Verbindung relationaler Datenbanksysteme und NoSQL-Produkte

Ein Überblick

Andreas Göbel
Friedrich-Schiller Universität Jena
Lehrstuhl für Datenbanken und Informationssysteme
Ernst-Abbe-Platz 2
07743 Jena, Germany
andreas.goebel@uni-jena.de

KURZFASSUNG

In den letzten Jahren entstanden verschiedene Open-Source-Systeme, die mit fundamentalen Konzepten und Regeln relationaler Datenbanksysteme brachen, um die Verwaltung von Daten in speziellen Einsatzbereichen zu optimieren. Die wesentlichen Gründe für die Entwicklung dieser so genannten NoSQL-Systeme sind jedoch nicht SQL oder das relationale Datenbankmodell, sondern sie ist auf die Implementierung relationaler Datenbanksysteme zurückzuführen. Der Beitrag verdeutlicht durch eine Gegenüberstellung von Oracle Real Application Cluster, IBM DB2 PureScale und MySQL Cluster die gegensätzlichen Implementierungen relationaler Clusterlösungen. An die Motivation der NoSQL-Produkte sowie einen Überblick ihrer Zielstellung, Vor- und Nachteile schließt sich das Aufzeigen von Möglichkeiten an, um Konzepte und Implementierungen beider Welten miteinander zu verbinden und so die Vorzüge zu vereinen.

Kategorien und Themenbeschreibung

H.2.4 [**Database Management**]: Systems—*Parallel databases* ; H.3.5 [**Database Management**]: Systems and Software—*Distributed systems*

Allgemeine Bestimmungen

Theory, Design, Reliability

Stichworte

Parallel Databases, NoSQL, Postrelational, Hybrid

23^{rd} GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken), 31.05.2011 - 03.06.2011, Obergurgl, Austria.

1. EINLEITUNG

Die zunehmende Verbreitung von Unternehmensnetzwerken, globalen Netzwerken wie dem Internet und mobilen Endgeräten gepaart mit dem Wunsch vieler Unternehmen nach Globalisierung führt vermehrt zur Nutzung zentraler (Datenbank-)Services für eine Vielzahl von Nutzern. Die unter dem Begriff Web 2.0 zusammengefassten Entwicklungen ermöglichen zunehmend Interaktion und Verknüpfungen in Netzwerken, was sowohl die Gestalt als auch die Menge der Daten auffallend beeinträchtigt. So werden Inhaber erfolgreicher Web-Anwendungen mit beachtlichen Datenmengen konfrontiert, die das Datenaufkommen in klassischen Anwendungen um ein Vielfaches übersteigen können.

Relationale Datenbanksysteme sind zentraler Bestandteil des Software-Stacks vieler Unternehmen und Behörden. Mittels der Verbindung eines mathematischen Fundaments, der Gewährleistung der ACID-Eigenschaften und der standardisierten deskriptiven Abfragesprache SQL stellen sie die Verfügbarkeit, Korrektheit und Auswertbarkeit der Unternehmensdaten sicher. Der vorliegende Beitrag motiviert, warum Betreiber vieler Web-Anwendungen trotz der auf der Hand liegenden Vorteile bewährter relationaler Produkte Eigenentwicklungen propietärer Spezialsysteme zur Datenverwaltung vorantreiben, die bewusst auf wesentliche Merkmale relationaler Systeme verzichten.

Nach einer Gegenüberstellung relevanter Implementierung relationaler Clusterdatenbanken werden die Herausforderungen und Einschränkungen der zu dem Schlagwort NoSQL zusammengefassten Systeme herausgearbeitet, einige aktuelle Entwicklungen zur Verbindungen von NoSQL und RDBMS zusammengefasst und die Notwendigkeit flexiblerer Implementierungen relationaler Datenbanksysteme aufgezeigt.

2. HERAUSFORDERUNGEN

Die Charakteristika zu verarbeitender Daten bei Web-Anwendungen führen zu folgenden Kern-Herausforderungen an zu verwendende Datenbanksysteme bzw. Datenspeicher.

Performance und Skalierbarkeit kennzeichnen die bedeutendsten Herausforderungen. Die damit verbundene Verringerung der Latenzzeit in Web-Anwendungen steht häufig in direktem Zusammenhang mit der Nutzerzufriedenheit und ist insbesondere in Bereichen wie Suchmaschinen oder dem E-Commerce-Sektor von essentieller Bedeutung. Die Performance resultiert aus der Grundperformance für Anfra-

gen und der Verarbeitungsgeschwindigkeit für steigende Datenvolumina, welche allgemeinhin als Skalierbarkeit bezeichnet wird. Eine zunehmende Datenmenge kann hierbei die Dauer von Aufgaben, die Anzahl der Aufgaben oder beides erhöhen. Die Skalierbarkeit eines Rechensystems kann durch den Einsatz leistungsfähigerer Hardware (vertikale Skalierbarkeit) oder durch das Verteilen der Aufgaben auf weitere Rechenressourcen (horizontale Skalierbarkeit) erzielt werden. Die Vorgänge müssen jeweils transparent zur Anwendung geschehen.

