OrientDB

Eine NoSQL-Datenbank

Jonas Schmeck
Universität Siegen
Fakultät IV
jonas.schmeck@student.uni-siegen.de

ABSTRACT

Diese Ausarbeitung beschäftigt sich mit der NoSQL-Datenbank OrientDB. Es werden grundsätzliche Konzepte von NoSQL (=NotOnlySQL) erläutert und deren Anwendung bei der Multi-Model Strategie von OrientDB. Multi-Model bedeutet, dass OrientDB, anders als übliche NoSQL-Datenbanken, mehrere Konzepte umsetzt, um deren Vorteile zu vereinen. Zusätzlich erfolgt eine Abgrenzung zu relationalen Datenbanken, indem diese mit OrientDB in wesentlichen Bereichen verglichen werden.

Keywords

OrientDB; NoSQL; Multi-Model

1. EINLEITUNG

Seit über 40 Jahren ist das Konzept von relationalen Datenbanken ein Standard in der Datenhaltung. Heutzutage treibt vor allem das Internet die Entwicklung neuer Technologien voran und davon bleiben auch Datenbanken nicht verschont. Es geht um immer größere Datenmengen und verteilte Systeme sind ein gängiges Modell. NoSQL-Datenbanken fokussieren sich auf diese beiden Bereiche und setzen neue Maßstäbe. Die folgende Ausarbeitung beschäftigt sich mit der seit 2010 entwickelten Datenbank OrientDB. Dazu werden zunächst grundlegende Konzepte der NoSQL-Bewegung behandelt, bevor auf die Implementierung von OrientDB und speziell dessen Multi-Model Konzept eingegangen wird.

2. RELATIONALE DATENBANKEN

Relationale Datenbanken sind der etablierte Standard in der Welt der Datenverwaltung. Hier werden Daten in Tabellen mit festen Spalten gespeichert. Ein Datensatz (Tupel) ist eine Zeile in einer Tabelle mit festen Attributen, die durch die Spalten vorgegeben werden. Diese Konvention schränkt die Flexibilität der Datenbank stark ein.

Referenzen werden im relationalen Modell mit Fremdschlüsseln umgesetzt. Ein solcher Wert entspricht dem Primär-Schlüssel eines Tupels in einer anderen Tabelle. Ein Primär-Schlüssel identifiziert einen Datensatz für die entsprechende Tabelle eindeutig. Eine einfache Referenz lässt sich noch effizient umsetzen, allerdings kann eine Referenz auch nicht vorhanden sein, das heißt nicht jedes Tupel in einer Tabelle speichert einen Fremdschüssel in der entsprechende Spalte. Noch häufiger allerdings kommt es zu einer mehrstelligen Referenz (n:m). Zur Verwaltung dieses Typs müssen relationale Datenbanken eine zusätzliche Tabelle anlegen, die Fremdschüssel für beide an der Referenz beteiligten Tabellen beinhaltet. Um bei Abfragen die Verbindung zwischen Elementen beider Seiten herstellen zu können, wird der sogenannte "Join" verwendet. Ein Join wird zum Abfragezeitpunkt berechnet, indem Fremd- und Primär-Schlüssel der am Join beteiligten Tabellen in Verbindung gebracht werden. Aufgrund der hohen Rechenleistung und des Speicherverbrauchs sind Join-Operationen der größte Kritikpunkt am relationalen Datenbankmodell. Ein beispielhafter Join wird in der Grafik 1 dargestellt.[16]

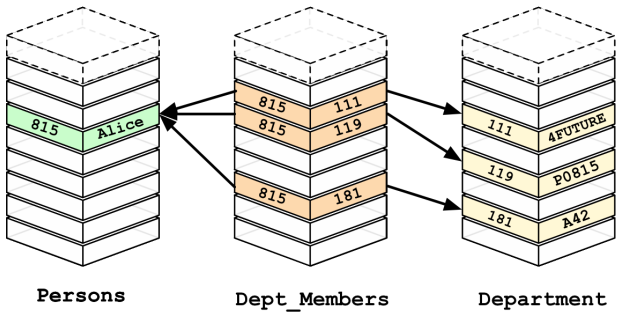

Figure 1: SQL-Join [1]

3. NOSQL-DATENBANKEN

OrientDB ist eine Multi-Model Datenbank. Das bedeutet, dass hier mehrere Konzepte der NoSQL-Bewegung umgesetzt und gemischt werden[9]. Die nächsten zwei Abschnitte behandeln daher die beiden wesentlichen verwendeten Modelle, bevor danach auf Implementierungsdetails von OrientDB eingegangen wird.

