Alles ist möglich

Einführung in MongoDB

Markus Alvermann

Für spezifische Probleme werden spezialisierte Datenbanken benötigt. MongoDB bietet für Webapplikationen, in denen eine große Anzahl von Daten, die nicht transaktionsbasiert vorgehalten werden müssen, eine hochperformante, skalierbare, schemafreie und dokumentenorientierte Lösung. In diesem Artikel wird die MongoDB vorgestellt und anhand von Beispielcode aufgezeigt, wie der Einsatz der Java-API funktioniert.

Einleitung

MongoDB wird von der Firma 10gen entwickelt. Im Jahr 2007 begann 10gen mit der Arbeit an einer Softwareplattform, die aus einem Server und einer Datenbank bestehen sollte. Als Einsatzgebiet war das Hosten und Skalieren von Anwendungen in der Cloud vorgesehen. Schnell stellte sich jedoch heraus, dass potenzielle Kunden zwar nicht bereit waren, dermaßen viel Kontrolle über ihre Produkte an die „Wolke" abzugeben, allerdings an der Datenbank der Softwareplattform selbst Gefallen fanden. Daraufhin stellte 10gen die Entwicklung an der Softwareplattform ein und konzentrierte sich auf die Datenbank – MongoDB war geboren.

Mongo steht als Abkürzung für „Hu*mongo*us", also enorm, und so gibt schon der Name einen Hinweis auf das Einsatzgebiet bzw. eine der wichtigsten Eigenschaften der Datenbank: die performante Verarbeitung großer Datenmengen. Von Beginn an für die Verwendung in Webapplikation und dem Internet vorgesehen, sind der Aufbau und die Datenstruktur von MongoDB genau für diesen Anwendungsfall optimiert worden (vgl. [Mong]).

Dokumente

Daten werden in MongoDB in Dokumenten abgespeichert. Ein Dokument besteht aus einem sortierten Satz von Eigenschaften, die aus einem Namen und einem Wert bestehen. Werte können einfache Datentypen, Felder oder andere Dokumente sein. Dokumente sind schemalos, d. h. sie haben keinen festen Aufbau, sondern werden dynamisch mit den Daten befüllt, die benötigt werden.

Das Format, in dem Dokumente abgespeichert werden, ist BSON, binäres JSON. BSON zeichnet sich durch Effizienz und Platzersparnis aus und kann durch seine auf C basierende Repräsentation von Typen performant codiert und decodiert werden (vgl. [Cho10] und [Bson]). Unabhängig vom eingesetzten Treiber wird intern jedes Dokument auf ein BSON-Konstrukt abgebildet.

```
{
  _id:ObjectID('123ab76fe554dcc4356bde9e'),   //1
titel:' MongoDB im Einsatz',
author:'mal',
  tags:[                                      //2
    'datenbank',
    'mongodb',
    'praxis'
  ],
kommentare:[                                  //3
    {
nutzername:'mustermann',
inhalt:'Wieder was gelernt.',
email:'max@mustermann.de'
    },
    {
nutzername:'dbgott',
inhalt:'Alles Schrott!',
      email:'dbgott@dodgit.com'
    }
  ]
}
```

Listing 1: Dokument

Am besten lässt sich die Dokumentenstruktur an einem konkreten Beispiel erklären. In Listing 1 sehen wir ein Dokument, welches Daten aus einem Blog-Eintrag widerspiegelt. In MongoDB bekommt jedes Dokument automatisch einen Primärschlüssel (Markierung 1). Der Primärschlüssel kann natürlich jederzeit den eigenen Bedürfnissen angepasst werden. An Markierung 2 sehen wir anhand der Eigenschaft `tags`, dass Daten mithilfe von eckigen Klammern in ein Array geschrieben werden. Die Kommentare bei Markierung 3 bestehen wiederum aus eigenen Dokumenten.

In einer relationalen, SQL-basierten Datenbank lässt sich ein Blog-Eintrag nicht so einfach abbilden. Hierzu benötigen wir eine Tabelle für die Blog-Einträge, die per Fremdschlüsselbeziehung mit einer Tabelle für die Kommentare und einer Tabelle für die Tags verknüpft ist. Leseoperationen funktionieren aufwendig mit Join-Operationen.

