Einführung in NoSQL-Datenbanken

Matthias-Christian Ott

1 Begriffsbedeutung und -herkunft

Ursprünglich geht der Begriff „NoSQL“ sehr wahrscheinlich auf eine gleichnamige von Carlo Strozzi seit 1998 entwickelte Datenbankverwaltungssoftware zurück [Pat99], die der Autor wie folgt beschreibt[1]:

> NoSQL is a fast, portable, relational database management system without arbitrary limits, (other than memory and processor speed) that runs under, and interacts with, the UNIX 1 Operating System. It uses the Operator-Stream Paradigm described in “Unix Review”, March, 1991, page 24, entitled “A 4GL Language”. There are a number of “operators” that each perform a unique function on the data. The “stream” is supplied by the UNIX Input/Output redirection mechanism. Therefore each operator processes some data and then passes it along to the next operator via the UNIX pipe function. This is very efficient as UNIX pipes are implemented in memory. NoSQL is compliant with the “Relational Model”.

Am 12. Mai 2009 kündigten Johan Oskarsson und Eric Evans ein Zusammentreffen unter dem Namen „NOSQL“, das „open source, distributed, non relational databases“, zum Thema haben sollte und bei dem die Datenbankensoftware Voldemort, Cassandra, Dynomite, HBase und Hypertable vorgestellt wurden. Ihre Definition fand im Umfeld dieser neuartigen und aufstrebenden Datenbanksoftware weite Verbreitung. Stefan Edlich fasst die heutige Bedeutung in einer eigentlich allgemein akzeptiert wie folgt zusammen[2]:

> NoSQL DEFINITION: Next Generation Databases mostly addressing some of the points: being non-relational, distributed, open-source and horizontally scalable. The original intention has been modern web-scale databases. The movement began early 2009 and is growing rapidly. Often more characteristics apply such as: schema-free, easy replication support, simple API, eventually consistent / BASE (not ACID), a huge amount, of data and more. So the misleading term “nosql” (the community now translates it mostly with “not only sql”) should be seen as an alias to something like the definition above.

[1] `http://web.archive.org/web/20080109003124/http://www.scriptaworks.com/cgi-bin/CSAI/tw7/en_US/NoSQL/Home%20Page`

[2] `http://nosql-databases.org/`

Im Gegensatz zu Strozzis Begriffswahl, die sich auf Umsetzung des relationalen Datenmodells gemäß der Unix Philosophie [Gan95] durch Pipes ohne die Anfragesprache SQL, also kein SQL, „NoSQL“, bezieht, ist der Begriff heutzutage also deutlich breiter gefasst: Als wesentliche technische Merkmale werden durch Edlich nicht mehr nur der Verzicht auf die Anfragesprache SQL, die aufgrund der wortreichen der Programmiersprache COBOL ähnelnden Syntax in mitunter einen schlechten Ruf hat, sondern neu ein nichtrelationales Daten- und Verbarbeitsunsmodell, verteilte Datenhaltung und Skalierbarkeit herausgestellt. Im Sinne dieser Definition müsste also eine NoSQL-Datenbank also unkompliziert sehr große Datenbank auf eine hohe Anzahl von Rechnern verteilen und auf diesen effizient verarbeiten können. Ferner soll NoSQL Software die vier Freiheiten Freier Software [Sta10] gewähren, also aus rein ökonomischer Sicht gesehen Anpassbarkeit ermöglichen, eine unabhängige Wartbarkeit sicherstellen und vor allem (gerade in Anbetracht der Rechneranzahl) Lizenzkosten sparen.

2 Geschichtliche Entwicklung und Einordnung

In den 1980er und 1990er Jahren verbilligten sich PCs und Netzwerkkomponenten wesentlich durch technischen Fortschritt, Standardisierung, Produktionsverlagerung und Massenproduktion (siehe Abbildungen 1 und 2). Gleichzeitig gab es in den Bereichen verteilte und parallele Systeme und paralleler Datenverarbeitung sowohl theoretisch als auch praktisch signifikanten Fortschritt[3] und (Quasi-)Standardisierung [Sun90, Wal94]. Als Folge dieser Entwicklung etablierte sich das mit dem Erfolg der Firma Google bekannt gewordene Geschäftsmodell des Commodity Computing [BDH03, HB09]. Anstatt wohl vorausplannend in teure Großrechner zu investieren, werden bedarfsorientiert billige, massengefertigte PCs gekauft und zu Clustern[4] über ein lokales Hochgeschwindigkeitsnetzwerk zusammengeschlossen. Da PC- und Netzwerk-Hardware zu einer austauschbaren Handelsware, einer „Commodity“, geworden waren, lässt sich ein besseres Preisleistungsverhältnis als bei Groß- oder Spezialrechnern erzielen und das Investitionsvolumen richtet sich ohne weit vorausschauende Planung nach dem Wachstum der Anwendung. Durch den Einsatz Freier Software können die im Vergleich zur Hardware nicht unwesentlichen Lizenzkosten proprietärer Software auch eingespart werden.

