Universität Leipzig

Fakultät für Mathematik und Informatik

Abteilung Datenbanken

Dozent: Prof. Dr. Erhard Rahm

Betreuer: Stefan Endrullis

Problemseminar NoSQL-Datenbanken

Semester: WS 11/12

Charakteristika und Vergleich von SQL- und NoSQL-Datenbanken

Tran Ngoc Ha

Kysirong2005@yahoo.de

Matrikelnummer: 1807352

12.12.2011

Inhaltsverzeichnis

1. **Einleitung**

2. **Einführung NoSQL-Datenbanken**

    2.1. Was sind NoSQL-Datenbank?
    2.2. Warum NoSQL-Datenbank?
    2.3. Definition von NoSQL-Datenbanksystemen

3. **NoSQL-Datenbanksystemen**

    3.1. Key-Value-Stores
    3.2. Document-Stores
    3.3. Wide-Column-Stores
    3.4. Graph-Datenbanken

4. **Vergleich von NoSQL- Familien**

5. **SQL vs. NoSQL**

6. **Zusammenfassung**

7. **Quellen**

1. Einleitung

Für die meisten Informatiker ist NoSQL-Datenbank kein fremder Begriff mehr. Seit 3 Jahren erfreuen sich NoSQL-Datenbanken an wachsender Beliebheit sowie zunehmender Wichtigkeit in der Forschung und Entwicklung. Mit dieser Ausarbeitung soll ein kurzer Überblick über NoSQL- Datenbanken gegeben werden. Dabei werden die Entstehung, Definition von NoSQL-Datenbanken sowie die wichtigsten NoSQL Familien behandelt. Um das Ganze abzuschließen werden wir die NoSQL-Datenbanken untereinander vergleichen und anschließend einen Vergleich zwischen SQL- und NoSQL-Datenbanken vornehmen.

2. Einführung NoSQL-Datenbanken

2.1. Was sind NoSQL-Datenabanken?

Auch wenn der Begriff der NoSQL-Datenbank erst seit 2009 für viele Informatiker geläufig ist, gibt es ihn schon lange. Im Jahr 1998 prägte Carlo Strozzi (IBM) den Begriff „NoSQL“ im Sinne von „no SQL“ (kein SQL) für seine zwar relationale Datenbank, aber ohne SQL-API. Erst seit 2009 wird „NoSQL“ als „Not only SQL“ verstanden und wurde als Sammelbegriff für zum relationalen Modell „alternative“ Datenbankentwicklung verwendet.

2.2. Warum NoSQL-Datenbanken?

Seit Jahren war das relationale Datenbank-Konzept die Lösung für alle Datenbank-Probleme. Das liegt daran, dass es RDBMS seit langem gab und es bisher immer weiterentwickelt bzw. optimiert wurde. Die Arbeit mit einer RDBMS wurde für den Benutzer immer einfacher und deshalb ist RDBMS für viele Datenbank-Probleme immer noch die erste Wahl, z.B. für klassische Geschäftsdatenverarbeitung mit strikten Konsistenzanforderungen.

Mit dem Aufkommen von Web 2.0 im Jahr 2000 haben sich die Anforderungen an Datenbanken drastisch verändert. Neue Anforderungen wurden an Datenbanken gestellt, die eine RDBMS nicht oder nur teilweise erfüllen können. Unter anderem wurde gefordert:

- Verarbeitung von großen bis sehr großen Datenmengen (Terra- und Petadatenbereich)
- relativ einfaches Schema
- geringere Anforderung bezüglich Konsistenz
- gute horizontale Skalierung wichtiger als vertikale Skalierung
- kurze Laufzeit der Anfragen

➔ NoSQL-Datenbanksystemen

Allerdings bleiben für viele Datenbank-Probleme das RDBMS als erste Wahl.

2.3. Definition von NoSQL-Datenbanken

Eine einheitliche Klassifikation von NoSQL-Datenbanken gibt es noch nicht. Eine mögliche Einteilung in Anlehnung an http://nosql-database.org/ wie folgt:

NoSQL-Kernsysteme:

- Key-Value-Stores
- Document-Stores
- Wide-Column-Stores
- Graph-Datenbanken

Nachgelagerte NoSQL-Systeme:

- Objektdatenbanken
- XML-Datenbanken
- Grid-Datenbanken
- und viele weitere nicht-relationale Systeme

Bei der Kategorisierung von NoSQL-Systemen werden die wichtigsten Merkmale der NoSQL-Datenbanksysteme betrachtet. Diese sind:

- kein relationales Datenmodell
- Eignung für verteilte und horizontale Skalierung
- schemafrei oder nur schwache Schemarestriktionen
- einfache Datenreplikation zur Unterstützung der verteilten Architektur
- einfache API
- kein ACID sondern BASE als Konsistenzmodell
- daneben gibt es auch Algorithmen und Protokolle, die allerdings nicht häufig verwendet werden

Diese Merkmale resultieren aus dem Grundkonzept der NoSQL-Datenbanksysteme.