3.1 Graphdatenbanken

OrientDB verwendet als Speicherkonzept Graphen, die aus Knoten und Kanten bestehen. Knoten stellen dabei die Hauptspeichereinheit dar, während Kanten die Verbindun-

gen zwischen Knoten modellieren, wobei eingehende und ausgehende Kanten unterschieden werden. Zusätzlich können Knoten Eigenschaften beziehungsweise Attribute zugewiesen werden.

Zur Indizierung von Graphelementen werden Strukturen, wie zum Beispiel B-Bäume, verwendet. Alle Elemente können so über einen eindeutigen Index angesprochen werden[7]. In einer solchen Datenbank ist der Datentyp den ein Knoten repräsentiert irrelevant, was die Flexibilität im Vergleich zum relationalen Modell deutlich steigert. Für Abfragen und Navigation werden lediglich die Indizes und die zugewiesenen Eigenschaften verwendet. Außerdem werden für jeden Knoten die eingehenden und ausgehenden Kanten verwaltet[2].

Graphdatenbanken verfügen über einige spezialisierte Graph-Algorithmen um Abfragen auf diesen Strukturen umzusetzen, wie zum Beispiel Traversierung, Auffinden direkter und indirekter Nachbarn, Berechnen des kürzesten Pfads zwischen zwei Knoten oder Finden und Identifizieren von Graphstrukturen wie Cliquen oder Hotspots. Simple Operationen wie Einfügen, Löschen und Suchen können garantiert in $\mathcal{O}(\log n)$ ausgeführt werden.

In Graphdatenbanken sind Referenzen sogenannte "first-class-citizens", das heißt sie sind ein Grundtyp des Modells und sind daher sehr einfach zu behandeln. Was in einer relationalen Datenbank ein Join übernimmt, wird in Graphen durch spezielle Algorithmen gelöst. Diese werden dabei durch den Aufbau der Knoten im Graph unterstützt. Da jeder Knoten eine Liste seiner Referenzen verwaltet, kann, wie in Grafik 2 zu sehen ist, direkt auf verbundene Knoten zugegriffen werden. Dadurch werden kostenintensive Such- und Vergleich-Algorithmen, wie bei einem Join, vermieden. Zur Verwaltung von stark vernetzten Datenmengen ist daher eine Graphdatenbank sehr gut geeignet.[11]

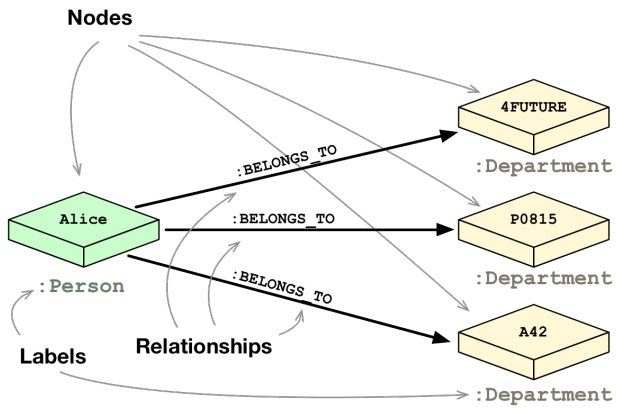

Figure 2: "Join" in Graphdatenbanken [1]

3.2 Dokumentenorientierte Datenbanken

Bei dokumentenorientierten Datenbanken, auch "Document Stores" genannt, werden Daten in Dokumenten gespeichert. Ein Dokument ist dabei eine Menge von sogenannten "key/value"-Paaren (Schlüssel und Wert). Dabei kann über den Schlüssel auf den zugehörigen Wert zugegriffen werden. Ein solches Dokument muss kein festes Schema einhalten und auch das Dateiformat für die Dokumente ist frei wählbar. Am geeignetsten sind natürlich strukturierte Dateien, wie zum Beispiel XML-Dokumente oder JSON-Objekte. Jedes Dokument in einer Datenbank bekommt einen eindeutigen Index, über den es angesprochen und referenziert werden kann. Zur Indizierung wird dabei häufig das Konzept der B-Bäume aufgegriffen. Viele dokumentenorientierte Datenbanken erlauben "Collections" (Sammlungen), um Dokumenten einen Typ zu geben und sie somit zusammenfassen zu können. Eine solche Collection entspricht in etwa einer Tabelle im relationalen Datenbank-Modell.[9]

Zusammenfassend lässt sich ausmachen, dass Document Stores viel Wert auf Flexibilität legen und weniger auf die Verwaltung von vernetzen Datenmengen.