Bei einem Dokument hingegen haben wir alle Informationen an einer Stelle, was Lese- und Schreiboperationen vereinfacht, und vor allen Dingen bei Projektstart für leichteres Arbeiten sorgt, da Dokumente dynamisch um benötigte Informationen erweitert werden können. Außerdem muss nicht jedes Dokument die gleichen Daten bereitstellen; es kann beispielsweise auch Blog-Einträge ohne Tags geben, die Eigenschaft `tags` entfällt dann für diesen Eintrag. Der zur Verfügung stehende Platz wird somit optimal genutzt, Platzhalter in Form von leeren Spalten bei relationalen Datenbanken existieren nicht.

Dokumente werden in MongoDB in Collections zusammengefasst. Analog zu relationalen Datenbanken stellen Dokumente das Pendant zu Spalten dar, während Collections das Gegenstück zu Tabellen sind. Collections können Dokumente mit völlig unterschiedlichem Aufbau beinhalten.

Hochperformant

Eine hohe Performance war von Beginn an eines der Hauptziele beim Entwurf von MongoDB. Viele Designentscheidungen sind gefällt worden, um die Performance der Datenbank zu verbessern. Die Kommunikation zwischen Server und Client läuft über ein leichtgewichtiges TCP/IP-Protokoll, welches viel weniger Overhead verursacht als HTTP/REST. Daten werden im Hauptspeicher verwaltet, wodurch das Speichermanagement vom Betriebssystem übernommen wird. Bei Abfragen greift im Hintergrund ein Abfrage-Optimierer, der sich den schnellsten Weg zum Ausführen einer Abfrage merkt.

Des Weiteren garantiert der bewusste Verzicht auf einige von relationalen Datenbanken bekannte Features, wie „Join-Operationen" oder Transaktionskontrolle, eine hohe Geschwindigkeit.

Eine weitere Möglichkeit, die MongoDB zur Performancesteigerung bietet, ist die Verwendung von Indizes. Jede Property eines Dokumentes kann mit dem Befehl `ensureIndex({property:wert})` indiziert werden – selbst Propertys eingebetteter Dokumente.

Der Wert kennzeichnet die Reihenfolge der indizierten Elemente (1 für aufsteigend, -1 für absteigend), zusammengesetzte Indizes werden per Kommanotation erstellt.

Replikation und Ausfallsicherheit

MongoDB bietet verschiedene Replikationsmodi für die Sicherung von Daten an, wobei der Master-Slave-Modus der gebräuchlichste ist. Er kann flexibel zur Datensicherung, Ausfallsicherung und Skalierung der Datenbank verwendet werden. Um dieses Feature nutzen zu können, müssen lediglich mindestens zwei Datenbankknoten gestartet werden, von denen ein Knoten im Master-Modus und die anderen Knoten im Slave-Modus laufen.

Einen Master starten wir mit dem Befehl `mongod --master`, den zugehörigen Slave mit `mongod --slave --sourcemaster_adresse`. Die Daten des Masters werden automatisch auf den laufenden Slaves repliziert. Die Anzahl der Slaves ist nicht begrenzt, wirkt sich jedoch bei einer größeren Anzahl auf die Performance des Master-Knotens aus. In der Praxis haben sich Cluster mit bis zu zwölf Slaves bewährt (vgl. [Cho10]).

Die Ausfallsicherheit wird durch sogenannte *Replica Sets* gewährleistet: In einem *Replica Set* gibt es keinen Master, sondern eine Primärdatenbank, die als Master fungiert, und eine beliebige Anzahl von Sekundärdatenbanken, welche die Slaves darstellen. Die Primärdatenbank wird vom Cluster ausgewählt. Fällt sie aus, wird bei allen verfügbaren Sekundärdatenbanken geprüft, welche als Letztes mit der Primärdatenbank synchronisiert wurde. Die so ermittelte Datenbank wird automatisch zur neuen Primärdatenbank. Wenn die ausgefallene Datenbank wieder zur Verfügung steht, reiht sie sich als Sekundärdatenbank in den Cluster ein.

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:

- *Shard*: Ein einzelner Datenbankserver, der eine Teilmenge, ein „Bruchstück" bzw. Shard, der Daten beinhaltet.
- *mongos*: Eine Art Router, der die Datenbankanfragen entgegennimmt, an die korrekten Shards weiterleitet und die Antworten der Shards zu einem kompletten Ergebnis zusammensetzt. Die Informationen, welche Daten sich in welcher Shard befinden, entnimmt `mongos` einem Konfigurationsserver.
- *Konfigurationsserver*: Hier werden die Informationen über die einzelnen Shards hinterlegt, welche von `mongos` synchronisiert werden.