Durch die billige Vernetzung von PCs, vor allem via Ethernet und Token Ring, ließen sich auch erstmals umfassend automatische Datenflüsse in Unternehmen und Institutionen abseits großer Universitäten und Großunternehmen erreichen, so dass sich zeitgleich auch das Datenaufkommen vor allem die durch Verbilligung von Speicher erhöhte. Es entstanden und enstehen zuvor schwer vorstellbare Datenmengen und Datenbanken, die heute unter dem mehrdeutigen Begriff „Big Data“ zusammengefasst werden. Diese Daten sind teilweise nicht mehr elegant im Sinne relationaler Datenbanken strukturiert und schematisiert und werden daher oft als unstrukturiert bezeichnet.

Mit der Verbreitung von objekt-orientierten Programmiersprachen in den 1990er Jahren

[3] Dies wird an den Erscheinungsdaten einschlägiger Publikationen (siehe z.B. Referenzen in [CGR11]) und den Erstveröffentlichungsdaten relevanter Lehrbücher deutlich.

[4] Damals oft unter dem Namen des gleichnamigen Prototyp „Beowulf Cluster“ bekannt [SSB+95, HWG98].

ergab sich auch das als Object Relational Impedance Mismatch bekannte Problem der Integration dieser Programmiersprachen mit dem relationalen Datenmodell, denn lange Zeit beschrieb die Darstellung von Relationen als Tabelle auch recht gut das Datenmodell relationaler Datenbanken: In Zeilen und Spalten einer Tabelle wurde Daten elementarer Datentypen wie Zahlen, Zeichenketten oder Wahrheitswerte notiert. Spätestens Anfang 1990er Jahre wurde der Gedanke, dass im relationalen Modell nicht ausschließlich elementare Datentypen der Programmiersprache, sondern alle Datentypen der Programmiersprache, bei auch objekt-orientieren Datenbanken also auch Objekte, in Relation stehen, deutlich, dass das Datenmodell relationaler Datenbanken nicht mehr mit dem moderner Programmiersprachen übereinstimmte und dass eine Abbildung zwischen diesen beiden Datenmodellen sich als schwierig gestaltete. Dies ist vor allem bei der Softwareentwicklung hinderlich. Später wurden als Rekation darauf in vielen relationalen Datenbankverwaltungssystemen, benutzerdefinierte Typen, Methoden, Operatoren und Module eingeführt, um diese Dichtonomie zu verringern. Allerdings waren diese neuen Datentypen von der Programmiersprache getrennt und eine enge Integration zwischen Datenbank und Programmiersprache, wie gefordert, blieb weitestgehend aus. Eine Folge davon war die Entwicklung von Objektdatenbanken Ende der 1990er Jahre, die sich aber nie wirklich durchsetzten.

Die Entstehung und geschichtliche Entwicklung von NoSQL-Datenbanksystemen ist daher auf ein verändertes, weitaus höheres Datenaufkommen, eine veränderte Programmiersprachenlandschaft, die rapide Entwicklung der Internetwirtschaft und die sich damit verändernden Dienstleistungs- und Softwarebereitstellungsmodelle zurückzuführen und wurde durch die Verfügbarkeit billiger, massenproduzierter PC-Cluster und passender Software und Algorithmen erst möglich.

3 Anwendungsgebiete

NoSQL-Datenbanken werden heutzutage in Bereichen eingesetzt, in denen große Datenmengen anfallen und verarbeitet werden, hohe Verfügbarkeit und Fehlertoleranz notwendig sind, Schemata klassischer relationaler Datenbanken keine Anwendung finden oder sie als die Softwareentwicklung verlangsamend wahrgenommen werden (Rapid Development), und gelten dabei wartungsarm, in dem Sinne, dass keine Datenbankarchitekten benötigt werden, und lizenzkostensparend.

Neben bekannten Anwendungen bei Suchmaschinen und sozialen Netzwerken werden NoSQL-Datenbanken bei auch bei weiteren Internetdiensten, bei denen ein schnelles Benutzerwachstum zu erwarten ist, große Datenmengen zu verarbeiten sind oder eine hohe Verfügbarkeit zu gewährleisten ist. Der Anwendungsbereich beschränkt sich dabei nicht auf klassische OLTP-Workloads sondern umfasst z.B. auch Web Analytics und verwandte und angeschlossene Data-Mining-Aufgaben oder Auswertung sozialer Netzwerke, wie z.B. markenbezogene Sentiment Analysis oder das Erstellen von Kunden- oder Werbeprofile. NoSQL-Datenbanken werden gerade auch im Bereich des Data- und Text-Mining, z.B. der Vorhersage von Börsenkursen, der Analyse von Logdateien, der Auswertung von Kundenbeschwerden oder der Verbrechensauswertung und -vorhersage, verwendet. In der Wissenschaft werden NoSQL-Datenbanken ferner z.B. zur Speicherung und Auswertung