4. ORIENTDB - MULTI-MODEL KONZEPT

OrientDB stellt dem Benutzer zwei verschiedene APIs zur Verfügung um dokumentenorientierte beziehungsweise graphenorientierte Datenbanken umzusetzen. Die Graph-API arbeitet eine Ebene über der Document-API und ist somit die erste Wahl für Einsteiger, weil damit ca. 80% aller Anwendungsfälle abgedeckt werden. Die übrigen 20% erfordern die flexiblere, weniger komplexe und für Nebenläufigkeit optimierte Document-API. Beide APIs arbeiten mit weiteren Konzepten, die auf niedrigeren Ebenen liegen.[6]

4.1 Key/Value

"Key/Value" ist ein simples Schema zur Datenspeicherung. Dabei wird einem Schlüssel ein Wert zugeordnet, was einen gezielten Zugriff ermöglicht. Dieses Schema nutzt OrientDB zum Beispiel bei Dokumenten. Ein Dokument besteht aus einer Menge von key/value-Paaren, die man als Felder bezeichnet. In einem key/value-strukturierten Dokument laufen Zugriffe, wie sie eine Abfragesprache umsetzt, sehr effizient ab.[9]

4.2 Class/Cluster

OrientDB setzt auch einige objektorientierte Konzepte um. Ein sehr wichtiges Konzept ist dabei die Klasse, "Class" in OrientDB, welche es erlaubt Dokumenten einen Typ zu geben. Auf diesem Weg werden gewisse Regeln zur Struktur festgelegt. Ein Beispiel: Die Klasse "Auto" legt fest, dass jedes Dokument von diesem Typ Felder wie Kennzeichen oder Erstzulassung hat. Natürlich kann eine Klasse auch abstrakt sein und somit nur zur Weitervererbung dienen.[8]

Ein weiteres Konzept in OrientDB sind Cluster. Durch Cluster lassen sich Klassen in Gruppen aufteilen, was die verteilte Speicherung von Elementen einer Klasse unterstützt. Standardmäßig legt OrientDB für jede Klasse einen Cluster an und ermöglicht dem Nutzer weitere Gruppeneinteilungen vorzunehmen. Beim Einfügen wird ein Element immer dem Default-Cluster zugewiesen, wenn kein Cluster explizit angegeben wird, während eine Abfrage immer auf allen Clustern arbeitet.[4] Diese Kapselung lässt sich besonders durch die Grafik 3 nachvollziehen.

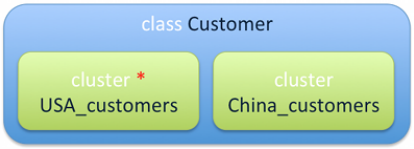

Figure 3: Aufteilung einer Klasse in Cluster [5]

4.3 OrientDB Graph-API

Die Graph-API von OrientDB setzt das Konzept der Graphdatenbanken um. Knoten werden in der Oberklasse V für Vertex zusammengefasst und speichern Eigenschaften sowie ein- und ausgehende Kanten (Referenzen). Ein Vertex ist dabei gleichzeitig ein Dokument mit variablem Schema, was noch einmal deutlich macht, dass die Graph-API eine Ebene über der Document-API arbeitet. Eine Kante aus der Oberklasse E für Edge ist in OrientDB eine bidirektionale Referenz zwischen zwei Vertices. Es wird dabei zwischen "leightweight" und "regular" Edges unterschieden. Eine "regular Edge" wird im Gegensatz zur "leightweight Edge "als Dokument gespeichert. Der Grundtyp in OrientDB ist ein Record, das "kleinste" Element das von der Datenbank verwaltet werden kann[2]. Ein Record ist entweder ein Dokument, eine Edge oder ein Vertex. Beim Einfügen eines Records bekommt dieser eine eindeutige Record-ID, die vom System nicht neu vergeben wird. Eine Record-ID setzt sich aus der Cluster-ID und der Position des Records im Cluster zusammen[10]. Zusätzlich kann OrientDB für Klassen einen Index anlegen. Ein Index ist ein key/value-Paar, wobei der Schlüssel über mehrere Eigenschaften der Klasse gebildet wird und der Wert die Record-ID ist. Es gibt vier Indizierungs-Algorithmen von denen standardmäßig der SB-Baum, ein erweitertes B-Baum Modell, verwendet wird[7].

