Allgemeine Untersuchung der NoSQL-Bewegung

Daniel Niecke

I. Einleitung

DIESE Arbeit geht auf grundlegende Unterschiede zwischen SQL- und NoSQL-Datenbanksysteme (NoSQL-DBMS) ein. Bevor die einzelnen Merkmale und möglichen Einsatzgebiete dargestellt werden, sollen SQL und NoSQL kurz definiert werden und es soll ein kleiner Einblick gegeben werden, worin die Motivation besteht NoSQL-Systeme zu entwickeln. Die in dieser Arbeit verwendeten Definitionen vor allem für NoSQL sollten nicht als allgemeingültig gesehen werden, da es keinen Konsens darüber gibt, welche Systeme zum NoSQL-Bereich gehören und welche nicht.
Zum Schluss wird der Nachholbedarf von NoSQL-DBMS gegenüber SQL-DBMS dargestellt und eine abschließende Zusammenfassung geben.

A. *S*tructured*Q*uery*L*anguage

Mitte der 1970er Jahre wurde SEQUEL (Structured English Query Language) [1] entwickelt, welches der Vorläufer des mitte der 1980er Jahre standardisierten SQL (Structured Query Language) [2] war.
Genau wie SEQUEL ist auch SQL eine Abfragesprache für das relationale Modell [3] und setzt somit die relationale Algebra in Anlehnung an die englische Sprache um. Das relationale Modell ist besten dazu geeignet, strukturierte Daten darzustellen und mit der relationalen Algebra komplexe Abfragen zu entwerfen. Deshalb werden SQL-Systeme fast ausschließlich dazu genutzt, um mit strukturierten Daten, welche vor allem in Unternehmen, zum Beispiel in der Buchführung vorkommen, zu arbeiten.
Eine Haupteigenschaft von SQL ist das sogenannte ACID-Prinzip [4] (atomicity, consistency, isolation, duration). Dieses Prinzip gibt maßgeblich das Verhalten von SQL-DBMS vor und wird im folgenden kurz erklärt:

1) Atomarität (atomicity): Das System bietet Sequenzen von Operationen an, die entweder vollständig oder überhaupt nicht ausgeführt werden. Eine solche Sequenz, die entweder eine Basisoperation des Systems oder eine vom Nutzer definierte Transaktion aus mehreren Basisoperationen ist, kann nicht von einer anderen Sequenz unterbrochen werden.

2) Konsistenzerhaltung (consistency): Herrschte vor einer Transaktion ein konsistenter Zustand bzgl. Primärschlüsseln und ähnlichem in der Datenbank, dann muss auch nach dieser Transaktion ein konsistenter Zustand herrschen, ansonsten wird die Transaktion rückgängig gemacht. Während einer atomaren Operation darf folglich ein inkonsistenter Zustand herrschen, am Ende der atomaren Operation hingegen nicht mehr.

3) Isolation (isolation): Sämtliche Operationen müssen klar voneinander getrennt ablaufen. Wenn eine Operation auf Daten in der Datenbank arbeiten möchte, sperrt sie die Daten für die Zeit der Bearbeitung, damit andere Operationen nicht zeitgleich auf den gleichen Daten arbeiten und es wohl möglich zu Lost-Updates kommt.

4) Dauerhaftigkeit (duration): Nach dem Abschluss einer Operation muss die an den Daten vorgenommene Änderung dauerhaft sein. Eine nachfolgende Operation kann die Daten erneut ändern, aber das System darf nicht eigenständig auf einen alten Stand zurückspringen.
Diese Sicherheiten sind für viele Anwendungsbereiche, vor allem in Unternehmen, von maßgebender Relevanz und ein Grund für den großen Erfolg von SQL. Bis heute sind SQL-Systeme die Marktführer im Datenbankbereich. Große DBMS sind zum Beispiel DB2 von IBM [5], Oracle vom gleichnamigen Unternehmen [6], MS SQL Server von Microsoft [7] und Postgres [8] aus dem Open-Source-Bereich.