Diese sind:

- Map/Reduce
- CAP-Theorem/Eventually Consistent (BASE)
- Consistent Hashing
- MVCC-Protokoll
- Vector Clocks
- Paxos
- REST

Auf eine nähere Erläuterung dieser Konzepte werden in dieser Ausarbeitung verzichtet.

Database	Status	Created by	Vendor	Type	Consistency/Availat tolerance	Users
HBase	Project	Yahoo	No	Column store	CP	Yahoo, Adobe, Trend Micro, Veoh Networks, Stumpleupon
Cassandra	Project	Facebook	No	Column store	AP	Facebook, Digg, Twitter, Reddit, Rackspace, Cisco, Openx
Hypertable	Project	Zvents	Hypertable Inc	Column store	CP	Zvents, Baidu, Rediff, Inepex, Endgame Systems
BigTable	Internal project	Google	No	Column store	CP	Google
CouchDB	Product	Couchio	Couchio	Document store	AP	BBC, Assay Depot, Engine Yard, Sauce Labs, Flirtbox
CouchDB	Service	Couchio	Cloudant	Document store	AP	Scoopler, Cellecta, Meteor
Riak	Product	Basho	Basho	Document store	AP	Comcast, Mochi, Cedexis, Collecta, TeleSkele
MongoDB	Product	10gen	10gen	Document store	CP	EA, New York Times, Disqus, Github, Sourceforge
Dynomite	Project	Cliff Moon	No	Graph	AP	?????
Neo4J	Product	Neo Technologies	Neo Technologies	Graph	??	Swami, The Swedish Defence Forces, Windh Technologies, Flextoll
InfoGrid	Product	NetMesh	NetMesh	Graph	??	?????
HypergraphDB	Project	Kobrix	Kobrix	Graph	??	?????
Dynamo	Internal project	Amazon	No	Key value store	AP	Amazon
SimpleDB	Service	Amazon	Amazon	Key value store	AP	Alexa, Livemocha, AdaptiveBlue, 37 Signals, Sonian
Redis	Project	Salvatore Sanfilippo	VMware	Key value store	CP	Engine Yard, Github, Craigslist, The Guardian
Tokyo Cabinet	Project	Mikio Hirabayashi	No	Key value store	AP	?????
MemcacheDB	Project	Steve Chu		Key value store	CP	?????
Membase	Product	NorthScale	NorthScale	Key value store	CP	Zynga, Heroku, NHN
Voldemort	Project	Linkedin	No	Key value store	AP	Linkedin

Abbildung 2.3: Übersicht der NoSQL-Systeme

3. NoSQL-Datenbanksysteme

In diesem Gliederungspunkt werden die Kernsysteme der NoSQL-Datenbanken nach Einteilung nach http://nosql-database.org/ erläutert. Dabei werden die wichtigsten Merkmale der jeweiligen Systeme vorgestellt. Schwerpunkt ist hier, Fragen wie:
Wie sind diese Systeme aufgebaut?
Welche Vor- und Nachteile haben diese Systeme?
Wo finden diese Systeme Anwendung?
zu beantworten.

Eine Ausführliche Erläuterung der einzelnen Systeme ist nicht Thema dieser Ausarbeitung.

3.1. Key-Value-Stores

Als eine der ältesten NoSQL-Systeme ist der Konzept der Key-Value-Stores schon seit den 70er Jahren im Einsatz. Heute gehört Key-Value-Stores zu den am schnellsten wachsenden NoSQL-Datenbanksystemen. Aufgrund der immer wachsenden Datenvolumen der Firmen wurden Key-Value-Stores immer häufiger verwendet, da diese Systeme sich sehr gut eignen, um Daten auf vielen verschiedenen Server zu verteilen.