Table 1: Modell-Vergleich

relationale DB	Graphdatenbank	OrientDB
Tabelle	Knoten/Kanten	Klasse
Zeile	Knoten	Knoten
Spalte	Eigenschaft	Eigenschaft
Referenz	Kante	Kante

4.4 OrientDB Document-API

Während die Graph-API sehr komplex ist, fokussiert sich die Document-API mehr auf Flexibilität und Nebenläufigkeit. Auf dieser Ebene gibt es nur key/value-strukturierte Dokumente als Records, mit eindeutiger Record-ID. Diese Dokumente werden mit Hilfe des Klassen-Prinzips getypt und können in Clustern gruppiert werden. In der Document-API wird eine Referenz als Link bezeichnet. Ein Link ist ein unidirektionaler Verweis auf einen anderen Record über die Record-ID. Links können in einem Dokument einzeln oder als Set, Liste oder Map gespeichert werden, je nach Anwendungsfall. Die Verwaltung von Referenzen für Dokumente unterscheidet OrientDB wesentlich von anderen dokumentenorientierten Datenbanken, wie auch der Tabelle 2 zu entnehmen ist.[11]

Auch für Klassen der Document-API kann ein Index angelegt werden[7].

Table 2: Modell-Vergleich

relationale DB	Document Store	OrientDB
Tabelle	Collection	Klasse
Zeile	Dokument	Dokument
Spalte	key/value-Paar	Dokument Feld
Referenz	-	Link

4.5 Die Abfragesprache von OrientDB

SQL ist ein etablierter Standard für Abfragesprachen. Viele Entwickler sind damit vertraut und haben viel Erfahrung gesammelt. Deshalb verwendet OrientDB SQL als Abfragesprache und erweitert diese, um die Arbeit mit Graphen und Dokumenten zu ermöglichen.[14]

Beim Aufsetzen einer Datenbank mit OrientDB werden anstelle von Tabellen zuerst Klassen erstellt, in die abhängig von der verwendeten API verschiedene Records eingefügt werden können, wie in Grafik 4 zu sehen. Beim Erstellen

Figure 4: Create Operationen

```
CREATE CLASS auto EXTENDS fahrzeug
CREATE CLASS besitzer EXTENDS E
CREATE PROPERTY auto.kennz STRING
CREATE PROPERTY auto.marke STRING
CREATE PROPERTY auto.ps INTEGER
-- insert document @rid #10:1
INSERT INTO auto (kennz, marke, ps)
VALUES ('NO-SQL-11', 'OrientDB', 2015)
-- insert vertex @rid #10:2
CREATE VERTEX auto SET kennz='NO-SQL-12',
marke='OrientDB', ps=15
-- insert edge @rid #12:1
CREATE EDGE besitzer FROM #10:2 TO #11:1
```

eines Records werden Attribute übergeben und es wird automatisch eine eindeutige Record-ID zugewiesen. Diese wird nachher von Links beziehungsweise Edges verwendet um Referenzen zu bilden.

In OrientDB hat ein Record direkten Zugriff auf alle referenzierten Elemente. In der Abfrage wird eine Referenz daher wie ein Attribut verwendet, wie im Beispiel zum "Join" in Grafik 5 zu sehen ist.