B. Motivation für NoSQL

Die anfangs in Datenbanken abgelegten Daten stammten von Unternehmen und wurden auch von diesen wieder verarbeitet. Für diese Aufgaben eigneten sich SQL-System hervorragend. Im Zuge des Web 2.0 wurden die dynamischen Inhalte auf Internetseiten, welche auch durch die Nutzer der Internetseiten erstellt werden konnten, zunehmend wichtiger und die Datenmengen stiegen extrem an. Ein neuer Trend, bekannt unter dem Synonym Big Data, hat das Datenwachstum erneut stark beschleunigt. Der wahrscheinlich jüngste Trend, der die Datenmengen vergrößert ist das Internet der Ding, wobei Embedded Computer in immer mehr Gegenständen des alltäglichen Lebens verbaut werden und untereinander Daten austauschen. Diese neuen Datenmengen haben drei Haupteigenschaften [9]

- **Datenvolumen**
  Das Datenvolumen ist deutlich höher als bei bisherigen Datenbanken, zum Beispiel in der Verwaltung.
- **Datengeschwindigkeit**
  Die Geschwindigkeit mit der neue Daten beispielsweise bei Sozialen Netzwerken entstehen oder sich alte Daten ändern ist extrem hoch.
- **Datenstruktur**
  Die Datenstruktur ist meistens sehr unterschiedlich und reicht von strukturierten Daten über semi-strukturierte Daten bis zu unstrukturierten Daten.

Durch eben diese Veränderung haben sich die Rahmenbedingungen verändert und man kann neue Anforderungen an DBMS ableiten. Im Folgenden werden die vier wichtigsten beschrieben.

1) Struktur: Hierbei ist zu unterscheiden, ob die Struktur der Daten häufig verändert wird, also praktisch bei keinem Datensatz vorhergesagt werden kann, welche Struktur er haben wird oder ob die Struktur sich regelmäßig ändert, durch zum Beispiel Erweiterung von Software, die mit der Datenbank arbeitet. In beiden Fällen ist ein konvertieren der Datenbank

teuer und sorgt für ein Zeitfenster, in dem die Datenbank nicht verfügbar ist.

2) Parallelisierung: DBMS müssen viele Anfragen bearbeiten und im besten Fall parallel. Ein pessimistisches Sperr-Verfahren, wie es bei SQL durch die Isolation verlangt wird, ist häufig hinderlich und würde bei einigen Anwendung zu stark erhöhten Antwortzeiten führen. Nur mit der effizienten Ausnutzung von Mehrkernarchitekturen kann die Datengeschwindigkeit in vielen Fällen erreicht werden.

3) Partitionierung: Auf Grund der großen Datenvolumen werden DBMS immer häufiger als Cluster aufgebaut, um die Daten auf mehrere Servermaschinen zu verteilen. In solchen Cluster muss sichergestellt werden, dass der Ausfall eines Knotens und das Hinzufügen eines Knotens oder das Neustarten eines ausgefallenen Knotens nicht zur Beeinträchtigung des Systems führen.

4) horizontale Skalierbarkeit: Die horizontale Skalierbarkeit unterstützt die Parallelisierung und die Partitionierung. Hierbei wird anders als bei der vertikalen Skalierung nicht eine Servermaschine durch genau eine leistungsstärkere Servermaschine ausgetauscht, sondern es wird ein meist heterogenes Netzwerk aus Maschinen aufgebaut [10]. Dabei wird versucht, möglichst Preis-Leistungs-starke Maschinen zu verwenden und alte Maschinen nicht zwingend zu ersetzen. Die horizontale Skalierbarkeit ist vor allem für große Datenbanken wichtig, da Hardware ab einem Punkt immer überproportional im Preis steigt.