Ausfallsicherheit ist für jedes zentrale (Datenverarbeitungs-)System eine wesentliche Herausforderung, um Nutzern dauerhafte Verfügbarkeit zu bieten. Neben ungeplanten Ausfällen eines Systems in Folge von Hardwaredefekten oder Systemfehlern müssen auch geplante Ausfälle – beispielsweise zur Aktualisierung des Systems – vermieden werden. Beide Ausfallarten können erheblichen wirtschaftlichem Schaden durch Kundenverlust oder Pönalen bei Verstoß gegen Service Level Agreements nach sich ziehen. Um Hochverfügbarkeit zu erreichen, sollten Single Points of Failure (SPoF) in einem System vermieden sowie binnen kurzer Zeit und automatisiert auf jegliche Art von Fehlern reagiert werden.

Schemaflexibilität bezeichnet den Verzicht auf ein vordefiniertes und stets omnipräsentes Datenbankschema, um den Umgang mit Datenbanken und -speichern flexibler zu gestalten. Dies ermöglicht die adäquate Verwaltung semistrukturierter und dokumenten-orientierter Daten, die nicht zuletzt aufgrund von Web-Standards und Auszeichnungssprachen wie XML oder RDF in Web-Anwendungen weit verbreitet sind. Schemaflexibilität spielt des Weiteren eine wichtige Rolle bei der Konsolidierung von heterogenen Nutzerdaten innerhalb eines Systems.

Kosten: Für viele Betreiber von Web-Anwendungen ist der Einsatz kostengünstiger Hard- und Software eine Grundvoraussetzung. Lizenz-, Support- und Administrationskosten für Datenbanksysteme sowie die Anschaffungs-, Administrations- und Betriebskosten von Datenbankservern machen meist einen nicht unerheblichen Teil der IT-Gesamtaufwendungen aus. Aus diesem Grund wird für Unternehmen die Nutzung von kostengünstigen Cloud-Services oder -Storages stets lukrativer. Entsprechend sollte ein geeignetes Lizenzierungskonzept angeboten und der Einsatz auf Commodity-Servern unterstützt werden.

Für viele Provider ist die Antwortzeit der Web-Anwendung derart wichtig, dass sie Einschränkungen der Datenkonsistenz in Kauf nehmen oder gar auf die Realisierbarkeit des Definierens von Konsistenzsicherungen verzichten, wenn diese einen Performance-Overhead mit sich bringen. Dies ist bemerkenswert, denn es kennzeichnet einen wahrnehmbaren Wandel der Anforderungen an Datenbanksysteme. In klassischen Unternehmensanwendungen stellt die Forderung nach Datenkonsistenz die oberste Prämisse dar und ist unentbehrlich. Die Herausforderung besteht hierbei im Wesentlichen in der Optimierung der Performance. Dementgegen verdeutlichen die obigen Herausforderungen, dass die Hauptaufgaben vermehrt in der Optimierung der Antwortzeit oder nach [3] gar in der Minimierung der (Hardware-)Kosten und Erhöhung des Konsistenzniveaus bei gegebenen Performance-Vorgaben zu sehen ist.

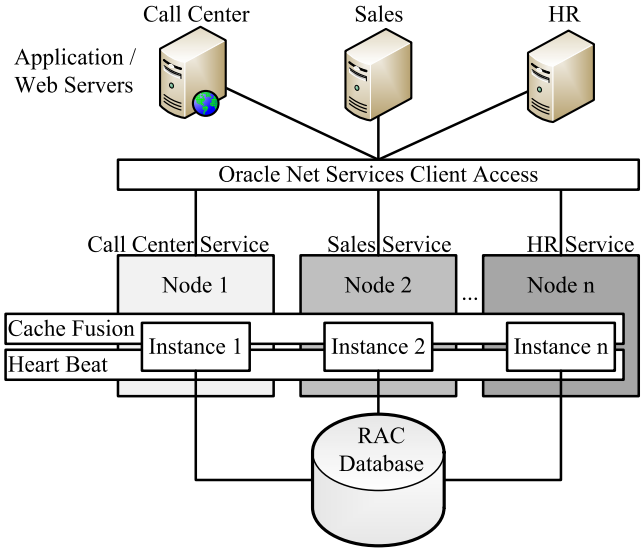

Abbildung 1: Architektur von Oracle RAC (nach [12])

3. RELATIONALE DATENBANKCLUSTER

Horizontale Skalierbarkeit und Hochverfügbarkeit unter Einsatz kostengünstiger Hardware bilden nach Abschnitt 2 die wesentlichen Herausforderungen an die Speicherung und Verarbeitung der Daten von Web-Anwendungen. Beinahe alle relationalen Datenbanksysteme bieten Mittel, um ihre Systeme vor Ausfällen und Datenverlust in diversen Fehlerszenarien zu schützen und eine hohe Verfügbarkeit zu erzielen. Sie bieten für dieses Ziel neben Sicherungs- und Wiederherstellungsmöglichkeiten von Datenbanken verschiedene Techniken zur Replikation. Zumeist erkennen sie ein Problem jedoch erst beim erfolglosen Datenzugriff statt unmittelbar nach dem Auftreten und erfordern im Fehlerfall einen manuellen Eingriff zum Umleiten auf das Replikat. Zudem sind die Replikate bei einigen Systemen ausschließlich im Fehlerfall einsetzbar und dienen im Normalbetrieb nicht der Lastbalancierung. Im Folgenden werden die Eckpunkte vorherrschender Hochverfügbarkeitslösungen gegenübergestellt, die diese Mängel nicht aufweisen und zudem horizontale Skalierbarkeit in Mehrrechnersystemen ermöglichen.