Figure 5: Join Operation

```
-- SQL Join
SELECT *
FROM Employee A, City B
WHERE A.city = B.id
AND B.name = 'Rome'
-- OrientDB "Join"
SELECT * FROM Employee
WHERE out('city')[0].name = 'Rome'
```

Die Abfragesprache von OrientDB entspricht bei simplen Abfragen dem SQL-Standard, weicht jedoch bei komplexeren Abfragen aufgrund der Strukturunterschiede zum Tabellenschema von diesem ab. Entwickler müssen sich daher beim Arbeiten mit NoSQL-Datenbanken anpassen. Was sich nach einem kleinen Problem anhört, ist in großen Unternehmen allerdings ein schweres Hindernis. Ein Umstieg würde bedeuten, dass Umschulungen der bisher eingesetzten Entwickler nötig sind, damit Anwendungen, die mit der Datenbank arbeiten, angepasst werden und neue entwickelt werden können. Ein Umstieg ist daher mit hohem Aufwand und Kosten verbunden, was viele Firmen abschreckt.

4.6 Einsatz im verteilten System

Heutzutage spielt es eine große Rolle ob eine Datenbank in verteilten Systemen eingesetzt werden kann. Eine moderne Datenbank sollte dieses Konzept unterstützen. OrientDB bietet dazu einige Strategien an.

4.6.1 Replikation

Zur Replikation wird ein Multi-Master Konzept implementiert, das heißt jeder Server kann Lese- und Schreiboperationen ausführen. Dadurch kann horizontal skaliert werden und es tritt kein "Bottleneck" auf. Die Replikate werden durch den Austausch von Update-Nachrichten konsistent gehalten (local-write Protokoll). Während einer Serverdowntime werden solche Nachrichten in einer Warteliste gespeichert, die der Server abarbeiten muss bevor er online geht.[12]

4.6.2 Sharding

Will man in OrientDB eine Datenmenge auf mehrere Serverknoten aufteilen, kann man dazu Sharding verwenden. Beim Sharding werden Daten auf Klassenebene aufgeteilt. Dazu greift die Datenbank auf das Cluster-Schema zurück. Eine Klasse wie Kunden kann zum Beispiel in Cluster für jedes Land unterteilt werden. So werden Kunden aus verschiedenen Ländern nur dem nächstliegendsten Server zugeteilt und von diesem verwaltet. Sharding erlaubt auch das gleichzeitige Speichern auf mehreren Servern, wobei immer ein Master-Server spezifiziert werden muss. Eine Abfrage auf einer in Cluster unterteilten Klasse wird immer auf allen Servern bzw. Clustern ausgeführt (Map). Die Antworten der einzelnen Server werden danach zusammengeführt (Reduce). Einen solchen Aufruf bezeichnet man als Map-Reduce-Operation.[13]

4.6.3 Transaktionen

OrientDB setzt sogenannte ACID-Transaktionen um. (ACID - Atomarität, Konsistenz, Isolation, Dauerhaftigkeit) Atomarität bedeutet in diesem Fall, dass eine Transaktion entweder komplett fehlerfrei abläuft oder gar nichts geändert wird. Zur Sicherstellung von Konsistenz verwendet OrientDB MVCC (Multi Version Control System). Tritt am Ende einer Transaktion beim "Commit" ein Fehler auf, wird die Transaktion rückgängig gemacht. In OrientDB werden Transaktionen auch voneinander isoliert. Wird innerhalb einer Transaktion eine Variable gelesen, kann deren Wert aus Sicht der aktuellen Transaktion nicht mehr durch andere Transaktionen geändert werden. Zusätzlich wird nur ein "commiteder" Wert gelesen. Dauerhaftigkeit wird gewährleistet indem Transaktionen nur auf Client-Seite ausgeführt werden und erst bei einem "Commit" in die Datenbank geschrieben wird. Dadurch bleibt bei einem Systemausfall der "Commit" aus und der Server kann alle ausstehenden Transaktionen rückgängig machen.[15]

4.7 OrientDB im Vergleich

OrientDB bietet aufgrund seines Multi-Model Ansatzes nahezu alle Konzepte von NoSQL-Datenbanken an. Durch die zwei verschiedenen APIs werden nahezu alle Anwendungsfälle abgedeckt. Der Benutzer hat die Wahl zwischen den Vorteilen der komplexen und mit hoher Funktionalität ausgestatteten Graph-API und der flexiblen und spezialisierten Document-API. Wo genau die Vorzüge von OrientDB liegen, zeigt der folgende Vergleich mit dem relationalen Datenbankmodell und anderen NoSQL-Technologien.