Das Sharding bei MongoDB funktioniert Collection-basiert. Es wird ein sogenannter *ShardKey* festgelegt, der als Kriterium fungiert, nachdem die Dokumente auf die verschiedenen Shards aufgeteilt werden. In unserem obigen Blog-Eintrag-Beispiel könnte der `author` als *ShardKey* festgelegt werden. Bei drei zur Verfügung stehenden Shards könnte der erste Shard alle Autoren von A-H, der zweite alle Autoren von I-P und der dritte alle Autoren von Q-Z enthalten. Die Applikation selbst bekommt vom Sharding gar nichts mit. Anfragen werden an `mongos` gestellt, die sich der Applikation wie eine einzelne Datenbank präsentieren.

Installation

Die Installation von MongoDB erfolgt in vier einfachen Schritten. Zuerst laden wir uns von [Mong] die aktuelle Version von MongoDB herunter und achten darauf, eine gerade Versionsnummer auszuwählen, da es sich bei den ungeraden Versionsnummern um Entwicklungsreleases handelt. Die heruntergeladene Datei entpacken wir an einen Ort unserer Wahl.

Als Nächstes legen wir im Root-Verzeichnis ein Unterverzeichnis `/data/db/` an, in dem MongoDB nach dem Start automatisch alle Datenbanken und Daten verwaltet. In einer Konsole wechseln wir in das `bin`-Verzeichnis der MongoDB-Installation und starten die Datenbank mit `./mongod` (bzw. `mongod.exe`).

Die MongoDB-Shell

Ebenfalls im `bin`-Verzeichnis befindet sich unter `./mongo` die Shell von MongoDB. Sie ist ein vollständiger JavaScript-Interpreter, was wir einfach mit der Eingabe einiger JavaScript-Befehle überprüfen können (bspw. `alert(„Test");`). Mithilfe der Shell können der Datenbankinhalt inspiziert, Abfragen abgesetzt und administrative Aufgaben ausgeführt werden.

In Tabelle 1 werden einige MongoDB-Befehle vorgestellt und ihrem SQL-Pendant gegenübergestellt. Als Java-Entwickler hat man die JavaScript-ähnliche Notation schnell verinnerlicht, da sie mehr an Programmierung als an SQL angelehnt ist. Mit `db.users.find({age:33}).sort({name:1})` werden beispielsweise alle Dokumente aus der Collection `users` ausgelesen, die die Property `age:33` enthalten, und nach der Property `name` aufsteigend sortiert ausgegeben.

	implizit
`CREATE TABLE USERS (a Number, b Number)`	
`INSERT INTO USERS VALUES(1,1)`	`db.users.insert({a:1,b:1})`
`SELECT a,b FROM users`	`db.users.find({}, {a:1,b:1})`
`SELECT * FROM users WHERE age=33 ORDER BY name`	`db.users.find({age:33}).sort({name:1})`
`UPDATE users SET a=1 WHERE b='q'`	`db.users.update({b:'q'}, {$set:{a:1}}, false, true)`
`CREATE INDEX myindexname ON users(name)`	`db.users.ensureIndex({name:1})`
`SELECT DISTINCT last_name FROM users`	`db.users.distinct('last_name')`

Tabelle 1: Vergleich SQL – MongoDB-Shell

Die Java-API

Um MongoDB in einem Java-Projekt einzusetzen, müssen wir uns lediglich das aktuelle mongo-x.x.jar von [Mong] herunterladen und in den Klassenpfad des Projektes einbinden. In Schritt 1 von Listing 2 sehen wir, wie die Verbindung zu einer Datenbank mithilfe des `Mongo`-Objekts und des `DB`-Objekts hergestellt wird und eine optionale Authentifizierung mit einem Datenbanknutzer erfolgt. Das Abfangen der Exceptions ist zugunsten der Lesbarkeit des Quellcodes gekürzt worden.

```
//Schritt 1
Mongo mongo = new Mongo();

DB db = mongo.getDB("test");

String username = "iks";
char passwd[] = { 'i', 'k', 's' };
booleanauth = db.authenticate(username, passwd);

//Schritt 2
DBCollectioncoll = db.getCollection("testCollection");

DBObjectmyDoc = coll.findOne();
```

```
System.out.println(myDoc);
coll.drop();

//Schritt 3
DBObject doc = new BasicDBObject();

doc.put("name", "MongoDB");
doc.put("type", "database");
doc.put("count", 1);

BasicDBObject info = new BasicDBObject();
info.put("x", 203);
info.put("y", 102);
doc.put("info", info);

coll.insert(doc);

//Schritt 4
BasicDBObject query = new BasicDBObject();
query.put("i", 71);
DBCursor cur = coll.find(query);
while (cur.hasNext()) {
  System.out.println(cur.next());
}
```

Listing 2: Basis-Operationen

In Schritt 2 wird mit einer **DBCollection** eine Collection aus der Datenbank ausgelesen und mit der **findOne()**-Methode der erste Eintrag aus dem Ergebnis in ein **DBObject** – der Java-Entsprechung eines Dokuments – geschrieben. Mit der Methode **coll.drop()** wird die gesamte Collection aus der Datenbank entfernt.