3.1 Oracle Real Application Cluster

Oracle Real Application Cluster (RAC) ermöglicht bis zu 100 Datenbankinstanzen einen parallelen Zugriff auf denselben Datenbestand und realisiert somit eine Shared-Disk-Architektur. Wie Abbildung 1 verdeutlicht, greifen die Application Server und Web Server über eine gemeinsame Service-Schnittstelle auf das System zu, die u.a. der Lastbalancierung dient. Sämtliche Dateien für Daten, Verwaltung und Konfigurationsparameter werden auf einem clusterfähigen Storage-System gespeichert und sind von allen Servern les- und schreibbar. Lediglich die Undo- und Redo-Logs bilden eine Ausnahme: Sie werden stets von der Besitzerinstanz geschrieben und können nur von deren Nachbarinstanzen gelesen werden, um die Besitzerinstanz bei einem Ausfall automatisch wiederherstellen zu können. Der Ausfall eines Knotens wird durch eine Heartbeat-Netzwerkverbindung in kürzester Zeit erkannt. Extended Distance Cluster bietet durch

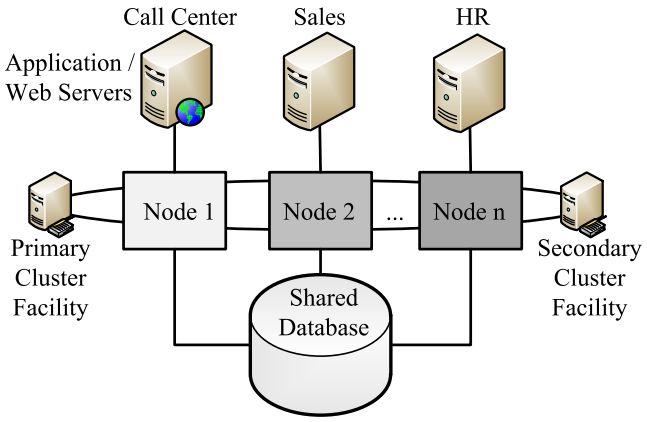

Abbildung 2: Architektur von IBM DB2 PureScale (nach [9])

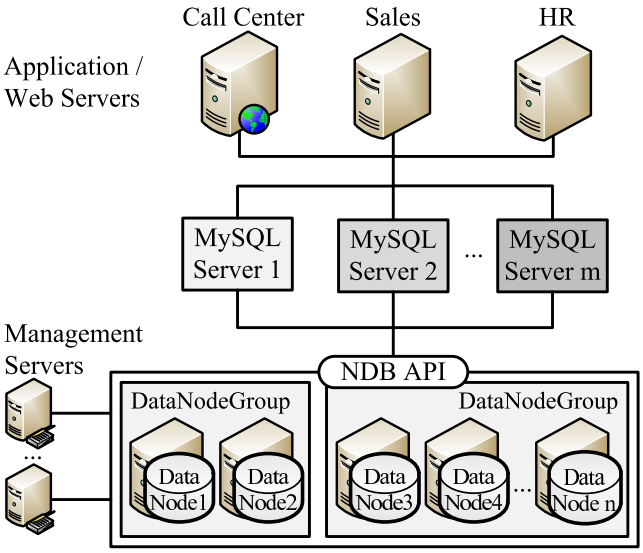

Abbildung 3: Architektur von MySQL Cluster (nach [10])

eine Systemspiegelung auf ein aktives System innerhalb weniger Kilometer das Reagieren auf Fehlerszenarien, die zum Ausfall des kompletten Clusters führen. Oracle Data Guard ermöglicht darüber hinaus das Spiegeln auf ein weiter entferntes Standby-System zur Realisierung einer Disaster Recovery.[12, 5]

Oracle RAC bietet Skalierbarkeit durch das Hinzufügen neuer Nodes, einen automatischen Lastausgleich und die parallelisierte Ausführung von Operationen auf mehreren Servern. Für den parallelen Zugriff mehrerer Instanzen auf denselben Datenbestand nutzt Oracle im Falle einer Datenmodifikation den Global Cache Service (GCS), um zu bestimmen, in welchen lokalen Knotencaches die betroffenen Blöcke liegen bzw. ob sie sich gegebenenfalls bereits auf dem Storage-System befinden. Nachdem die Position bekannt ist, werden die Blöcke durch ein In-Memory-Blockinventar (Global Resource Directory, GRD) und Global Enqueue Service auf aktive Schreibsperren und weitere wartende Instanzen geprüft, um anschließend eigene Schreibsperren zu setzen, die wiederum im GRD vermerkt und anderen Nodes bekannt gemacht werden. Die verwendeten Komponenten werden unter dem Begriff Cache Fusion zusammengefasst und ermöglichen zudem beim Datenzugriff das direkte Versenden von Daten zwischen Buffer-Caches verschiedener Nodes. Oracle RAC vermeidet somit Cache-Kohärenz und ein SPoF durch einen globalen Cache, jedoch auf Kosten von sehr viel Kommunikation.[12, 5]

