Apache Mahout: Massendaten mit Java und Scala verarbeiten ▸ 70

Deutschland €9,80 Österreich € 10,80 Schweiz sFr 19,50 Luxemburg € 11,15 6.2015

JAVA Mag

Java magazin

Java • Architekturen • Web • Agile

www.javamagazin.de

Einstieg in MVC
Neues Webframework in Java EE 8 ▸ 30

Microservices testen
Consumer-driven Contracts ▸ 58

Docker rockt Java
Container in der Amazon Cloud ▸ 64

INTERNET OF THINGS

MQTT: Standardprotokoll für IoT aus Java-Sicht ▸ 36

Tutorial: Bindings mit Eclipse SmartHome leicht gemacht ▸ 48

Binäre Formate: Jenseits von XML, JSON und Co. ▸ 41

4 194586 709801 06

java.util.Optional<T>: Funktionaler Programmierstil in JDK 8 ▸ 12

Prozedurales Java: Es muss nicht immer OO sein ▸ 20

Android: Globale Sprachen unterstützen ▸ 86

Teil 2: Nach SQL kommt NoSQL

Eine kleine Reise durch NoSQL

Die Welt der Speichertechnologien befindet sich momentan im Umbruch. NoSQL-Datenbanken adressieren Probleme, für die relationale Datenbanken nicht geeignet sind. Wir werden hierauf dezidierter eingehen. Die Lösungsmechanismen der NoSQL-Datenbanken kommen allerdings nicht ohne Nebeneffekte daher. Wir wollen Parallelen zu den Entwicklungen ab den 1970er-Jahren aufzeigen und versuchen zu motivieren, dass aus diesen Entwicklungen gelernt werden kann.

von Christian Mennerich und Joachim Arrasz

Relationale Systeme genießen heute nicht mehr den Ruf als Universallösungen, den sie lange Zeit innehatten. Mit der zunehmenden Verteilung von Daten im Web und den damit einhergehenden Anforderungen haben sie ihre Vormachtstellung verloren. NoSQL-Datenbanken dagegen haben messbare Anteile am Datenbankmarkt gewonnen [1]. Nachdem wir den ersten Teil unserer Artikelserie den relationalen Datenbanksystemen und ihrer Entstehung gewidmet haben, wollen wir in diesem Teil die NoSQL-Datenbanken beleuchten und in diesem Zusammenhang Missverständnisse und Wahlmöglichkeiten diskutieren.

Mit den heute weit verbreiteten NoSQL- bzw. Not-only-SQL-Datenbanken sollen Probleme adressiert werden, für die relationale Systeme ungeeignet erscheinen oder bei denen diese schlicht versagen. Es ist schwierig zu definieren, was unter NoSQL genau zu verstehen ist. Die deutschsprachige Definition nach dem NoSQL-Archiv [2] kann auch unter http://nosql-database.org/ nachgelesen werden [3]. Demnach weist eine NoSQL-Datenbank „meistens einige“ der folgenden Eigenschaften auf:

- Die verwendeten Datenmodelle und Schemata sind nicht relational.
- Restriktionen an die Schemata sind schwach oder gar nicht vorhanden.
- Die verwendeten Technologien sind Open Source.
- Ein Hauptaugenmerk der Datenbank liegt von vornherein auf guter horizontaler Skalierbarkeit.
- Das System bringt Mechanismen zur einfachen Replikation der gespeicherten Daten mit.
- Als Konsistenzmodelle können BASE und Eventual Consistency Anwendung finden.
- Die APIs für Anfragen an das System sind einfach gehalten und unterstützen oft keine komplexen Anfragen.

Die Vielfalt an Interpretationen, die eine solch schwache Definition zulässt, kann schnell zu Missverständnissen führen und Neu- und Quereinsteigern den Zugang erschweren. Unter dem Begriff NoSQL sind eine Vielzahl von Systemen vereint; das NoSQL-Archiv [2] listet derzeit 150. An vielen Stellen wird mit althergebrachten Paradigmen gebrochen, sodass die neuen Systeme oft erst im Kontext ihrer Entwicklung gut zu verstehen sind. Die unterschiedliche Deutung von Begriffen, auch aus der relationalen Welt, verstärkt die Verwirrung oftmals noch.

