Dokumentenorientierte Datenbanken

Tristan Eisermann
Fakultät Informatik
Hochschule Furtwangen
Furtwangen, Deutschland
tristan.elias.eisermann@hs-furtwangen.de

Marius Gerold
Fakultät Informatik
Hochschule Furtwangen
Furtwangen, Germany
marius.gerold@hs-furtwangen.de

***Abstract*—Durch die immer größer werdenden Datenmengen und den in diesem Zusammenhang fallenden Begriff "Big Data" kommen relationale Datenbanksysteme an ihre Grenzen. Daher rücken NoSQL-Datenbanken immer mehr in den Fokus. Eine Unterart der NoSQL Datenbanksysteme sind die dokumentenorientierten Datenbanksysteme. Diese speichern Daten schemalos in Dokumenten. Diese Arbeit soll eine Einführung in dokumentenorientierte Datenbanken bieten. Dazu werden einige Systeme historisch betrachtet. Auch die Relevanz von NoSQL und insbesondere dokumentorientierten Datenbanken soll verdeutlicht werden. Außerdem soll auf den momentanen Stand der Forschung eingegangen werden. Besonderer Fokus liegt dabei auf Clustering, ACID-Kompatibilität und Security Aspekten von dokumentenorientiterten Datenbanken. Darüber hinaus wird auf aktuelle Neuheiten in diesen Systemen eingegangen.**

***Index Terms*—document-oriented Database, mongodb, couchbase, document-oriented functionality, acid, clustering, shards, security**

I. Introduction

Lange Zeit dominierten relationale Datenbanken den Markt. Seit Mitte bis Ende der 2000er Jahre gibt es auch immer mehr NOSQL-Datenbanken. Diese lassen sich grob in vier Kategorien klassifizieren. Key-Value Datenbanken, Dokumentendatenbanken, *Extensible record data stores*, die in der Literatur auch als Column-Datenbanken bezeichnet werden und Graphdatenbanken [1]. Diese Arbeit soll einen allgemeinen Überblick zu dokumentenorientierten Datenbanken liefern. Nach dem TOPDB-Datenbank-Index database Index, welcher auf Google-Suchen beruht, befindet sich die dokumentenorientierte Datenbank MongoDB auf dem fünften Rang der beliebtesten Datenbank-Systeme [2]. Sowohl CouchDB als auch MongoDB traten zum ersten mal 2010 in diesem Ranking auf. Die Firma 10gen begann die Entwicklung von MongoDB im Oktober 2007. Im Jahr 2013 folgte die Umbenennung der Firma zu MongoDB Inc [3]. MongoDB was bis 2018 Open Source. Seit 2018 wird MongoDB unter der Server Side Public License (SSPL) vertrieben .

Der Name CouchDB steht für das Akronym "**C**luster **o**f **u**nreliable **c**ommodity **h**ardware **D**ata **B**ase". CouchDB wird seit 2005 als freie Software entwickelt. Dabei wurde CouchDB bis 2011 intern für das Linux Betriebssystem Ubuntu genutzt. Das heißt, damals war CouchDB durchaus relevant. Nach TOPDB-Datenbank-Index ist CouchDB heute weniger relevant. Stattdessen tritt der Name Couchbase auf. Einige der Entwickler von CouchDB sind 2011 zu CouchBase gewechselt.

Neben klassischen dokumentenorientierten Datenbanken gibt es auch noch eine dokumentenorientierte Suchmaschine, die hier erwähnt wird. ElasticSearch wird häufig zur Log-Aggregation im Umfeld der Web-Entwicklung genutzt. Die Speicherung der Daten basiert auf Dokumenten. Entwickelt wurde die Suchmaschine von Shay Bannon. Die Entwicklung begann, weil der Vorgänger Compass zu wenig skalierbar war. Die erste Version von ElasticSearch wurde im Februar 2010 veröffentlicht. Elasticsearch basiert auf der Suchmaschinen-Software Bibliothek Apache Lucene [4].

Da der TOPDB-Datenbank-Index auf Google-Suchen basiert, werden bestimmte Datenbanken stärker bewertet. Beispielsweise ist Oracle auch der Firmenname, und man weiß nicht, ob die Datenbank oder die Firma gegooglet wurde. Daher wird bei der Bewertung der Relevanz noch die StackOverflow-Umfrage 2021 betrachtet. Bei ungefähr 73000 Teilnehmenden wurde die relationale Datenbank MySQL mit 50.18% am populärsten bewertet. Die genau Frage war, welche Datenbank man extensiv im letzten Jahr genutzt hat und im kommenden Jahr nutzen möchte. Auf Platz 2 befindet sich die relationale Datenbank PostgreSQL mit 40.42%. Sowohl MySQL als auch PostgreSQL bieten zusätzlich zu relationalen Funktionalität auch dokumentenorientierte Funktionalität. Auf diese Funktionalität wird in IV genauer eingegangen. Die dokumentenorientierten Datenbanken, MongoDB(27.7%), Firebase(16.7%), ElasticSearch(13.27%) und Couchbase(1.57%) befinden sich auf den Rängen 4,8,9 und 14 [5].

Wie man anhand dieser Umfrage erkennen kann, spielen dokumentenbasierte Datenbanken durchaus eine Rolle unter allen Datenbank Management Systemen. Allerdings muss immer in Frage gestellt werden, wie repräsentativ solche Umfragen sind, und ob die Datenbanksysteme, die dabei am meisten genutzt werden, auch bei anderen Entwicklern so beliebt sind.