C. NoSQL

In den letzten Jahren wurden die DBMS, die von den herkömmlichen SQL-DBMS in ihrer Datenspeicherung und -verarbeitung stark abweichen unter dem Begriff NoSQL zusammengefasst. Hierbei ist hervorzuheben, dass NoSQL für „Not only SQL“ steht und der Begriff, nach einigen heute zu NoSQL gezählten Datenbanken, entstand. Es geht also nicht darum, SQL zu ersetzen, sondern es zu ergänzen. Die grundlegende Idee von NoSQL ist für Problemstellungen, bei denen SQL ungeeignet ist, neue Ansätze zu verfolgen, anstatt wie vorher häufig versucht wurde, das Problem auf Umwegen mit SQL zu lösen. Die Vorteile dieses Vorgehens sind klar: Das Speichern und Verarbeiten von Daten, die nicht effizient mit einem relationalen Schema dargestellt werden können, ist in SQL nur sehr schwer möglich und geht daher immer mit Performance-Verlusten einher. Eine geeignete NoSQL-Lösung hingegen kann dies sehr effizient lösen. Zum besseren Verständnis der unterschiedlichen Philosophien hinter SQL und NoSQL hilft das CAP-Theorem [10] [11] (siehe Abbildung 1). Dabei geht es um die drei Eigenschaften: Verfügbarkeit, Konsistenz und Partitionierung. Aus diesen drei sind zwei zu wählen die am wichtigsten für die Problemstellung sind. Allerdings wird das CAP-Theorem von vielen Vertretern von NoSQL fehlinterpretiert. Es ist keine Rechtfertigung dafür eine der drei Eigenschaften vollständig aufzugeben, um die übrigen beiden zu verbessern. Es geht dabei viel mehr um einen Trade-Off, indem eine Eigenschaft zur Stärkung der anderen beiden abgeschwächt wird. Im Zuge des CAP-Theorem wurde das BASE-Prinzip entwickelt [11],

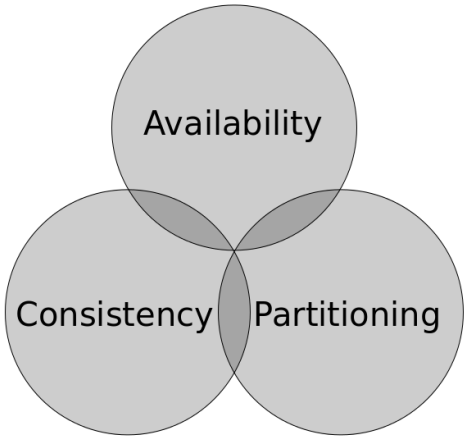

Abbildung 1. CAP-Theorem

welches als Gegenstück zum ACID-Prinzip zu sehen ist und die Grundlage vieler NoSQL-Datenbanken darstellt. BASE beschreibt die folgenden Eigenschaften eines Systems:

*1) **B**asically **A**vailable:* Die Verfügbarkeit der Datenbank wird als maßgebend angesehen. Sie soll möglichst maximiert werden und im optimalen Fall nicht durch verstärkte Anfragen oder das Ausfallen von Hardware bedingt sein.

*2) **S**oft state:* Der Zustand einer Datenbank kann sich ändern, auch ohne das dies direkt auf Input zurückzuführen ist. Dies passiert zum Beispiel, wenn sich verschiedene Servermaschinen synchronisieren.

*3) **E**ventual consistency:* Die Datenbank kann sich, wenn sie auf mehrere Maschinen verteilt ist, für ein gewisses Zeitfenster in einem inkonsistenten Zustand befinden. Das System versucht allerdings, diese Zustände mit der Zeit wieder zu beheben. Die Datenbank erreicht also nach einer ungewissen Zeit wieder einen konsistenten Zustand.

NoSQL-DBMS lassen sich in verschiedene Klassen einteilen (siehe Abbildung 2). Meistens werden NoSQL-DBMS in die drei Klassen Key-Value Store, Document Store und Wide Column Store eingeteilt. Die hier verwendete Klassifizierung ist um Graph DBMS und Mulit-Mode-DBMS erweitert. Graph DBMS gibt es schon seit einigen Jahren und deutlich länger als NoSQL, sie können trotzdem zu NoSQL gezählt werden, da sie kein SQL verwenden und ihre Idee der Grundidee von NoSQL gleicht. Multi-Mode-DBMS bestehen aus Komponenten, die zu verschiedenen NoSQL-Klassen gehören und können teilweise sogar mit einem relationalen Schema arbeiten. Dies hat den Vorteil, dass Daten auf verschiedene Art abgelegt und verarbeitet werden können, ohne ein zusätzliches DBMS zu installieren.