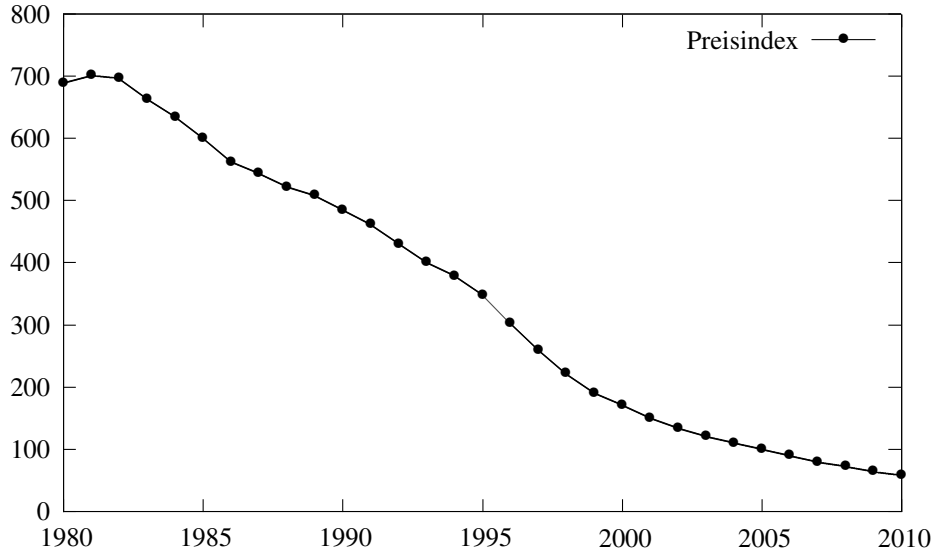

Abbildung 1: Preisindex für „Video, audio, photographic, and information processing equipment and media“ nach [NIP11, Table 2.4.4]

von Messreihen in der Physik und Chemie oder von genetischen Daten in der Bioinformatik genutzt.

4 Ausgewählte Theoretische Grundlagen

Folgende theoretische Grundlagen werden oft im Zusammenhang mit NoSQL-Datenbanken genannt und zur Unterscheidung herangezogen[5].

[5] Synchronisations-, Fehlerdetektions- und Replikationsalgorithmen werden hier ausgelassen, da sich das Thema in aufgrund der verschiedenen Fehler-, Kommunikations- und modelle hier nicht angemessen darstellen lassen (auch wenn bei vielen heutigen NoSQL-Datenbanken nur einfache Modelle, zumeist Fail-Stop und ein durch TCP bestimmtes Kommunikationsmodell, eingesetzt werden, was aber nicht wirklich mehr als was ein TCP-Socket-API bietet, was aber keiner eigentlich Erklärung bedarf), es sei aber exemplarisch auf [CGR11] verwiesen.

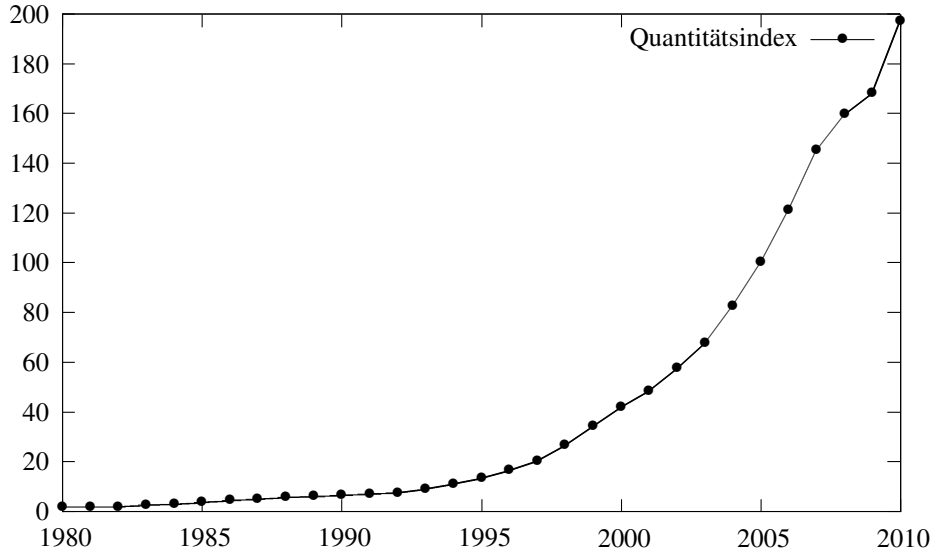

Abbildung 2: Quantitätsindex für „Video, audio, photographic, and information processing equipment and media“ nach [NIP11, Table 2.4.3]

4.1 CAP-Theorem

Das CAP-Theorem ist ein durch Erik Brewer bekanntes und durch Nancy Lynch und Seth Gilbert bewiesenes Theorem[6], dass die Grenzen verteilter Systeme aufzeigt [GL02]. Das Theorem besagt, dass nur zwei der drei Eigenschaften, Konsistenz (Consistency), Verfügbarkeit (Availability) und Partitionstoleranz (Partition Tolerance). Die Eigenschaften sind wie folgt definiert:

Konsistenz Ein verteiltes System ist konsistent, wenn alle (in Bezug auf das Fehlermodell) korrekten Rechner des verteilten Systems zum gleichen Zeitpunkt die gleichen Daten vorhalten.

Verfügbarkeit Ein verteiltes System ist verfügbar, wenn alle korrekten Rechner sets Anfragen bearbeiten können.

Partitionstoleranz Ein verteiltes System ist partitionstolerant, wenn es bei einer Unterteilung des Systems in mehrere Teilsysteme, zwischen denen keine Kommunikation möglich ist, trotzdem korrekt funktioniert.