Das Datenmodell

Der Aufbau von Key-Value-Stores ist ziemlich einfach und ähnelt den relationalen DBMS. Ein bestimmter Schlüssel zeigt auf einen Wert (Key/Value), der eine strukturierte oder willkürliche Zeichenkette sein kann. Dabei können die Werte (Value) außer String auch Listen, Sets oder auch Hashes enthalten. Zugriff auf einen Wert kann nur über einem einzigen Schlüssel erfolgen, d.h. hinter einem Schlüssel steckt ein eindeutig identifizierbares Objekt. Man kann die Key-Value-Stores in zwei Gruppen, die In-Momery-Variante und die On-Disk-Variante, unterteilen. Die In-Memory-Variante behält die Daten im Speicher und sorgt für eine hohe Performanz. Deswegen bieten sich In-Memory-Datenbanken als verteilte Cache-Speichersysteme an. Einige Vertreter dieser Gruppen sind: Redis, Oracle Coherence, memcached und Sanssouci. Die On-Disk-Variante speichern Ihre Daten direkt auf der Festplatte und werden hauptsächlich als Datenspeicher genutzt. Wichtige Vertreter dieser Gruppe sind: BigTable, Redis, Chordless, MongoDB und Keyspace. Zusätzlich kann man Ryak, Cassandra, Dynamo, Voldemort, Hibari zu einer Gruppe zusammenfassen, die man als Key-Value-Stores mit eventually consistent- Eigenschaften bezeichnen kann.

Vorteile

Durch das einfache Schema der Key-Value-Stores haben solche Systeme sehr gute Skalierbarkeit sowie schnelle und effiziente Datenverwaltung. Wenn also Firmen mit großen Datenmengen (Terra- bis Petabytebereich) eine Schemaänderung an einem Key-Value-Stores-System durchführt, müssen sie ihren Server nicht für mehrere Stunden lahm legen. Denn bei Key-Value-Stores wird der Ansatz der Datenversionierung verfolgt. Dabei kann z.B. beim Hinzufügen eines neuen Feldes (Schemaänderung) die Anwendung von Anfang an die Daten schreiben, während ein weiterer Prozess die Daten im Hintergrund konvertiert und als neue Version speichert.

Nachteile

Durch das einfache Schema ist bei einem Key-Value-Stores-System die Abfragemöglichkeit eingeschränkt. Einfache Abfragen werden schneller verarbeitet. Dafür ist es aber oft nicht möglich, komplexere Abfragen zu schreiben. Es ist auch besonders nicht geeignet, um komplexere Objekt-Beziehungsmodelle darzustellen, da es sich lediglich um einen Key-Value-Stores handelt und nötige Integritätsbedingungen in die Anwendung ausgelagert werden müssen.

Anwendungen:

Mit großem Erfolg werden Key-Value-Stores für die Shopping-Cart von Amazon (Amazon Dynamo) und für die Posteingangssuche von Facebook (Cassandra) eingesetzt. Weitere Einsatzgebiete für diese Datenspeicher sind Anwendungen, die hochverfügbar sind und gleichzeitig sehr geringe Reaktionszeit aufweisen müssen.