II. Merkmale

Im kommenden Abschnitt werden einige Merkmale, die häufig mit NoSQL verbunden werden, erläutert. Dabei wird versucht, kurz auf Vor- und Nachteile einzugehen und soweit dies Möglich ist, soll auch auf äquivalente Lösungen mit SQL verwiesen werden.

Key-Value Store	Redis
	Memcached
	Riak
Document Store	MongoDB
	CouchDB
Graph DBMS	Neo4j
	Titan
Wide Column Store	Cassandra
	HBase
	Hypertable
Multi-Mode	OrientDB
	Amazon DynamoDB

Abbildung 2. NoSQL-Klassifizierung [12]

A. *Schema oder Schemalos*

Grundsätzlich liegt NoSQL-Datenbanken kein relationales Schema zu Grunde. Dies ist einer der am häufigsten genannten Vorteile von NoSQL. Zu Beginn eines Projektes sollte jedoch immer genau analysiert werden, ob die Daten wirklich nicht mit einem relationalen Schema dargestellt werden können. Sollte dies nicht der Fall sein, wird man mit NoSQL immer auf Probleme stoßen, die das gesamte Projekt gefährden können.
Des Weiteren folgt durch die Wahl des NoSQL-DBMS immer ein gewisses Schema, in dem die Daten abgelegt werden. Ein Key-Value Store legt zum Beispiel alle Daten in Key-Value-Paaren ab, wodurch eine gewisse Struktur vorgegeben ist. Diese Strukturen sind allerdings deutlich simpler und lassen meistens die Definition von Integritätsbedingungen, welche sehr hilfreich beim Betrieb eines Systems sind, nicht zu.
Für agile Softwareentwicklung wären Schema-Evolutions-Werkzeuge [13] innerhalb des NoSQL-DBMS sehr interessant. Hierbei würde ein Schema entwickelt werden, was mit Hilfe des Werkzeugs deutlich schneller geändert werden kann, als dies in SQL möglich ist. Würden dem Schema nämlich Attribute oder ähnliches hinzugefügt werden, könnte das DBMS alle neuen Datensätze mit den neuen Attributen anlegen, ohne die alten zwingend überarbeiten zu müssen.
Die Gefahr ist bei all diesen Überlegungen jedoch, dass Stück für Stück ein starres Schema in der Anwendungsebene über der Datenbank entsteht. In diesem Fall würde man praktisch alles, was von SQL-DBMS standardmäßig unterstützt wird, neu entwickeln. Das hierbei Fehler passieren, die verheerende Folgen haben können, ist sehr wahrscheinlich.

B. *in-Memory*

NoSQL hat einige sogenannte Hauptspeicherdatenbanken hervorgebracht, welche ihre Daten ausschließlich im Hauptspeicher haben und damit auch ohne Verbesserung der verwendeten Algorithmen eine deutlich höhere Performance haben. Die Daten im Hauptspeicher zu halten, um schneller auf sie zugreifen zu können, ist allerdings keine Erfindung der NoSQL-Bewegung. SQL-DBMS haben immer schon die wichtigsten Bereiche wie die Schlüssel der Tabellen im Hauptspeicher vorgehalten. Jedoch war lange Zeit der Preis für diese Art von Speicher zu hoch, als das es sich gelohnt hätte die gesamte Datenbank im Hauptspeicher zu halten.
Auch SQL-DBMS wie Oracle oder die SAP HANA haben seit einiger Zeit eine in-Memory-Funktionen. Es sollte auch keinen Unterschied machen, abgesehen von den Zugriffszeiten, ob die Daten im Hauptspeicher, auf einer herkömmlichen HDD oder auf einem Magnetband liegen, da die Kernaufgabe eines DBMS gerade die physische Datenunabhängigkeit ist.
Eine Gefahr, die sowohl SQL also auch NoSQL betrifft, ist die Flüchtigkeit von Hauptspeicher. Kommt es zu einer Unterbrechung der Stromversorgung, führt dies zum Verlust der Daten. Also müssen regelmäßig Snapshots der Datenbank angefertigt werden und auf einem nicht flüchtigen Speicher abgelegt werden. Dies bedeutet jedoch, dass bei einer Unterbrechung der Stromversorgung, die zuletzt veränderten Daten mit sehr großer Wahrscheinlichkeit verloren sind. Wirklich Abhilfe schafft hier nur die Spiegelung der Daten oder die Verwendung von nicht flüchtigem Arbeitsspeicher [14].

