MongoDB

Seminararbeit im Modul NoSQL-Datenbanken

Sebastian Volke
Institut für Informatik, Universität Leipzig

Inhaltsverzeichnis

1 Einführung

In den letzten Jahren hat im Bereich der Datenbanken das Stichwort NoSQL zunehmend Bedeutung gewonnen.

Auf vielen Einsatzgebieten von Datenbanksystemen werden zunehmend mehr Daten in kürzerer Zeit gespeichert. In wissenschaftlichen Anwendungen liegt das an immer genaueren Messgeräten und neuen Verfahren (wie etwa Genexpression), die große Datenmengen generieren. In der Wirtschaft haben die Unternehmen immer größere Kundenstämme, nicht zuletzt durch die Industrialisierung, und immer größere Datenaufkommen durch ständig mehr tägliche Transaktionen. Auch im Unterhaltungssektor werden durch weltweit verfügbare und ständig wachsende soziale Netzwerke sehr viele Daten erzeugt.

Traditionell werden Datensätze in einem relationellen Datenbankmanagementsystem auf einem zentralen Server gespeichert. Damit das System mit den Anforderungen Schritt halten kann, muss die die Hardware ständig aufgerüstet und die Software immer weiter optimiert werden. Gerade im Bereich der Hardware werden aber zunehmend Grenzen erreicht. Die CPUs lassen sich kaum über 3GHz takten und hier muss mit Parallelisierung gearbeitet werden und auch im Bereich der Speichersysteme kann zwar die Speicherdichte stetig weiter erhöht werden, aber die Zugriffszeiten verbessern sich nur noch unwesentlich.

Gerade durch die bei Google entwickelten Speichertechnologien setzt sich inzwischen ein neuer Ansatz durch: es werden Cluster aus Rechnern mit durchschnittlicher und somit billiger Hardware gebildet. Die Software ist so ausgelegt, dass Hardware-Ausfall kompensiert werden kann, indem Daten geeignet repliziert werden. Durch die Verteilung der Datensätze auf verschiedene Knoten im Cluster kann auch sehr große horizontale Skalierbarkeit gewährleistet werden. Auch hier ergeben sich Grenzen, wie es z.B. das CAP-Theorem zeigt, die aber für viele Datenbankanwendungen zu verschmerzen sind.

Unter diesem neuen Paradigma bildeten sich verschiedenste Datenbanksysteme heraus, die als NoSQL-Systeme zusammengefasst werden, weil sie auf SQL als Anfragesprache verzichten oder darüber hinaus Anfragemöglichkeiten bieten („Not Only SQL“). Neben key-Value-Datenbanken, Graphdatenbanken und Objektdatenbanken gibt es auch die Gruppe der Dokumentdatenbanken.

Ein Vertreter dieser Gruppe, MongoDB soll im Rahmen dieser Arbeit vorgestellt werden. Dabei soll gemäß dem Anspruch, den MongoDB an sich selbst stellt, gezeigt werden, dass MongoDB im Funktionsumfang herkömmlichen relationalen Datenbanksystemen in nichts nachsteht und auch Pluspunkte bei der Performance sammeln kann.

Dazu wird neben einem Überblick über MongoDB das Hauptaugenmerk auf der Anfragesprache liegen und eine Diskussion der für SQL-Systeme sehr zentralen ACID-Eigenschaften komplexer Transaktionen liegen. Abschließend soll ein Geschwindigkeitsvergleich zu MySQL und ein kurzer Vergleich zu CouchDB, dem anderen wichtigen Dokumentdatenbanksystem, gezogen werden.

2 Überblick MongoDB

MongoDB ist eine verteilte Dokumentendatenbank.

Beim Design des Systems standen vier Ziele im Fokus: Flexibilität, Mächtigkeit, Geschwindigkeit/Skalierbarkeit und einfache Benutzbarkeit. [1]

Flexibilität bedeutet ein schemaloses Datenmodell, dass mit der Datenbankanwendung mitwachsen und sich verändern kann. Es soll auch einen einfachen Zugang zum Datenbanksystem für den Programmierer geben. Der Gedanke ist, dass neue Systeme das Erstellen von Software vereinfachen

und erschweren sollen.

Mächtigkeit bedeutet hoher Funktionsumfang. Trotz der Flexibilität des Datenmodells soll der Benutzer nicht auf die Funktionalität verzichten müssen, die er von SQL gewohnt ist. So stellt MongoDB ein mächtige Anfragesprache, Index-Strukturen und viele weitere Funktionen zur Verfügung.

Bei relationalen Datenbanksystemen wird durch Datenbanksperren und Konsistenzbedingungen oft Geschwindigkeit eingebüßt. MongoDB stellt an sich den Anspruch, den Funktionsumfang von SQL zu bieten, aber gleichzeitig die Geschwindigkeit und Skalierbarkeit von Key-Value-Stores zu erreichen. Dies wird in der Abbildung 1 verdeutlicht. Neben schnellen Antwortzeiten soll auch die Speicherkapazität des Systems im Online-Betrieb leicht zu erweitern sein.

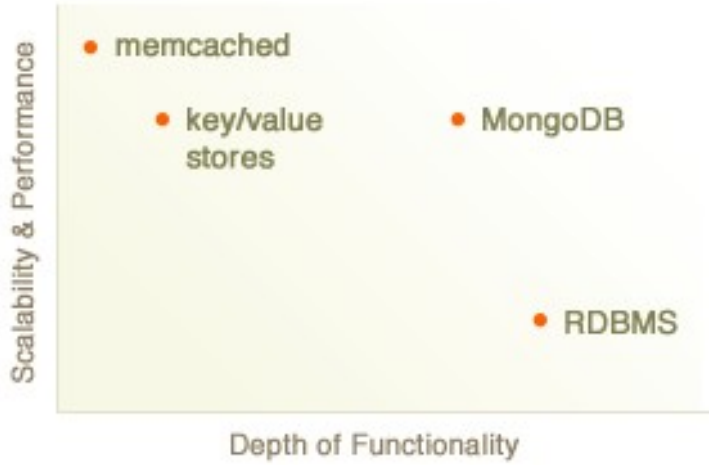


Abbildung 1: Anspruch von MongoDB (Quelle: [1])

Als letzter Punkt bleibt einfache Benutzbarkeit. Es soll einfach, ohne Training und praktisch überall möglich sein, eine MongoDB-Instanz laufen zu lassen, sei es auf eigenen Servern, in einem Cloud-Service oder auf dem privaten Rechner. Es gibt ein kurzes Tutorial, in dem beschrieben wird, wie man in fünf Minuten einen MongoDB-Server aufsetzt und benutzt [2].

