NoSQL, NewSQL und Cloud-native Datenbanken

Andreas Buckenhofer, Daimler TSS

Die ersten NoSQL-Datenbanken sind um das Jahr 2000 entstanden. Unternehmen wie Google, Amazon oder Twitter entwickelten eigene Datenbanken für ihre spezifischen Anforderungen. Im Laufe der Zeit stehen viele dieser Datenbanken als Open Source zur Verfügung.

In den 2000er-Jahren entstanden eine Vielzahl von Datenbanken, die als NoSQL zusammengefasst sind. Sie zeichnen sich dadurch aus, dass die Datenspeicherung verteilt erfolgt, Replikation zum Einsatz kommt, hohe Verfügbarkeit besteht und die Daten nicht in relationalen Tabellen gespeichert sind. Durch den Verzicht auf SQL soll der Impedance Mismatch vermieden werden, der auftritt, wenn man Objekte einer objektorientierten Programmiersprache in einer relationalen Datenbank speichert.

NoSQL-Datenbank-Kategorien

NoSQL basiert auf speziellen Datenmodellen, die im Gegensatz zum relationalen Datenbankmodell nicht mehr universell einsetzbar sind, sondern einen bestimmten Anwendungszweck erfüllen. Es wird zwischen vier verschiedenen NoSQL-Datenbank-Kategorien unterschieden:

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

Key-Value Stores bieten ein leicht verständliches Datenmodell. Daten lassen sich einfach verteilen und somit hohe Skalierbarkeit erreichen. Key-Value Stores werden häufig als Caches eingesetzt, entweder als eigenständige Datenbank oder als Caching-Layer für bestimmte Daten zwischen der Anwendung und einer relationalen Datenbank. *Abbildung 1* zeigt das Datenmodell.

Key-Value Stores bestehen aus einem eindeutigen Zugriffsschlüssel (Key) und den Nutzdaten (Value). Die Nutzdaten können vielfältig sein, etwa ein einzelner Wert oder komplexe Strukturen. Als Operatoren stehen „put“, „get“ und „delete“ zur Verfügung. Typischerweise kann nur der Key indexiert werden, da über diesen der Zugriff erfolgt.

Oracle NoSQL, Redis, Amazon Dynamo und Riak sind typische Vertreter dieser Kategorie. Der Übergang zu Document Stores ist fließend. Oracle NoSQL hat beispielsweise auch Eigenschaften von Document Stores.

Document Stores basieren auf einem Datenmodell, in dem zu einem Schlüssel komplex strukturierte Daten („Dokumente“) als Wert zugeordnet sind *(siehe Abbildung 2)*. Solche Dokumente sind typischerweise JavaScript Object Notation (JSON) oder XML-Strukturen. Der Zugriff auf Daten erfolgt nicht nur über den Schlüssel im Vergleich zu Key-Value Stores, sondern auch über Daten im Dokument. Dazu ist eine Indexierung der Zugriffsschlüssel nötig, um eine performante Suche zu ermöglichen.

JSON wird zur Serialisierung von vielen Datenbank-Systemen als primäres (etwa bei MongoDB, CouchDB, Couchbase) oder zusätzliches Datenformat unterstützt. JSON-Strukturen erlauben die Einbettung vieler Datenelemente in ein Dokument, beispielsweise Personendaten einschließlich Adressen und Blog-Artikel. Diese Normalisierung vermeidet Joins. Besonders aufwendig sind dagegen Updates, da aufgrund der Denormalisierung unter Umständen viele Datensätze aktualisiert werden müssen.

Eine Standard-Abfragesprache wie SQL sucht man für Document Stores vergeblich. Proprietäre Sprachen, die teilweise SQL-artige Elemente übernehmen, müssen zum Erzeugen, Ändern und Löschen von Dokumenten genutzt werden; so gibt es beispielsweise Operationen wie „insertOne“ oder „insertMany“ zum Einfügen eines oder mehrerer Dokumente.