II. Related work

Da NoSQL und insbesondere dokumentenorientierte Datenbanken im letzten Jahrzehnt an Bedeutung gewonnen haben, gibt es zu diesem Thema viel Literatur. Im Folgenden werden verschiedene wissenschaftliche Arbeiten erwähnt und kurz beschrieben.

Samantha et al untersuchen die Performance zwei unterschiedlicher Mongo-DB Cluster. Es überrascht dabei, dass der selbstkonfigurierte Cluster eine bessere Performance aufweist als der vordefinierte Cluster eines Cloud-Dienstes [6].

Sanchez et al bewerten die NoSQL-Datenbanken Redis, Cassandra und MongoDB bezüglich ihrer Sicherheit. Dabei fällt die Bewertung für das dokumentenbasierte MongoDB am schlechtesten aus. Cassandra ist nach diesen Tests am sichersten [7].
Ob zusätzliche Indexe über eine SQL-Datenbank die Performance von MongoDB verbessern können, untersuchen Ceresnak et al. Eine Performancesteigerung um 12% deutet sich dabei an [8].
In [9] vergleichen Chopade et al die dokumentenbasierten Datenbanken Couchbase und MongoDB bezüglich ihrer Performance bei der Bildspeicherung. Couchbase ist schneller beim Schreiben der Bilder und MongoDB ist schneller beim Lesen der Bilder. Vera-Olivera et al untersuchen Methoden zur Modellierung von NoSQL-Datenbanken. Dabei werden Erweiterungen zum ursprünlich für relationale Datenbanken entwickelten Entity Relationship(ER)-Modell und der Modellierungssprache Unified Modeling Language (UML) untersucht. Auch auf Alternativen zu diesen Technologien wird eingegangen. Dabei wird insbesondere die Generierung von Schemata in JSON, XML und XMI behandelt. Außerdem werden Methoden zur Modellierung der Systemanforderungen bezogen auf Verfügbarkeit, Konsistenz und Skalierbarkeit betrachtet [10].

III. Grundlagen

Mark Drake von Digital Ocean definiert dokumentenorientierte Datenbanken wie folgt: "Document oriented databases, or document stores, are NoSQL databases that store data in the form of documents" [11]. Ein Dokument ist dabei die Speicherform, welche beispielsweise im Format JSON gespeichert sein kann. Dabei werden Daten anhand von *Collections* aufgeteilt. Eine *Collection* ist dabei einer Sammlung zusammengehörender Dokumente. Diese ist dabei vergleichbar mit einer Tabelle in einer relationalen Datenbank. Ein *Document* entspricht einer *Row* in einer relationalen Datenbank. Zuletzt entspricht das *Field* einer dokumentenorientierten Datenbank ungefähr einer *Column* in einer relationalen Datenbank.
Die meisten dokumentenorientierten Datenbanken basieren auf dem Format JSON. Allerdings gibt es auch xml-basierte Datenbanken. Für MongoDB wurde zusätzlich noch ein binäres JSON-Format entwickelt. Diese Format wird als BSON bezeichnet, was für Binary JSON steht. Zu diesem Format gibt es auch eine Spezifikation [12].
Normalerweise sind Daten in dokumentenorientierten Datenbanken schemalos. Collections definieren keine Felder, wie das beispielsweise relationale Tabellen tun. Das heißt die Daten liegen in keiner normalisierten Form vor. Allerdings gibt es auch JSON-Schemata, über welche sich gewisse Einschränkungen bezüglich der Form von Dokumenten definieren lassen [13]. Dokumente lassen sich dann anhand eines Schemas validieren. Beispielsweise kann bei einem Insert das einzufügende Dokument erst anhand eines Schemas überprüft werden. Über eine Schema lassen sich beispielsweise verpflichtende Felder und Feldtypen definieren.