4.7.1 Flexibilität

Hohe Flexibilität wird bei beiden Ansätzen von OrientDB angestrebt. Da die Graph-API eine Ebene höher arbeitet als die Document-API kann sie auf deren Konzepte zugreifen und somit Knoten und Kanten als Dokumente umsetzen. Das macht eine mit OrientDB umgesetzte Graphdatenbank flexibler als Konkurrenzmodelle und fast so flexibel wie dokumentenorientierte Datenbanken oder OrientDB Document Stores. Das Konzept von Klassen und Vererbung macht OrientDB äußerst attraktiv für objektorientierte Anwendungen und bringt im Vergleich mit der Konkurrenz weitere Vorteile ein. Relationale Datenbanken können hier nicht mithalten. Weder Vererbung noch flexible Elemente lassen sich hier so einfach und geschickt umsetzen.

4.7.2 Skalierbarkeit

Skalierbarkeit bleibt ein zentraler Aspekt in Zeiten von Big Data. Bereits in den vorherigen Abschnitten wurde das Problem von relationalen Datenbanken mit stark vernetzten und großen Datenmengen thematisiert. Der Join, den relationale Datenbanken bei Referenzen verwenden um Tabellen miteinander in Verbindung zu bringen, wurde dabei, aufgrund des hohen Aufwands einer solchen Operation, als größter Kritikpunkt genannt. Der Aufwand entsteht bei der Suche und dem Vergleichen der Schlüsselwerte aus beiden Tabellen. In OrientDB werden Referenzen über Edges bzw. Links umgesetzt. Immer wenn eine Operation äquivalent zum SQL-Join ausgeführt wird hat die Datenbank direkten Zugriff auf alle referenzierten Elemente und vermeidet eine aufwendige Suche und Vergleichsoperationen. Hier gewinnt OrientDB auch den Vergleich mit anderen NoSQL-Datenbanken wie MongoDB (Document Store), die keine direkte Unterstützung für Referenzen bieten.

4.7.3 Verteilte Systeme

Im Allgemeinen fällt die Bewertung für NoSQL-Datenbanken im Bereich Nebenläufigkeit und Parallelität sehr gut aus. Die meisten Vertreter dieser Bewegung setzen standardmäßig Konzepte zur Bewältigung von Problemen in verteilten Systemen um. Dazu müssen beim Verwenden einer relationalen Datenbank zusätzliche Anwendungen entwickelt werden, die diese Aufgaben für das Datenbank-Management-System übernehmen. Speziell OrientDB implementiert bereits viele Konzepte, setzt aber noch nicht alles um und kann sich an einigen Stellen, zum Beispiel beim Sharding, noch verbessern.

4.7.4 Abfragesprache

Ein weiterer großer Vorteil von OrientDB ist die an SQL angelehnte Abfragesprache, die vielen Entwicklern den Einstieg beziehungsweise Umstieg von SQL erleichtern soll. Die meisten anderen Vertreter der NoSQL-Bewegung setzen auf eine eigene Abfragesprache. Das hat oft den Vorteil, dass eine solche Sprache optimierter und minimal effizienter ist. Daher ist die SQL ähnliche Abfragesprache vor allem ein Komfort-Faktor von OrientDB, dessen Bedeutung jeder Kunde selbst bewerten muss.[14]

5. FAZIT

Insgesamt zeichnet sich OrientDB durch das Multi-Model Konzept als der vielseitigste Vertreter der Branche aus. Die zwei verschiedenen APIs, key/value-Strukturen, objektorientierte Konzepte, die an SQL angelehnte Abfragesprache und die Eignung für den Einsatz im verteilten System machen

OrientDB unglaublich komplex. Es wird ein großer Bereich an Anwendungsfällen abgedeckt. Diese hohe Komplexität hat zur Folge, dass OrientDB sich auf keinen Bereich wirklich spezialisiert. Wer zum Beispiel nur eine dokumentenorientierte Datenbank umsetzen will und sich nicht für die Graph-API von OrientDB interessiert, findet daher bei der Konkurrenz eventuell ein effizienteres Modell. Dennoch besticht OrientDB mit vielen Komfort-Elementen, die der Konkurrenz fehlen.

