Vorteil ist, dass verschiedene DBMS verwendet werden können, während die Verwendung des ODM Frameworks gleich bleibt.

Schema

Ein ODM Framework braucht ein vom Entwickler erstelltes Schema, indem die Attribute mit den Datentypen abgebildet sind. In diesem Schema kann wenn es unterstützt wird auch die Validierung der Daten stattfinden. Auch Attribute die auf andere Daten referenzieren müssen gekennzeichnet werden.

Vereinfachung des Zugriffes

Die Objekte die das ODM Framework aus den Daten des DBMS generiert können beliebig verändert werden. Über eine `save()` Methode kann das Objekt wieder abgespeichert werden. Das Framework kümmert sich dabei um die Validierung und Versionierung der Daten.

Validierung der Daten

Während das verwendete DBMS, vor allem bei NoSQL Systemen, schemalos ist, kann mithilfe des Frameworks eine einfache Validierung geschehen. Dabei können einerseits die Datentypen für verschiedene Felder definiert werden, aber auch Überprüfungen ob die eingegebenen Daten zulässig sind.

Probleme

Durch die Verwendung eines ODM Frameworks könnte die Performance verringert werden.

Referenzen

Die meisten NoSQL Datenbankmanagementsysteme, vor allem dokumentorientierte, sind nicht in der Lage Referenzen unter den Dokumenten abzubilden. Das ODM Framework übernimmt dabei die Auflösung der Referenzen um komplexe Objektgraphen abzufragen.

Wie diese Referenzen dargestellt werden ist vom ODM Tool abhängig. Meist wird aber der Identifier vom Ziel Objekt genutzt.

Zirkulare Abhängigkeiten

Bei der Auflösung der Referenzen durch das ODM Framework muss vor allem auf Zirkulare Abhängigkeiten geachtet werden um zu verhindern, dass verschiedene Instanzen von dem selben Objekt auftreten.

Konsistenz

Da NoSQL Systeme durch ihre Eigenschaften, vor allem Skalierbarkeit und Performance, meist keine Transaktionen unterstützen, muss der Entwickler auf die Konsistenz der Daten achten. Eine einfache Möglichkeit der dafür ist die Verwendung von Versionsnummern. Die Versionsnummer wird bei jeder Veränderung eines Objektes inkrementiert. So kann das Framework erkennen, ob sich inzwischen etwas verändert hat.

Implementierungen

Mongoose

Mongoose ist ein Javascript ODM Framework speziell für NodeJS und MongoDB entwickelt. Es werden Schemata erstellt gegen die die Daten validiert werden. Auch das Bauen von Queries wird mit dem Framework vereinfacht.

Simple Webapplikation mit NodeJS und MongoDB

Für dieses Beispiel wird koa, eine HTTP Middleware für NodeJS, und Mongoose als ODM Framework genutzt.

Erklärung

Zuerst muss die Verbindung zum DBMS aufgebaut werden. Dies geschieht über `mongoose.connect('mongodb://localhost/test');`. `localhost` ist dabei der Hostname und `test` die Datenbank.

Nun kann ein Schema erstellt werden. Für dieses Beispiel wurde ein Dokument erstellt um Katzen abzulegen. Jede Katze hat einen Namen (Erforderlich) und ein Geburtsdatum. Um ein Schema zu erstellen wird die `mongoose.model()` Funktion genutzt. Der erste Parameter gibt an wie die dazugehörige Collection in der Datenbank heißt. Der Zweite Parameter ist ein JSON Objekt mit den Eigenschaften (siehe Code).

Das erstellte Cat Objekt kann nun zum Zugriff auf die Daten genutzt werden. Es gibt eine vielzahl an Funktionen die verschiedene Queries erlauben. Eine Liste aller Funktionen gibt es in der Dokumentation[25], ein paar Funktionen wurden im Code genutzt.