Verteilte Systeme und Konsistenz

Mit der Verteilung der Daten im Web gehen neue Herausforderungen einher. Auf dem Symposium PODC 2000 präsentierte Brewer die folgende Vermutung [4]: „In keinem verteilten Informationssystem können zur gleichen Zeit alle Rechnerknoten dieselbe Sicht auf alle Daten haben (Consistency, C), alle Anfragen in akzeptabler Zeit beantwortet (Availability, A) und Ausfälle

Artikelserie

Teil 1: Relationale Datenbanksysteme
Teil 2: NoSQL-Datenbanken
Teil 3: NewSQL und Ausblick

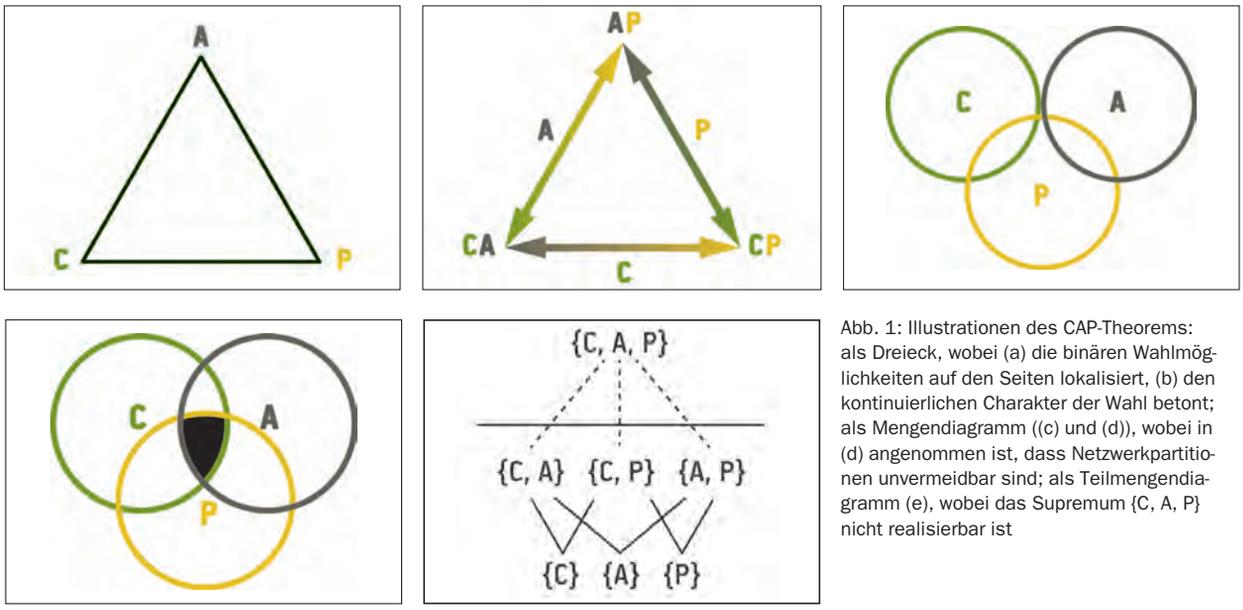

Abb. 1: Illustrationen des CAP-Theorems: als Dreieck, wobei (a) die binären Wahlmöglichkeiten auf den Seiten lokalisiert, (b) den kontinuierlichen Charakter der Wahl betont; als Mengendiagramm ((c) und (d)), wobei in (d) angenommen ist, dass Netzwerkpartitionen unvermeidbar sind; als Teilmengendiagramm (e), wobei das Supremum {C, A, P} nicht realisierbar ist

von Teilen des Netzwerks toleriert werden (Partition Tolerance, P)."

Diese Vermutung ist heute als CAP-Theorem bekannt und wurde 2002 von Gilbert und Lynch innerhalb eines formalen Netzwerkmodells streng bewiesen [5]. Akzeptiert man dieses Modell, so ist CAP eine „binäre Entscheidung": Ein verteiltes Informationssystem ist entweder

- konsistent und ausfallsicher (CA),
- ausfallsicher und partitionstolerant (AP) oder
- konsistent und partitionstolerant (CP).

Wichtig zu verstehen ist, dass das gleichzeitige Gelten der drei Eigenschaften (C, A und P) ausgeschlossen ist. Reale Systeme bieten Spielraum für das Feintuning der drei Größen. Die Wahl kann von daher eher als kontinuierlich angesehen werden, wie auch Brewer eingeräumt hat [6]. Die Toleranz gegen Netzwerkausfälle (P), welche als unvermeidlich gelten, wird häufig zum wichtigsten Parameter erklärt [7]. Die Auswahl findet dann zwischen Verfügbarkeit und Konsistenz statt. **Abbildung 1** zeigt unterschiedliche Arten der Darstellung des CAP-Theorems.