6. AUSBLICK

Die Vorteile von NoSQL-Datenbanken sind immens, dennoch dominieren bis heute relationale Datenbanken den Markt. Wo liegt das Problem mit den neuen Technologien? Auf diese Frage gibt es mehrere Antworten. Zunächst einmal sind die etablierten Datenbanken schwierig zu verdrängen, da ein kompletter Umstieg zeit- und kostenintensiv ist. NoSQL-Datenbanken arbeiten daher an einer effizienten Möglichkeit Tabellenschemata in ihre jeweilige Repräsentation umzuwandeln, um den Umstieg für Firmen attraktiver zu machen. Ein weiteres Problem für die Unternehmen ist die teilweise stark vom SQL-Standard abweichende Abfragesprache. Ein Umstieg bedeutet auch, dass Umschulungen der bisher eingesetzten Entwickler nötig sind, damit Anwendungen, die mit der Datenbank arbeiten, angepasst werden und neue entwickelt werden können. Ein Umstieg ist daher mit hohem Aufwand und Kosten verbunden, was viele Firmen abschreckt. Zusätzlich ergeben sich erst bei sehr großen Datenmengen spürbare Unterschiede zwischen relationalen- und NoSQL-Datenbanken. Andere Probleme wie Nebenläufigkeit und Parallelität, die relationale Datenbanken nicht behandeln, werden oft durch externe Anwendungen gelöst. Zusätzlich zu den bereits genannten Problemen wird NoSQL-Datenbanken vorgeworfen, dass sie nicht ausgereift sind. Oft wird vor allem die Stabilität und das zu häufige Auftreten von Bugs kritisiert. Die Vorwürfe sind angesichts der Menge an markierten Problemen und Bugs auf der Github-Seite von OrientDB nachvollziehbar[3].

Zusammenfassend ist besonders der kosten- und zeitintensive Umstieg auf NoSQL-Datenbanken, wie OrientDB, das Hauptproblem für Unternehmen. Was die neuen Technologien brauchen ist vor allem Entwicklungszeit. Es muss an der Stabilität und dem Funktionsumfang gearbeitet werden bevor Unternehmen bereit sind den endgültigen Schritt zu wagen. Außerdem sollte der Umstieg durch Algorithmen zur Datenkonvertierung aus dem relationalen Modell unterstützt werden. Angesichts der stetig wachsenden Datenmengen wird die Zeit für effizientere Modelle als relationale Datenbanken kommen. NoSQL ist eine zukunftsweisende Bewegung, deren Vertreter sich erst noch etablieren müssen.

APPENDIX

A. REFERENCES

[1] Neo4j, graph vs rdbms, Dezember 2015. http://neo4j.com/developer/graph-db-vs-rdbms/.

[2] Orientdb, basic concepts, Dezember 2015. http://orientdb.com/docs/last/Concepts.html.

[3] Orientdb, bugs und probleme, Dezember 2015. https://github.com/orientechnologies/orientdb/issues/.

[4] Orientdb, cluster, Dezember 2015. http://orientdb.com/docs/last/Tutorial-Clusters.html.

[5] Orientdb, cluster bild, Dezember 2015. http://orientdb.com/images/class-clusters.png.

[6] Orientdb, graph or document api, Dezember 2015. http://orientdb.com/docs/last/Choosing-between-Graph-or-Document-API.html.

[7] Orientdb, indizierung, Dezember 2015. http://orientdb.com/docs/last/Indexes.html.

[8] Orientdb, klassen, Dezember 2015. http://orientdb.com/docs/last/Tutorial-Classes.html.

[9] Orientdb, multi-model, Dezember 2015. http://orientdb.com/docs/last/Tutorial-Document-and-graph-model.html.

[10] Orientdb, record-id, Dezember 2015. http://orientdb.com/docs/last/Tutorial-Record-ID.html.

[11] Orientdb, referenzen, Dezember 2015. http://orientdb.com/docs/last/Tutorial-Relationships.html.

[12] Orientdb, replikation, Dezember 2015. http://orientdb.com/docs/last/Replication.html.

[13] Orientdb, sharding, Dezember 2015. http://orientdb.com/docs/last/Distributed-Sharding.html.

[14] Orientdb, sql-dialekt, Dezember 2015. http://orientdb.com/docs/last/SQL.html.

[15] Orientdb, transaktionen, Dezember 2015. http://orientdb.com/docs/last/Transactions.html.

[16] Prof. dr. udo kelter, das relationale datenbankmodell, Februar 2016. http://pi.informatik.uni-siegen.de/kelter/lehre/11w/lm/lm_rdbm_20081110_a5.pdf.