Das dokumentorientierte Datenmodell kommt den meisten Programmiersprachen sehr entgegen, da die Dokumente fast direkt dem Datentyp des Objekts entspricht. Folglich wird in MongoDB auch die Javascript-Objekt-Notation zur Dokumentrepräsentation verwendet. Darüber hinaus verwendet MongoDB Javascript für serverseitige Skripte und als Programmiersprache für Map-Reduce-Anfragen.

Es existiert ein Client-Programm für MongoDB, dass eine Javascript-Shell [3] zur Verfügung stellt, mit der Datenbankoperationen einfach und direkt im Terminal durchgeführt werden können.

Darüber hinaus existieren für viele Programmiersprachen Bindings und Treiber, sodass MongoDB in praktisch jeder Umgebung eingesetzt werden kann (einen Überblick siehe [4]).

Im Rahmen dieser Arbeit soll aber die Javascript-Schnittstelle verwendet werden, um ein beispielhaft das Arbeiten mit MongoDB zu verdeutlichen. Dazu befinden sich gelegentlich am rechten Rand graue Boxen mit kurzen Listings, die direkt in der Javascript-Shell ausgeführt werden können.

3 Datenmodell

In diesem Kapitel soll das Datenmodell von MongoDB genauer beschrieben und ein kurzer Blick auf die Modellierung in diesem Datenmodell geworfen werden.

Als Dokumenten-Datenbank bildet in MongoDB das Dokument einen zentralen Bestandteil des

Datenmodells. Ein Dokument ist im wesentlichen eine Sammlung von zusammengehörigen Daten.

In MongoDB ist ein Dokument einfach eine Menge von Feldern, wobei ein Feld ein Schlüssel-Wert-Paar ist. Der Schlüssel ist gleichzeitig der Name des Feldes und wird als String repräsentiert.

Der Wert kann ein primitiver Datentyp sein, eine Liste von Werten oder ein Dokument. Das bedeutet insbesondere, dass jedes Dokument Subdokumente enthalten kann. Die primitiven Datentypen sind die üblicherweise verfügbaren Boolean, Integer, Float und String [5].

Die Schlüssel-Namen unterliegen fast keinen Einschränkungen. MongoDB verbietet allerdings Bezeichner, die mit einem „$“ beginnen oder einen Punkt enthalten, da solche für die MongoDB-eigene Query-Sprache verwendet werden.

```
{
        „boolValue“     : true,
        „_id“           : „eindeutige_ID_12351asd“,
        „array“         : [ true, „value2“, 3, [1,2,3] ],
        „object“        :
                          {
                                  „name1“ : „value1“,
                                  „name2“ : „value2“
                          }
        „string“        : „value“
}
```

Abbildung 2: Beispiel-Dokument

Des weiteren gibt es eine besonderes Feld, das jedes Dokument aufweist: das „_id“-Feld enthält einen innerhalb der Collection (siehe unten) eindeutigen Wert, der als Primärschlüssel verwendet wird.

In MongoDB werden Dokumente als BSON-Objekte dargestellt, also mithilfe einer speichereffizienten, binären Variante von JSON. Eine sehr übersichtliche Grammatik der JSON-Beschreibungssprache ist unter [6] zu finden. Die Spezifikation von BSON befindet sich hier [7]. Ein Beispiel-Dokument befindet sich in Abbildung 2.

Eine wichtige Einschränkung ist die maximale Dokumentgröße von 16MB [8]. Diese hat keine technischen Gründe, sondern dient mehr als eine Art Konsistenz-Check. Wenn Dokumente diese Datenmenge überschreiten, spricht das für eine schlechte Datenmodellierung.

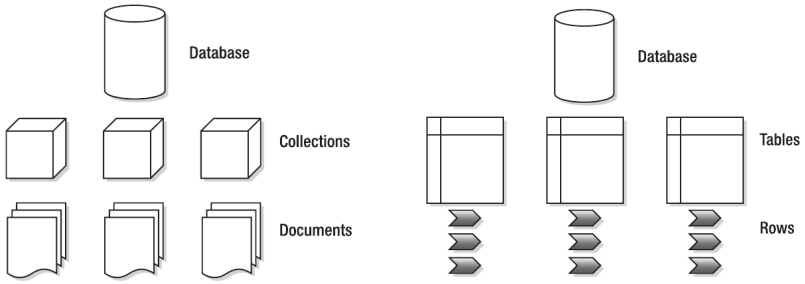


Abbildung 3: Vergleich MongoDB – SQL-Schema

Dokumente werden in Collections zusammengefasst. Eine Collection ist eine benannte Menge von Dokumenten. Jede Collection ist in einer Datenbank gespeichert und jedes MongoDB-System enthält eine Menge von Datenbanken.

Dadurch ergibt sich eine gewisse Ähnlichkeit zu der Strukturierung in relationalen Datenbankensystemen (siehe Abbildung 3). Auch in letzteren sind die Daten in Datenbanken gespeichert, die wiederum Tabellen enthalten. Diese Tabellen entsprechen in etwa den Collections in MongoDB. Innerhalb einer Tabelle sind die Datensätze als Zeilen gespeichert, wobei jede Zeile ein Tupel aus einem oder mehreren Werten ist. Dies entspricht in MongoDB einem Dokument als Sammlung von Schlüssel-Werte-Paaren.

In einem RDBMS unterliegt aber die Tabelle einem Schema, sodass alle Zeilen der Tabelle den

gleichen Aufbau haben, d.h. die gleiche Anzahl von Werten mit jeweils den gleichen Datentypen. In MongoDB unterliegen die Dokumente keinem festen Schema, sondern jedes Dokument hat seine eigene Strukturierung. Die Dokumente eine Collection können Ähnlichkeiten aufweisen, müssen es aber nicht.