3.2 IBM DB2 PureScale

Das Design der IBM-Clusterlösung DB2 pureScale basiert auf der Architektur des bewährten Parallel Sysplex für System z[1]. Sie ermöglicht durch eine Shared-Disk-Architektur den gemeinsamen Zugriff von bis zu 128 Datenbankservern auf einen gemeinsamen Datenbestand, der durch IBMs General Parallel File System zur Verfügung gestellt wird. Die Abbildung 2 zeigt, dass der Cluster neben den Datenbankservern aus Cluster Facilities (CF) besteht. Um einen SPoF zu verhindern, ist diese Komponente meist doppelt ausgelegt. Sie kann ein eigenständiges System sein oder auf einem Clusterknoten betrieben werden. Die CF ermöglicht die zentrale Verwaltung der Sperren (Global Lock Table, GLT) und des Caches (Group Buffer Pool, GBP), wodurch analog zum GRD und GCS des Oracle RAC die Informationen der Datenblöcke verwaltet und allen Servern zur Verfügung gestellt werden. Die Server sind untereinander sowie mit der CF mittels eines Hochleistungsnetzwerks verbunden, welches einen direkten Fernzugriff auf den Arbeitsspeicher (RDMA) in wenigen Mikrosekunden ermöglicht. Bei einem Schreibvorgang ermöglicht diese schnelle Verbindung das synchrone Aktualisieren der zentralen Sperrtabelle in Form von Zeilen- und Seitensperren und des zentralen wie auch anderer relevanter Caches. Beim Lesevorgang eines Nodes wird nach erfolgloser Suche im lokalen Cache im GBP nach den Blöcken gesucht. Werden die Daten vom Festspeicher in den lokalen Cache geladen, wird dies ebenfalls dem GBP bekannt gemacht. [9, 6]

Ein integrierter Watchdog-Prozess überwacht permanent die Verfügbarkeit sämtlicher Knoten. Wird der Ausfall eines Knotens bemerkt, stehen bis zum Instanzneustart lediglich die momentan von diesem Knoten aktualisierten Tupel nicht zur Verfügung. Logs werden im Gegensatz zu Oracle RAC auf den gemeinsamen Festspeicher geschrieben und sind für die Recovery von anderen Knoten lesbar.

3.3 MySQL Cluster

MySQL Cluster basiert im Gegensatz zu den Lösungen von IBM und Oracle auf einer Shared-Nothing-Architektur, weshalb die bis zu 255 Datenknoten nicht parallel auf einen gemeinsamen Datenbestand zugreifen, sondern jeder Datenknoten einen Teil des Gesamtdatenbestands verwaltet. Die Tabellen werden bei diesem Ansatz horizontal partitioniert. MySQL Cluster stellt keine spezifischen Voraussetzungen an zu verwendende Netzwerke oder Server und unterstützt In-Memory- als auch Festpeicher-Datenspeicherung. Auf das System wird über vollwertige MySQL-Server zugegriffen. Sie sind mit einer Schnittstelle zur NDB-Engine versehen und werden zudem für verschiedene Funktionen wie Views, Trigger oder Volltext-Indizes verwendet, die von der NDB-Engine nicht unterstützt werden. Die Management-Server sind für die Konfiguration des Clusters zuständig, während die

[1] Auf eine gesonderte Beschreibung des Parallel Sysplex wird aufgrund analoger Konzepte verzichtet.

Datenknoten zur Speicherung der Daten und der Verwaltung von Transaktionen dienen. Knoten, die deckungsgleiche Inhalte verwalten, werden zu einer Datenknotengruppe zusammengefasst, in der die synchrone Replikation der Knoten dazu führt, dass der Ausfall von Knoten keine aufwändige Instanz -Wiederherstellung nach sich zieht. Somit müssen Undo- und Redo-Dateien anderen Knoten nicht sichtbar gemacht werden und das System ist verfügbar, solange ein Datenknoten je Gruppe erreichbar ist. Da beim Ausfall eines Knotens die Aktualisierung einer zentralen Sperr- und Pufferverwaltung nicht nötig ist, können sehr geringe Failover-Zeiten erzielt werden. Zudem werden asynchron Checkpoints auf einen Festspeicher geschrieben, um auf den Ausfall kompletter Gruppen reagieren zu können bzw. einen System-Reboot zu ermöglichen. Durch den Einsatz der MySQL Cluster Carrier Grade Edition kann Hochverfügbarkeit durch die Realisierung geografischer Replikation erzielt werden.[10, 11, 5]

3.4 Bewertung

Die vorgestellten Datenbankcluster ermöglichen Skalierbarkeit sowohl durch Einsatz leistungsstärkerer Server als auch durch das Hinzufügen weiterer Server. Trotz verschiedener Realisierungen verfügen sie über effiziente Strategien für die wesentlichen Herausforderungen im Kontext der Skalierbarkeit: Logging, Locking und die Verwaltung von Zwischenspeichern[13]. Im Gegensatz zum Shared-Nothing-Ansatz von MySQL Cluster basieren Oracle RAC und DB2 pureScale auf einer Shared-Disk-Architektur und benötigen wegen ihrer nahen Knotenkopplung schnelle Kommunikation mittels Hochleistungsnetzwerken. Diese ist für die aufwändige Kommunikation der Sperr- und Cachingverwaltung bei gegebener Performance notwendig.