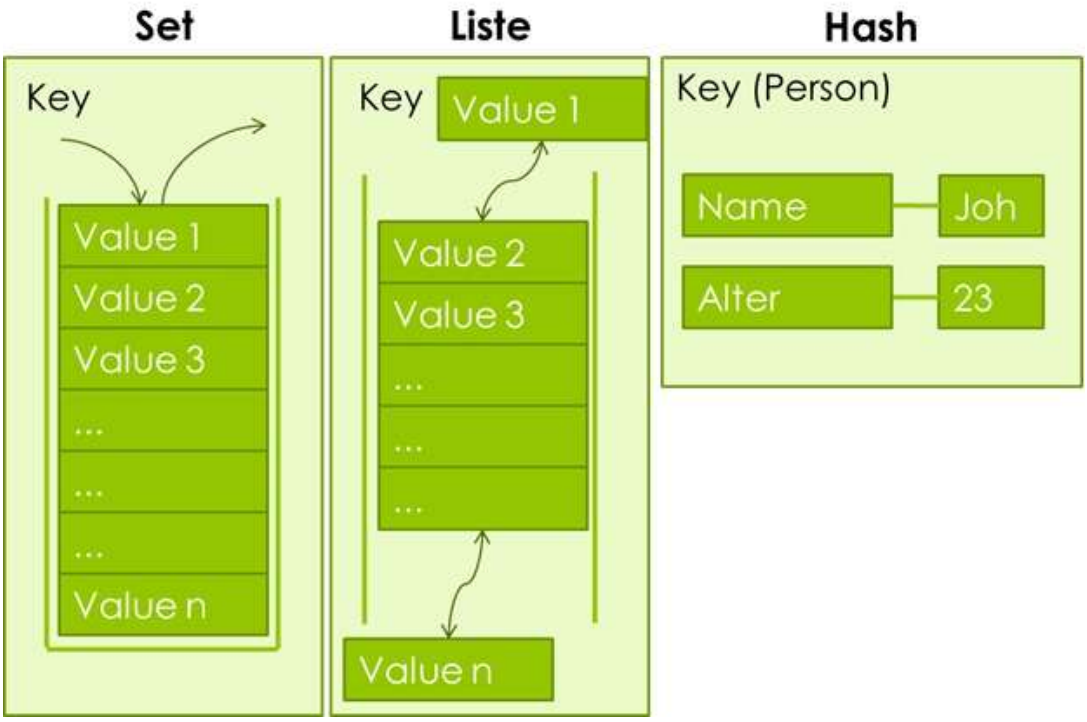

Abbildung 3.1: Value-Typen in Redis

3.2. Document-Stores

Wie Key-Value-Stores gehört Document-Stores zu den Haupt-NoSQL-Datenbanken und erfreut sich über die wachsende Beliebtheit seit dem Aufkommen von Web 2.0.

Das Datenmodell

Nicht wie bei Key-Value-Stores, deren Aufbau dem relationalen DBMS ähnelt, ist der Aufbau von Document-Stores anders als beim relationalen DBMS. Hier werden die Daten nicht in Form von Tabellen gespeichert, sondern in Form von Dokumenten (wie der Name schon sagt). Ein Dokument wird hier nicht im Sinne von einer durch einen Textbearbeitungsprogramm bearbeitete Datei verstanden. Ein Dokument bei Document-Stores ist eine Zusammenstellung von strukturierten Daten und kann sowohl ein gewöhnliches Tupel aus einem relationalen DBMS, eine Tabelle oder auch Objekte

enthalten. Ein Schema, die die ganze Datenbank umfasst, gibt es nicht. Man spricht hier auch von Schemafreiheit. Es ist demnach also nicht vorgeschrieben, welchen Typen ein Dokument annehmen darf. Deswegen kann in jedem einzelnen Dokument eine ganz andere Struktur herrschen. D.h. jedes Dokument ist für sich eine geschlossene Einheit und kann ein beliebiges Schema sowie beliebigen Inhalt haben. In den Dokumenten werden die Daten in Form von Key-Value-Paaren gespeichert. Jedem Schlüssel wird einen Wert zugewiesen. Der Wert kann hier eine beliebige Information sein. Es kann ein String, Text oder auch Liste und Array sein, der wiederum geschachtelte Datentypen enthalten kann. Durch die Schemafreiheit existieren zwischen den Dokumenten keine Relationen. Die Dokumente sind in der Datenbank durch einen Bezeichner gekennzeichnet. Über den Bezeichner kann man auf die Dokumente zugreifen.

Vorteile

Document-Stores speichern zusammenhängende Daten in einem Dokument. So ist es zum Beispiel beim Erstellen von Views nicht notwendig, die Information aus mehreren Tabellen auszulesen und zu kombinieren, da die zusammenhängenden Daten in einem Dokument enthalten sind. Ein Document-Stores bietet also die Möglichkeit, real existierende Dokumente angemessen abzubilden und zu speichern. Deswegen eignen sich Document-Stores sehr gut für die Speicherung der von HTML-Formularen übertragenden Daten. Außerdem verfügen Document-Stores auch über eine hohe Skalierbarkeit.

Nachteile

Durch die Schemafreiheit des Document-Stores ergeben sich die Nachteile. Da die Struktur der Dokumente selbst festgelegt und kontrolliert werden muss, müssen auf viele komfortable Funktionen der relationalen Datenbanksysteme, wie zum Beispiel selbstdefinierte Constraints und Trigger, verzichtet werden. Außerdem müssen in den Anwendungsprogrammen Prüfungsmechanismen für die eingegebenen Werte in den Dokumenten implementiert werden. Auch die Operationen mit den in Dokumenten gespeicherten Daten müssen dementsprechend programmiert werden, da die Datenbank über keine ähnlich mächtige Abfragesprache wie SQL verfügt. Generell ist bei einem Document-Stores wegen Schemafreiheit mehr Aufwand erforderlich.

Anwendungen:

Document-Stores kommen in Bereich Web- Applikation oft zum Einsatz und sind immer beliebter geworden. Wichtige Vertreter der Document-Stores sind Lotus Note von IBM, CouchDb und MongoDB. Ein möglicher zukünftiger Anwendungsbereich ist die Synchronisation von mobilen Endgeräten, wo leicht gewichtige Datenspeicher gefordert sind. Document-Stores eignen sich, um unstrukturierte Daten zu speichern.