Schwächen relationaler Datenbanken sind ein Antrieb zur Entwicklung von NoSQL-Datenbanken (vgl. den ersten Teil dieser Serie). Die Unzulänglichkeit relationaler Systeme ist 2005 von Stonebraker, einem der Pioniere der relationalen Welt und Çetintemel in einem Fachbeitrag eindrucksvoll demonstriert worden [10]. Sie zeigen am Beispiel einer Streamverarbeitungsanwendung, wie ein für seine Aufgabe optimiertes System ein relationales um ganze Größenordnungen abhängen kann.

Dabei nutzen sie die Tatsache aus, dass sowohl die Anforderungen, als auch die Domäne der Anwendung genau bekannt sind. Die Anwendung ist tolerant gegenüber dem Verlust von (wenigen) Datensätzen. Die Anforderungen an die Datenkonsistenz ist abgeschwächt. So erreichen sie Datendurchsätze, die mit einem relationalen System und dem Overhead, den dieses mit sich bringt, schlichtweg unmöglich sind. In ihrer Arbeit erklären Stonebraker und Çetintemel die Aufrechterhaltung der Universalität relationaler Datenbanksysteme zur reinen Marketingillusion und besiegeln damit das Ende einer Ära.

In verteilten Systemen gelten eigene Regeln. Das bietet Spielraum für alternative Konsistenzmodelle. Weit verbreitet in der NoSQL-Welt ist BASE (Basically Available, Soft State, Eventually Consistent). BASE zielt auf die Verfügbarkeit eines Systems ab. Der Systemzustand ist weniger strikt und das Antwortverhalten weniger vorhersagbar. Gleichzeitig sind die Konsistenzanforderungen gelockert.

Als Beispiel schauen wir wieder auf unseren bereits aus dem ersten Teil bekannten Webshop. Wir nehmen an, dass die dort zum Verkauf angebotenen Artikel in einer verteilten Datenbank gespeichert sind. Wird ein neuer Artikel eingestellt, so kann es sein, dass einige Benutzer diesen früher sehen können als andere. Dies hängt davon ab, durch welchen Knoten eine Anfrage an das System beantwortet wird. Die Eventual Consistency stellt jedoch sicher, dass nach und nach alle Knoten mit den nötigen Informationen versorgt werden und schließlich der Artikel für jeden Benutzer sichtbar wird – vorausgesetzt, er wird nicht vorzeitig geändert oder gelöscht.

Info

Ein verteiltes System ist, einfach gesprochen, eine Menge verteilter Rechner, die sich nach außen hin wie ein System präsentiert. Wikipedia gibt einen guten Einstieg [8]. Die folgende Definition von verteilten Systemen wird Lamport zugeschrieben [9]: „A distributed system is one in which the failure of a computer that you didn't even know existed can render your own computer unusable."

Klassifikation von NoSQL-Datenbanken

Die Vielfalt der verfügbaren NoSQL-Datenbanken ist groß. Die Systeme werden oftmals in vier Kategorien eingeordnet:

- Key-Value Stores
- Wide Column Stores
- Dokumentenorientierte Datenbanken
- Graphdatenbanken

Key-Value Stores haben ein einfaches Datenmodell: Schlüssel werden auf Werte abgebildet. Die Schlüssel sind dabei zumeist einfache Zeichenketten. Abhängig vom Store können die Werte, auf die die Schlüssel zeigen, komplexe strukturierte Objekte sein. Die Anwendbarkeit reicht von der Sessionverwaltung bis hin zu Warenkörben in Webshops. Bekannte Vertreter sind hier Redis oder Riak.

Wide Column Stores bieten ein komplexes Datenmodell, das geeignet ist, dynamische Spalten abzubilden. Terminologisch orientieren sich diese Datenbanken oft stark an Begriffen aus der relationalen Welt. Im Kern ist das Datenmodell ein mehrdimensionales assoziatives Array. Wichtige Beispiele sind Cassandra oder HBase, eine Implementierung von Google BigTable.

Info

Konsistenzlevel in verteilten Systemen beziehen sich auf die Sicht, die die Rechnerknoten des Systems auf die Daten haben (das C in CAP). Das stärkste Level ist die lineare Konsistenz. Es erfordert, dass jederzeit auf alle Anfragen mit dem neuesten Datum geantwortet wird. Das andere Ende des Spektrums bildet die schwache Konsistenz.
Die in den NoSQL-Systemen häufig anzutreffende Eventual Consistency liegt dazwischen: Das System kann zeitweise dieselbe Anfrage unterschiedlich beantworten. Allerdings ist garantiert, dass irgendwann alle Rechnerknoten auch die gleiche Antwort liefern werden. Voraussetzung dafür ist, dass hinreichend lange keine Updates auf die angefragte Information erfolgen. Es gibt verschiedene Arten der Eventual Consistency [10]:

- Causal Consistency: Wenn ein Prozess A ein Datum aktualisiert und dieses einem Prozess B mitteilt, so wird B nur noch den aktualisierten Wert verwenden.
- Read Your Own Writes: Hat ein Prozess einen Wert aktualisiert, so wird er auch nur noch diesen neuen Wert lesen, keine ältere Version mehr.
- Session Consistency: Bei wiederholtem Lesen eines aktualisierten Werts innerhalb einer Session wird Read Your Own Writes gewährleistet, nicht jedoch über die Session hinaus.
- Monotonic Read: Wenn ein Prozess eine Version eines Werts gelesen hat, so wird er in folgenden Leseoperationen niemals eine ältere Version sehen.
- Monotonic Write: Das verteilte System garantiert jedem Prozess die Serialisierung seiner Schreiboperationen.

In der Praxis sind die Level kombinierbar, für reale Systeme sind in der Regel Read Your Own Writes und Monotonic Read wünschenswerte Eigenschaften.

In dokumentenorientierten Datenbanken gespeicherte Daten kann man sich in der Regel als JSON-Dokumente vorstellen. Die wohl bekanntesten Vertreter dieser Gattung sind MongoDB und CouchDB. Auch Elasticsearch kann zu dieser Kategorie gezählt werden. Was einige Eigenschaften wie Transaktionen, Sicherheit und Verfügbarkeit angeht, ist hier jedoch Vorsicht geboten, obwohl Elasticsearch in diesen Bereichen zuletzt stark zugelegt hat [11].

Graphdatenbanken haben eine gewisse Sonderstellung in der NoSQL-Welt. Das Datenmodell ist ein Graph, bestehend aus durch Kanten verbundenen Knoten, die jeweils Attribute haben können. Neo4j, Titan (als Aufsatz auf andere Datenbanken) oder HyperGraphDB sind Vertreter dieser Kategorie.

Die Grenzen zwischen den Kategorien sind oft fließend. Redis beispielsweise bietet eine Vielzahl von Möglichkeiten, die Werte hinter einem Schlüssel zu strukturieren und zu manipulieren. Auch werden Multimodelldatenbanken wie ArangoDB immer populärer. Andere NoSQL-Datenbanken bringen Abstraktionsschichten mit, die die Zugehörigkeit zu verschiedenen Kategorien nahelegen. Beispiele sind das oben erwähnte Titan, oder der Key-Value Store FoundationDB.

Eine NoSQL-Datenbank gehört aber nicht nur einer dieser vier Kategorien an, sondern lässt sich zusätzlich gemäß des CAP-Theorems klassifizieren. Demnach sind Systeme darauf ausgelegt, primär CA, CP oder AP zu erfüllen. Dies führt zu einer hohen Komplexität mit vielen Auswahlmöglichkeiten, die es bei der Wahl einer Datenbank zu beachten gilt.

NoSQL-Missverständnisse

Zu den ohnehin komplexen Auswahlmöglichkeiten kommen Unwissenheit über die Eigenheiten verteilter Systeme über die Herkunft und Hintergründe der NoSQL-Ansätze sowie ungünstige natürlichsprachliche Gegebenheiten hinzu. Dadurch wird eine fundierte Auswahl einer NoSQL-Datenbank erschwert. Zudem halten sich viele Missverständnisse hartnäckig.

Dies beginnt bereits bei dem Begriff der „Relation“, der sich nicht auf die Beziehungen zwischen Daten in Tabellen, sondern auf eine Relation im Sinne der mathematischen Mengentheorie bezieht (vgl. auch den ersten Teil dieser Serie). Eine Relation ist schlicht eine Menge von Tupeln. Referenzielle Integrität taucht hier als Konzept noch gar nicht auf. Diese wird für das Modell extern sichergestellt.

Unglücklich ist auch die Wahl des Begriffs „Consistency“ für das C in den Akronymen ACID und CAP. Während sich das C in ACID auf Konsistenz im Sinne der referenziellen Integrität bezieht, steht es in CAP für die Konsistenz in verteilten Systemen. Die beiden Konsistenzbegriffe beziehen sich also auf grundlegend verschiedene Konzepte. Plakativ ließe sich das als $C_{ACID} \neq C_{CAP}$ darstellen.

Fast schon tragisch ist die Ähnlichkeit des englischen Worts „eventually“, das „schließlich“ bedeutet, zum

deutschen Wort „eventuell“. Ein BASE-System erreicht aber nicht „vielleicht einmal“ einen konsistenten Zustand, sondern „garantiert einmal.“

Genauso ist der Begriff der Versionierung nicht unproblematisch. Einige NoSQL-Datenbanken halten unterschiedliche Versionen eines Datensatzes vor. Dies geschieht zum Zwecke der Versionskontrolle, nicht zur Historisierung. Nicht mehr benötigte Versionen können prinzipiell jederzeit vom Datenbanksystem gelöscht werden. Um eine verlässliche Änderungsverfolgung der Daten muss sich der Anwendungsentwickler jedoch selbst kümmern.