Wide Column Stores verwenden ein Datenmodell, das aus beliebig vielen Spalten besteht. Einzelne Zeilen können dabei verschiedene Spalten aufweisen *(siehe Abbildung 3)*. Die Zellen von Spalten können atomare Daten oder komplexe Strukturen wie Dokumente beinhalten; der Zugriff erfolgt über den

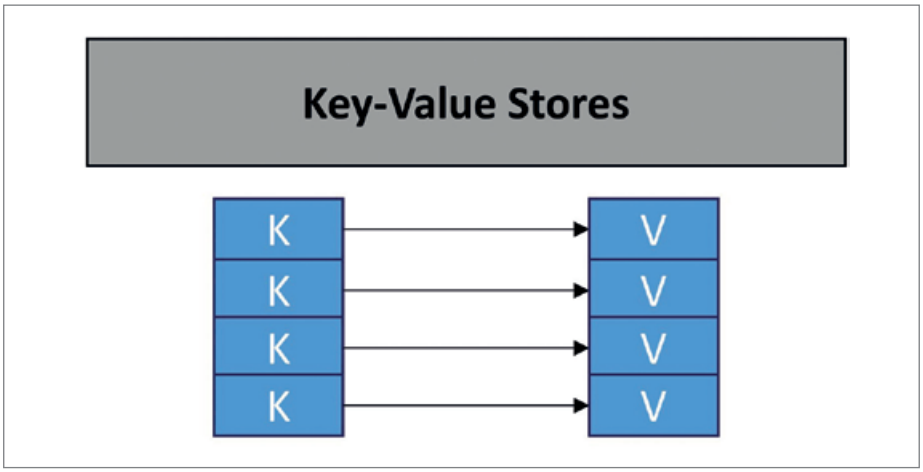

Abbildung 1: Key-Value Stores

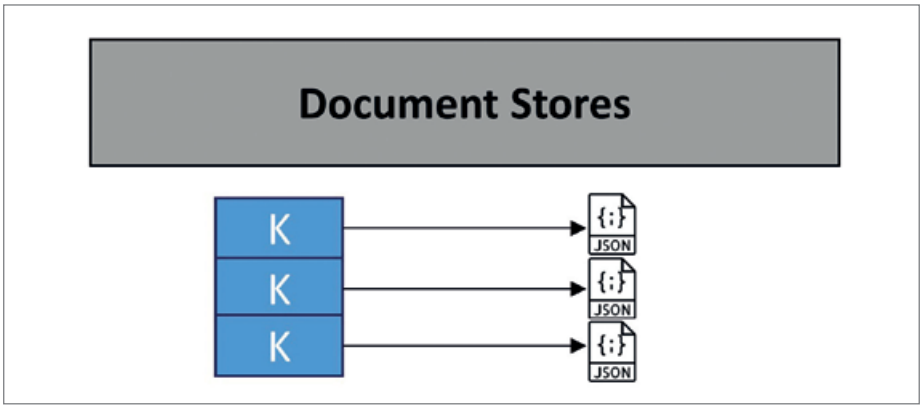

Abbildung 2: Document Stores

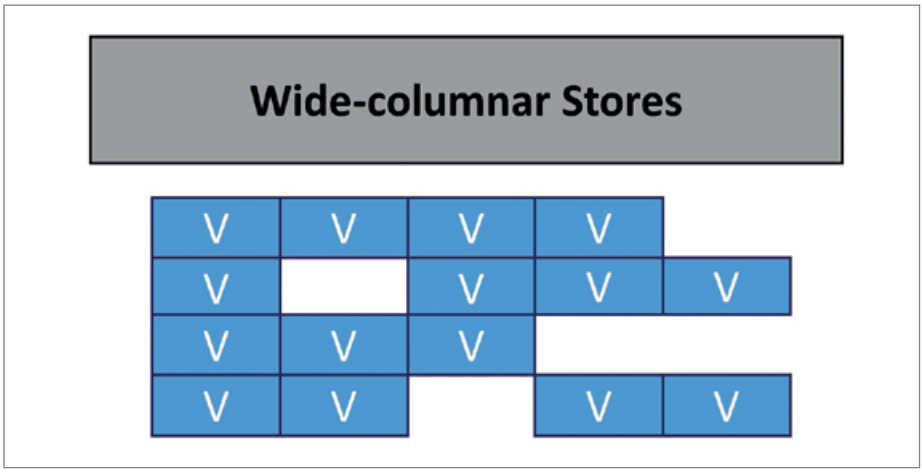

Abbildung 3: Wide Column Stores

Schlüssel. Diese Datenbanken sind hochskalierbar beim Schreiben sowie beim Lesen weniger Datensätze über den Schlüssel. Bekannteste Vertreter dieser Kategorie sind Cassandra und HBase als Bestandteil des Hadoop-Stacks. Ein Einsatz-Szenario von Cassandra ist die Speicherung von Ausstattungsmerkmalen bei der individuellen Zusammenstellung eines Fahrzeugs in einem webbasierten Online-Fahrzeugkonfigurator.