Im Vergleich zu Key-Value-Stores ist MongoDB mächtiger (siehe Abbildung 4). Key-Value-Stores verwenden auch Collections um darin Key-Value-Paare abzulegen. Dabei sind aber meist nur primitive Datentypen erlaubt und Suchanfragen beschränken sich auf die Schlüssel. Im Gegensatz dazu haben die Dokumente in MongoDB (vom „_id“-Feld abgesehen) keinen ausgezeichneten Schlüssel-Wert, der sie voneinander unterscheidet. Es muss also möglich sein, Dokumente nach ihrem Inhalt zu suchen.

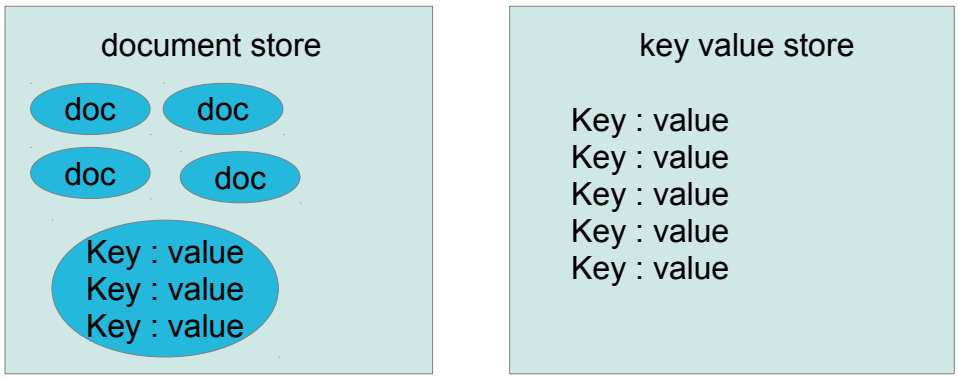


Abbildung 4: Vergleich der Collections bei MongoDB und Key-Value-

Bei der Datenmodellierung ist die Ähnlichkeit zu relationalen Datenmodellen von Vorteil. Viele relationale Modelle können fast direkt übernommen werden. Aufgrund der Schemalosigkeit ist aber größere Flexibilität möglich, wenn sich Datensätze oder Typen von Datensätzen nur in kleinen Details auf der Schema-Ebene unterscheiden. In RDBMS sind oft größere Verrenkungen nötig, z.B. für Vererbungshierarchien. In der Dokumenten-Datenbank können die Daten dagegen einfach flach gespeichert werden. Je nachdem, welche Eigenschaften ein bestimmter Datensatz abhängig von seiner Klasse oder seinem Typ hat, enthält das Dokument unterschiedliche Felder.

Ein Problem der Dokumenten-Datenbank ist aber referenzielle Integrität. So etwas wie Fremdschlüsselbeziehungen gibt es nicht. MongoDB verwendet stattdessen das Konzept von Einbettung und Verlinkung. Ein Link ist ein Verweis auf ein anderes Dokument. Er muss auf der Ebene der Anwendungsprogramme interpretiert und das referenzierte Dokument in einer zusätzlichen Abfrage eingelesen werden. Bei der Einbettung werden die referenzierten Datensätze als Unterdokument in das referenzierende Dokument eingebettet. Beispielsweise enthält ein Dokument, dass eine CD repräsentiert, ein Unterdokument für jeden Track auf der CD, das dann die Eigenschaften wie Tracknummer, Künstler, Titel, etc. enthält. Man könnte dies auch als „pre joined“ bezeichnen, da die Zusammenführung der Daten, die in SQL mit einem Join umgesetzt werden würde, bereits geschehen ist.

Das bedeutet, dass die Datensätze sozusagen im Vergleich zu relationalen Datenmodellen „unnormalisiert“ abgelegt werden. Mehr Informationen zur Datenmodellierung findet sich z.B. hier [9].

4 Anfragesprache

Im Folgenden soll die Anfragesprache von MongoDB vorgestellt werden. Zunächst wird überblicksartig das Prinzip der Anfragesprache und einfache CRUD-Anweisungen beschrieben. Anschließend soll die Herangehensweise bei komplexen Anfragen und Änderungsoperationen, sowie von Map-Reduce-Anfragen gezeigt werden.

4.1 Überblick

Ebenso wie bei dem Datenmodell, steht auch bei der Anfragesprache das Dokument im Mittelpunkt. MongoDB geht soweit, dass jeder Datenbankbefehl selber ein Dokument ist.

DBs anzeigen:
```
show dbs;
```
Neue DB anlegen (implizit durch benutzen):
```
use test;
show collections;
```
Collections entstehen implizit, indem Dokumente angelegt werden.

Der einfachste Fall ist das Speichern eines neuen Dokuments in der Datenbank. Dazu wird einfach das zu speichernde Dokument an das Datenbanksystem übergeben und von diesem in der entsprechenden Collection hinterlegt.

Für eine Anfrage ermittelt das Datenbanksystem alle zu einem Beispieldokument passenden Dokumente einer Collection. Das Beispieldokument hat dabei ungefähr die Funktion der WHERE-Klauses in einer SQL-Anfrage. Es werden alle Dokumente gefunden, die alle Felder des Beispieldokuments enthalten und jeweils in den Werten übereinstimmen.

Im Falle eines Subdokuments bedeutet dies vollständige Übereinstimmung, d.h. eine Anfrage der Form

```
{ „a“ : { „b“ : 1 } }
```

findet alle Dokumente, die ein Feld mit Namen „a“ haben, dessen Wert genau das Dokument `{ „b“ : 1 }` ist. Will man stattdessen alle Dokumente finden, deren Subdokument „a“ einen Schlüssel „b“ mit Wert 1 enthält, dann kann man die Dot-Notation verwenden:

```
{ „a.b“ : 1 }
```

Dokumente einfügen:
```
db.col.insert({a:1, b:2, c:[1,2,3]});
db.col.insert({a:2, b:2, c:“hallo“});
```
Alle Dokumente auflisten:
```
db.col.find();
```
Dokumente nach Muster finden:
```
db.col.find({a:1});
```
Dokumente löschen:
```
db.col.remove({b:2});
```
Dokument verändern:
```
db.col.update({a : 1}, {b : 3, c : [1]});
```

Ebenso, wie eine Anfrage, kann auch eine Löschoperation durchgeführt werden. Dabei werden alle Dokumente aus der Datenbank entfernt, die auf die Suchanfrage mit dem Beispieldokument passen.