Die Praxis: So wahr uns Codd helfe?

Mit Beginn der 1970er-Jahre wurden Applikationen mühsam mithilfe normalisierter relationaler Datenbankschemata von den lästigen Abhängigkeiten befreit, die der Datenbanktheoretiker und Turing-Award-Gewinner Edgar F. Codd (1923–2003) angeprangert hatte. Datenmodell und Datenhaltung werden vorbildlich getrennt. Modell und Anfragen sind vor Veränderungen der Datenhaltung auf dem Festplattensystem geschützt, wie wir im ersten Teil dieser Serie diskutiert haben. Mit NoSQL kehren nun viele der Abhängigkeiten wieder zurück. Hier sind in der Entwicklung von NoSQL Analogien zu Codds Zeit zu beobachten. Ein paar Beispiele sollen dies illustrieren.

Von HBase werden Daten gruppiert gehalten, um ähnliche Daten physikalisch nahe zueinander zu speichern. Die Sortierung erfolgt zunächst alphabetisch nach Spaltenschlüssel, dann nach Spaltenfamilie und Spaltenqualifikator gruppiert. Von jedem Datensatz können verschiedene Versionen gespeichert werden. Macht eine Anwendung implizit von dieser von der Datenbank gelieferten Sortierung Gebrauch, so ändert sich ihre Semantik, wenn die Datenbank ihre Sortierstrategie ändert oder der Store ausgetauscht wird.

Ein Beispiel einer interessanten, evolutionären Änderung liefert die Graphdatenbank Neo4j. In der Anfragesprache Cypher [12] gab es bis zur Version 2.0 eine direkte Abhängigkeit der Anfragen von Indexen (vgl. Codds Kritik der Indexabhängigkeit). Eine Anfrage musste mit einer *START*-Klausel beginnen, die auf der rechten Seite einen Indexnamen enthielt. So auch im Beispiel des Index *Movies*:

```
START movie = node:Movies('title=Matrix')
```

Die *START*-Klausel gibt es nicht mehr, die Sprache wurde von ihrer Indexabhängigkeit – Codd sei Dank – befreit. Nach den Namen der Schauspieler aus Matrix kann heute ohne *START* und dank Knotenlabeln (im folgenden Beispiel *Movie* und *Actor*) dennoch sehr einfach und effizient gefragt werden:

```
MATCH (matrix:Movie {title='Matrix'}) -[:ACTED_IN]-> (actor:Actor)
RETURN actor.name;
```

Ebenfalls eine Abhängigkeit von Indexen weist Apaches Cassandra auf. In der an SQL angelehnten Anfragesprache CQL [13] können WHERE-Klauseln nicht ausgewertet werden, wenn auf den dort referenzierten Attributen kein Index existiert. Die Frage nach den Namen aller Marketingmitarbeiter

```
SELECT name FROM employees WHERE department = 'Marketing'
```

kann also nur beantwortet werden, wenn ein Index für die Abteilungen existiert. Anfragen sind allerdings frei von Referenzen auf die Indexe, was sie robust gegen Änderungen der Indexstrukturen macht.

Derartige Probleme gibt es in SQL erst einmal nicht. Anfragen können formuliert und beantwortet werden, ohne dass Indexe vorhanden oder die Art der Datenrepräsentation bekannt sein müssen. Indexe können unabhängig erstellt, gepflegt und auch gelöscht werden.

Datenhaltung und -repräsentation sind in NoSQL-Systemen wieder stark gekoppelt, die Kenntnis der Datenablage wird zur Steigerung der Anfrageperformanz genutzt. In Systemen wie Cassandra wird das Datenmodell von vornherein auf Anwendungsfälle hin optimiert entwickelt. Die universelle Flexibilität, die das relationale Datenmodell bietet, gibt es hier nicht. Es ist möglich, dass die Beantwortung gewisser Anfragen im Datenmodell ausgeschlossen ist, da beispielsweise *Join Operation* per Design nicht vorgesehen ist. Die Datenbank forciert die denormalisierte Datenhaltung. Daher muss gegebenenfalls für neue Anwendungsfälle nachmodelliert werden.