[6]Das Theorem war zumindest informell auch vorher bekannt.

Ein formaler Beweis soll an dieser Stelle nicht gegeben werden, jedoch ist das Theorem auch intuitiv ersichtlich[7]. Es lassen sich dabei drei Fälle unterscheiden:

1. Angenommen die verteilte Datenbank soll konsistent und verfügbar sein. Dann kann sie nicht partitionstolerant sein, denn kommt es zu einer Partitionierung der Datenbankserver, so können Änderungen nicht an alle Server kommuniziert werden und damit könnte, würden die Änderungen ausgeführt, die Konsistenz nicht mehr gewährleistet werden.

2. Angenommen die verteilte Datenbank soll konsistent und partitionstolerant sein. Dann kann sie nicht konsistent sein, denn in dem vorherigen Szenario würden dann einfach die Änderungen ausgeführt und die Konsistent aufgegeben.

3. Angenommen die verteilte Datenbank soll konsistent und partitionstolerant sein. Dann kann sie nicht stets verfügbar sein. Wird sie bei einer Partition in zwei oder mehr Teile unterteilt und würden in den Teilen Schreiboperationen durchgeführt, so könnte die Konsistenz nicht mehr gewährleistet werden, die Schreibanfrage müsste abgewiesen werden und die verteilte Datenbank wäre (zumindest in Bezug auf Schreiboperationen) nicht mehr verfügbar.

Klassische verteilte relationale Datenbanken sind konsistent und verfügbar.

Im CAP-Theorem werden die Eigenschaften allerdings sehr streng angenommen und durch eine Abschwächung einer oder mehrerer Eigenschaften könnten auch weniger kompromisslose verteilte Datenbanken entwickelt werden. So wäre es z.B. denkbar, dass eine klassische relationale Datenbank einen Teil der Datenbank nicht mehr beschreiben kann, aber Lesezugriff oder zumindest Zugriff auf die restlichen Teile angeboten werden könnten, was in der ein oder anderen Form.

Eine im Bereich von NoSQL populäre Abschwächung der Eigenschaften des CAP-Theorems wird als BASE (Basically Available, Soft state, Eventual consistency; Grundsätzliche Verfügbarkeit, Veränderlicher Zustand, irgendwann konsistent) bezeichnet [Vog09]. Eventual Consistency meint dabei, dass nach einem gewissen Zeitraum, in dem keine Schreiboperationen mehr durchgeführt werden, alle Rechner der verteilten Datenbank die selben Daten haben. Das bedeutet jedoch nicht, dass die Daten dann im Sinne der ACID-Definition konsistent sind. Veränderlicher Zustand bedeutet, dass sich der Zustand des Systems auch nachträglich, z.B. durch Replikation ändern ändern kann.

4.2 Map und Reduce

Bereits in der letzten Hochzeit paralleler Computer war die vor allem automatische Parallelisierung von Programmen ein großes Thema, da angenommen wurde, dass sich rein funktionale Programmiersprachen basierend auf dem Church-Rosser-Theorem des Lambda-Kalküls einfach parallelisieren lassen.

[7] Zur Wiedergabe des in [GL02] gegebenen Beweises wäre noch die Einführung mehrerer Kommunikationsmodelle verteilter Systeme nötig, was allerdings den Umfang und Anspruch dieser Ausarbeitung übersteigt

Die automatische Parallelisierung stellte sich im Nachhinein doch schwieriger heraus als gedacht. Durch Google würde später aber die Nutzung der elementaren Funktionen map und reduce zur Parallelisierung von Datenverarbeitung popularisiert [DG08]. Die beiden Funktionen sind Funktionen höherer Ordnung, die eine Funktion und eine Liste als Argument erhalten. Die Funktion map wendet die Funktion auf jedes Element der Liste an und gibt die resultierende Liste zurück. Die Funktion reduce nimmt jeweils zwei Elemente der Liste wende die Funktion auf die Elemente an und ersetzt die Elemente durch das Ergebnis, bis nur noch ein Element, das dann das Ergebnis ist, übrig ist. Auch wenn diese Funktionen turing-vollständig sind, lassen sich natürlich nicht alle Probleme elegant durch map und reduce lösen.

Listing 1: Einfache Implementierung von map und reduce in Haskell

```
map f [] = []
map f (x:xs) = f x : map f xs

reduce f z []     = z
reduce f z (x:xs) = f x (reduce f z xs)
```

Da sämtliche Funktionsanwendungen in der map Funktion unabhängig sind, lassen sie sich einfachst parallelisieren, indem die Listenelemente auf die einzelnen Rechner verteilt werden und die Ergebnisse dann schließlich wieder zu einer Liste zusammengefügt werden. Die reduce Funktion lässt sich ähnlich parallelisieren, in dem die Liste unterteilt wird, die Teillisten „reduziert“ werden und auf die Liste der Ergebnisse der Anwendung der reduce Funktion wiederum die reduce Funktion angewendet wird.