Code

```
// Benötigte Module: koa: ^1.2.0, koa-body: ^1.4.0, koa-router: ^5.4.0, mongoose: ^4.4.19
const Koa = require('koa');
const KoaRouter = require('koa-router');
const KoaBody = require('koa-body');
const ObjectId = require('mongodb').ObjectId;
// Anbindung zu MongoDB erstellen
const mongoose = require('mongoose');
// In diesem Fall wird die test Datenbank genutzt
mongoose.connect('mongodb://localhost/test');
// Schema für eine Katze
var Cat = mongoose.model('Cat', {
    // Name der Katze
    name: {
        type: String, // Datentyp
        required: true, // Markiert dieses Feld als erforderlich
        index: true // Erstellt einen Index auf dieses Feld
    },
    // Geburtsdatum der Katze
    birthdate: Date
});
// Web Middleware mit Koa
var app = Koa();
var body = KoaBody();
var router = KoaRouter();
// Index, einfache ausgabe von Hello World
router.get('/', function *() {
    this.body = "Hello World!";
});
// Definition der cat API
// Alle grundlegenden CRUD Operationen
// wurden Implementiert
router.get('/cat', function *() {
    // Holt sich alle vorhandenen Katzen
    // Beispiel: GET /cat/
    this.body = yield Cat.find();
}).get('/cat/:id', function *() {
    // Holt sich das Dokument zu einer Katze
    // Beispiel: GET /cat/574cb8f6ff9915215d5cd7f3
    this.body = yield Cat.findOne({_id: this.params.id})
}).post('/cat', body, function *() {
    // Erstellt eine neue Katze mit den Informationen im Body
    // Beispiel: POST /cat/
    //           {
    //             "__v": 0,
    //             "name": "Klaus",
    //             "_id": "574cb9082d5e8d365db5771c"
    //           }
    let cat = this.request.body;
    delete cat._id;
    this.body = yield new Cat(this.request.body).save();
}).put('/cat/:id', body, function *() {
    // Updated eine bestehnde Katze
    // Beispiel: PUT /cat/574cb9082d5e8d365db5771c
    //           {
    //             "name": "Ottwin"
    //           }
    if (!ObjectId.isValid(this.params.id)) return; // ID validieren
    let cat = this.request.body;
    // _id und __v vom eingehenden Objekt entfernen um Fehler zu verhindern
    delete cat._id;
    delete cat.__v;
    this.body = yield Cat.findOne({_id: this.params.id}).update(cat);
}).delete('/cat/:id', function *() {
    // Löscht eine Katze
    // Beispiel: DELETE /cat/574cb8f6ff9915215d5cd7f3
    this.body = yield Cat.findOne({_id: this.params.id}).remove();
});
// Routen zur Middleware hinzufügen
app
    .use(router.routes())
    .use(router.allowedMethods());
// Webapplikation auf Port 3000 starten
app.listen(3000);
```

Morphia

Morphia ist ein ODM Framework für Java. Das Schema der Daten wird mit Java POJOs abgebildet. Das Framework bietet neben dem Mapping der Dokumente auch eine Query Builder die es erlauben, einfach Querys zu bauen und auszuführen.

Simple Webapplikation mit Spring und Morphia

Als Beispiel wird das Cat Example von der Mongoose Einführung in Java implementiert. Dafür wird neben Morphia Spring genutzt um eine einfache REST Schnittstelle zur Verfügung zu stellen.

Erklärung

Im ersten Schritt müssen anhand bestehender Informationen zum Schema die POJOs erstellt werden. Dazu wurde eine Klasse `Cat` im `entity` Package erstellt. Mithilfe der Attribute werden die benötigten Felder abgebildet. Mit Annotationen können weitere Informationen festgelegt werden. In diesem Fall wird die Klasse mit `@Entity("Cat")` annotiert, um Morphia mitzuteilen wie die entsprechende Collection in MongoDB heißt. Auch werden die Indizes auf das `name` Feld erstellt. Der Identifier muss mit `@Id` annotiert werden.

Mithilfe der `prePersist` Methode wird eine einfache Validierung der Daten gemacht. In diesem Fall wird überprüft, ob das `name` Feld gesetzt ist.

```
@Entity("Cat")
@Indexes(@Index(value = "name", fields = @Field("name")))
public class Cat {
    @Id
    private ObjectId id;
    private String name;
    private Date birthday;
    @PrePersist
    public void prePersist() {
        if (name == null || name.isEmpty()) throw new IllegalStateException("a name has
to be set");
    }
}
```

Weiters wurden diverse Konstruktoren und Getter und Setter Methoden erstellt damit man auf die Attribute zugreifen kann.