Für eine Änderungsoperation werden zwei Dokumente benötigt. Alle auf das Suchdokument passenden Dokumente werden durch das Änderungsdokument ersetzt. Das Suchdokument entspricht dabei der WHERE-Klausel im UPDATE-Statement und das Änderungsdokument in etwa dem SET-Teil des SQL-Befehls.

Die Änderung eines Dokuments ist atomar, aber nicht die Änderung mehrerer Dokumente. Dies verhindert komplexe Transaktionen, wird aber durch die weiter unten vorgestellten Änderungsoperationen aufgewogen.

4.2 Komplexe Anfragen

SQL bietet die Möglichkeit, komplexere Anfragen zu stellen als eine reine Konjunktion von Gleichheitsbedingungen. Das ist auch bei MongoDB möglich durch Operatoren, das sind spezielle Felder in den Anfragedokumenten [10]. Diese beginnen immer mit einem „$“-Zeichen und veranlassen das Datenbanksystem zur Auswertung einer Operation.

Eine wichtige Operatorengruppe verknüft Anfragedokumente. Auf diese Weise können Oder-, Und- und Weder-Noch-Verknüpfungen zwischen Anfragebedingungen gemacht werden. Dazu wird ein Dokument der folgenden Form zur Suche verwendet:

```
{ $or : [Suchdokument1, Suchdokument2, …] }
```

Die anderen beiden Operationen können mit `$and` bzw. `$nor` durchgeführt werden.

Die andere wichtige Operatorengruppe erlaubt etwas allgemeinere Suchbedingungen für Werte. Wie man auf Gleichheit prüft, haben wir ja bereits im vorherigen Unterkapitel gesehen. Wenn der zu vergleichende Wert ein Dokument mit einem Vergleichsoperator ist, so wird stattdessen dieser Operator angewendet. Alle Dokumente, die ein Feld „a“ mit einem Wert größer 5 enthalten, werden z.B. von diesem Dokument gefunden:

```
{„a“ : {$gt : 5} }
```

Die nachfolgende Tabelle zeigt eine Übersicht der verfügbaren Operatoren:

Operator	**Feldbezeichnung**	**Erwarteter Datentyp**
$>, <, \geq, \leq, \neq$	`$gt, $lt, $gte, $lte, $ne`	Integer, Float
Wert (nicht) aus einer Liste	`$in, $nin`	beliebig
Array-Werte in/nicht in/alle in Liste	`$in, $nin, $all`	beliebig
Feld existiert / existiert nicht	`$exists`	Boolean
Modulo	`$mod`	Liste: [Divisor, Rest]
Länge einer Liste	`$size`	Integer
Datentype des Feldes	`$type`	String, Typbezeichnung

Ein weiterer Operator existiert, der auf Operatoren angewendet werden kann und diese negiert. Er hat die Bezeichnung `$not`.

Als letztes soll noch der `$where`-Operator vorgestellt werden. Er kann auch direkt im Suchdokument verwendet werden und verlangt als Wert Javascript-Code, der für jedes Dokument ausgewertet wird und über einen Boolean zurückmeldet, ob das Dokument zum Suchergebnis zählt.

4.3 Cursor

Ähnlich wie in SQL-Systemen ist das Ergebnis einer Suchanfrage ein Cursor. Dieser kann verwendet werden, um die Suchmenge zu traversieren, aber er bietet auch Funktionen, um die Menge weiter einzuschränken oder zu verarbeiten.

Möglich ist das Sortieren mit `sort`, wobei ein Dokument mit der Sortierrichtung übergeben wird und das Einschränken der Ergebnismenge, indem mit `skip` eine Anzahl Dokumente übersprungen und mit `limit` nur eine bestimmte Anzahl ermittelt wird.

Projektion:
```
db.col.find({a:1}, {b:1});
//nur Ausgabe des b-Feldes
```
Sortierung:
```
db.col.find().sort({a:1}); //aufsteigend
```
Limit:
```
db.col.find({a:1}).skip(1).limit(2);
```
Aggregation:
```
db.col.count({b:2});
db.col.find({b:2}).count();
```

Die `min`- und `max`-Funktionen liefern den minimalen bzw. maximalen Wert eines Feldes innerhalb der Menge und die `count`-Funktion zählt die Anzahl der Ergebnisdokumente. Auf diese Weise können einfache Aggregationen durchgeführt werden.

4.4 Änderungsoperationen

Am Anfang des Kapitels wurden bereits einfache Änderungsbefehle vorgestellt. Diese ersetzen die zu ändernden Dokumente vollständig durch ein neues Dokument. In der Praxis will man aber oft nicht das gesamte Dokument ersetzen, sondern nur Änderungen daran vornehmen.

Zu diesem Zweck unterstützt MongoDB Änderungeoperationen [11]. Diese werden ebenso verwendet, wie die Operatoren in den Anfragedokumenten, aber befinden sich in den Änderungsdokumenten.

Man spricht auch von dem „Finde und Verändere“-Paradigma. Alle nach einem bestimmten Kriterium gefundenen Dokumente werden entsprechend den Anweisungen im Änderungsdokument verändert.

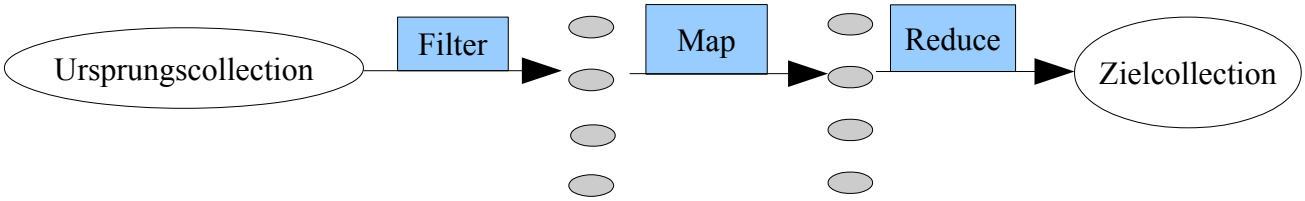
Die folgende Tabelle listet die Änderungsoperatoren und ihre Wirkung:

Operator	**Wirkung**
`$inc`	Erhöhe Feld um gegebenen Betrag
`$set, $unset`	Setze den Wert eines Feldes und lege Feld ggf. an / entferne Feld aus dem Dokument
`$push, $pushAll, $addToSet`	Füge einen / mehrere Werte zu einem Array hinzu (betrachte Array ggf. als Menge)
`$pop, $pull, $pullAll`	Lösche Werte aus einem Array
`$rename`	Benenne ein Feld um
`$bit`	Bitweise Operationen auf Integern

4.5 Map-Reduce

Die Anfragesprache von MongoDB ist sehr umfassend und funktionsreich. Es gibt aber Szenarien, in denen sie nicht ausreicht. In diesen Fällen kann aber auf ein Map-Reduce-Verfahren zurückgegriffen werden [12]. Ein Beispiel wären Anfragen, die man in SQL mithilfe von GROUP BY realisieren würde. Zukünftige Versionen von MongoDB sollen zwar ein GROUP BY implementieren, aber im Moment ist diese Funktionalität noch nicht zu gebrauchen.

Map-Reduce-Anfragen folgen einem festen Workflow:



Ausgehend von einer Ursprungscollection werden als normale Suchanfrage Dokumente einer Collection ausgewählt. Diese dienen als Eingabe der Map-Funktion, die für jedes Dokument einmal aufgerufen wird und Zwischenergebnisdokumente erzeugt. Die Reduce-Funktion wird dann ggf. mehrmals aufgerufen um die Zwischenergebnisse zusammenzufassen. Die Ergebnisse werden dann in der Zielcollection gespeichert.

Die Funktionen werden auf dem Server ausgeführt und in Javascript formuliert.

Die Map-Funktion hat folgende Signatur: `map()`

Innerhalb der Map-Funktion kann mit this auf das aktuelle Dokument zugegriffen und mit

`emit(key, value)` ein Ergebnisdokument zu einem bestimmten Schlüssel erzeugt werden. Einen Rückgabe-Wert gibt es nicht.

Die Reduce-Funktion hat die Signatur `reduce(key, value)`

`value` ist dabei eine Liste von Dokumenten, die zu dem gleichen Schlüssel gehören. Der

Rückgabewert der Funktion ist der aggregierte Wert, der sich aus den Dokumenten der Liste ergibt. Die Reduce-Funktion muss dabei indempotent sein, d.h. wenn sie mehrfach auf ihrer eigenen Ausgabe wieder ausgeführt wird, darf sich das Ergebnis nicht ändern. Auf diese Weise kann die Reduce-Funktion inkrementell immer wieder aufgerufen werden, wenn die Map-Funktion neue Ergebnisdokumente liefert.

Optional kann am Ende der Berechnung noch eine Finalize-Funktion zum Einsatz kommen. Diese hat im Prinzip die gleiche Aufgabe wie eine Reduce-Funktion, aber sie wird nur ein einziges Mal aufgerufen und muss daher auch nicht zwingend indempotent sein.

```
DB wechseln:
use firma;
Map-Reduce:
db.lager.mapReduce(
function() {
      emit( this.Artikel, this.Menge );
},
function(key, values) {
      var result = 0;
      values.forEach(function(value) {
            result += value;
      });
      return result;
},
{out : "bestand"});
Resultat auflisten:
db.bestand.find();
```

Üblicherweise benötigen die Map-Reduce-Anfragen deutlich mehr Zeit zur Ausführung. Die Anfrage mittels Dokumenten und den Anfrageoperatoren ist optimiert und kann zudem Indexstrukturen nutzen. Bei Map-Reduce ist das nicht nötig. Solche Anfragen laufen also völlig unoptimiert ab, sind aber deutlich flexibler.

Ein Vorteil der Möglichkeit, die Ergebnisse in einer Zielcollection abzulegen, liegt darin, dass inkrementelles Map-Reduce durchgeführt werden kann. Wenn in der Ursprungscollection neue Dokumente erstellt werden, können diese über einen Filter gesammelt und als Eingabe für die Map-Reduce-Anfrage genutzt werden. Durch die Indempotenz der Reduce-Funktion können die neuen Ergebnisdokumente problemlos mit den bereits in der Zielcollection vorhandenen Ergebnisdokumenten zusammengeführt werden.

5 Replikation und Sharding

In diesem Kapitel sollen die Fähigkeiten von MongoDB bzgl. Skalierbarkeit und Ausfallsicherheit besprochen werden.

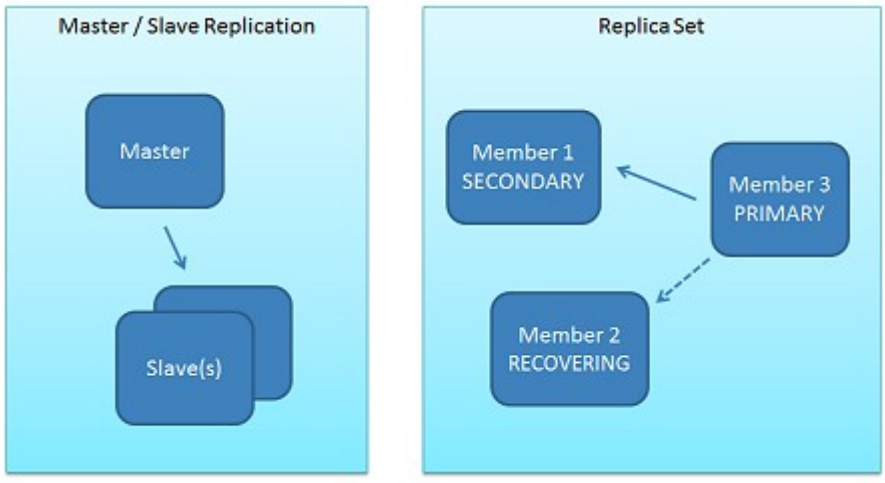


Abbildung 5: Replikationsverfahren (Quelle: [13])

Wie in der Einleitung bereits erwähnt, sind moderne NoSQL-Systeme für typische Standard-Hardware ausgelegt und müssen somit mit regelmäßigen Hardware-Ausfall rechnen. Um dennoch Datensicherheit zu garantieren, muss das Datenbanksystem über einen Replikationsmechanismus verfügen. MongoDB bietet sogar zwei zur Auswahl [13]. Ursprünglich wurde eine Master/Slave-

Replikation implementiert. Alle Datenbankoperationen wurden auf dem Master ausgeführt, der diese dann an die Slaves weiterleitet. Auf diese Weise kann strenge Konsistenz gewährleistet werden, also Serialisierbarkeit der Datenbankoperationen. Genügt bereits eventually consistent als Qualitätsbedingung, so können Leseoperationen auch über den Slave abgewickelt und somit bereits eine Lastenbalancierung vorgenommen werden. Fällt der Master-Knoten aus, so kann einer der Slave-Knoten in den neuen Master-Knoten umgewandelt werden.