In Schritt 3 legen wir ein eigenes **DBObject** an und befüllen es mit der **put()**-Methode, die als ersten Parameter den Namen und als zweiten Parameter den Wert der Eigenschaft erhält. So ist es auch auf einfache Art und Weise möglich, ein eingebettetes Dokument hinzuzufügen. Mit der **coll.insert()**-Methode wird das Dokument in die Collection geschrieben.

In Schritt 4 verwenden wir ein **BasicDBObject**, um eine Abfrage abzusetzen. In diesem Fall werden alle Dokumente aus der Collection, die **i:71** als Eigenschaft beinhalten, in einen **DBCursor** geschrieben. Mithilfe des Cursors können wir über das Ergebnis iterieren. Mit Literalen und Punktnotation können wir auch komplexere Abfragen erstellen.

```
query.put("i", newBasicDBObject("$gt", 20).append("$lte", 30));
coll.find(query);
```

Listing 3: Parametrisierte Abfrage

Die Methoden aus Listing 3 lesen alle Dokumente aus der Collection mit **20 < i < 30**. Eine vollständige Übersicht über die zur Verfügung stehenden Literale und Befehle findet sich auf [Mong].

Auch der Umgang mit großen Dateien stellt mit MongoDB kein Problem dar. Mit GridFS, einer leichtgewichtigen Spezifikation zum Speichern von Dateien, die auf der normalen Dokumentenspezifikation aufsetzt, werden große Dateien einfach in kleinere Chunks aufgesplittet und jeder Chunk wird als eigenes Dokument in der Datenbank abgelegt. MongoDB stellt mit **fs.chunks** eine eigene Collection für GridFS-Dateien zur Verfügung. Die Dokumente in dieser Collection bestehen aus einer **id**, einer **file_id**, die festlegt, in welchem Dokument sich die Metadaten für den Chunk befinden, einer Chunk-Nummer und den binären Daten des Dateifragments. Die **file_id** verweist auf eine andere Collection namens **fs.files**, die die Dateiinformationen enthält, die folgende Angaben umfasst: eine **id**, die Größe der Datei in Bytes, eine Angabe über die Chunk-Größe, das Upload-Datum und eine md5-Checksumme, mit deren Hilfe Nutzer überprüfen können, ob die Datei korrekt hochgeladen wurde.

```
final byte[]fileAsByte = readFile(file);

GridFSgridFS = new GridFS(db);
GridFSInputFileinputFile = gridFS.createFile(byteArray);
inputFile.save();
String name = inputFile.getFilename();

List list = gridFS.find(name);
```

Listing 4: Dateien speichern mit GridFS

In der Java-API stellt die Klasse **GridFS** die Funktionalität zum Abspeichern von Dateien zur Verfügung. Listing 4 zeigt die Schritte, die nötig sind, um eine Datei in der Datenbank abzulegen, wobei das Einlesen der Datei in der Methode **readFile()** abstrahiert wurde. **GridFS** kann über ein **byte**-Array aus einer Datei ein **GridFSInputFile** kreieren, welches dann einfach per **save()**-Methode abgespeichert werden kann. Anhand des Dateinamens können wir das Dokument per **find()**-Methode wieder aus der Datenbank auslesen. Um die einzelnen Chunks muss sich der Programmierer nicht kümmern, da dies automatisch von MongoDB übernommen wird.