Sobald die Entities fertig modelliert und erstellt wurden, kann Morphia konfiguriert werden. Dafür wird eine Instanz der Morphia Klasse erstellt und dieser mitgeteilt, wo sie die Entities finden kann:

```
Morphia morphia = new Morphia();
// tell morphia where to find your classes
// can be called multiple times with different packages or classes
morphia.mapPackage("at.renehollander.morphiaexample.entity");
```

Weiters wird ein Datastore erstellt. Dieser Datastore ist die eigentliche Verbindung zur Datenbank.

```
// create the Datastore connecting to the database running on the default port on the
local host
Datastore datastore = morphia.createDatastore(new MongoClient(), "test");
// drop old database and create indizes
datastore.getDB().dropDatabase();
datastore.ensureIndexes();
```

Nun kann Morphia verwendet werden:

Erstellung einer neuen Katze und Speicherung in der Datenbank:

```
Cat cat = new Cat("Klaus");
datastore.save(cat);
```

Abfrage aller vorhandenen Katzen:

```
List<Cat> cats  = datastore.createQuery(Cat.class).asList();
```

Suche nach einer bestimmten Katze mithilfe das Namen:

```
return datastore.createQuery(Cat.class).field("name").equal("Klaus").asList();
```

Löschen einer Katze:

```
WriteResult results = datastore.delete(Cat.class, new
ObjectId("574dd0162332238280690655"));
```

MongoEngine

Erklärung und kurzes Python Example mit Webfrontend

Performance

Performance gegenüber Raw Zugriff

rubedo CMS

rubedo ist ein Content Management System, dass in PHP geschrieben wurde. Es verwendet eine NoSQL Datenbank, MongoDB, als DBMS und ElasticSearch als Such Engine. [26]

Auf ihrer Webseite behaupten sie, es sei die erste „Behavior Driven Content and Commerce soltion“. Was das bedeutet weiß ich leider nicht.

Das Framework selber ist gratis und Open Source. Neben der Installation auf dem eigenen Gerät, kann die sogenannte rCloud genutzt werden. Sie bietet rubedo als Service ab 20€ pro Monat an.

Das Github Repository von rubedo zeigt, dass es regelmäßg Commits gibt. Auch Updates werden in regelmäßigen Abständen veröffentlicht. Die Dokumentation die auf der Webseite gefunden werden kann, macht einen ersten guten Eindruck.

Installation

Um das System kurz zu testen wurde es laut der Installationsanleitung auf der Webseite auf einer Debian 8 Virtual Machine installiert. Es gibt es vollautomatisches Skript, dass die Installation in nur einem Befehl durchführt:

```
curl -Ls http://bit.ly/QuickInstallRubedo | sudo -H sh -s Your_Github_Token
```

Leider funktionierte das Skript nur auf Debian 7. Auch mussten die Wheezy backports Repositories zu apt hinzugefügt werden damit alles funktioniert.

Sobald die Installation fertig ist, muss man noch die htaccess anpassen und kann dann die Grundeinstellungen von Rubedo im Browser vornehmen.

Verwendung

Nach der Grundinstallation wird man vom Backoffice begrüßt. Es ist wie ein Desktop aufgebaut und kann dort seine Einstellungen vornehmen sowie Webseiten gestalten (siehe Abbildung).

Ohne viel in der Dokumentation nachzulesen war es möglich eine Seite mit Inhalt zu erstellen. Dafür musste zuerst eine „Site“ angelegt werden. Dies erfolgt einfach und unkompliziert im Sites Editor der über das Startmenü → Studio → Sites erreichbar ist. Nach dem klicken auf den „Add“ Button wird man von einem Wizard durch die Erstellung begleitet. Sobald das geschehen ist, kann man schon eine leere Seite unter der eingestellten Domain betrachten.

Um die Homepage anzupassen wird der „Pages“ Editor über das Startmenü aufgerufen. In der rechten Toolbar kann man Homepage anklicken um die Startseite zu bearbeiten. Mit einem Klick auf „New Block“ kann ein neues Element hinzugefügt werden. Zum Testen eignet sich ein

einfaches Text Element. Sobald es hinzugefügt wurde, kann man den Inhalt über die Toolbar auf der linken Bildschirmseite bearbeiten.