Das Master/Slave-Modell gilt für MongoDB neuerdings als veraltet, da es einen relativ eingeschränkten Funktionsumfang hat und zudem großen Administrationsaufwand bedeutet. Es wird daher die Verwendung von Replica Sets empfohlen. Darunter versteht man eine Gruppe von Knoten, die Kopien voneinander sind. Innerhalb dieser Gruppe wird ein primärer Knoten gewählt, der sich aber ansonsten nicht von den anderen unterscheidet. Alle Schreiboperationen werden über diesen primären Knoten abgewickelt und von diesem an die anderen Knoten der Gruppe weitergereicht. Auf diese Weise werden die Daten redundant abgelegt. Abhängig vom Datenbanktreiber können so auch verschiedene Konsistenzbedingungen durchgesetzt werden, z.B. dass ein Schreibzugriff erst als erfolgreich gemeldet wird, wenn mehr als die Hälfte der Knoten im Replica Set die Änderung übernommen haben.

Auf diese Weise kann ein beliebiger Knoten aus dem Replica Set automatisch übernehmen, wenn der Primärknoten ausfällt, was den Administrationsaufwand minimiert. Die Redundanz erlaubt auch große Skalierbarkeit der Leseoperationen, da nahezu beliebige Knoten im Set zum Lesen verwendet werden können.

Eine praktische Idee ist auch die Verwendung eines leicht verzögerten Sekundär-Knotens, der als Katastrophenschutz eingesetzt werden kann um destruktive Vorgänge wie das versehentliche Löschen einer Collection rückgängig zu machen.

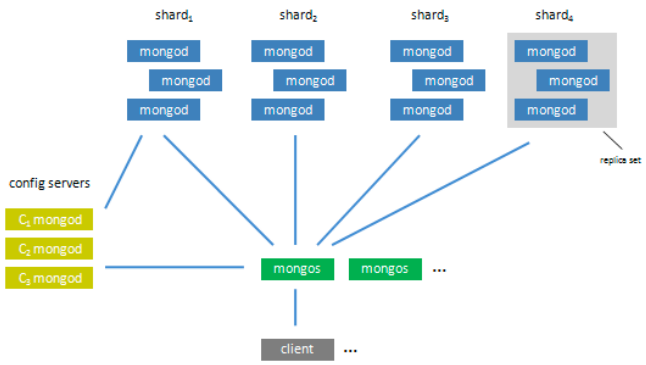


Abbildung 6: Sharding-Architektur

Neben der Replikation zur Ausfallsicherheit und Lastbalancierung ist auch das Partitionieren der Datenbestände auf verschiedene Server ein wichtiges Mittel um die Performance des Datenbanksystems zu steigern. Man spricht auch von horizontaler Skalierung, da nicht das Hardware/Software-System hochgerüstet wird, sondern die Daten selber skaliert werden. In MongoDB wird das durch Sharding erreicht.

Die Grundidee beim Sharding ist das Aufteilen der Dokumente einer Collection in kleinere Gruppen, auch Chunks genannt, die dann auf verschiedene Replica Sets verteilt werden können. Diese bezeichnet man als Shards. Zur Aufteilung wird ein bestimmter Schlüssel, der Sharding Key, verwendet. Dies kann ein einzelnes Feld oder ein zusammengesetzter Schlüssel sein. Neben den Shards gibt es noch Konfigurationsserver im System, welche die Meta-Daten verwalten und Informationen darüber speichern, welche Chunks in welchem Shard zu finden sind.

Das Kernstück der Sharding-Architektur sind aber die mongos. Das sind die Server, an die der

Client Anfragen richtet. Sie sehen nach außen aus wie normale MongoDB-Server, haben aber keinen inneren Zustand, sondern beziehen alle Daten mithilfe der Informationen aus den Konfigurationsservern von den Shards. Die Antworten der Shards werden gesammelt, ggf. weiterverarbeitet und dann dem Client zugänglich gemacht.

Weitere Details finden sich hier [14].

6 Weitere Features

In diesem Kapitel sollen drei weitere Feature von MongoDB vorgestellt werden: Index-Strukturen, Server-side functions und stored procedures.

MongoDB erlaubt das Erzeugen von Index-Strukturen auf Collections, die konzeptuell denen in RDBMS entsprechen. Für die Implementierung wurden B-Bäume verwendet [15].

Ein Index über das „_id“-Feld ist in jedem Fall enthalten. Weitere Indizes über einzelne Felder oder auch über Felder in eingebetteten Dokumenten (Dot-Notation) können angelegt werden. Zusätzlich erlaubt MongoDB auch Indizes über mehrere Felder, sogenannte zusammengesetzte Indizes. Um den Index im Blick auf die erwarteten späteren Anfragen zu optimieren, kann für jedes Feld die Indizierungsrichtung angegeben werden. Auf diese Weise kann das Sortieren oder Berpeichsanfragen durch den Index sehr beschleunigt werden. Je nach Bedarf können in dem Index alle Dokumente, oder nur diejenigen, die das Index-Feld enthalten, gespeichert werden.

Über Unique-Indizes kann man serverseitig einfordern, dass die Werte für ein bestimmtes Feld innerhalb der Collection eindeutig sind und optional bei Index-Erstellung Duplikate entfernen lassen.

Bei Anfragen oder Änderungsoperationen über das Query-Interface von MongoDB werden die erzeugten Indexstrukturen automatisch zur Beschleunigung herangezogen und führen zu erheblichen Geschwindigkeitssteigerungen. Sie können also zur Optimierung der Anfragezeiten herangezogen werden in einer ganz ähnlichen Weise, wie man es von RDBMS kennt.