Abbildung 3.2: Beispiel eines möglichen Dokuments, dargestellt in JSON-Format

3.3. Wide-Column-Stores

Obwohl die Idee eines Wide-Column-Stores aus dem 80er stammte, existieren erst seit den 90er Jahren spaltenorientierte Datenbanksysteme (sog. Wide-Column-Stores). Hinter Wide-Column-Stores verstecken sich superskalierbare, mit Petabyte von Daten arbeitende Datenbanken. Als Begründer dieses Ansatzes kann Google mit BigTable gesehen werden.

Das Datenmodell

Wide-Column-Stores gruppiert die Daten mehrerer Einträge nach ihren Spalten und nicht nach ihren Zeilen wie bei relationalen Datenbanken. Dabei besteht jeder Eintrag aus den Namen der Spalte, die Daten und einem Zeitstempel (timestamp) zur Aktualitätsprüfung. Zusammenhängende Spalten (Columns) bilden eine so genannte Column-Family, die wie bei relationalen Datenbanken eine Tabelle ist. In einer Column-Family existiert keine logische Struktur und auch keine Einschränkung. So kann eine Column-Family Tausend oder sogar Millionen von Columns enthalten. Deswegen wurden sie auch Wide-Column-Stores genannt. Die Column-Families werden in der Datenbank über Schlüssel identifiziert.

Vorteile

Wide-Column-Stores bringen viele Vorteile mit sich. Sie sind superskalierbar und eignen sich daher sehr gut für Datenbanken mit großem Datenvolumen. Das liegt daran, dass durch die spaltenorientierte Struktur die Daten auf mehreren Servern verteilt werden und somit die Lasten eines einzelnen Servers gering gehalten werden können. Außerdem wird der Leseprozess bei einem Wide-Column-Stores beschleunigt, da keine unnötigen Daten gelesen werden, sondern nur die Daten, die ausgesucht worden sind. Auch der Schreibprozess wird beschleunigt, wenn es nur um eine einzelne Spalte geht.

Nachteile

Der Nachteil eines Wide-Column-Stores ist der Aufwand bei Schreibprozessen, die über mehreren Spalten geht. Zum Beispiel wenn zu einer bestehenden Person das Alter, die Adresse, das Gehalt hinzugefügt werden soll, dann muss man auf mehrere Columns (jeweils die Columns Alter, Adresse und Gehalt) zugreifen. Das verlangsamt den Schreibprozess.

Anwendungen

Wide-Column-Stores sind geeignet für analytische Informationssysteme, wo eine direkte Verarbeitung der Daten notwendig sind, für Data-Mining für Business Reporting für den Vertrieb (Marketing), für Business Process Management-Systeme (Business Process Management) und für Systeme für Finanzberichte und Budget-Projektionen. Durch die gute Skalierbarkeit sind Wide-Column-Stores auch für Firmen mit großen Datenmengen geeignet wie Google, Youtube, Facebook, Digg, Twitter. Die wichtigsten Wide-Column-Systeme sind Cassandra, Hypertable, Hbase, Amazon SimpleDB und natürlich Googles BigTable.