Alle Systeme bieten hohe Ausfallsicherheit bis hin zu Unterstützung einer Disaster-Recovery über die Anbindung entfernter Standby-Systeme. Serverausfälle werden beinahe unmittelbar erkannt, die Wiederherstellung ist in kürzester Zeit möglich und führt kaum zu wenig Einschränkungen. Während bei Oracle RAC im Fehlerfall bis zum Neuaufbau des CGS für einen Augenblick keine Datenmodifikation durchgeführt werden können, stehen bei DB2 PureScale die vom ausgefallenen Knoten aktuell veränderten Daten bis zur Instanz-Wiederherstellung nicht zur Verfügung. MySQL Cluster besitzt durch den Shared-Nothing-Ansatz in Verbindung mit synchroner Replikation der In-Memory-Daten im Fehlerfall kaum Einschränkungen.

Die wesentlichen Nachteile von Oracle RAC und DB2 pureScale bestehen im Kontext der Anforderungen in Abschnitt 2 vor allem in den enormen Kosten für Lizenzen, spezielle Hardware und Wartung im Vergleich zu MySQL Cluster. Insbesondere sind hier der vor Ausfällen zu schützende Shared Storage, das Cluster-Dateisystem sowie leistungsstarke Netzwerke für die Clusterkommunikation und Cache Fusion bzw. die Cluster Acceleration Facilities zu nennen. Oracle RAC wurde zudem in den vergangenen Jahren um diverse Features ergänzt, die zu einer System-Komplexität führten, die eine intensive Einarbeitungszeit unabdingbar macht.

Ein wesentlicher Vorteile von Oracle RAC und DB2 pureScale ist hingegen die einfache Migration von Anwendungen auf die Clustersysteme, da keine Änderung des Anwendungscodes notwendig ist. Da die NDB-Engine von MySQL Cluster nur einen Teil der Funktionen von InnoDB und MyISAM unterstützt, müssen fehlende Funktionalitäten auf die MySQL-Server ausgelagert werden, um deren Performance und Verfügbarkeit manuell gesorgt werden muss. Daher bietet sich der MySQL Cluster vor allem in Szenarien mit einer Vielzahl simpler Anfragen und hohen Latenz- und Verfügbarkeitsanforderungen an, während die Einsatzmöglichkeiten von Oracle RAC und DB2 pureScale kaum begrenzt sind.

4. NOSQL-BEWEGUNG

In den letzten Jahren gewinnen so genannte NoSQL-Systeme zur Verwaltung von Daten zunehmend an Bedeutung. Einige Kritikpunkte bei der Verwendung relationaler (Cluster-)Systeme in der Welt der Services regten Unternehmen zur Eigenentwicklung von Systemen zur Datenspeicherung und -verarbeitung an, die bewusst auf Merkmale relationaler DBMS verzichten, um sich auf einen Anwendungsfall zu spezialisieren. Ausgehend von den technischen Beschreibungen von Systemen bekannter Internetgrößen entstanden im Laufe der letzten Jahre eine Vielzahl von Open-Source-Systemen. Diese kopierten, kombinierten und erweiterten die Konzepte der Ausgangssysteme mit dem Ziel, den Anforderungen der Unternehmen gerecht zu werden. Der Begriff „NoSQL“ umfasst all jene Systeme und wird inzwischen üblicherweise als „Not only SQL“ ausgelegt. Das Ziel dieser Systeme besteht im Aufzeigen von Alternativen zu relationalen Datenbanksystemen und nicht in deren Ablösung.

4.1 Zielstellungen

Mangels einer anerkannten Definition des Begriffs „NoSQL“ werden im Folgenden entsprechend der in Abschnitt 2 beschriebenen Herausforderungen die wesentlichen Zielstellungen der NoSQL-Systeme zusammengefasst, wobei diese als Obermenge der Ziele jedes einzelnen NoSQL-Systems zu sehen sind.

Performance, Skalierbarkeit: Untersuchungen wie [14] zeigen, dass die Performance moderner RDBMS in verschiedenen Bereichen um ein Vielfaches übertroffen werden kann. Als Grund wird vor allem die nach wie vor auf System R basierende und stets erweiterte Architektur gesehen, welche in der Client-Server-Welt hervorragende Dienste leistet, für die Welt der Services und die verschiedenden Leistungs- und Kapazitätsverhältnisse von Prozessoren, Fest- und Arbeitsspeicher jedoch neuer Architekturansätze bedarf [16]. Das Hauptziel der meisten NoSQL-Datenspeicher ist das Erreichen linearer horizontaler Skalierbarkeit zur Verarbeitung riesiger Datenmengen. Sie nutzen hierfür überwiegend Shared-Nothing-Architekturen in Verbindung mit horizontaler Partitionierung der Daten. Das im Jahre 2002 bewiesene Eric Brewers CAP-Theorem besagt, dass nur zwei der drei folgenden Eigenschaften eines verteilten Systems erfüllt sein können [4].