4.3 Consistent Hashing

Durch Nutzung von Hashfunktionen lassen sich, vorausgesetzt die Hashfunktion liefert gleichverteilte Werte (wie z.B. kryptografischen Hashfunktionen), Daten in $O(1)$ auf Server gleichverteilen. Der zu einem Datum gehörende Server könnte durch $h(x) \pmod n$, wobei h eine Hashfunktion ist und n die Anzahl der Server, bestimmt werden. Allerdings führt bei einer solchen Zuordnung das Hinzufügen oder Entfernen von Servern zu einer großen Umverteilung von Daten, was bei einer linear skalierbaren Datenbankverwaltungssoftware nicht akzeptabel ist.

Consistent Hashing [DHJ+07, KLL+97] löst dieses Problem, in dem den Servern eine (gleichverteilte) Zufallszahl (z.B. durch einen kryptographischen Pseudozufallsgenerator) innerhalb des Wertebereichs zugewiesen wird und jeder Server die Werte, die am nächsten an seiner Zufallszahl liegen, aufnimmt. Bei Einfügen eines Servers müssen daher nur die Daten der Server mit der nächst höheren bzw. nächst kleiner Zahl eventuell umverteilt werden und nicht die Daten der gesamten Server. Durch Mehrfachvergabe einer Zufallszahl kann des Weiteren Ausfallsicherheit erreicht werden und durch Vergabe mehrerer Zufallszahlen an einen Server kann eine Lastanpassung vorgenommen werden, da so die Last nicht mehr gleichverteilt ist.

5 Einteilung

Aufgrund der derzeitigen technologischen Entwicklungen und der Schwierigkeit NoSQL-Datenkbanksoftware nach den Gesichtspunkten Datenmodell, Skalierbarkeit und Verfügbarkeit einzuteilen, existiert keine allgemein akzeptierte und überzeugende Einteilung von NoSQL-Datenbanksoftware [Näs12]. Die in [EFHB10] aufgestellte und durch Stefan Edlich popularisierte ausschließlich auf datenmodellenbasierte Einteilung in Schlüsselwertdatenbanken, Column-Stores, Dokumentdatenbanken und Graphdatenbanken scheint jedoch einen oft herangezogenen kleinsten gemeinsamen Nenner darzustellen:

	Datenmodell	**Bekannte Vertreter**
Schlüsselwertdatenbanken	Schlüsselwertpaare	Amazon DynamoDB, memcached, Redis, Riak, Project Voldemort, Tokyo Tyrant
Column-Stores	Spalten	Google BigTable, Apache Cassandra, Apache HBase
Dokumentendatenbanken	Dokumente (Schlüsselwertmenge und Dokumenteninhalt)	Apache CouchDB, MongoDB, RavenDB
Graphdatenbanken	Graphen, graphähnliche Strukturen	Neo4J, AllegroGraph, HyperGraphDB

Allerdings wird dadurch diese Datenmodell basierte Einteilung auch wesentlich ältere Software, z.B. die weitverbreitete 1979 entwickelte DBM-Schnittstelle [Sel07], die heutzutage z.B. von GNU dbm oder Berkley DB implementiert wird, (hierachische) Dokumentenverwaltungs- und Dateisysteme, die Speicherung und Indexierung von Metadaten erlauben, wie z.B. das Be File System [Gia99], oder das wissenschaftliche Datenerfassungsformat HDF [HDF02], als NoSQL Datenbanken eingeteilt. Ferner weist Edlich auch daraufhin, dass gemäß der Einteilung nach Datenmodellen, auch Objekt-, XML- und Mehrwertedatenbanken als NoSQL-Datenbanken gelten könnten, die er als „Soft NoSQL Systems“ bezeichnet.

Auch wenn eigentlich nicht sinnvoll ist, NoSQL-Datenbanken allein anhand des Datenmodells einzuteilen, und fraglich ist, ob eine eindimensionale Einordnung an sich überhaupt möglich ist[8], wird folgend die vorherig aufgeführte Einteilung aufgrund ihrer Verbreitung weiter ausgeführt.

[8] Vielmehr sollte eine Datenbankverwaltungssoftware nach weiteren Kriterien, wie Form und Art der verteilten Datenhaltung- und -verarbeitung, Fehlertoleranz und Sicherheit, bewertet werden, da praktisch nicht die Wahl eines Datenbanktyps, sondern die Übereinstimmung der Eigenschaften der Datenbank mit den Anforderungen, erforderlich ist.

5.1 Schlüsselwertdatenbanken

Schlüsselwertdatenbanken sind wesentlich ein (zumeist persistentes) assoziatives Datenfeld, das einem beliebigen Schlüssel einen beliebigen Wert zuordnet, wie auch in vielen Programmiersprachen vorhanden. Die zugeordneten Werte können dabei aus Sicht der Datenbankverwaltungssoftware unstrukturierte Folgen von Bytes oder strukturierte Daten, wie z.B. elementare Datentypen (Zahlen, Symbole, Zeichenketten usw.), Tupel, Listen oder Datenfelder sein.

Da das Datenmodell nur einen Zugriff von Schlüsseln auf Werte erlaubt, existiert im Allgemeinen auch keine server-seitig verarbeitete Anfragesprache, so dass komplexere Anfragen entweder von einer server-seitigen Middleware oder client-seitig, dann aber mit einer aus den mehrfachen Anfragen resultierenden Latenz, aus mehreren Anfragen erstellt werden, was aber ohne durch die Schnittstelle der Datenbanksoftware abstrahierte genaue Informationen über die Speicherstruktur zu einer ineffizienteren Anfrageoptimierung und -plan führen kann. Ferner erlischt so bei komplexen, zusammengesetzten Anfragen auch der Geschwindigkeitsvorteil gegenüber einer auf komplexe Anfrage ausgelegten Software.