C. *Verteilte Systeme*

Eine weitere wichtige Eigenschaft von NoSQL-DBMS ist die Realisierung von verteilten Systemen, dabei ist es häufig nicht nur wichtig, einen Cluster aus einer großen Anzahl von Servermaschinen aufzubauen, sondern auch verschiedene Cluster zu vernetzen.
Für große Datenbanken werden beispielsweise häufig Shared-Nothing-Architekturen verwendet. In einer Shared-Nothing-Architektur speichert jede Maschine ihre eigenen Daten. Meistens werden die Daten nach Nutzern aufgeteilt, sodass die Anfragen eines bestimmten Nutzers immer an eine Maschine weitergeleitet werden. Diese Architekturen haben den Vorteil, dass sie praktisch beliebig erweitert werden können, durch das Hinzufügen neuer Maschinen, unter der Voraussetzung, dass eine Möglichkeit existiert, die Daten sinnvoll aufzuteilen. Wird vom System allerdings gefordert, dass jede Maschine eine eigene Kopie der Gesamtdaten speichert, müssen komplexe Koordinationsprotokolle verwendet werden und das System benötigt eine ausgezeichnete Vernetzung der Maschinen, sowie eine gewisse Toleranz gegenüber Hardwareausfällen.
Eine Shared-Nothing-Architektur, in der die Nutzer auf die Maschinen verteilt werden, hat bei einer naiven Implementierung allerdings n Single-Points-of-Failure, wobei n der Anzahl an Maschinen entspricht. Fällt eine Maschine aus, können alle Nutzer, die dieser zugeteilt wurden, nicht mehr weiter arbeiten. Daher werden die Daten häufig repliziert, dies führt aber zu Konsistenzproblemen. Replikationen werde normalerweise nach zwei Arten von Muster aufgebaut:

1) Master-Slave: Eine Replikation wird häufig als Master-Slave-Architektur aufgebaut, in der es einen Master gibt, der Anfragen entgegen nimmt und die Daten auf die Slaves spiegelt. Fällt der Master in diesem System aus, wird er durch einen Slave ersetzt und der alte Master wird nach einem erfolgreichen Neustart zu einem Slave.

2) Multi-Master: Eine weitere Möglichkeit bieten Multi-Master-Achritekturen, bei denen jede Maschine als Master

auftritt und Anfragen entgegennehmen kann. Hier müssen Abstimmungsverfahren implementiert werden, damit die Maschinen Veränderungen an den Daten untereinander propagieren können. Werden die Daten von verschiedenen Anfragen über verschiedene Master verändert, liegt es im Aufgabenbereich des DBMS zu entscheiden, welche Daten übernommen werden sollen oder ob ein manuelles Eingreifen vonnöten ist.
Allgemein ist das Verteilen von Datenbanken auf eine große Anzahl von Maschinen ein sehr komplexer Bereich, der von SQL-Systemen häufig gemieden wurde. Bei NoSQL wurde dieser Bereich direkt betrachtet, weshalb die großen NoSQL-DBMS meistens das Arbeiten in einer verteilten Umgebung nativ unterstützen.

D. Sicherheit

In Bezug auf die Sicherheit hinken die meisten NoSQL-DBMS den SQL-DBMS und dem was in den meisten Projekten benötigt wird extrem weit hinterher. Da es eigene Arbeiten zum Thema Sicherheit in NoSQL gibt [15], wird hier lediglich auf einige wichtige Punkte verwiesen. Diese Punkte sollten unbedingt geprüft werden, bevor ein NoSQL-DBMS eingesetzt wird:

- Gibt es eine Möglichkeit, die Daten auf dem Datenträger automatisch durch das DBMS zu verschlüsseln?
- Sind Injection-Attacken möglich?
- Wird die Kommunikation mit den Clients ausreichend verschlüsselt?

Da die meisten NoSQL-DBMS häufig aktualisiert werden, ist es nur schwer möglich, eine dauerhafte Aussage über die Sicherheit eines solchen Systems zu treffen.