- Consistency: Zu jedem Zeitpunkt sehen alle Knoten denselben Datenbestand.
- Availability: Knoten können Datenbestände jederzeit schreiben und lesen.
- Partition tolerance: Das System arbeitet trotz einer Zerteilung in Teilsysteme weiter.

Während relationale Datenbanksysteme stets auf die Wahrung der Konsistenz bestehen und dies zur Beeinträchtigung der Performance und Skalierbarkeit nach sich zieht, verfolgen viele NoSQL-Systeme den im Abschnitt 2 aufgefassten

Ansatz, strenge Konsistenzforderungen zugunsten der Performance aufzugeben.

Ausfallsicherheit: Ein Großteil der NoSQL-Systeme bietet hervorragende Replikations- und Failovertechniken, um Ausfälle von Knoten innerhalb einer Shared-Nothing-Architektur zu kompensieren, indem das System vor Datenverlust geschützt und der laufende Betrieb minimal beeinflusst wird.

Schemaflexibilität: NoSQL-Systeme verdeutlichen, dass neben dem relationalen Datenbankmodell andere Datenmodelle existieren, die Daten gemäß ihrer Eigenschaften adäquat speichern, ohne sie in ein fixes Datenbankschema zu fügen. Für einfache, schemafreie Daten bieten Key-Value-Stores die Möglichkeit, mehrattribute Objekte anhand eines eindeutigen Schlüssels zu speichern und abzufragen. Dokumenten-basierte Systeme erlauben zudem das Speichern komplexerer Inhalte wie verschachtelte Daten und bieten durch leistungsfähigere Abfragesprache beispielsweise das Suchen auf beliebigen Attributen. Wide-Column-Stores vereinen hingegen Vorzüge des Relationenmodells mit Funktionalitäten wie flexiblen Schemata und Versionierung. Diese Datenmodelle werden beispielweise durch die Graphen-DBS ergänzt. Die Komplexität des Datenmodells spiegelt sich meist in der zur Verfügung gestellten Programmierschnittstelle bzw. Abfragesprache wieder, es existieren für die Datenmodelle kaum standardisierte Notationen und standardisierte, deskriptive Sprachen. Entsprechend ihrer Zielstellung bieten sie häufig auf REST basierende Schnittstellen.[15]

Kosten: Das Gros der Systeme wird als Open Source und mit wenigen Nutzungseinschränkungen zur Verfügung gestellt. Die Installation und Verwendung der Systeme ist meist unkompliziert. Zudem ist häufig ein Betrieb auf günstigen Commodity Servern möglich, da Einschränkungen bezüglich der zu verwendenen Hardware kaum vorhanden sind und geläufige Betriebssysteme unterstützt werden.

4.2 Bewertung

NoSQL-Datenspeicher sind hervorragend geeignet, um kostengünstig skalierbare und hochverfügbare Datenspeicherung und -verarbeitung in einem begrenzten Anwendungsfall bereitzustellen. Aus Sicht dieser Systeme sind die größten Hürden beim Einsatz relationaler Systeme für Web-Applikationen nicht das relationale Datenbankmodell, ACID oder gar SQL. So zeigen aktuelle Entwicklungen im Bereich relationaler Datenbanksysteme wie VoltDB[2] oder HyPer[3], dass diese Merkmale eine lineare Skalierbarkeit nicht zwingendermaßen ausschließen. Im Zentrum der Beanstandungen stehen hingegen die Tatsachen, dass keine bewährten parallelen und hochverfügbaren Open Source RDBMS existieren und die *Implementierung* bewährter relationaler DBMS häufig keine hinreichende Skalierbarkeit zulässt.

Die Spezialisierung der NoSQL-Systeme auf eine wenige Anwendungsgebiet verwehrt in vielen Fällen den Einsatz bei sich ändernden Anforderungen, wie beispielsweise dem Wunsch komplexer Abfragen auf Daten bei simplen Datenmodellen. Bei der Nutzung eines relationalen DBMS wären hierbei kaum Änderungen vonnöten, während ein NoSQL-System angepasst oder gar ausgetauscht werden muss. Ein Austausch gestaltet sich zudem schwierig, da es den Systemen an standardisierten Notationen und Schnittstellen mangelt. Zudem werden statt deskriptiven Sprachen je nach Datenmodell meist Low-Level-Abfragesprachen verschiedenen Komplexitäts- und Mächtigkeitsgrades genutzt, was aus Sicht des Programmierers ein Fortschritt, aus Sicht eines Datenbänklers aber durchaus als Rückschritt gesehen werden kann [2]. Insbesondere der Verzicht einiger NoSQL-Systeme auf die Gewährleistung der ACID-Eigenschaften führt dazu, dass ein Großteil von Unternehmen den Einsatz dieser Systeme ausschließen wird.