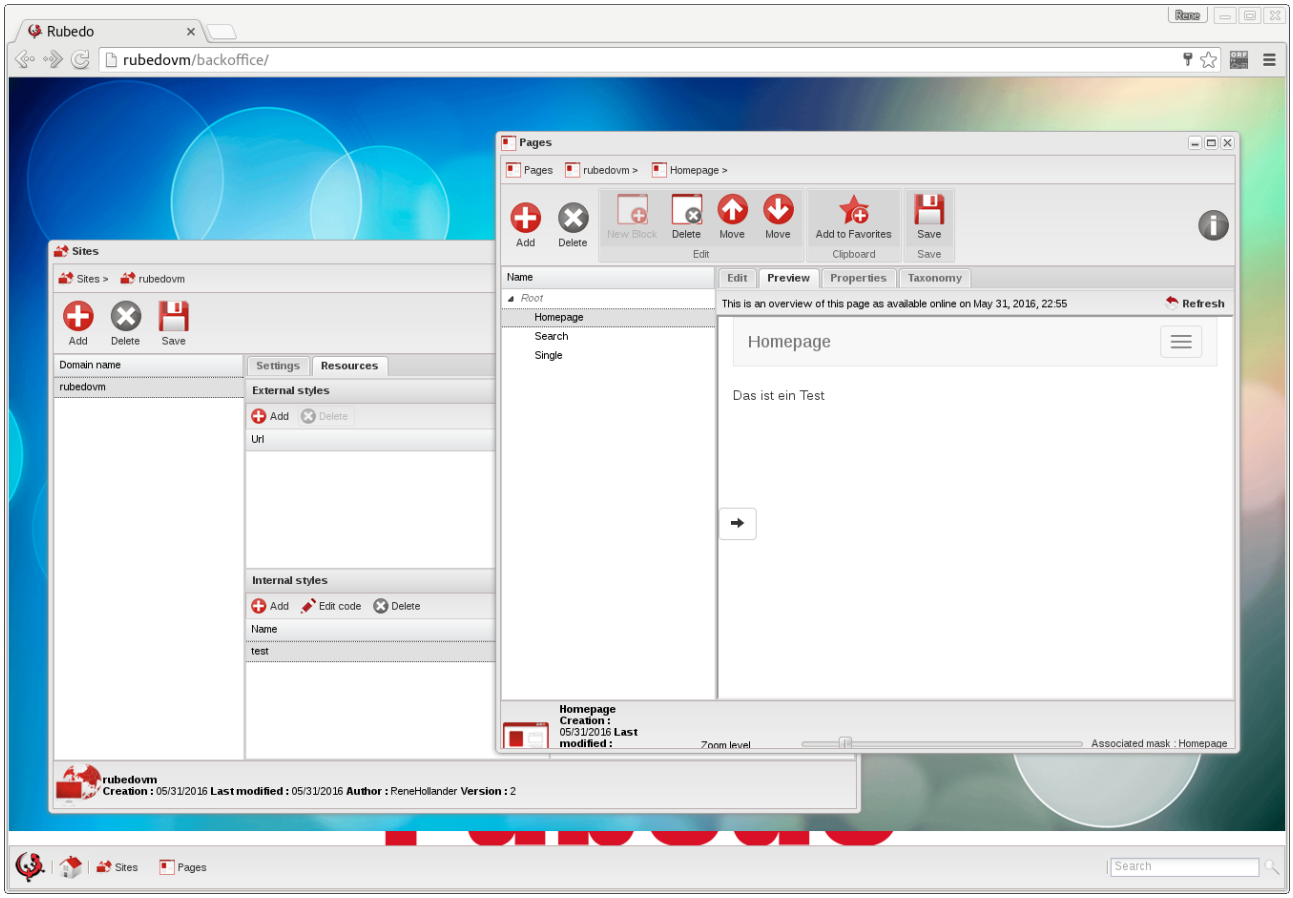


Abbildung 4: Rubedo Backoffice

Das Framework ist vor allem auf die Verwendung im e-Commerce Bereich ausgelegt.

Mit ein paar Klicks konnte eine Unterseite über Essen erstellt werden. Dafür wurde zuerst ein neuer Artikel im Contents Editor erstellt. Auch wurde ein Bild hinzugefügt.

Sobald der Artikel angelegt ist, kann im Pages Editor eine Unterseite angelegt werden. Auf der angelegten Seite wird ein Content list Block eingefügt. Wenn der Block erstellt wurde, wird in der linken Toolbar ein Simple Query erstellt, der alle Artikel aus der Articles Kategorie abfragt. Wenn dieser Query angelegt ist, kann die Arbeit im Preview Tab geprüft werden. Ist alles korrekt kann die Seite gespeichert werden und verwendet werden (siehe Abbildung).