E. verschiedene APIs oder SQL

Im Gegensatz zu SQL gibt es bei NoSQL keinen offiziellen Standard, daher ist es häufig deutlich schwerer, sich in ein System einzuarbeiten. Doch auch der SQL-Standard wird nicht von allen Systemen gleich oder in gleichem Umfang umgesetzt. Dennoch ähneln sich SQL-DBMS deutlich mehr, als dies NoSQL-DBMS tun.
Auch wenn die bekannten NoSQL-DBMS eine gewisse Einheitlichkeit ihrer APIs anstreben, sind sie doch noch sehr weit von dem, was für SQL gilt, entfernt. Durch das Aufkommen immer neuer Systeme wird dies wahrscheinlich auch noch längere Zeit so bleiben. Ein Wechsel von einem NoSQL-DBMS zu einem anderen NoSQL-DBMS ist somit aufwändig und dadurch sehr kostspielig.
Dies gilt für SQL jedoch ebenfalls. Im Allgemeinen gilt: wenn in einer Firma einmal ein DBMS eingeführt wurde und alle Anwendungen damit arbeiten, ist es häufig unmöglich, dieses System auszutauschen, ohne das enorme Kosten entstehen.

III. Einsatzgebiete

Während einige Jahrzehnte lang SQL für alle Probleme der Datenverarbeitung eine passende Lösung anbot, ist dies bei aktuellen Anwendungen öfter nicht der Fall. Daher werden bei NoSQL sehr spezielle Lösungen für Problemstellungen, bei denen SQL scheitert oder zumindest nicht die gewünschte Performance bietet, entwickelt.

A. Projekt-Abhängigkeit

Wenn in einem Projekt ein DBMS zum Einsatz kommen soll, muss man genau abwägen welches DBMS eingesetzt werden soll. Erst recht, wenn ein NoSQL-DBMS eingesetzt werden soll, müssen die Vor- und Nachteile genau abgewogen werden. Die folgenden drei Punkte geben einen Überblick darüber, was zu beachten ist:

1) Projektart: Obwohl Vertreter des NoSQL-Bereich die Beschaffenheit der Daten häufig als das ausschlaggebende Kriterium sehen, ist die Projektart meistens entscheidender wenn es um die Wahl eines DBMS geht. Interessant ist hier das Budget, denn mächtige SQL-DBMS sind in der Anschaffung und im Betrieb teuer, während viele NoSQL-DBMS Open-Source-Projekte sind und somit kostenfrei. Natürlich ist die Installation und der Betrieb sowohl von SQL- als auch von NoSQL-DBMS kostenintensiv. Zudem ist es relevant, ob es sich um ein staatliches, wirtschaftliches oder privates Projekt handelt. Sowohl im staatlichen als auch im wirtschaftlichen Bereich sind SQL-DBMS beliebter, da sie sich über Jahre hinweg bewährt haben, wohingegen NoSQL immer noch sehr weit am Anfang steht.

2) Datenart: Die Datenart ist in sofern von Interesse, als das es Daten gibt, wie zum Beispiel Graphen oder wirklich unstrukturierte Daten, die nicht vernünftig in ein relationales Schema passen. Doch häufig werden Daten völlig ohne Grund als unstrukturiert deklariert, da es einen gewissen Aufwand mit sich bringt, ein relationales Schema für einige Daten zu entwerfen. Jedoch passen die meisten Daten, die Objekte aus der wirklichen Welt beschreiben, in ein relationales Schema und auch viele andere Daten.

3) gesetzliche Bestimmungen: Die Wahl eines DBMS wird häufig auch durch gesetzliche Bestimmungen eingeschränkt. Für Bereiche der Buchführung oder die Finanzwirtschaft eigenen sich SQL-DBMS auf Grund des ACID-Prinzips. Generell eigenen sich SQL-DBMS immer, wenn die Daten durch Gesetze festgelegt werden, da aus den Gesetzgebungen meist sehr einfach ein relationales Schema mit sehr strikten Integritätsbedingungen abgeleitet werden kann. Alle Bereiche des Handels oder der staatlichen Verwaltung benötigen ein deterministisches System, was im Zuge von BASE für NoSQL meistens nicht zutrifft.
Für Unternehmen und Staaten ist es zudem sinnvoll, Software von großen, schon lange existierenden Unternehmen einzusetzen, da dies ein gewisses Maß an Sicherheit gibt. Bei der großen Anzahl von NoSQL-DBMS [12] und den raschen Veränderungen in diesem Bereich, ist es nicht sicher, ob es in einigen Jahren das ein oder andere NoSQL-DBMS noch geben wird.