[2]http://voltdb.com/

[3]http://www3.in.tum.de/research/projects/HyPer/

5. VERBINDUNG BEIDER WELTEN

Relationale Datenbanksysteme bieten aufgrund jahrelanger Forschung und Entwicklung u.a. eine enorme Verbreitung und Bekanntheit, ein ausgereiftes mathematisches Fundament, die Datenbanksprache SQL und nicht zuletzt zugesicherte Transaktionseigenschaften durch ACID. Auf der anderen Seite existieren NoSQL-Systeme, deren Verbreitung sich in der Regel auf wenige Web-Anwendungen beschränkt. Charakterisiert durch die in Abschnitt 4 zusammengefasste Eigenschaften sowie die verwendeten Konzepte, weisen sie zum Teil Zielstellungen auf, die sich deutlich von der Zielstellung klassischer relationaler Datenbanksysteme unterscheidet.

Eine Verbindung von Konzepten und Implementierungen relationaler Datenbanksysteme und NoSQL Data Stores kann dazu genutzt werden, die Vorteile beider Welten zu vereinen. Im Folgenden werden mögliche Ansätze zur Vereinigung anhand stellvertrender Beispiele vorgestellt.

5.1 Erweiterungen von NoSQL-Produkten

NoSQL-Systeme wurden in der Regel für ein spezielles Anwendungsgebiet entwickelt. Durch eine Erweiterung der Systeme kann ihr Einsatzbereich vergrößert werden, wodurch sie die Aufmerksamkeit von mehr Unternehmen auf sich ziehen können. Somit wird neben der Erweiterung der Funktionalität auch die Bekanntheit des Produkts gesteigert. Ein Beispiel für diesen Ansatz ist das Produkt Hive[17], welches das NoSQL-System Hadoop um die deskriptive, SQL-ähnliche Sprache HiveQL erweitert und Schnittstellen in Form eines CLIs, einer Web-Gui und JDBC/ODBC bietet. Zudem schafft es durch komplexe Analysen und das Absetzen von Ad-hoc-Abfragen die Voraussetzung, Hadoop für Data Warehousing zu nutzen.

5.2 Hybridsystem

Hybride Systeme wie HadoopDB[1] führen zu einem Kompromiss zwischen zwei unterschiedlichen Produktwelten und erschaffen dabei Produkte mit neuen Funktionalitäten. Das Open-Source-Produkt HadoopDB kombiniert MapReduce in Form der Implementierung Hadoops sowie Hive und PostgreSQL, wobei das System bereits mit anderen Datenbanksystemen getestet wurde. HadoopDB kann sowohl SQL-Anfragen als auch MapReduce-Jobs entgegennehmen und bietet den Zugriff auf Hadoops verteiltes Dateisystem HDFS oder alternativ auf ein Datenbanksystem wie PostgreSQL an. In der Folge sind Nutzer durch die Verwendung von HadoopDB in der Lage, mittels SQL auf ein Shared-Nothing-DBMS zuzugreifen.

5.3 Anpassung von RDBMS

Der Abschnitt 4 verdeutlicht, dass die bewährten fundamentalen Konzepte hinter dem relationalen Datenbankmodell mit Anforderungen wie enormer Skalierbarkeit vereinbar sind, es hierzu jedoch einer Anpassung der von System R abstammenden Architektur bedarf. Die Implementierungen

von DBMS müssen sich durch geeignete Konfigurationsmöglichkeiten weit mehr als bisher an verschiedene Einsatzzwecke anpassen lassen. Realisiert werden kann dies beispielsweise durch die Ausnutzung der Austauschbarkeit von Komponenten in modularen DBMS-Architekturen wie [7] oder die Implementierung von adaptierbaren DBMS-Komponenten.

Ein möglicher Ansatzpunkt dieses Konzepts könnte das Anbieten einer wahlweisen Speicherung auf langsamen, persistenten Festspeichern oder im schnellen, flüchtigen Arbeitsspeicher oder einer kombinierten Lösung sein, was bereits in einigen Systemen wie dem in Abschnitt 3.3 beschriebenen MySQL Cluster möglich ist. Hierdurch bieten sich entsprechend der Charakteristika und des Umfang der zu speichernden Daten sowie den Zugriffseigenschaften verschiedene Einsatzmöglichkeiten. Orthogonal kann wahlweise eine spalten- oder zeilenbasierte Speicherung angeboten werden, um sowohl im OLTP- als auch im OLAP-Bereich überzeugende Leistungskennzahlen zu erzielen. Für die Implementierung bieten einige Systeme bereits verschiedene Storage-Engines innerhalb eines Systems.

Auch die Transaktionsverwaltung relationaler Datenbanksysteme bietet sich bezüglich einer Erweiterung an, indem neben den harten Anforderungen von ACID und den heute wählbaren Isolationsszenarien weitere Transaktionskonzepte mit schwächeren Anforderungen integriert werden und Administratoren die Wahl des Transaktionskonzepts überlassen wird. Aus Sicht des in Abschnitt 4 angesprochenen CAP-Theorems könnten je nach Konfiguration des Systems verschiedene CAP-Eigenschaften erfüllt werden und somit das Datenbanksystem an verschiedene Einsatzzwecke angepasst werden. Die Realisierung kann beispielsweise über ein autonomes Modul zur Transaktionsverwaltung in einer modularen DBMS-Architektur gemäß [8] erfolgen.