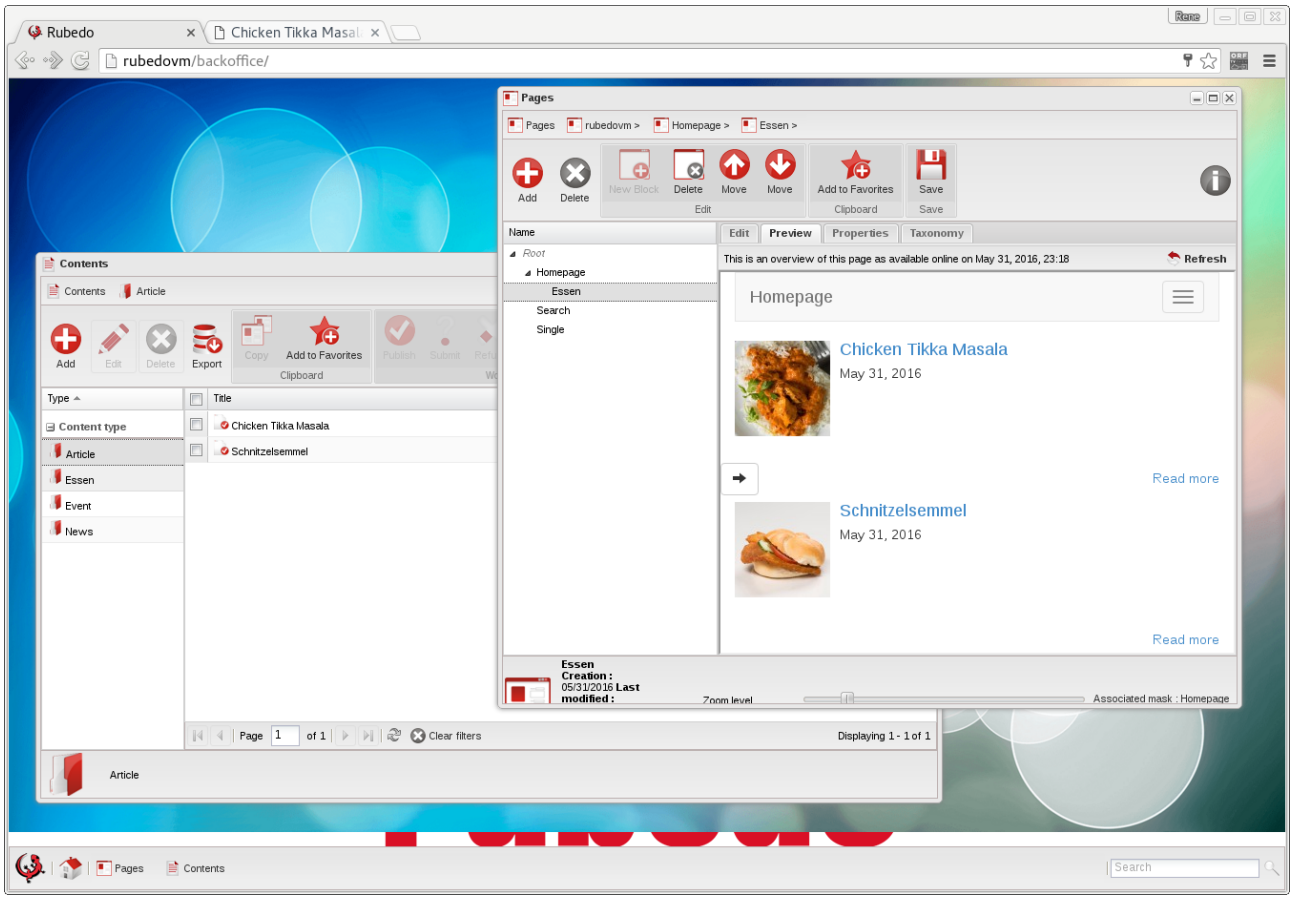


Abbildung 5: Webseite mit 2 angelegten Artikel

Erstellte Artikel bekommen automatisch ihre eigene Seite. Dort wird dann der gesamte Body angezeigt (siehe Abbildung).