IV. Dokumentenorientierte Funktionalität in relationalen Datenbanken

Auch in relationalen Datenbanken wurde aufgrund der Popularität von dokumentenorientierten Datenbanken, dokumentenorientierte Funktionalität eingebaut. In diesem Kapitel soll auf die eingebauten Funktionalitäten von PostgreSQL und MySQL eingegangen werden, da diese, wie in Kapitel I beschrieben, populäre Datenbanksysteme sind.
PostgreSQL erlaubt die dokumentenbasierten Formate XML und JSON als Feldtypen im Datenbanksystem. Diese könnten auch als Text gespeichert werden. Durch den JSON Datentyp wird sichergestellt, dass jeder gespeicherte Wert den JSON-Regeln gerecht wird. Neben dem JSON Typen gibt es auch noch einen jsonb-Typen. Diese akzeptieren die gleichen Eingaben. Jsonb ist vergleichbar zu BSON in MongoDB. Durch ein binäres Format ist das Lesen der Daten schneller, da diese zur Übertragung nicht umgewandelt werden müssen. Allerdings ist das Einfügen in die Datenbank langsamer, da hier die Daten in das binäre Format umgewandelt werden müssen. Beim Einfügen ist Serialisierung notwendig, beim Auslesen dagegen nicht. Ein weiterer Unterschied ist, dass Leerzeichen und mehrere Werte für einen Schlüssel im JSON Format gespeichert werden, während diese bei jsonb entfernt werden. Bei doppelten Werten für einen Schlüssel wird bei jsonb nur der letzte gespeichert [14]. Für die dokumentenbasierten Typen gibt es in PostgreSQL zudem Operatoren. Beispielsweise lassen sich damit Elemente aus einem JSON-Array extrahieren und in einen anderen Datentyp umwandeln. Für jsonb gibt es noch Containment-Operatoren. Der Operator `@>` gibt dabei an ob der erste Wert den zweiten beinhaltet. Beispielweise würde `'"a":1, "b":2'::jsonb @> '"b":2'::jsonb` true zurückliefern, da sich der Wert 2 im ersten Array befindet. Neben Operatoren gibt es noch JSON-Funktionen. Diese werden in Creation-Funktionen und Processing-Funktionen aufgeteilt. Ein Beispiel für eine Creation-Funktion wäre `to_json ( anyelement )` zur Umwandlung eines Elements in JSON. Ein Beispiel für eine Processing-Funktion wäre `json_array_length ( json )` zur Rückgabe der Länge eines *JSON*-Arrays [15].
Bei MySQL wird die dokumentenorientierte Funktionalität durch einen Document Store realisiert. In Fig. 1 sieht man die Architektur des MySQL Document Store. Die MySQL Connectors bilden die Schnittstelle zu verschiedenen Programmiersprachen. Die ermöglichen den Zugriff auf den Document Store aus diesen Programmiersprachen. Die Abbildung ist dabei teilweise aus [16] übernommen. Dabei gibt es Verbindungen zu Node.js, PHP, Python, Java, .NET und C++. Die MySQL Shell ist ein Terminal zur Administration des MySQL Servers auf dem der Document Store liegt. Die X DevAPI führt Collections für CRUD-Operationen auf dem Document Store ein. X Protocol ist ein neues Protokoll, um sowohl SQL als auch dokumentenbasierte CRUD Operationen zu ermöglichen [16]. Das Arbeiten auf dem Document Store ist ähnlich zum Arbeiten mit MongoDB. Beispielsweise lassen sich auch mit `db.getCollections()` die mo-

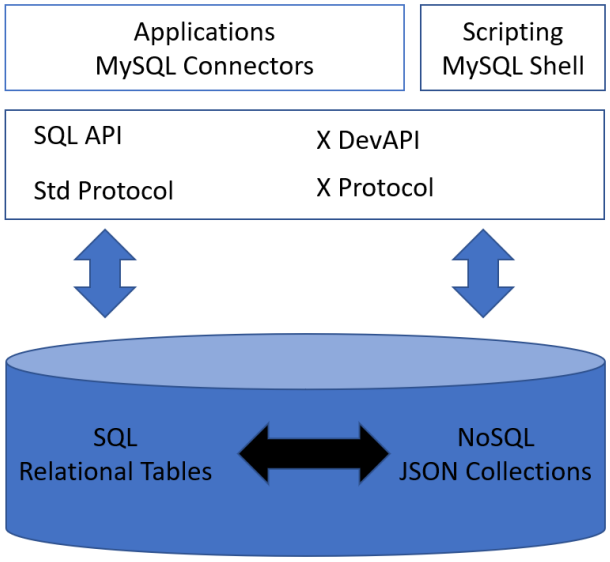

Fig. 1. Architektur: MySQL Document Store

mentanen Collections in der Datenbank ausgeben. Oder mit `db.name.find()` lassen sich Werte aus der Datenbank *db* in der Collection *name* zurückgeben [17].

V. Neuerungen in dokumentenorientierten Datenbanken

Um momentane Neuerungen in dokumentenorientierten Datenbanken zu analysieren werden beispielhaft die Release Notes des letzten Major-Release von MongoDB betrachtet. Release 5.0.0 wurde am 13. Juli 2021 veröffentlicht. Neuerungen waren unter anderem die *Time Series Collections*. Diese dienen zur effizienten Speicherung von Zeitreihen-Daten, die über einen Zeitraum gesammelt wurden. Dabei soll die Abfrageeffizienz und der Speicherbedarf optimiert werden.
Im zeitlichen Kontext gibt es auch neue Operatoren, wie beispielsweise `$dateAdd`/`dateSubtract` zum inkrementieren/dekrementieren eines Datums oder `$dateDiff` zum Ausrechnenn eines Zeitraums zwischen zwei Datumsangaben. Außerdem gibt es neuerdings ein `$count` mit einem Akkumulator. Ein weiterer neuer Operator ist `$rand` zum generieren einer Zufallszahl zwischen 0 und 1.
`$setWindowFields` ist ein Operator um Daten in einer MongoDB-Aggregation-Pipeline in Windows einzuteilen. Eine Aggregation-Pipeline besteht aus einzelnen Schritten um Daten aus Dokumenten zu verarbeiten [18]. Dadurch lassen sich dann Berechnungen auf den spezifischen Daten ausführen, zum Beispiel Verkaufsranglisten oder die summierten Verkaufszahlen eines Zeitfensters.
Seit MongoDB 5.0.0 lassen sich auch Cluster resharden. Das heißt, die Daten eines Clusters werden neu auf den Knoten des Clusters verteilt. Der MongoDB-Cluster wird noch genauer in VI erläutert. Die Schema-Validation-Fehler wurden überarbeitet, sodass klarer wird, welche Aspekte dem Schema nicht gerecht wurden.
Um die Sicherheit zu erhöhen, wurde eine *Online Certificate Rotation* zwischen den Sicherheitstechnologien Transport Layer Security(TLS), Certificate Revocation List(CRL) und Certificate Authority(CA) ermöglicht.
Da eine Vielzahl neuer Kommandos hinzugefügt wurde, wurden auch einige alte Kommandos entfernt. Beispielsweise gibt es den Befehl `db.collection.ensureIndex()` nicht mehr. Stattdessen soll `db.collection.createIndex()` verwendet werden. Außerdem wurde ein Befehl entfernt, der das Clustering zurücksetzten sollte. Dies soll ab Version 5.0.0 nicht mehr möglich sein. Zudem wurde der Operator `$geoSearch` entfernt. Als Ersatz kann `$geoNear` oder ein anderer geospatial query operator genutzt werden.
Wie man an dieser Menge an Änderungen sehen kann, entwickeln sich dokumentenorientierte Datenbanken auch heute noch.