Auf der Suche nach der richtigen Datenbank

Im besten Falle diktiert das Wissen über die Anwendungsanforderungen auch die Auswahl der Speichertechnologie. NoSQL-Datenbanken sind oft weniger universell einsetzbar als relationale Datenbanken, wollen dies aber auch nicht sein. Wenn aber die Architektur des Systems von vorne herein verteilt ist und die Belange der einzelnen Anwendungen gut erfasst werden, kann man sich dezidiert über die einzelnen Teile Gedanken machen.

Dazu gehören natürlich die Anforderungen Datenkonsistenz und Verfügbarkeit im Sinne des CAP-

Info

Viele NoSQL-Datenbanken bringen eigene Anfragesprachen mit. Häufig bieten die Systeme nur ein simples Anfrage-API an, das das Einfügen, Aktualisieren und Auslesen von Daten ermöglicht. Die Aussagekraft einer NoSQL-Anfragesprache wie Cypher mit der der relationalen Algebra zu vergleichen, erscheint schwierig. Cypher beruht nicht unmittelbar ersichtlich auf einer abgeschlossenen Algebra (beispielsweise einer Pfadalgebra, wie sie von Rodriguez und Neubauer vorgeschlagen wird [14]), sondern erzeugt aus den Knoten, Kanten und Wegen eines Neo4j-Graphen eine tabellarische Ausgabe der angefragten Informationen. Für andere Graphanfragesprachen wie SPARQL sind Analysen und Äquivalenzergebnisse zur relationalen Algebra erhältlich [15].

Theorems: Es ist zu entscheiden, ob vorübergehende Inkonsistenzen tolerabel sind, oder ob es verschmerzbar ist, wenn das System (in seltenen Fällen) zeitweise nicht erreichbar ist. Im Bereich statistischer Auswertungen ist die Datenkonsistenz oft nicht so strikt. Wenn es um Abrechnungen geht, sieht es in der Regel anders aus. Hier ist es wichtiger, korrekt abzurechnen und Kosten sowie Verlusten durch fehlende oder falsche Abrechnungen vorzubeugen.

Schauen wir auf ein Beispiel und nehmen an, wir schreiben eine Software für ein Unternehmen, das Handhelds und Smartphones vertreibt. Es sollen ausreichend viele Daten über das Nutzerverhalten gesammelt werden, weshalb jedes Gerät sekündlich eine Nachricht schickt. Diese immensen Datenmengen werden in einem Wide Column Store gesammelt, beispielsweise einer Cassandra (AP). Da nicht jedes Gerät immer Netzzugang hat, gehen Daten verloren. Dennoch können mit diesen gesammelten Daten Statistiken im Sinne eines Data Warehouse erstellt werden. Benutzerstammdaten sind selten veränderlich, einander ähnlich, aber nicht immer gleich. Einige Nutzer haben mehrere Geräte, Adressen, Telefonnummern etc. Diese Daten werden in einem Dokumenten-Store, wie beispielsweise MongoDB (CP) gespeichert. Hier ist ausreichend Funktionalität geboten, um einfache Benutzerdatenanalysen erstellen zu können. Die Daten werden weiterhin nach Standorten geshardet gespeichert und in verschiedene Rechenzentren repliziert. Der Store unterstützt hierbei nativ.

Für Abrechnungen wird ein relationales System verwendet (CA). Im angeschlossenen Webshop werden Benutzereinkäufe in einer Graphdatenbank wie Neo4j (CA) gespeichert, um einfache Kaufempfehlungen an Benutzer geben zu können (**Abb. 2**).

Es ist also durchaus möglich, Aufgaben dezidiert zu verteilen und polyglott über Services zu verbinden. Kunden, die behaupten, sie hätten in einem Abrechnungszeitraum ihr Gerät nicht verwendet, kann also widersprochen werden, auch wenn es nicht alle Daten eines Geräts in die Cassandra geschafft haben. Gelegentlicher Datenverlust ist hier tolerabel, die Anforderungen bestimmen die Wahl der Datenhaltung.