MongoDB bietet auch eine server-seitige Programmierschnittstelle. Diese verwendet Javascript als Programmiersprache und erlaubt es einem Client, nahezu beliebigen Code auszuführen. So können z.B. spezielle Suchprogramme, Abläufe oder Berechnungen programmiert und auf dem Server ausgeführt werden.

Da Javascript-Code einfacher Text ist und sich so leicht in Dokumenten ablegen lässt, bietet MongoDB auch die Möglichkeit von stored procedures. Es gibt eine spezielle Collection „system.js“, in der Dokumente mit Javascript-Befehlen abgelegt werden können. Ihr „_id“-Feld dient als Funktionsname für einen späteren Aufruf. Ein Beispieldokument könnte etwa so aussehen:

```
{ _id : "foo" , value : function( x , y ){ return x + y; } }
```

Es gestattet einen späteren Aufruf als `foo(3, 5)`.

So erzeugte Funktionen können z.B. für spezielle Suchanfragen mit $where verwendet werden.

7 Transaktionen

In diesem Kapitel soll MongoDB unter dem Gesichtspunkt von Transaktionen mit SQL-Systemen verglichen werden. Da es im Sprachgebrauch von MongoDB keine Transaktionen gibt, muss der Begriff zunächst definiert werden.

Eine Transaktion kapselt mehrere, hintereinander ausgeführte Datenbankoperationen (Anfragen, Änderungen, etc.). Die Ausführung einer Transaktion ist atomar: entweder werden alle gekapselten

Operationen ausgeführt, oder gar keine.

In SQL-Systemen wird auch noch gefordert, dass die Transaktionen voneinander isoliert sind, eine erfolgreiche Transaktion nicht mehr verloren gehen kann und die Konsistenzbedingungen eingehalten werden. Dies führt zu den ACID-Eigenschaften.

Es stellt sich die Frage, ob man mit MongoDB auch die ACID-Eigenschaften für den etwas allgemeineren Transaktionsbegriff als Hintereinanderausführung von Datenbankoperationen erreichen kann.

Von Haus aus ist nur die Änderung eines Dokuments atomar. Es können auch praktisch keine Lese- oder Schreibsperren, wie in einem RDBMS, gesetzt werden.[1] Demzufolge kann die Forderung der Isolierung nicht erfüllt werden und es ist mit allen üblichen Folgen, wie etwa „phantom reads“ zu rechnen.

Durability kann MongoDB allerdings zusichern, zumindest wenn Replikation verwendet wird und die Datenbank so gegen Ausfälle gesichert ist. Die Einhaltung von Konsistenzbedingungen garantiert MongoDB ebenfalls nicht, da in dem System keine Constraints oder referenzielle Integrität vorgesehen ist. Wenn so etwas benötigt wird, muss es auf der Ebene der Anwendungssoftware durchgesetzt werden.

Ein Hauptgrund für die Einführung von Transaktionen mit ACID-Eigenschaften in SQL-Systemen ist, dass Informationsverlust nicht auftreten darf. Wenn zwei Parteien einen Wert lesen und verändern, überschreibt die zweite Partei den geänderten Wert der ersten und Information ist verloren gegangen. Durch Transaktionsmanagement kann verhindert werden, dass die zweite Partei ihren neuen Wert schreibt ohne den neuen Wert der ersten Partei zu kennen. Für Bankanwendungen ist dies besonders wichtig.

In MongoDB kann dieser Informationsverlust aber durch andere Mittel als Isolation verhindert werden. In den meisten Fällen verändert eine Transaktion zwei Datensätze. Der eine wird inkrementiert und der andere dekrementiert, wie es etwa bei Bank-Anwendungen häufig vorkommt. Dabei ist es im Prinzip nicht wichtig, dass die beiden Datensätze gleichzeitig und untrennbar voneinander verändert werden, sondern nur, dass beide verändert werden und spätere Änderungen die vorangegangenen nicht überschreiben, sondern berücksichtigen. Im Beispiel der Bank steckt die Information auch nur indirekt in dem neuen Kontostand, sondern mehr in der Höhe des überwiesenen Betrags. MongoDB bietet mit den Änderungsoperatoren (siehe Kapitel 4.4) eine Möglichkeit, genau diese Information zu verwenden. Es kann eine Anfrage mit einer Inkrementoperation und eine zweite mit einer Dekrementoperation ausgeführt werden. Das Datenbanksystem garantiert Atomarität der Änderung eines Dokuments und führt das Inkrement selbst aus, sodass nachfolgende Änderungen vorangehende nicht einfach überschreiben können.

Durch die Verwendung von Änderungsoperationen können also die selben Anwendungsfälle abgedeckt werden, die sich auf die Isolierung und Atomarität von Transaktionen in SQL-Systemen verlassen. Gleichzeitig kann aber durch das Vermeiden von Sperren die Geschwindigkeit gegenüber RDBMS stark gesteigert werden.

8 Bewertung

In diesem Kapitel soll abschließend ein kurzer Vergleich zu einer anderen wichtigen Dokumentendatenbank und auch zu einem relationalen Datenbanksystem gezogen werden.

1 Eingeschränkt sind Schreibsperren möglich. Die Update-Anweisung hat eine Option, die Schreibzugriffe während dem Update unterbindet, sodass innerhalb einer einzigen Update-Anweisung mehrere Dokumente atomar geändert werden können.

8.1 Vergleich mit CouchDB

CouchDB [16] ist eines der ersten Dokumentendatenbanksysteme. Im Vergleich zu MongoDB fällt allerdings zunächst die schlechte Dokumentation auf, was die Bewertung erschwert.

CouchDB verwendet als Datenprotokoll nicht BSON, sondern JSON. Die Kommunikation zwischen Client und Server wird nicht über ein BSON-basiertes Protokoll, sondern über ein REST-Interface realisiert.

Laut Dokumentation unterstützt CouchDB Transaktionen und kann die vollen ACID-Eigenschaften sicherstellen. Im Gegenzug dazu bietet MongoDB keine Transaktionen an, kann dies aber durch eine ausgeklügelte und sehr mächtige Query-Sprache ausgleichen. Die Ausdrucksmächtigkeit von MongoDB ist in dieser Klasse von NoSQL-Datenbanken einzigartig.