Abbildung 3.3.1: Struktur einer Spalte

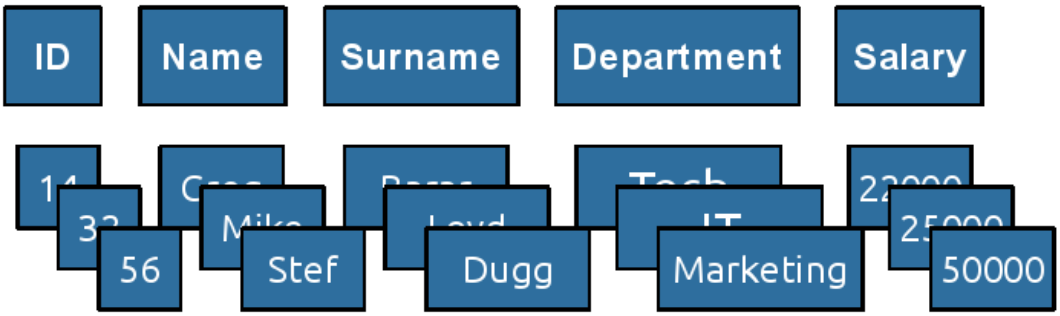

Abbildung 3.3.2: Column Family

3.4. Graph-Datenbanken

Beziehungen zwischen verschiedenen Elementen ist einer der resourcenintensivsten und kompliziertesten Dinge, die man in einer SQL-Datenbank speichern kann wie z.B. Freunde, Follower, wer kennt wen über wem usw. in sozialen Netzen. Bekannt ist, dass man viele Probleme im Bereich Informatik mit Hilfe von Graphen lösen kann. Daher kommt die Idee, eine Datenbank zu entwickeln, die auf Graphentheorie basiert, um somit eine Lösung für einige Datenbankprobleme zu bieten. Es entsteht so genannte Graph-Datenbanken. Graph-Datenbanken bieten eine optimale Speicherung von Graphen. Die Daten werden in Form von Graphen gespeichert und somit lassen sich auch Traversierungen an den Daten ausüben.

Das Datenmodell

Wie auch bei einem Graph bestehen Graph-Datenbanken aus Knoten und Kanten. Die Knoten repräsentieren die Tupel aus der relationalen Datenbank und die Kanten die Beziehungen zwischen den Knoten. In solchen Datenbanken können durch einfache Traversierung teure Datenbankabfragen wie mehrere rekursiv geschachtelte Joins vermieden werden. Somit bieten Graph-Datenbanken eine bessere Performance als relationale DBMS. Die Traversierung erfolgt wie beim Graphen als Breiten- und Tiefensuche, algorithmische Traversierung und randomisierte Traversierung. Wie es auch verschiedene Graphenarten gibt, z.B. gerichtet, ungerichtet, gewichtet, usw. so gibt es auch verschiedene Modelle der Graph-Datenbanken. Hier muss nach Art der Anwendung entschieden werden, welches zugrunde liegende Modell sinnvoll ist. Wenn die Anzahl der Knoten und Kanten für einen Server zu groß wird, muss der Graph partitioniert werden. Die Partition ist eine Art der horizontalen Skalierung und wird auch bei anderen NoSQL-Datenbanken verwendet. Dabei wird versucht, der Gesamtgraph in Teilgraphen aufzuteilen. Dies erwies sich manchmal als schwierig, eine entsprechende Stelle für die Teilung zu finden. Es existiert hierfür auch keine mathematisch exakte Methode, sondern nur ein Paar heuristische Algorithmen wie z.B. Clustering-Algorithmen. Manche Graphen können nicht aufgeteilt werden. Dann müssen manche Knoten in 2 oder mehrere Teilgraphen auftauchen. Das ist die so genannte überlappende Partitionierung. Handelt es sich um eine Graph-Datenbank mit viel Lese- und relativ wenig Schreiblast, bietet sich eine Replikation an, was auch von anderen Datenbankarten verwendet wird. Auch die Replikation ist eine horizontale Skalierung. Dabei wird der Graph vervielfacht und auf

mehreren Servern gespiegelt. So ist die Last eines einzelnen Servers geringer und somit wird eine bessere Performance erreicht.

Vorteile

Der wichtigste Vorteil einer Graph-Datenbank ist die Fähigkeit, vernetzte Informationen zu speichern und zu verarbeiten. Außerdem bieten Graph-Datenbanken einfache und effiziente Traversierung und hat dadurch bessere Performance als relationale Datenbanken. Durch die Struktur können auf komplexen Anfragen wie rekursiv geschachtelte Joins verzichtet werden.

Nachteile

Der große Nachteil einer Graph-Datenbank ist die Abfragesprache. Bisher existiert noch keine einheitliche Abfragesprache. Es gibt viele Abfragesprachen sowie Graphmodelle.

Anwendungen

Der Anwendungsbereich von Graph-Datenbanken beschränkt sich bis jetzt auf Datenbanken, wo die Beziehungen der Daten eine große Rolle spielen.
Graph-Datenbanken finden bei Twitter und Google als große Firmen Anwendung. Der wichtigste Anbieter ist Neo4j.