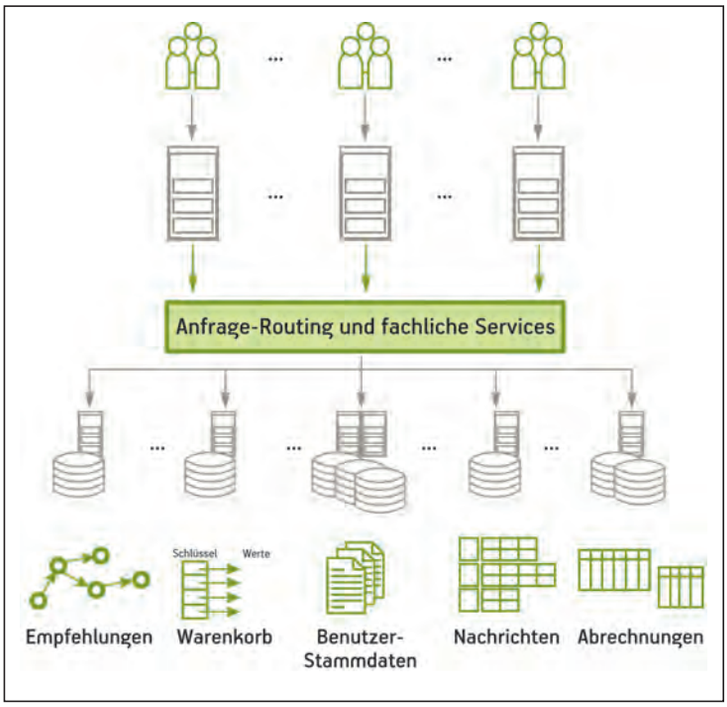

Abb. 2: Verschiedene Services und dezidierte anwendungsspezifische Speichertechnologien

Programmieren gegen NoSQL-Datenbanken

Auch in der neuen NoSQL-Welt erhält der Java-Entwickler gute Unterstützung von unterschiedlichen Frameworks. Analog zu den objektrelationalen Mappern gibt es Objekt-Mapper zu anderen Datenmodellen, die oft wesentlich natürlicher in der Bedienung sind als objektrelationale Mapper. Der Impedance Mismatch der relationalen Welt entfällt, da in der Modellierung nicht künstlich normalisiert wird und die Datenhaltung von Beginn an redundanter sein darf.

Das Spring-Data-Projekt [16] unterstützt eine Vielzahl von NoSQL-Datenbanken mit ähnlichen Mechanismen, wie sie sich in relationalen Datenbanken etabliert haben. So ist es mit Spring-Templates für MongoDB, Redis, Neo4j und nahezu alle weiteren wichtigen Datenbanken wie gewohnt möglich, transparent zu programmieren. Für einige der erwähnten Datenbanken gibt es bereits weitergehende Unterstützung in Form eines Entwicklungsmodells, das sehr vergleichbar zu Spring Data JPA ist. Insbesondere für Neo4j [17] ist diese sehr weit fortgeschritten (Listing 1).

Der programmatische Umgang mit den oben genannten unterschiedlichen Konsistenzleveln bleibt allerdings erhalten und ist grundlegend verschieden vom Programmieren mit ACID-Transaktionen. Die Sicherheiten, welche diese bieten, sind nicht mehr vorhanden. Auf inkonsistente (Zwischen-)Zustände muss nun der Entwickler in der Anwendung reagieren.

Listing 1

```
interface MovieRepository extends GraphRepository<Movie> {
  // developer provided statement for user ratings of movies
  @Query("MATCH (movie)<-[rating:RATED]-(user)
  RETURN rating")
  Iterable<Rating> getRatings(Movie movie);
  // autogenerated cypher statement
  Iterable<Person> findByActorsMoviesActorName(String name);
}

vs.

Neo4jOperations neo = new Neo4jTemplate(graphDatabaseService);
neo.query("MATCH ({person}) <-[:WORKS_WITH]-(other) RETURN other.name",
map("person", thomas)).to(String.class).single());
```

Eventual Consistency als Programmiermodell fühlt sich grundlegend anders an. Insbesondere gegen Systeme, die keine Monotonic-Write- oder Read-Your-Own-Write-Konsistenz gewährleisten, ist das Programmieren sehr ungewohnt. Der Entwickler hat sehr viel mehr Verantwortung für die fachliche Korrektheit der gespeicherten Daten. Er muss wissen, was er wie ablegt. Dies bürdet ihm auch mehr Dokumentationsarbeit auf, damit Wartung und korrekte Weiterentwicklung im angedachten Modell gewährleistet sind.

Im Umkehrschluss bietet es aber deutlich mehr Flexibilität in der Entwicklung. Für noch nicht vorliegende Daten beispielsweise müssen keine Workarounds mehr erdacht werden. Entwickler können sie einfach zum Zeitpunkt der Einforderung hinzufügen. Die relationale Welt fordert dies durch Constraints (NOTNULL) ein.

Fazit

NoSQL ist noch jung und die Uneinigkeiten über ein gemeinsames Verständnis des Begriffs sind stark spürbar. Die schwache Definition von NoSQL und die daraus resultierenden unterschiedlichen Interpretationen führen immer wieder zu Verstimmungen und Diskussionen in der NoSQL-Gemeinschaft.