B. sicherheitskritische Bereiche

In sicherheitskritischen Bereichen ist das ACID-Prinzip von entscheidender Wichtigkeit, weshalb NoSQL in diesen Bereichen wahrscheinlich nie relevant wird. In Systemen, von denen Menschenleben abhängen können, muss eine Konsistenz der Daten sichergestellt werden, hier ist ein unbestimmtes Zeitfenster der Inkonsistenz nicht akzeptabel. Des Weiteren existieren in solchen Systemen meistens viele

Integritätsbedingungen, die es zu Überwachen gilt. Hierfür sind SQL-DBMS deutlich besser geeignet.
In Bereichen, die solche Sicherheitsbedingungen nicht aufweisen, können NoSQL-DBMS durchaus eingesetzt werden und das teilweise sogar sehr effizient. Als Beispiel sei hier auf Soziale Netzwerke verwiesen. Die Einträge in einem sozialen Netzwerk müssen nicht sofort aktuell sein, hier ist es wichtiger, dass das System möglichst immer Verfügbar ist.

C. Data Mining

Beim Data Mining wird versucht in großen Datenmengen, zum Beispiel Log-Daten von Internetseiten, ein Muster zu erkennen. Hierbei sind NoSQL-DBMS häufig sehr hilfreich, da man riesige Mengen von unstrukturierten Daten überprüfen muss und es mit vielen NoSQL-DBMS ebenfalls möglich ist, diese Überprüfung auf eine große Anzahl von Servermaschinen aufzuteilen. Hierfür wird häufig das MapReduce-Verfahren verwendet, welches in einigen NoSQL-DBMS implementiert wurde. [16]
Auch Graphen-Datenbanken werden vermehrt im Data Mining eingesetzt und sind deutlich effizienter als vergleichbare Lösungen in SQL. [17]

D. Koexistent von SQL und NoSQL

Einige große Datenbank-Entwickler versuchen, die Vorteile, die NoSQL bietet, in ihre eigenen Systeme zu übernehmen. Die DB2 wurde zum Beispiel in der Version 10.5 um einige Funktionalitäten erweitert [18], die BLU genannt werden und vor allem die Datenanalyse verbessern sollen. SAP versucht, mit der HANA Database, die Teil der SAP HANA ist, ebenfalls einige der neuen Anforderungen zu erfüllen [19].
Das Problem der hier beschriebenen Systeme ist jedoch, dass es sich um Closed-Source handelt, daher ist es sehr aufwendig nachzuvollziehen, ob es wirklich möglich ist zum Beispiel Graphen in einem eigenen Format abzuspeichern oder ob diese nicht wohl möglich in ein relationales Schema konvertiert werden und der Geschwindigkeitsverlust, der dabei entsteht, durch verbesserte Hardware ausgeglichen wird.

IV. Nachholbedarf der NoSQL-Bewegung

Es lassen sich drei Punkte ableiten, in denen die meisten NoSQL-DBMS enormen Nachholbedarf haben:

1) Sicherheit: Da die meisten NoSQL-DBMS auch in Zukunft an das Internet angeschlossen sein werden, müssen vor allem die Verschlüsselungen verbessert werden. In Puncto Sicherheit lässt sich viel aus den Erfahrungen mit SQL lernen und es ist nicht nötig, die selben Fehler erneut zu machen.

2) Standardisierung der Schnittstellen: Damit NoSQL-DBMS für Unternehmen in Zukunft interessanter werden, wäre es hilfreich, wenn zumindest innerhalb der verschiedenen NoSQL-Klassen sich ein starker Standard durchsetzt.