Listing 2: Project Voldemort Client API

```
public interface StoreClient<K,V> {
    public boolean applyUpdate(UpdateAction<K,V> action);
    public boolean applyUpdate(UpdateAction<K,V> action,
                               int maxTries);
    public boolean delete(K key);
    public boolean delete(K key, Version version);
    public Versioned<V> get(K key);
    public Versioned<V> get(K key, Object transforms);
    public Versioned<V> get(K key, Versioned<V> defaultValue);
    public Map<K,Versioned<V>> getAll(Iterable<K> keys);
    public Map<K,Versioned<V>> getAll(Iterable<K> keys,
                                      Map<K,Object> transforms);
    public List<Node> getResponsibleNodes(K key);
    public V getValue(K key);
    public V getValue(K key, V defaultValue);
    public Version put(K key, V value);
    public Version put(K key, Versioned<V> versioned);
    public Version put(K key, V value, Object transforms);
    public boolean putIfNotObsolete(K key,
                                    Versioned<V> versioned);
}
```

Allgemein lassen sich durch das recht einfache Datenmodell und vor allem ohne sofortige Persistenz eine hohe Verarbeitungsgeschwindigkeiten erreichen, sofern viele einfache Anfragen gestellt werden.

Mit unter wird solche Software daher auch als (zumeist im Hauptspeicher liegender) Zwi-

schenspeicher für klassische relationale Datenbanken verwendet, wobei oft keine Konsistenz des Zwischenspeichers vorliegt, d.h. zwischengespeicherte Daten können verschieden zu den in der eigentlichen Datenbank gespeicherten Daten sein. Auf diese Weise können die zwischengespeicherten Daten auf den dem Datenbankserver vorgeschalteten Anwendungsservern vorgehalten werden und müssen nicht erst vom Datenbankserver, der bei klassischen relationalen Datenbanken natürlich auch einen, allerdings konsistenten, Zwischenspeicher besitzt, übertragen werden. Durch den Verlust der Konsistenz enstehen allerdings auch Situationen, in denen dem Benutzer, zumindest für eine kurze Zeit, ein veraltete oder nicht mehr existierende Daten angezeigt werden, wie man es heutzutage bei einigen größeren Webseiten beobachten kann. Oft werden solche Situationen jedoch als Alternative zur Nichtverfügbarkeit des Dienstes durch Überlast toleriert.

5.2 Column-Stores

Column-Stores sind tabellenähnliche Datenbanken die anstatt einer festen, in jeder Zeile gleichen Anzahl von Spalten eine von Zeile zu Zeile verschiedene Anzahl von Spalten erlauben und eine Versionierung der Zeilen erlauben. In dem von Google BigTable [CDG+06] popularisierten von Column-Store-Modell sind die Zeilen dabei ähnlich zu Schlüsselwertdatenbankn über einen eindeutigen Schlüssel auffindbar und Daten innerhalb der Zeile können über Spaltenname und (eventuell) Zeitstempel ausgelesen werden, so dass ein Datum als Tripel aus Zeilenschlüssel, Spaltenname und Zeitstempel eindeutig identifiziert werden kann. Ferner können Spalten noch weiter in Spaltenfamilien unterteilt werden und werden dann durch Spaltenname und Spaltenfamilie benannt.

5.3 Dokumentdatenbanken

Dokumentdatenbanken sind Mengen von (im Allgemeinen nicht schematisierten) Dokumenten, die aus Schlüsselwertpaaren, deren Wertkomponente wiederum rekursiv verschachtelt aus Schlüsselwertpaaren bestehen kann, und einem (je nach Sofware) etwaigen Dokumenteninhalt bestehen. Innerhalb einer Dokumentendatenbank lassen sich Dokumente zur besseren Unterscheidung abhängig von der Datenbankverwaltungssoftware in Kategorien oder Verzeichnishierachien einsortieren. Mittels eindeutiger Kennungen können Dokumente aufgerufen werden und andere Dokumente referenzieren, wobei aber in der Regel keine referenzielle Integrität gewährleistet wird, da Dokumente als eigenständig betrachtet werden und die Referenzierung konventionell erfolgt.