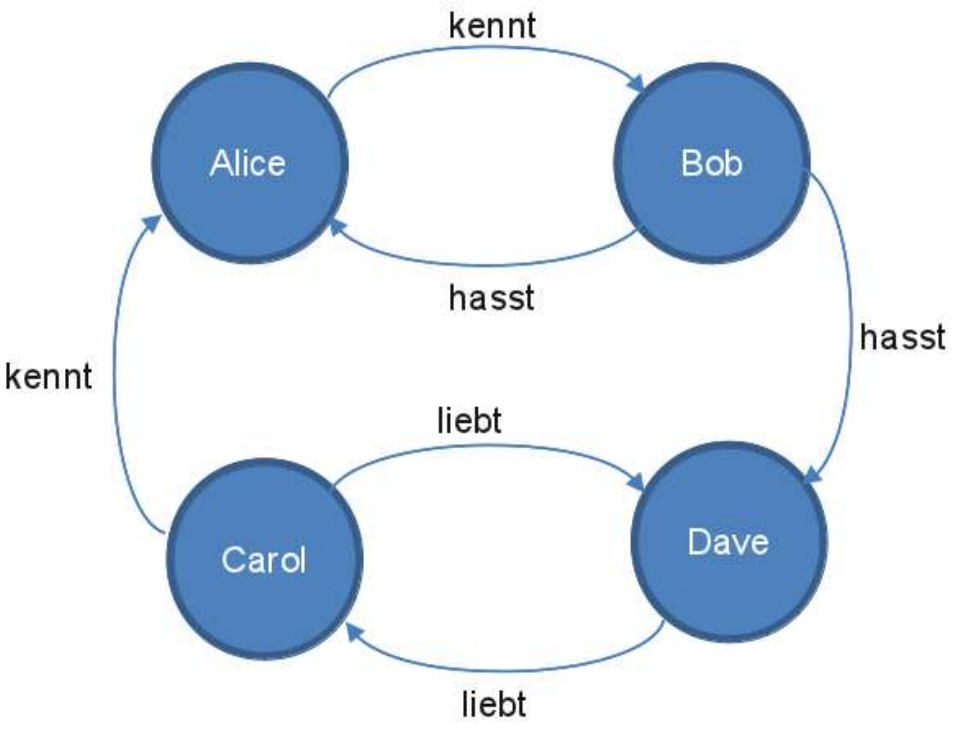

Abbildung 3.4: Beispiel Graph-Datenbank

4. Vergleich von NoSQL-Familien

Gemeinsamkeiten

Die oben beschriebenen NoSQL-Kernsysteme und auch die nachgelagerten NoSQL-Systeme sind NoSQL-Datenbanken. Sie erfüllen die im Punkt 2.3 aufgeführten Eigenschaften und nutzen die im Punkt 2.3 aufgelisteten Konzepte. Manche NoSQL-Systeme können zu mehreren NoSQL-Familien zugeordnet werden. Sie alle wurden entwickelt, um Datenbankprobleme, die mit einem relationalen Datenbanksystem nicht oder nicht ganz gelöst werden können. Die NoSQL-Familien weisen jedoch viele Unterschiede auf.

Unterschiede

Die NoSQL-Familien haben wesentliche Unterschiede im Datenmodell und der daraus resultierende Merkmale, Funktionen sowie Vor- und Nachteile. Deswegen finden die NoSQL-Systeme der unterschiedlichen Familie auch in anderen Verwendungsgebieten ihren Einsatz. Um die Nutzung der NoSQL-Systeme zu optimieren, soll je nach Einsatz das am besten geeignete NoSQL-System ausgewählt werden. Es existieren zwischen den NoSQL-Familien und sogar in den NoSQL-Familien unterschiedliche Abfragemöglichkeiten und API. Es richtet sich oft nach ihrem Anbieter. Zwar werden NoSQL-Systemen immer beliebter, trotzdem gehören sie noch zu dem Minderheiten. Daher ist der Support der NoSQL-Systeme noch dünn und richtet sich nach der Beliebtheit des jeweiligen Systems.

5. SQL vs. NoSQL

Zwischen SQL-Systemen und NoSQL-Systemen gibt es sehr viele Unterschiede in verschiedenen Aspekten. Wir werden in diesem Abschnitt die SQL- und NoSQL-Systeme nach dem Aspekten: Allgemein, Skalierung, Performance, Konsistenz und die Realisierung von Beziehungen vergleichen.

Allgemein

Im Allgemeinen sind SQL-Datenbanken, im Gegensatz zu NoSQL-Datenbanken, keine Minderheit, sondern immer noch die Lösung für die meisten Datenbankprobleme. SQL-Datenbanken gibt es schon sehr lange und fast jeder kennt SQL. Zusätzlich gibt es im Laufe der Entwicklung immer viele Erweiterungen der SQL-Systeme, so dass die Arbeit mit einer SQL-Datenbank immer einfacher wird. Man hat eine große Auswahl an Anbieter mit einer großen Anzahl an Tutorien, Support usw. Doch der wichtigste Unterschied ist, dass SQL-Datenbanken eine einheitliche und sehr mächtige Abfragesprache besitzen, nämlich SQL. Deswegen bleibt SQL-Datenbank immer noch die erste Lösung für alle Datenbankprobleme, die mit einer SQL-Datenbank realisiert werden können.