Einsatzgebiete

Obwohl MongoDB ein noch recht junges Projekt ist, bietet die Datenbank eine Fülle von Features und positioniert sich damit – wie gewollt – zwischen den schnellen, aber featureärmeren und unflexibleren Schlüssel-Wert-Speichern und den komplexeren, schwergewichtigeren relationalen Datenbanken und wird auch schon in vielen Produktivumgebungen eingesetzt. MongoDB wird in der Regel nicht exklusiv verwendet, sondern dient als Lösung für bestimmte Teilaufgaben. Einige bekanntere Firmen, die MongoDB einsetzen sind

- *Foursquare*: Ein ortsabhängiges soziales Netzwerk, mit dem Nutzer an Orten virtuell einchecken und zu diesen Orten Tipps und Bewertungen schreiben. Foursquare setzt MongoDB exklusiv für die Eincheck-Funktion ein (‚Who's here?').
- *New York Times*: Hier wird MongoDB für ein frei konfigurierbares Foto-Upload-Formular verwendet. Aufgrund der Schemalosigkeit der Dokumente kann jeder Anwender sein eigenes, maßgeschneidertes Formular erstellen.
- *Sourceforge*: Die Projekt-Webseiten und die Download-Seiten aller Projekte werden mit MongoDB verwaltet.
- *GitHub*: Die soziale Coding-Plattform setzt MongoDB für internes Reporting ein.
- *IGN Entertainment*: Die Computerspiele- und Unterhaltungsseite analysiert in Echtzeit den Seiten-Verkehr mit MongoDB.

Da die Datenbank von Anfang an für den Einsatz in Webapplikationen vorgesehen war, lassen sich mit MongoDB Aufgaben wie Dokumentenmanagement, das Verwalten von Anwender- und Sitzungsdaten, Logging, Echt-Zeit-Analysen und generell Aufgaben mit einem hohen Datenaufkommen einfach und bequem lösen. Nicht geeignet ist MongoDB für Systeme mit komplexen Transaktionen und für traditionelle Business-Intelligence-Projekte, da die dafür nötigen Strukturen nicht zur Verfügung stehen.

Fazit

MongoDB ist eine Datenbank, die bei den oben aufgeführten Einsatzgebieten ohne Schwierigkeiten eingesetzt werden kann. Datenreplikation und Skalierbarkeit funktionieren ein-

fach und automatisch. Innerhalb weniger Minuten lässt sich ein neuer Server in einen bestehenden Cluster einbinden und nutzen; vertikales Skalieren, also das Aufrüsten eines bestehenden Servers und die damit verbundene Ausfallzeit, entfallen vollständig. Der Failover-Mechanismus funktioniert ebenfalls automatisch und garantiert somit eine sehr hohe Datenverfügbarkeit.

MongoDB ist hochperformant. Atomare Updates von Dokumenten sind um ein Vielfaches schneller als bei relationalen Datenbanken. In einem interessanten Blog-Eintrag [Word] beschreiben die Entwickler von Wordnik, dass sie aufgrund von Performance-Problemen den Umstieg von MySQL auf MongoDB vollzogen haben. Die Geschwindigkeit verschiedener Abfragen hat sich signifikant verbessert. Das Abfragen von Wörterbucheinträgen dauert statt 20 ms nur noch 1 ms, Abfragen, die auf den Plattenspeicher zugreifen, dauern nur noch 60 ms statt 400 ms.

Die Java-API ist ausgereift und einfach zu verwenden. Durch die JavaScript-ähnliche Syntax finden sich Programmierer sehr leicht in MongoDB zurecht.

Natürlich gibt es Nachteile: Um eine hohe Geschwindigkeit zu erreichen, wird die gesamte Datenbank im RAM gehalten. Kommt es zum unsauberen Beenden einer Master-Datenbank, sind die Daten oftmals korrupt. Deswegen ist es zwingend notwendig, MongoDB mit Replikation laufen zu lassen. Außerdem sollte MongoDB auf einem 64-Bit-System installiert werden, um die RAM-Beschränkung von 32-Bit-Systemen zu umgehen.

Literatur und Links

[Ban10] K. Banker, MongoDB in Action (MEAP), Manning, 2010

[Bson] Homepage BSON,
`http://www.bsonspec.org`

[Cho10] K. Chodorow, M. Dirolf, MongoDB: The Definitive Guide, O'Reilly, 2010

[Mong] Homepage MongoDB,
`http://www.mongodb.org`

[Word] Wordnik-Blog,
`http://blog.wordnik.com/12-months-with-mongodb`

Markus Alvermann ist als IT-Berater bei der iks Gesellschaft für Informations- und Kommunikationssysteme mbH tätig. In seinen Kundenprojekten beschäftigt er sich seit vielen Jahren mit der Entwicklung moderner JEE-Anwendungen. Ein besonderer Interessenschwerpunkt ist die Evaluierung neuer Technologien.
E-Mail: m.alvermann@iks-gmbh.com