VI. MongoDB-Cluster

In diesem Kapitel sollen verschiedene Konzepte des MongoDB-Clustering erläutert werden. Dabei wird hauptsächich auf die Konzepte in der MongoDB-Dokumentation zum Sharding eingegangen [19]. In Fig. 2 ist der grundlegende Aufbau eines Mongo-Clusters zu sehen. Dieser Aufbau

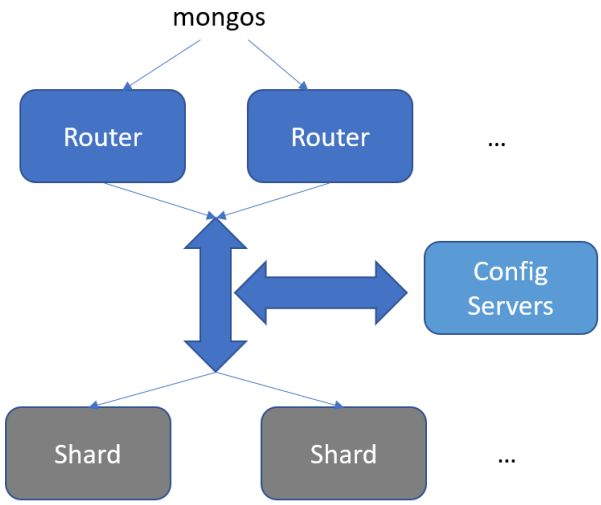

Fig. 2. Aufbau Mongo Cluster

besteht aus drei Kernkomponenten. Auf den Shards liegen die verteilten Daten. Ein Shard is also ein Server auf dem Teile der Daten liegen. Dabei werden die Daten anhand des Shard-Keys aufgeteilt. Es muss mindestens zwei Shards in einem Cluster geben. Die Router verwalten die Anfragen und leiten diese an die Shards weiter. Es muss mindestens einen Router geben. Die Anfragen in einem Mongo-Cluster werden mit dem Terminal-Befehl `mongos` realisiert. Die Config-Server halten die Metadaten und Konfigurationen des Servers. Sowohl Shards als auch Config-Server befinden sich in einem Replicaset. Bei Ausfahl eines Shards/Config-Servers wird ein Ersatz über das Replicaset genutzt.
Wie schon beschrieben, werden die Daten anhand eines Shard-Keys auf die Shards aufgeteilt. Ein Shard-Key besteht aus

einem oder mehreren Feldern einer Collection. Dieses Feld kann seit der Version 4.4 auch `null` sein. Davor war ein Eintrag in diesem Feld verpflichtend. Seit Version 5.5 lassen sich die Daten auf den Shards auch neu verteilen. Dieser Prozess nennt sich *Resharding*. Um eine Collection zu aufzuteilen muss ein Index auf dem Shard-Key angelegt werden. Bei einer leeren Collection legt MongoDB automatisch einen Index auf dem Shard-Key an, falls dieser noch nicht existiert.
In einem Cluster kann es gleichzeitig sharded und non-sharded Collections geben. Non-sharded Collections befinden sich nur auf einem einzelnen Shard, während sharded Collections auf mehreren Shards verteilt sind. Es gibt verschiedene Strategien beim Sharden von Collections. Beim *Hashed Sharding* wird ein Hash basierend auf dem Wert des Shard-Keys berechnet. Die Daten werden dann anhand dieses Hashes in *Chunks* aufgeteilt. Ein *Chunk* ist ein Teil der Daten mit inklusiver unterer Grenze und exklusiver oberer Grenze. Im Gegensatz dazu wird beim *Ranged Sharding* direkt der Wert des Shard-Keys verwendet. Anhand dieses Wertes werden die Daten in *Ranges* aufgeteilt. Diese *Ranges* werden dann *Chunks* zugeordnet. Eine schlechte Wahl des Shard-Keys kann zu einer ungleichmäßigen Aufteilung der Daten auf den Shards führen. Dadurch können einige Vorteile vom Sharding aufgehoben werden.
Sharding in MongoDB hat einige Vorteile. Beispielsweise werden Reads und Writes auf die Shards aufgeteilt, was zu einer Performance Steigerung führen kann. Außerdem kann die Größe des Speichers auf den Shards durch Hinzufügen von neuen Shards skaliert werden. Da sowohl Shards als auch Config-Server Replicasets definieren müssen, ist auch die Verfügbarkeit in einem MongoDB-Cluster höher, als wenn kein Clustering genutzt wird.