Graph-Datenbanken *(siehe Abbildung 4)* zählen ebenfalls zur NoSQL-Kategorie, auch wenn Skalierbarkeit und Verteilung nicht zu den Stärken dieser Datenbanken gehören. Sie speichern Daten als Knoten und Kanten ab. Haupt-Anwendungsgebiet der Graph-Datenbanken ist

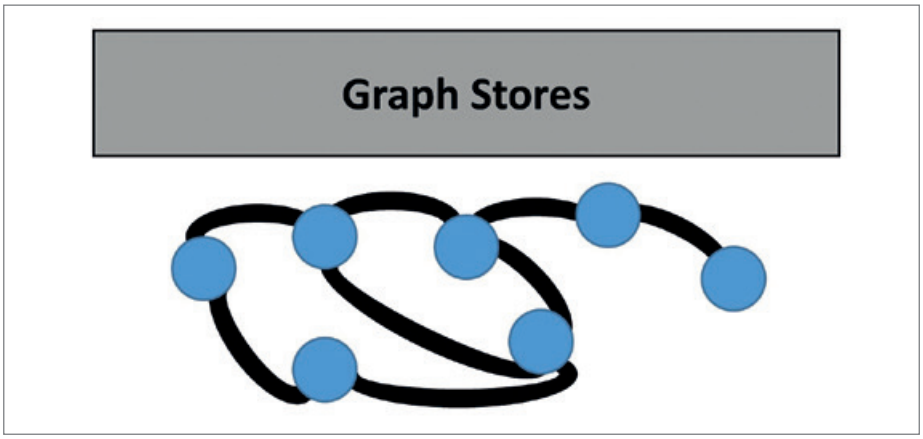

Abbildung 4: Graph Stores

die Abbildung von Beziehungen. Diese Datenbanken ermöglichen ein sehr performantes Traversieren von Kanten. Bekannteste Datenbank dieser Kategorie ist Neo4j.

ACID, BASE und CAP

ACID ist eine bekannte Eigenschaft von Transaktionen in relationalen Datenbanken, insbesondere zur Sicherstellung von Konsistenz. Konsistenz wird bei NoSQL sehr häufig aufgeweicht. Laut dem CAP-Theorem von Brewer kann ein verteiltes System zwei der folgenden Eigenschaften annehmen: Konsistenz (C = consistency), Verfügbarkeit (A = availability) und Partitionstoleranz (P = partition tolerance). AP-Systeme geben strenge Konsistenz auf zugunsten von Verfügbarkeit und Ausfallsicherheit.