6. ZUSAMMENFASSUNG

In diesem Beitrag wurde die Notwendigkeit adaptierbarer flexibler RDBMS-Implementierungen aufgezeigt. Als Grundlage diente der Vergleich von Oracle RAC, IBM DB2 PureScale und MySQL Cluster. Er verdeutlichte, dass die Hersteller zum Erreichen des Ziels eines horizontal skalierbaren und hochverfügbaren Clusterdatenbanksystems gemäß verschiedener Implementierungsansätze verfahren. Während Oracle RAC und IBM DB2 PureScale sich durch gute Lastbalancierung, effizientes Logging, Locking und Caching sowie einfache Migration von Anwendungen auf die Clustersysteme hervorheben, ist MySQL Cluster vor allem durch geringe Ansprüche bezüglich der verwendeten Hardware und unkomplizierte Fehlerbehandlung aufgrund der Shared-Nothing-Architektur gekennzeichnet.

NoSQL Data Stores stellen vermehrt eine Alternative zu RDBMS dar, die Systeme sind jedoch meist auf den Einsatz in wenigen Anwendungsgebieten limitiert. Zudem mangelt es ihnen an Standardisierung und vor allem die Low-Level-Abfragesprachen sind aus Sicht der Datenbankforschung als Rückschritt zu werten.

Durch die Verknüpfung bewährter Konzepte und Implementierungen der RDBMS mit Ansätzen der NoSQL-Bewegung können Vorteile beider Welten vereint werden. Die Erweiterung eines NoSQL-Systems führt nicht nur zu zusätzlichen Funktionalitäten, sondern steigert zudem die Bekanntheit und eröffnet neue Einsatzbereiche. Eine weitere Möglichkeit stellt eine Kombination von RDBMS- und NoSQL-Implementierungen in Form eines hybriden Systems dar, wovon bereits wenige Beispiele existieren. Als Mittel der Wahl zur Vereinigung von Konzepten relationaler Datenbanksysteme und NoSQL-Systeme zeichnen sich jedoch aus Sicht des Autors flexible RDBMS-Implementierungen ab, die sich gezielter als in aktuellen Systemen an verschiedene Einsatzzwecke anpassen lassen. Als mögliche Ansatzpunkte wurden die Implementierung verschiedener Storage-Engines und weiterer Transaktionskonzepte vorgeschlagen.

7. LITERATUR

[1] A. Abouzeid, K. Bajda-Pawlikowski, D. Abadi, A. Silberschatz, and A. Rasin. HadoopDB: an architectural hybrid of MapReduce and DBMS technologies for analytical workloads. In *VLDB '09*, pages 922–933. VLDB Endowment, 2009.

[2] D. J. DeWitt and M. Stonebraker. MapReduce: A major step backwards, 2008.

[3] D. Florescu and D. Kossmann. Rethinking cost and performance of database systems. *SIGMOD Rec.*, 38:43–48, June 2009.

[4] S. Gilbert and N. Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33:51–59, June 2002.

[5] T. Grebe. Gruppendynamik – Oracle Real Application Cluster vs. MySQL Cluster. *databasepro*, (6):46–63, 2010.

[6] IBM. Transparent Application Scaling with IBM DB2 pureScale. Technical report, IBM, 2009.

[7] F. Irmert, M. Daum, and K. Meyer-Wegener. A new approach to modular database systems. In *EDBT Workshop der SETMDM '08*, pages 40–44, New York, NY, USA, 2008. ACM.

[8] F. Irmert, C. P. Neumann, M. Daum, N. Pollner, and K. Meyer-Wegener. Technische Grundlagen für eine laufzeitadaptierbare Transaktionsverwaltung. In *BTW '09*, pages 227–236, Münster, Germany, 2009.

[9] A. Maslo. Unendliche Weiten – IBM DB2 pureScale für Power Systems. *databasepro*, (1):82–86, 2010.

[10] MySQL. Hochverfügbarkeitslösungen von MySQL – Ein Überblick über die Hochverfügbarkeitslösungen von MySQL. Technical report, MySQL AB, 2007.

[11] Oracle. MySQL Cluster 7.0 & 7.1: Architektur und neue Funktionen. Technical report, Oracle, Inc., 2010.

[12] Oracle. Oracle Real Application Clusters Administration and Deployment Guide, 11g Release 2. Technical report, Oracle Corporation, 2010.

[13] M. Stonebraker. The NoSQL Discussion has nothing to do with SQL. Blog-Eintrag, 2010.

[14] M. Stonebraker, C. Bear, U. Çetintemel, M. Cherniack, T. Ge, N. Hachem, S. Harizopoulos, J. Lifter, J. Rogers, and S. Zdonik. One size fits all - Part 2: benchmarking results. In *In CIDR*, 2007.

[15] M. Stonebraker and R. Cattell. Ten Rules for Scalable Performance in „Simple Operation" Datastores. *Communications of the ACM*, 2010.

[16] M. Stonebraker, S. Madden, D. J. Abadi, S. Harizopoulos, N. Hachem, and P. Helland. The end of an architectural era (it's time for a complete rewrite). In *VLDB '07*, pages 1150–1160. VLDB Endowment, 2007.

[17] A. Thusoo. Hive - A Petabyte Scale Data Warehouse using Hadoop. Technical report, Facebook Inc., 2009.