VII. ACID

Um ACID genauer zu betrachten, sollten zuerst Transaktionen definiert werden. Eine Transaktion ist eine Folge einer oder mehrerer Operationen, die als eine Einheit behandelt werden. ACID steht für Atomicity, Consistency, Isolation und Durability [20]. Die ACID Prinzipien beschreiben vier Eigenschaften, die eine Transaktion besitzen kann. Eine Datenbank, deren Transaktionen alle vier Eigenschaften besitzt ist ACID konform. Ein Dokumentenorientertes Datenbanksystem, welches von sich selbst behauptet ACID konform zu sein, ist Couchbase [21]. Im folgenden werden die einzelnen Eigenschaften genauer erläutert und anhand von Couchbase exemplarisch veranschaulicht, wie eine Umsetzung in Dokumentenorientierten Datenbanksystemen aussehen kann.

A. Atomicity

Die Atomizität ist eine wichtige Eigenschaft von Transaktionen. Diese garantiert, dass die Menge der Aktionen einer Transaktion entweder vollständig ausgeführt werden, oder alle fehlschlagen [20].
Bei der Betrachtung von Atomizität im Bereich von Dokumentenorientierten Datenbanksystemen fällt auf, dass diese häufig nur Garantien auf "Single-document" Ebene anbieten [21]. Die Fähigkeit atomare Transaktionen in verteilten Systemen und über mehrere Dokumente hinweg durchzuführen, wurde in Couchbase erst im Januar 2020 mit Version 6.5 eingeführt [22]. Dennoch ist Couchbase bereits seit 2012 in Produktionssystemen im Einsatz [23] und das ohne verteilte atomare Transaktionen. Warum also Firmen Couchbase bereits im Einsatz hatten, bevor verteilte atomare Transaktionen überhaupt in der Diskussion zur Umsetzung standen, lässt sich mit der Natur der Daten in Dokumentenorientierten Datenbanksystemen erklären. Daten in Dokumentenorientierten Datenbanksystemen sollten primär denormalisiert sein. Diese Denormalisierung führt dazu, dass zusammengehörige Daten meistens vollständig in einem Dokument vorliegen. So betreffen Transaktionen, die ebenfalls zumeist zusammengehörige Daten betreffen, entsprechend nur ein Dokument. Aus diesem Grund sind Dokumentenorientierte Datenbanksysteme bereits seit Jahren auch ohne verteilte Transaktionen im Einsatz in Produktionssystemen. Laut einem Blog Artikel von MongoDB benötigen schätzungsweise 80% bis 90% keine verteilten oder Multidokument Transaktionen, da Daten denormalisiert vorliegen [24].

B. Consistency

Konsistenz bedeutet, dass eine Operation nicht gegen deklarierte Systemintegritätsbeschränkungen verstoßen darf [20]. Eine konsistente Transaktion stellt also sicher, dass sowohl vor, als auch nach der Transaktion alle Systemintegritätsbeschränkungen eingehalten Couchbase etwa besitzt einige Beschränkungen, beispielsweise muss jedes Dokument eine valide ID besitzen und ein valides JSON Dokument sein. Jedoch ist durch die meist schemalose Art von Dokumentenorientierten Datenbanksystemen nicht gegeben, dass mehrere Dokumente dem gleichen Schema folgen müssen. Andere Dokumentenorientierte Datenbanksysteme wie beispielsweise MongoDB ermöglichen jedoch das Erstellen von JSON-Schemata, welche derartige Validierung zu den Beschränkungen hinzufügen kann.
Eine etwas andere Auffassung der Konsistenz ist im Zuge der sogenannten Query Konsistenz. Couchbase erstellt und aktualisiert Indizes von Dokumenten asynchron [21]. Die Grundeinstellung von Couchbase liefert für Anfragen den aktuellen Zustand der Daten zurück, was dazu führen kann, dass eine Anfrage nach Erstellung eines Dokuments ein leeres Ergebnis zurück liefern kann. Um dies zu Vermeiden kann in Couchbases Abfragesprache N1QL eine ScanConsistency festgelegt werden, welche eine vollständige Indizierung der ganzen Datenbank oder einer Instanz garantiert. Hierdurch kann, falls gewünscht, garantiert werden, dass alle Ergebnisse, die zum Zeitpunkt der Abfrage existieren, zurückgeliefert werden. Diese Konsistenz kommt jedoch nicht ohne zeitliche Kosten, weil hierdurch eine Abfrage möglicherweise länger braucht, da erst auf Indizierung gewartet werden muss.

C. Isolation

Isolation bedeutet, dass die Lese- und Schreibzugriffe einer Transaktion nicht von anderen Lese- oder Schreibzugriffen