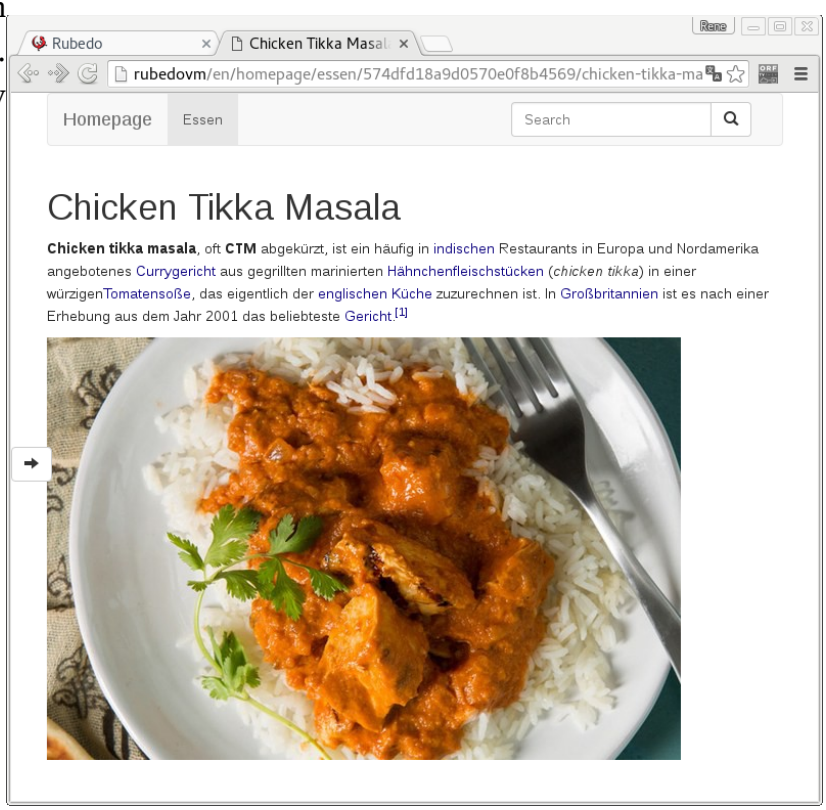


Abbildung 6: Artikelseite

Literaturverzeichnis

1: Seth Gilbert, Nancy Lynch, Brewer's Conjeture and the Feasibility ofConsistent, Available, Partition-Tolerant Web,
2: Andrew S. Tanenbaum, Maarten van Steen, Verteilte Systeme, 2007
3: Ben Smith, Beginning JSON, 2015
4: , ,
5: , ,
6: Christof Strauch, NoSQL Databases,
7: Stephen Yen, NoSQL is a horseless carriage, 2009, http://dl.getdropbox.com/u/2075876/nosql-steve-yen.pdf
8: Alex Popescu, Ana-Maria Bacalu, Presentation: NoSQL @ CodeMash - An Interesting NoSQL Categorization, 2010, http://nosql.mypopescu.com/post/396337069/presentation-nosql-codemash-an-interesting
9: , MongoDB Cloud Manager, , https://www.mongodb.com/cloud
10: , MongoDB Cloud Manager Frequently Asked Questions, , https://docs.cloud.mongodb.com/faq/#what-versions-of-mongodb-does-mms-manage
11: , Couchbase Web Console, , http://developer.couchbase.com/documentation/server/4.1/admin/ui-intro.html
12: Couchbase Entwickler, Database Querying with N1QL,
13: , MongoDB Documentation, , https://docs.mongodb.com/
14: , MongoDB Sharding Introduction, , https://docs.mongodb.com/manual/core/sharding-introduction/
15: , MongoDB Read Isolation, Consistency, and Recency, , https://docs.mongodb.com/manual/core/read-isolation-consistency-recency/
16: , MongoDB Atomicity and Transactions, , https://docs.mongodb.com/manual/core/write-operations-atomicity/
17: , MongoDB Installation, , https://docs.mongodb.com/manual/installation/
18: , Redis Webseite, , http://redis.io/
19: , An introduction to Redis data types and abstractions, , http://redis.io/topics/data-types-intro
20: , Redis Persistance, , http://redis.io/topics/persistence
21: , Saving Data in SQL Databases, , https://developer.android.com/training/basics/data-storage/databases.html
22: , Grundkonzepte hinter IndexedDB, , https://developer.mozilla.org/de/docs/IndexedDB/Grundkonzepte_hinter_IndexedDB
23: , UnQLite Webseite, , https://unqlite.org/
24: , Couchbase Mobile, , http://www.couchbase.com/nosql-databases/couchbase-mobile
25: , Mongoose Dokumentation, , http://mongoosejs.com/docs/guide.html
26: , RubedoCMS, , http://www.rubedo-project.org/en/

Abbildungsverzeichnis