Der Verlauf der Entwicklungen in der NoSQL-Welt zeigt hier Parallelen zu den von Edgar F. Codd kritisierten Aspekten auf. Zwischenzustände, wie die oben diskutierte Indexabhängigkeit in Neo4j, sind nicht pauschal negativ zu bewerten. Sie zeigen aber, dass die Evolution der neuen Systeme nicht vor zeitweisen Rückschritten gefeit ist. Die Forderung nach Ehrlichkeit an die Anbieter von NoSQL-Datenbanken wird daher immer lauter. Denn nur, wenn alle Anbieter die Eigenschaften ihrer Systeme offenlegen, kann auch bewusst die richtige Wahl für eine Speichertechnologie getroffen werden. Oftmals ist es schwer, anhand der Informationen auf der Homepage einer NoSQL-Datenbank zu verstehen, für welche Anwendungsfälle diese wirklich gut geeignet ist und auf welche Eigenschaften im CAP-Sinne die Datenbank spezialisiert ist [18]. Gelegentlich liegt sogar die Vermutung nahe, dass ein Hersteller seine NoSQL-Datenbank zum neuen Universalsystem machen will.

Ebenso ist der Blick in die Vergangenheit wichtig, vielleicht sogar bis hin zu den Entstehungszeiten der relationalen Datenbanken. Vieles ist aus den Entwicklungen der Geschichte zu lernen. Eben dies kann helfen, Fehlentwicklungen und problematische Trends zu vermeiden. Nur wer sein System verstanden hat und Defizite ehrlich benennt, ermöglicht die fundierte Wahl einer Datenbank.

Im dritten und letzten Teil dieser kleinen Artikelserie werden wir einen Blick in die Zukunft werfen und aus aktuellen Trends Prognosen ableiten. Dazu legen wir das Augenmerk natürlich auf die so genannten NewSQL-Systeme. Diese setzen auf die Stärken altbekannter Systeme und kombinieren sie mit neuen Technologien. Aber auch in Bezug auf die Auswirkungen von zunehmend günstiger werdendem Speicher und immer höherer Rechenleistung auf die Art der Datenspeicherung werden wir uns an einer Einschätzung versuchen.

Christian Mennerich hat Diplom-Informatik studiert, einer seiner Schwerpunkte lag auf Theorie und Implementation von Datenbanksystemen. Er arbeitet als Entwickler bei der synyx GmbH & Co. KG in Karlsruhe, wo er sich unter anderem mit NoSQL beschäftigt.

@cmennerich

Joachim Arrasz ist als Software- und Systemarchitekt in Karlsruhe bei der synyx GmbH & Co. KG als Leiter der Code Clinic tätig. Darüber hinaus twittert und bloggt er gerne.

@arrasz **http://blog.synyx.de/**

Links & Literatur

[1] http://www.techrepublic.com/article/nosql-databases-eat-into-the-relational-database-market

[2] http://nosql-database.org/

[3] Edlich, Stefan; Friedland, Achim; Hampe, Jens et al.: „NoSQL: Einstieg in die Welt nichtrelationaler Web 2.0 Datenbanken“, Carl Hanser Verlag, 2011

[4] Brewer, Eric: „Towards Robust Distributed Systems“, Keynote at PODC 2000

[5] Gilbert, Seth; Lynch, Nancy: „Brewer’s conjecture and the feasibility of consistent, available, partition-tolerant web services“, ACM SIGACT News, Vol. 33:2, 2002

[6] Brewer, Eric: „CAP Twelve Years Later: How the “Rules” Have Changed“, Computer, IEEE Computer Society, 2012

[7] Vogels, Werner: „Eventual Consistent“, Communications of the ACM, Vol. 52:1, 2009

[8] http://de.wikipedia.org/wiki/Verteiltes_System

[9] http://research.microsoft.com/en-us/um/people/lamport/pubs/distributed-system.txt

[10] Stonebraker, Michael; Çetintemel, Uğur: „One Size Fits All: An Idea Whose Time Has Come and Gone“, Proceedings of the 21st ICDE, 2005

[11] http://www.quora.com/Why-should-I-NOT-use-ElasticSearch-as-my-primary-datastore

[12] http://neo4j.com/docs/stable/cypher-query-lang.html

[13] https://cassandra.apache.org/doc/cql3/CQL.html

[14] Rodriguez, Marko A.; Neubauer, Peter: „A Path Algebra for Multi-Relational Graphs“, CoRR, 2010

[15] Angles, Renzo; Gutierrez, Claudio: „The Expressive Power of SPARQL“, ISWC 2008

[16] http://projects.spring.io/spring-data/

[17] http://projects.spring.io/spring-data-neo4j/

[18] http://blog.foundationdb.com/on-lowered-expectations-transactions-scaling-and-honesty