beeinflusst werden [20]. Zusätzlich sollten sich Transaktionen, die sich gegenseitig beeinflussen könnten, in eine geordnete Reihenfolge, in der sie ausgeführt wurden, bringen lassen [20]. Couchbase bietet eine sogenannte Read-Commit Isolation auf Ebene einzelner Dokumente [21]. Eine Transaktion führt nach erfolgreicher Durchführung einen Commit durch, sodass die Schreibzugriffe der Transaktion persistent im System existieren und für andere sichtbar sind. Dies bedeutet, dass Anfragen nur Daten, die tatsächlich Committed wurden, liefern können. Um gleichzeitige Schreibzugriffe zu verhindern, werden sogenannte compare-and-swap (CAS) Werte benutzt. Jedes Dokument besitzt einen CAS Wert, welcher sich ändert, wenn das Dokument modifiziert wird. Dieser CAS Wert kann nun mit zwei Strategien verwendet werden, die als Optimistisches und Pessimistisches Locking bezeichnet werden [21].
Beim Optimistischen Locking wird bei einem Lesezugriff der CAS Wert des Dokuments ausgelesen. Beim Aktualisieren des Dokuments wird der selbe CAS Wert wieder übergeben und anschließend überprüft. Wenn der übergebene und der aktuelle CAS Wert des Dokuments übereinstimmen, wird die Aktualisierung zugelassen und das Dokument beschrieben.
Falls das Dokument in der Zwischenzeit jedoch bereits bearbeitet wurde und der CAS Wert entsprechend nicht mehr übereinstimmt, wird der Schreibzugriff verweigert und die Operation schlägt fehl. Um dennoch das Dokument zu bearbeiten, muss die Aktion wiederholt werden. Aus diesem Grund wird diese Strategie Optimistisch genannt, sie geht optimistisch davon aus, dass während einer Aktion keine anderen Aktionen durchgeführt werden und eignet sich entsprechend nur für nicht stark beschriebene Dokumente.
Dokumente, für die ein häufiger Schreibzugriff erwartet wird, sollten mit dem Pessimistischen Locking bearbeitet werden [21]. Beim Pessimistischen Locking wird beim initialen Lesen des zu bearbeitenden Dokuments der aktuelle CAS Wert ausgelesen, aber auch ein Lock erstellt, welches sowohl Schreibzugriffe, als auch weitere Locks ausschließt und verhindert. Hierdurch kann garantiert werden, dass das zu bearbeitende Dokument zwischen Auslesen und Bearbeitung nicht von anderen Operationen beeinflusst werden kann und der Schreibzugriff erfolgreich abläuft. Nach erfolgreichem Schreibzugriff mit passendem CAS Wert wird das Dokument wieder entsperrt und kann von anderer Stelle bearbeitet werden. Zu beachten ist hier, dass ein Lock auf einem Dokument Schreibzugriffe logischerweise nicht beeinflusst.

D. Durability

Die Beständigkeit einer Transaktion garantiert, dass Änderungen einer Transaktion, die erfolgreich Comitted wurde, auch persistent bestehen [20]. Diese Persistenz von Änderungen sollte auch im Falle von Störungen garantiert sein, auch wenn die Störungen unmittelbar nach der erfolgreichen Transaktion geschehen.
Couchbase verfolgt eine "Memory-first" Architektur, das heißt, dass Änderungen bereits bestätigt werden, wenn diese im RAM existieren [21]. Standardmäßig besitzt Couchbase also keine garantierte Persistenz von erfolgreichen Transaktionen. Dennoch lässt sich Couchbase mithilfe von Parametern so konfigurieren, dass eine Beständigkeitsgarantie erreicht werden kann. Couchbase unterstützt hierbei zwei Parameter für Operationen: `ReplicateTo` und `PersistTo` [21]. Der Parameter `ReplicateTo` gibt die Anzahl der Nodes in einem Cluster an, welche die Änderung mindestens erhalten haben müssen, um den Schreibzugriff als erfolgreich anzusehen. Der Parameter `PersistTo` gibt an, wie viele Nodes die Änderungen zusätzlich tatsächlich persistent geschrieben haben müssen. Diese Parameter können auch in Kombination angegeben werden. Zusätzlich existiert die Möglichkeit der Konfiguration des gesamten Systems mit Beständigkeitsleveln [25]. Das erste Level `majority` garantiert, dass eine Änderung auf mindestens einem Großteil der Nodes existiert. Das zweite Level `majorityAndPersistActive` garantiert zusätzlich, dass die Änderung auf der aktiv angesprochenen Node im Speicher persistiert wird. Das dritte Level `persistToMajority` garantiert, dass eine Änderung auf einem Großteil der Nodes persistent im Speicher existiert.

VIII. Security

Vielen Dokumentenorientierten Datenbanksystemen fehlen einige grundsätzliche Sicherheitsgrundlagen und gute Standardkonfigurationen, die eine hohe Sicherheit begünstigen [26], [27], [28], [29]. MongoDB ist besonders anfällig für sogenannte NoSQL Injection Angriffe, welche im folgenden genauer betrachtet werden [26], [27].
SQL Injection ist bereits seit 1998 bekannt [30] und mittlerweile für die meisten Menschen, die mit Datenbanken arbeiten ein Begriff. Was jedoch ist eine sogenannte NoSQL Injection? Ein NoSQL Injection Angriff ist eine Angriff, bei dem Code an eine Applikation geschickt wird, welche diesen Code in ihrer NoSQL Datenbank interpretiert und ausführt, wobei Daten entwendet oder unautorisierter Zugriff erfolgen kann.
Im Falle von MongoDB gibt es einige Anfragen, die für NoSQL Injection anfällig sind, falls Nutzereingaben nicht oder unvollständig überprüft und bereinigt werden.
Eine anfällige Anfrage könnte etwa so aussehen: `User.find({"username": "%username%", "password": "%password%"})`, wobei die mit % gekennzeichneten Wörter durch die Nutzereingaben ersetzt werden.
Unter der Annahme, dass Nutzereingaben nicht bereinigt werden und ein Nutzername, im Folgenden "guest", bekannt ist, könnte ein Angreifer als "password" Eingabe ein JSON-Objekt übergeben. Diese könnte beispielsweise so aussehen: `{"$ne": 1}`. Durch diese Eingabe würde insgesamt diese Anfrage folgendermaßen aussehen: `User.find({"username": "guest", "password": "{"$ne": 1}"})`. Bei dieser Abfrage würde nicht überprüft werden, ob es einen Nutzer mit exakt dem eingegebenen Nutzernamen und dem Passwort gibt, sondern, ob es einen Nutzer mit dem Nutzernamen gibt, bei dem das Passwort nicht gleich 1 ist, was eigentlich immer

der Fall sein sollte, wenn der Nutzername existiert. Hierdurch würde ein unbefugter Angreifer Zugang erhalten, was je nach Anwendungsfall bereits größere Probleme bereiten kann, vorallem, wenn der übernommene Nutzer für administrative Aufgaben autorisiert ist. Auch wenn die Anwendung selbst durch diesen Angriff keine Informationen oder ausnutzbare Zugriffsrechte preisgibt, erhält der Angreifer dennoch Zugang zu den persönlichen Daten des Nutzers.