Listing 3: Vereinfachtes C# Modell eines Angestellten mit RavenDB

```
public class Employee
{
    public string Name { get; set; }
    public string[] Specialities { get; set; }
```

```
    public DateTime HiredAt { get; set; }
    public double HourlyRate { get; set; }
}
```

Listing 4: JSON Repräsentation des Modell

```
{
  "Name": "John Doe",
  "Specialities": ["C#", "RavenDB"],
  "HiredAt": "1970-01-01T00:00:00.000000-00:00",
  "HourlyRate": "100.0"
}
```

5.4 Graphdatenbanken

Graphdatenbanken speichern Graphen oder graphähnliche mathematische Strukturen. Je nach Datenbank können dabei Knoten und Kanten mit aus Schlüsselwertpaaren bestehenden Eigenschaften versehen werden (Property-Graph-Model), Kanten gerichtet oder ungerichtet sein oder Kanten mehrere Knoten verbinden (Hypergraphen). Die Suche in Graphdatenbanken kann entweder deklarativ, d.h. die konkrete Traversierungsabfolge wird anhand einer deklarativen Abfragespache oder eines Beispiel Untergraphen (Query-By-Example) durch die Datenbankverwaltungssoftware bestimmt, oder imperativ (aus Sicht der Datenbankverwaltungssoftware) durch Ausführung eines benutzerbestimmen Travesionsalgorithms- oder -programms erfolgen, so dass Graphdatenbankenverwaltungssoftware die direkte Ausführung graphtheoretischer Algorithmen erlauben und teilweise oft verwendete Algorithmen bereits eingebaut sind. Durch Interpretation von Knoten als Objekte, Kanten als Beziehungen zwischen Objekten und Angabe von Eigenschaften für Objekte und Kanten haben Graphdatenbanken Ähnlichkeit zu objektrelationalen Datenbanken.

Graphdatenbanken sind zur Speicherung komplexer und vernetzter Daten und Informationen, die zwar (sofern sinnvoll schematisierbar) auch in klassischen relationalen Datenbanken gespeichert werden hätten könnten, aber durch (physische) Speicherung als Graphen und vor allem durch Nutzung benutzerbestimmenter Traversierungs- und Graphenalgorithmen wesentlich schneller verarbeitet werden können und entsprechende Anwendungen, wie z.B. die Speicherung und Analyse großer Wissenbasen oder (sozialer) Netzwerke, überhaupt erst effizient möglich werden.

6 Zusammenfassung und Ausblick

In vielen Anwendungsbreichen bietet NoSQL-Datenbankverwaltungssoftware eine interessante Alternative zu klassischen relationalen Datenbanken auch jenseits von OLTP, wenn man mit den Trade-Offs leben kann, und machen eine Anwendungen mit sehr großen Da-

tenbanken durch Commodity Computing mitunter erst möglich.

Durch erst mit proprietären Technologien von Firmen wie Google oder Amazon und später durch Freie Software Projekte allgemein verfügbare gemachte Technologien, die eine lineare Skalierung bestimmter Anwendungen erlauben, hat NoSQL neue Maßstäbe in Bezug auf Rechneranzahl und potentielle Rechenkapazität gesetzt, auch wenn sicher eine große Anzahl Anwender nicht in Dimensionen wie besagte Unternehmen denken und planen.

Durch den Einsatz Freier Software konnten im Vergleich zu den etablierten Herstellern klassicher (proprietärer) relationaler Datenbanken kleinere Firmen den Markt für Datenbank- und Datenanalysesoftware in Teilen, v.a. bei neuartigen Anwendungen, aufschütteln und beginnen derzeit eigene Produkte im NoSQL-Umfeld, die sich in ihr existierendes Produkt-Portofolio integrieren, anzubieten, so dass diese Technologien auch in konservativen und weniger experimentierfreudigen Unternehmen in der ein oder anderen Form Einzug halten werden und eine Entwicklung hin zum „Plateau of Productivity“ gemäß dem Gartner Hype Cycle Model[9] absehbar ist, währenddessen sich aber auch zeigen wird, was genau aus dem NoSQL-Spektrum sich bewähren wird und sicher auch die ein oder andere Software des derzeitig großen Angebots an Datenbanken nicht weiterentwickelt werden wird oder in die Bedeutungslosigkeit abdriften wird.

Im derzeitigen Rausch der Geschwindigkeitssteigerung sollte allerdings nicht übersehen werden, die Abweichung von Datenmodell und ACID-Eigenschaften klassischer relationaler Datenbanken auch neben einem Umdenken auch Chancen für innovative Anwendungen und der Steigerung der Sicherheit und Fehlertoleranz, die in einer sich immer schneller ändernden Welt, in der diese Eigenschaften bisher oft unter dem Gesichtspunkt, dass die benötigte Software überhaupt erstmal entwickelt werden und bestehen muss, ignoriert wurden und man sich auch verstärkt die „Nachhaltigkeit“ (im übertragenen Sinne) von Software kümmern muss. In der Welt von morgen werden Anwendungen nicht mehr in einzelnen Programmiersprachen programmiert sein, nicht mehr auf einzelnen Computer laufen und nicht mehr einen einzelnen Datenbestand haben, sondern verteilt auf mehrere Computer mit leider verschiedenartigen (teilweise) unstrukturierten oder teilstrukturierten Daten arbeiten und mit Asynchronität und Inkonsistenzen klar kommen müssen. In der Welt von morgen werden Computer und Daten zur Organisation von Gesellschaften mit der fortschreitenden Digitalisierung eine immer wichtigere Rolle einnehmen und die Anforderungen an Verlässlichkeit, Verfügbarkeit und Sicherheit von Computern immer höher werden, denn im Gegensatz zu bisherigen physischen Realität, bei der Fehler lokal begrenzt auftreten und manuell ausgenutzt werden müssen, haben in Fehler in digitalen Welten im Sinne der Softwaregleichheit und -verbreitung globale Folgen (Class Breaks) und lassen sich automatisch ausnutzen, wie Angriffe oder Ausfälle im häufiger eingesetzter zentraler Datenbanken zeigen, und in einer Zeit des digitalen Umbruchs ist noch nicht

[9] `http://www.gartner.com/technology/research/methodologies/hype-cycle.jsp`

klar, was das genau bedeuten kann[10,11].