3) Entwicklung von Werkzeugen: Die Entwicklung von Werkzeugen für Administratoren könnte die Betriebskosten der NoSQL-DBMS deutlich reduzieren. Vor allem im Bereich der Datenmigration und der Schema-Evolution [13] wären viele hilfreiche Werkzeuge denkbar.

V. Zusammenfassung

NoSQL ist für spezielle Probleme interessant, wird jedoch in klassischen Bereichen, wie der Unternehmensverwaltung, nicht relevant werden. Aktuell wird der NoSQL-Bereich sehr überschätzt, wodurch einige Behauptungen und Meinungen zu schnell als Wahrheiten verstanden werden. Diese Einstellung muss sich ändern, damit man auch von anderen Bereichen, wie zum Beispiel SQL, lernen kann. Es bleibt abzuwarten, wie viel in einigen Jahren von diesem Hype noch übrig ist. Allerdings ist nicht damit zu rechnen, dass sich der gesamte Bereich auflösen wird, sondern es werden auch in Zukunft einige NoSQL-Klassen weiterhin von Bedeutung sein.

Literatur

[1] M. M. Astrahan and D. D. Chamberlin, “Implementation of a structured english query language,” *Commun. ACM*, vol. 18, no. 10, pp. 580–588, 1975.

[2] X. ANSI, “American national standard for information systems: Database language sql,” *American National Standards Institute, NY*, vol. 135, 1986.

[3] E. F. Codd, “A relational model of data for large shared data banks,” *Commun. ACM*, vol. 13, no. 6, pp. 377–387, 1970.

[4] T. Haerder and A. Reuter, “Principles of transaction-oriented database recovery,” *ACM Comput. Surv.*, vol. 15, no. 4, pp. 287–317, 1983.

[5] IBM DB2. http://www.ibm.com/software/data/db2/. Erreicht 07.07.2015.

[6] Orcale Database 12c. https://www.oracle.com/database/. Erreicht 07.07.2015.

[7] MS SQL Server. http://www.microsoft.com/de-de/server-cloud/products/sql-server/. Erreicht 07.07.2015.

[8] PostgreSQL. http://www.postgresql.org/. Erreicht 07.07.2015.

[9] D. Abadi, R. Agrawal, and A. Ailamaki, “The beckman report on database research,” *SIGMOD Rec.*, vol. 43, no. 3, pp. 61–70, 2014.

[10] E. A. Brewer, “Towards robust distributed systems,” in *PODC*, vol. 7, 2000.

[11] E. Brewer, “Cap twelve years later: How the rules have changed,” *Computer*, vol. 45, no. 2, pp. 23–29, 2012.

[12] NoSQl-Databases.org. http://nosql-database.org/. Erreicht 07.07.2015.

[13] M. Klettke, S. Scherzinger, and U. Störl, “Datenbanken ohne schema?” *Datenbank-Spektrum*, vol. 14, no. 2, pp. 119–129, 2014.

[14] “White paper: Putting flash on the memory bus,” SanDisk Corporation, Tech. Rep. WP008, Dezember 2013.

[15] L. Okman, N. Gal-Oz, Y. Gonen, E. Gudes, and J. Abramov, “Security issues in nosql databases,” in *Trust, Security and Privacy in Computing and Communications (TrustCom), 2011 IEEE 10th International Conference on*. IEEE, 2011, pp. 541–547.

[16] J. Dean and S. Ghemawat, “System and method for efficient large-scale data processing,” Jan. 19 2010, uS Patent 7,650,331.

[17] V. Kolomičenko, M. Svoboda, and I. H. Mlýnková, “Experimental comparison of graph databases,” in *Proceedings of International Conference on Information Integration and Web-based Applications Services*, ser. IIWAS ’13. ACM, 2013, pp. 115:115–115:124.

[18] M. Päßler and M. Nicola, “Db2 10.5 mit blu acceleration: Optimiert für komplexe analytische anfragen,” *Datenbank-Spektrum*, vol. 14, no. 1, pp. 47–52, 2014.

[19] F. Färber, S. K. Cha, J. Primsch, C. Bornhövd, S. Sigg, and W. Lehner, “Sap hana database: data management for modern business applications,” *ACM Sigmod Record*, vol. 40, no. 4, pp. 45–51, 2012.