8.2 Vergleich mit MySQL

Im Rahmen dieser Arbeit wurde bereits gezeigt, dass der Funktionsumfang von MongoDB fast alle Sprachmittel abdeckt, die man von SQL-Systemen kennt. MongoDB stellt an sich den Anspruch, nicht nur den Funktionsumfang von SQL zu bieten, sondern gleichzeitig die hohe Geschwindigkeit von Key-Value-Stores. Im Folgenden soll dies exemplarisch anhand einer Studie überprüft werden.

Im Rahmen des Betriebsdatenpraktikums am Lehrstuhl Betriebliche Informationssysteme, Fakultät für Informatik und Mathematik, Universität Leipzig,[18] hat Martin Junghanns die Geschwindigkeit von MongoDB und MySQL gegenübergestellt. Seine Ergebnisse finden sich hier [17].

Die zugrundeliegenden Daten sind Zeitreihen mit bis zu 30 Millionen Datensätzen. Anfragen sind typische Bereichsanfragen, wie etwa die Messpunkte ab Tag x oder in einem bestimmten Zeitbereich.

Zu acht verschiedenen Anfragen wurden Zeitmessungen angefertigt. Auf den ersten Blick wird bereits deutlich, dass die benötigte Zeit zur Ergebnisermittlung relativ konstant bzgl. der Anzahl der Dokumente in der Collection ist, also von der Ergebnismenge abhängt. Dies lässt auf eine hohe Qualität der Indexstrukturen schließen.

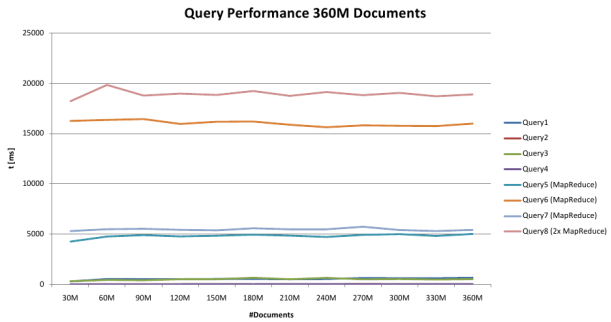


Abbildung 7: Geschwindigkeit verschiedener Anfragen (Quelle: [17])

Ebenfalls gut erkennbar ist, dass Map-Reduce-Anfragen trotz Vorselektierung der Dokumente deutlich länger brauchen, als Anfragen mit der MongoDB-Query-Sprache. Ein Vergleich mit Anfragen auf einer MySQL-Datenbank (wurde von Stefan Thomas ebenfalls im Rahmen dieses Praktikums vorgenommen), zeigt, dass MySQL etwas doppelt so lange benötigt.

Ein großer Nachteil von MongoDB ist an dieser Stelle allerdings die Schemalosigkeit. Für jedes Dokument werden die Feld-Bezeichner vollständig abgelegt, wodurch erheblich mehr Speicher benötigt wird, als bei MySQL. Bei 35 Millionen Messpunkten betrug die Größe der Datenbank ca. 44 GB.

9 Zusammenfassung

MongoDB ist eine verteilte Dokumentendatenbank mit einem großen Anspruch: möglichst den vollen Funktionsumfang eines relationalen Datenbanksystems bei gleichzeitig so großer Geschwindigkeit wie Key-Value-Stores zu erreichen.

In dieser Arbeit wurde die umfangreiche Anfragesprache und die Werkzeuge zum Erstellen einer Datenbankanwendunge mit MongoDB vorgestellt und gezeigt, dass die wichtigsten Sprachmittel von SQL und viele Features der großen RDBMS zur Verfügung stehen [19].

Auch wurde gezeigt, dass es durch die Änderungsoperationen von MongoDB auch ohne Datenbanksperren und komplexe Transaktionen möglich ist, zusammengesetzte Aktionen auf mehreren Datensätzen sicher und konsistent durchzuführen.

Abschließend wurde die Geschwindigkeit von MongoDB anhand eines Beispiels kurz demonstriert und die Überlegenheit gegenüber MySQL gezeigt.

Insgesamt ist MongoDB ein sehr einfach einzusetzendes System mit einer aktiven Community und einer sehr guten Dokumentation. Die Fähigkeiten des Systems werden ständig ausgebaut. Gerade auf dem Gebiet der Replikation und des Sharding wird viel investiert und sind schnelle Fortschritte zu beobachten (bereits in dem kurzen Zeitraum des Seminars ist eine neue Replikationsmethode dazugekommen). Durch die Treiber für viele bekannte Programmiersprachen und die schnelle, unkomplizierte Einsatzfähigkeit sowie die leichte Administration ist MongoDB für viele Anwendungen sehr attraktiv. Gerade in Anwendungen, in denen es auf Echt-Zeit-Verarbeitung ohne komplexe Transaktionen ankommt, kann MongoDB punkten.

Es wurde also ein konkurrenzfähiges Produkt geschaffen, dass seinem Anspruch gerecht wird.

Literaturverzeichnis

[1] http://www.mongodb.org/display/DOCS/Philosophy
[2] http://www.mongodb.org/display/DOCS/Tutorial
[3] http://www.mongodb.org/display/DOCS/Overview+-+The+MongoDB+Interactive+Shell
[4] http://www.mongodb.org/display/DOCS/Drivers
[5] http://www.mongodb.org/display/DOCS/Data+Types+and+Conventions
[6] http://json.org/
[7] http://bsonspec.org/
[8] http://www.mongodb.org/display/DOCS/Documents
[9] http://www.mongodb.org/display/DOCS/Inserting
[10] http://www.mongodb.org/display/DOCS/Advanced+Queries
[11] http://www.mongodb.org/display/DOCS/Updating
[12] http://www.mongodb.org/display/DOCS/MapReduce
[13] http://www.mongodb.org/display/DOCS/Replication
[14] http://www.mongodb.org/display/DOCS/Sharding+Introduction
[15] http://www.mongodb.org/display/DOCS/Indexes
[16] http://couchdb.apache.org
[17] https://github.com/s1ck/MongoDB_eval
[18] http://bis.informatik.uni-leipzig.de/de/Lehre/1112/WS/LV/BDE?v=15xu
[19] http://www.mongodb.org/display/DOCS/SQL+to+Mongo+Mapping+Chart