Literatur

[BDH03] Luiz André Barroso, Jeffrey Dean und Urs Hölzle. Web Search for a Planet: The Google Cluster Architecture. *IEEE Micro*, 23:22–28, March 2003.

[CDG+06] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes und Robert E. Gruber. Bigtable: a distributed storage system for structured data. In *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation - Volume 7*, OSDI '06, Seiten 15–15, Berkeley, CA, USA, 2006. USENIX Association.

[CGR11] Christian Cachin, Rachid Guerraoui und Luís Rodrigues. *Introduction to Reliable and Secure Distributed Programming*. Springer, 2. Auflage, 2011.

[DG08] Jeffrey Dean und Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, Januar 2008.

[DHJ+07] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall und Werner Vogels. Dynamo: amazon's highly available key-value store. *SIGOPS Oper. Syst. Rev.*, 41(6):205–220, Oktober 2007.

[EFHB10] Stefan Edlich, Achim Friedland, Jens Hampe und Benjamin Brauer. *NoSQL: Einstieg in die Welt nichtrelationaler Web 2.0 Datenbanken*. Hanser Fachbuchverlag, Oktober 2010.

[Gan95] Mike Gancarz. *The UNIX philosophy*. Digital Press, Newton, MA, USA, 1995.

[Gia99] Dominic Giampaolo. *Practical File System Design with the Be File System*. Morgan Kaufmann Publishers, San Francisco, CA, USA, 1999.

[GL02] Seth Gilbert und Nancy Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33(2):51–59, Juni 2002.

[HB09] Urs Hölzle und Luiz André Barroso. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan and Claypool Publishers, 2009.

[HDF02] The HDF Group, Champaign, IL, USA. *HDF5 Table Specification*, 1. Auflage, Mai 2002.

[HWG98] Jim Hill, Michael Warren und Patrick Goda. I'm Not Going to Pay a Lot for This Supercomputer! *Linux Journal*, 1998(45), Januar 1998.

[10] In diesem Zusammenhang sei auch auf das Askemos Projekt (`http://askemos.org`) verwiesen, dessen Begründer und Hauptentwickler Jörg F. Wittenberger sich schon vor über zehn Jahren, weit vor NoSQL Gedanken zur Verteilung und Sicherung Rechenprozessen machte, und mit Askemos eine verteilte virtuelle Maschine, die gegenüber byzantinischen Fehlern tolerant ist, konzipierte und entwickelte und somit einen Teil dieses Gedankens bereits umsetzte.

[11] Allgemein wurden in dieser Richtung auch schon wichtige theoretische Resultate erzielt und im Rahmen des letzten Hypes verteilter System in den 1980er und 1990er Jahren als durchaus einsetzbare Software, wie z.B. verteilte Authentifikations-, Datei- und Objektsysteme, großenteils im wissenschaftlichem Umfeld implementiert, haben aber nur ein einigen Bereichen Verbreitung gefunden.

[KLL+97] David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine und Daniel Lewin. Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the World Wide Web. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, STOC '97, Seiten 654–663, New York, NY, USA, 1997. ACM.

[Näs12] Petter Näsholm. Extracting Data from NoSQL Databases – A Step towards Interactive Visual Analysis of NoSQL Data. Masterarbeit, Chalmers University of Technology, Göteborg, 2012.

[NIP11] National Income and Product Accounts Tables. U.S. Dept. of Commerce, Bureau of Economic Analysis, Washington, DC, USA, August 2011.

[Pat99] Giuseppe Paterno. NoSQL Tutorial: A comprehensive look at the NoSQL database. *Linux Journal*, 1999(67), November 1999.

[Sel07] Margo I. Seltzer. Berkeley DB: A Retrospective. *IEEE Data Engineering Bulletin*, 30(3):21–28, 2007.

[SSB+95] Thomas L. Sterling, Daniel Savarese, Donald J. Becker, John E. Dorband, Udaya A. Ranawake und Charles V. Packer. BEOWULF: A Parallel Workstation for Scientific Computation. In Prithviraj Banerjee, Hrsg., *International Conference on Parallel Processing*, Jgg. 1, Seiten 11–14. CRC Press, 1995.

[Sta10] Richard M. Stallman. The Free Software Definition. In *Free software, Free society: Selected Essays of Richard Stallman*, Seiten 3–6. Free Software Foundation, Boston, MA, 2. Auflage, 2010.

[Sun90] Vaidy S. Sunderam. PVM: A framework for parallel distributed computing. *Concurrency: Practice and Experience*, 2(4):315–339, 1990.

[Vog09] Werner Vogels. Eventually consistent. *Commun. ACM*, 52(1):40–44, 2009.

[Wal94] David W Walker. The design of a standard message passing interface for distributed memory concurrent computers. *Parallel Computing*, 20(4):657 – 673, 1994.