Wenn ein Angreifer jedoch keine Nutzernamen kennt, lässt sich auch dies mithilfe von NoSQL Injection beheben. Hier könnte die "username" Eingabe etwa so aussehen: `{"$gt": "a"}`.

Diese Eingabe würde den ersten Nutzer, dessen Nutzername alphabetisch größer als "a" ist, zurückgeben und den Angreifer als diesen authentifizieren. Durch Kombination und gezielte Anwendung des `$regex` Operators könnte auf diese Weise sowohl Passwort und Nutzername eines beliebigen, oder sogar aller Nutzer herausgefunden werden [31].

Bisher sind wir davon ausgegangen, dass beliebige Eingaben nicht bereinigt werden. Falls die Anwendung, die angegriffen wird, jedoch Eingaben angemessen bereinigt, lassen sich beispielsweise keine JSON-Objekte übergeben.

Doch auch dieser Fall umgeht den Angriffsvektor unter Umständen nicht vollständig. Falls die Bereinigung in einem Web Frontend geschieht, lassen sich Anfragen mit bösartigen JSON-Objekten über Anwendungen wie beispielsweise Postman [32] direkt an das Backend schicken.

Aus all diesen Beispielen lässt sich in Essenz ein Fazit ziehen, welches bei SQL Datenbanken bereits weit verbreitet und im Bewusstsein vieler Leute ist [33]:

NEVER TRUST USER INPUT

Alle bisher gezeigten Beispiele lassen sich durch angemessenes Bereinigen von Nutzereingaben beheben. Auch Eingaben, die wahrscheinlich aus dem eigenen Frontend kommen, sollten bereinigt werden, da diese auch von einem Proxy manipuliert werden könnten.

Die zu Beginn des Kapitels genannte Behauptung, dass MongoDB besonders Anfällig für NoSQL Injection ist, ist etwas irreführend, da dies primär aus der großen Menge an Features von MongoDB herkommt. Das resultierende Problem entsteht dadurch, dass viele Programmierer, ohne eigene Schuld, nicht die Risiken und Sicherheitsprobleme einiger MongoDB Operatoren kennen, wodurch diese an vermeidbaren Stellen eingesetzt werden [34], [35].

Demnach sollte beim Einsatz von Dokumentenorientierten Datenbanken, wie auch beim Einsatz anderer Technologien, die Nutzereingaben übergeben bekommen, darauf geachtet werden, dass diese Eingaben bereinigt werden um derartige Angriffe zu vermeiden.

IX. Fazit

Um eine Einführung zu dokumentenorientierten Datenbanken zu geben wurde erst auf einige historische Fakten eingegangen. Außerdem wurde die Relevanz dieser Datenbanksysteme anhand eines beliebten Datenbank-Index und einer StackOverflow-Umfrage analysiert. Das Ergebnis war, dass dokumentenorientierte Datenbanken durchaus relevant sind. Auch die Vielzahl an anderer wissenschaftlicher Literatur zu diesem Thema lässt darauf schließen. Um das Thema verständlich darzustellen, wurden kurz die Grundlagen dieser Datenbanksysteme erläutert. Auch auf dokumentenorientierte Funktionalität in relationalen Datenbanken wurde eingegangen. Traditionelle Datenbanksysteme übernehmen teilweise Konzepte von dokumentenorientierten Datenbanken. Um momentane Entwicklungen zu betrachten, wurden die Release Notes der neusten MongoDB-Version beschrieben. Daran sieht man, dass sich dokumentenorientierte Datenbanken stetig weiterentwickeln. Skalierbarkeit, Verfügbarkeit und Performance eines MongoDB-Datenbanksystems lassen sich durch Clustering verbessern. Couchbase und andere vergleichbare Datenbanken können auch ACID kompatibel sein. Allerdings muss was die Sicherheit von dokumenten-orientierten Datenbanken angeht noch einiges getan werden. Insbesondere das Bewusstsein, dass es auch NoSQL-Injections gibt sollte dabei stärker werden.

References

[1] J. Ahmed and M. Ahmed, "A study of big data and classification of nosql databases," in *2020 IEEE International Conference on Technology, Engineering, Management for Societal impact using Marketing, Entrepreneurship and Talent (TEMSMET)*, 2020, pp. 1–8.

[2] Pierre Carbonnelle, "Topdb top database index," Jan 2022. [Online]. Available: https://pypl.github.io/DB.html

[3] MongoDB, Inc, "Mongodb blog post about company name change," Jan 2022. [Online]. Available: https://www.mongodb.com/press/10gen-announces-company-name-change-mongodb-inc

[4] E. B.V, "Was ist elasticsearch," Jan 2022. [Online]. Available: https://www.elastic.co/de/what-is/elasticsearch

[5] Stack Exchange Inc, "Stackoverflow survey," Jan 2022. [Online]. Available: https://insights.stackoverflow.com/survey/2021

[6] A. K. Samanta and N. Chaki, "Performance monitoring of mongodb on varied cluster configuration: An experimental approach," in *2021 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT)*, 2021, pp. 525–530.