Skalierung

Ein sehr wichtiger Aspekt, ein Grund warum NoSQL-Datenbanksystemen überhaupt entwickelt wurden, ist die Skalierung, insbesondere die horizontale Skalierung. Es ist hier kein Vergleich von vertikaler und horizontaler Skalierung, denn es existieren auch horizontale Skalierungsstrategien für SQL-Systeme. Aber durch den bei horizontaler Skalierung in SQL-Systemen erforderlichen Verwaltungsmehraufwand führen diese Lösungen ab einen bestimmten Punkt dazu, dass sich die Vorteile von SQL ins Negative kehren und die Performance deutlich sinkt. Dies gilt umso mehr, je größer der Skalierungsbedarf wird. Bei NoSQL ist das genau das Gegenteil. Alle NoSQL-Systeme besitzen eine hohe Skalierbarkeit aufgrund des einfachen Schemas oder es existiert gar kein Schema. NoSQL-Systeme sind so geschaffen, dass ihre gute Skalierbarkeit trotz hohen Datenvolumens konstant bleibt. Damit sind NoSQL-Datenbanken hinsichtlich Skalierung dem SQL-Datenbanken überlegen.

Performance

Mit ähnlichen Gründen wie bei der Skalierung sind die NoSQL-Datenbanken performanter als die SQL-Datenbanken. Eine der Gründe warum NoSQL existiert, ist, dass SQL-Systeme von der Leistung her an ihren Grenzen stoßen. Egal ob bei Lese- oder Schreibleistung ist NoSQL dem SQL im Voraus. Der Unterschied wird deutlicher, wenn das Datenvolumen zunimmt. Das liegt an der Schemarestriktion sowie der Skalierung der beiden Systeme.

Konsistenz

Hier ist es offensichtlich, dass SQL-Datenbanken mit ihren ACID-Eigenschaften eine bessere Konsistenz besitzen als NoSQL-Datenbanken. NoSQL besitzen hinsichtlich der Konsistenz die Eigenschaft „eventually consistency“ oder auf Deutsch „schlussendlich konsistent“. Diese Eigenschaft garantiert nicht, dass nach dem Update immer derselbe Wert zurückgegeben wird. Eine Reihe von Bedingungen müssen erfüllt sein, bis alle denselben Wert bekommen. Es gibt auch NoSQL-Datenbanken, die sowohl „eventually consistency“ als auch ACID-Eigenschaften besitzen. Dabei wird festgelegt, welcher Vorgang sich nach dem ACID-Eigenschaften richtet und welche nach dem „eventually consistency“. Aber trotzdem ist NoSQL hinsichtlich der Konsistenz nicht so stark wie SQL, da sie eine absolute Konsistenz besitzen. Benutzer müssen hier entscheiden, was für ihn wichtiger ist: eine gute Konsistenz oder eine gute Performance.

Realisierung von Beziehungen

Wie schon im Punkt 3.4 erwähnt wurde, sind Beziehungen eine der schwierigsten und resourcenintensivsten Dinge, die man mit Hilfe eine SQL-Datenbank erstellen kann. Natürlich lassen sich Beziehungen wie Assoziationen definieren. Aber das Speichern von vernetzten Informationen oder zusammenhängenden Objekten kann man bei SQL-Datenbanken sehr schwer realisieren, und wenn dann mit einem sehr hohen Aufwand. NoSQL bietet hier die Graph-Datenbanken an, die darauf spezialisiert sind, vernetzte Information zu speichern.

6. Zusammenfassung

Mit dem immer zunehmenden Datenvolumen der Firmen und dem zunehmenden Schwerpunkt auf Leistung sowie der zunehmenden Wichtigkeit, Beziehungen in Datenbank zu definieren, werden die NoSQL-Datenbanken immer beliebter und weitverbreitet. Trotzdem können sie SQL nicht ersetzen und anders herum können sie auch nicht durch SQL ersetzt werden. Je nach Zweck der Verwendung soll zwischen den Systemen gewählt werden, welche die bessere Lösung ist. SQL und NoSQL werden parallel existieren und einander ergänzen.

7. Quellen

1. http://blogs.the451group.com/information_management/2010/03/15/categorizing-the-foo-fighters-making-sense-of-nosql/
2. http://de.wikipedia.org/wiki/Graphdatenbank
3. http://wikis.gm.fh-koeln.de/wiki_db/Category/NoSQL
4. http://www.heise.de/open/artikel/NoSQL-im-Ueberblick-1012483.html

sowie die vorgegebenen Literatur