[7] R. A. G. Sánchez, D. J. M. Bernal, and H. D. J. Parada, "Security assessment of nosql mongodb, redis and cassandra database managers," in *2021 Congreso Internacional de Innovación y Tendencias en Ingeniería (CONIITI)*, 2021, pp. 1–7.

[8] R. Čerešňák, K. Matiaško, and A. Dudáš, "Improvement of data searching in mongodb with the use of oracle database," in *2021 18th International Multi-Conference on Systems, Signals Devices (SSD)*, 2021, pp. 1388–1393.

[9] M. R. M. Chopade and N. S. Dhavase, "Mongodb, couchbase: Performance comparison for image dataset," in *2017 2nd International Conference for Convergence in Technology (I2CT)*, 2017, pp. 255–258.

[10] H. Vera-Olivera, R. Guo, R. C. Huacarpuma, A. P. B. Da Silva, A. M. Mariano, and M. Holanda, "Data modeling and nosql databases - a systematic mapping review," *ACM Comput. Surv.*, vol. 54, no. 6, jul 2021. [Online]. Available: https://doi.org/10.1145/3457608

[11] "A comparison of nosql database management systems and models." [Online]. Available: https://web.archive.org/web/20190813163612/https://www.digitalocean.com/community/tutorials/a-comparison-of-nosql-database-management-systems-and-models

[12] "Bson specification," Jan 2022. [Online]. Available: https://bsonspec.org/

[13] json-schema org, "Json schema," Jan 2022. [Online]. Available: https://json-schema.org/

[14] The PostgreSQL Global Development Group, "Postge sql doc json types," Jan 2022. [Online]. Available: https://www.postgresql.org/docs/current/datatype-json.html

[15] ——, "Postge sql doc json functions and operators," Jan 2022. [Online]. Available: https://www.postgresql.org/docs/current/functions-json.html

[16] Oracle Corporation, "My sql document store," Jan 2022. [Online]. Available: https://www.mysql.com/de/products/enterprise/document_store.html
[17] ——, "Documents and collections," Jan 2022. [Online]. Available: https://dev.mysql.com/doc/refman/8.0/en/mysql-shell-tutorial-javascript-documents-collections.html
[18] I. MongoDB, "Aggregation pipeline," Jan 2022. [Online]. Available: https://docs.mongodb.com/manual/core/aggregation-pipeline/
[19] MongoDB, Inc, "Mongodb documentation cluster," Jan 2022. [Online]. Available: https://docs.mongodb.com/manual/tutorial/deploy-shard-cluster/
[20] (2022, Jan) Atomic consistent isolated durable. [Online]. Available: http://wiki.c2.com/?AtomicConsistentIsolatedDurable
[21] ACID properties of couchbase: Part 1. [Online]. Available: https://blog.couchbase.com/acid-properties-couchbase-part-1/
[22] Couchbase supports multi-document ACID transactions. [Online]. Available: https://blog.couchbase.com/couchbase-brings-multi-document-acid-transactions-to-json-database/
[23] A. Brust. (2022, Jan) Couchbase 2.0 released; implements JSON document store. [Online]. Available: https://www.zdnet.com/article/couchbase-2-0-released-implements-json-document-store/
[24] "ACID Transactions Basics," Jan 2022. [Online]. Available: https://www.mongodb.com/basics/acid-transactions
[25] Durability | couchbase docs. [Online]. Available: https://docs.couchbase.com/server/current/learn/data/durability.html
[26] R. A. G. Sánchez, D. J. M. Bernal, and H. D. J. Parada, "Security assessment of nosql mongodb, redis and cassandra database managers," in *2021 Congreso Internacional de Innovación y Tendencias en Ingeniería (CONIITI)*, 2021, pp. 1–7, ISSN: 2539-4320.
[27] A. Ron, A. Shulman-Peleg, and E. Bronshtein, "No SQL, no injection? examining NoSQL security," 2015, version: 1. [Online]. Available: http://arxiv.org/abs/1506.04082
[28] L. Okman, N. Gal-Oz, Y. Gonen, E. Gudes, and J. Abramov, "Security issues in NoSQL databases," in *2011IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications*, 2011, pp. 541–547, ISSN: 2324-9013.
[29] A. Zahid, R. Masood, and M. A. Shibli, "Security of sharded NoSQL databases: A comparative analysis," in *2014 Conference on Information Assurance and Cyber Security (CIACS)*, 2014, pp. 1–8.
[30] rain.forest.puppy. (2022, Jan) Phrack magazine. [Online]. Available: http://phrack.org/issues/54/8.html#article
[31] (2022, Jan) $regex MongoDB manual. [Online]. Available: https://docs.mongodb.com/manual/reference/operator/query/regex/
[32] "Postman API Platform | Sign Up for Free." [Online]. Available: https://www.postman.com/
[33] B. Edmunds, *Securing PHP Apps*. Springer, 2016.
[34] (2022, Jan) $where MongoDB manual. [Online]. Available: https://docs.mongodb.com/manual/reference/operator/query/where/
[35] (2022, Jan) A NoSQL injection primer (with MongoDB). [Online]. Available: https://nullsweep.com/a-nosql-injection-primer-with-mongo/