Für diese Datenbanken gilt das BASE-Prinzip (Basically Available, Soft state, Eventual consistency). Die Konsistenz stellt sich im Laufe der Zeit ein, sobald alle verteilten Datenbankknoten die Änderung erhalten haben. Das Verhalten einer Anwendung ohne strenge Konsistenz lässt sich beispielsweise auf Twitter regelmäßig beobachten: Neue Follower werden namentlich gelistet, der Zähler für die Anzahl der Follower wird oft jedoch erst später aktualisiert.

Das folgende Beispiel zeigt die Auswirkungen von BASE. Angenommen, der Wert der Variable X ist 1 und es werden die Operationen X = 5 und danach auf einem anderen Netzwerkknoten Y = X + 2 durchgeführt. Ein System mit strenger Konsistenz wird den Wert 7 liefern. Ein BASE-System dagegen liefert 3 oder 7 beziehungsweise nach einiger Zeit nur noch 7. Ein sehr lesenswerter Artikel zu CAP und BASE stammt von Daniel Abadi *[1]*.

NewSQL-Datenbanken

Das Fehlen strenger Konsistenz in der Persistenz-Schicht macht eine Anwendung sehr komplex. Die Verlagerung der Konsistenz-Sicherung in die Anwendung erweist sich als zu große Herausforderung, zu aufwendig und viel zu fehleranfällig. In einem White Paper über die NewSQL-Datenbank F1 bringt Google die Problematik auf den Punkt: „We also have a lot of experience with eventual consistency systems at Google. In all such systems, we find developers spend a significant fraction of their time building extremely complex and error-prone mechanisms to cope with eventual consistency and handle data that may be out of date. We think this is an unacceptable burden to place on developers and that consistency problems should be solved at the database level. Full transactional consistency is one of the most important properties of F1“ *[2]*.

Auch SQL als standardisierte Anfragesprache wurde schnell als wichtig erkannt. Datenbanken wie VoltDB versprechen die Flexibilität von NoSQL sowie wichtige Features wie strenge Konsistenz und SQL.

Cloud-native Datenbanken

Cloud-native ist der neueste Trend und basiert wie NoSQL und NewSQL auf Verteilung, Replikation und Hochverfügbarkeit. Cloud-native Datenbanken sind idealerweise bereits für die Nutzung in der Cloud entwickelt. Eigenschaften solcher Anwendungen sind von der Cloud-native Computing Foundation (CNCF) definiert:

- *Ubiquitous and flexible*
  Die Datenbank muss in beliebigen Container-Technologien in beliebigen Cloud-Umgebungen lauffähig sein
- *Resilient and scalable*
  Der Ausfall von Knoten darf nicht durch hohe Verfügbarkeit, Redundanz und automatischen Fail-Over bemerkbar sein
- *Dynamic*
  Upgrades erfolgen automatisch
- *Automatable*
  Alles ist im Sinne einer Infrastructure as Code automatisiert
- *Observable*
  Loggin, Tracing, Metriken müssen auch noch verfügbar sein, wenn der Container nicht mehr existiert
- *Distributed*
  Nutzung der Vorteile einer verteilten Cloud

Beispiele für Cloud-native Datenbanken sind Google Spanner sowie die Open-Source-Variante CockroachDB.

Ausblick

Wie reagieren die relationalen Datenbank-Hersteller auf die Herausforderungen durch NoSQL, NewSQL und Cloud-native Datenbanken? In der Vergangenheit gab es bereits Ergänzungen, die längst in den SQL-Standard eingeflossen sind:

- In den 1990er-Jahren wurden objektorientierte Konzepte als Antwort auf objektorientierte Datenbanken integriert
- In den 2000er-Jahren wurden XML-Konzepte als Antwort auf XML-Datenbanken integriert

Wird der Einsatz einer NoSQL-, NewSQL- oder Cloud-ready Datenbank in Erwägung gezogen, sind Security-Features kritisch zu betrachten („Enterprise-readiness“). Relationale Datenbanken verfü-

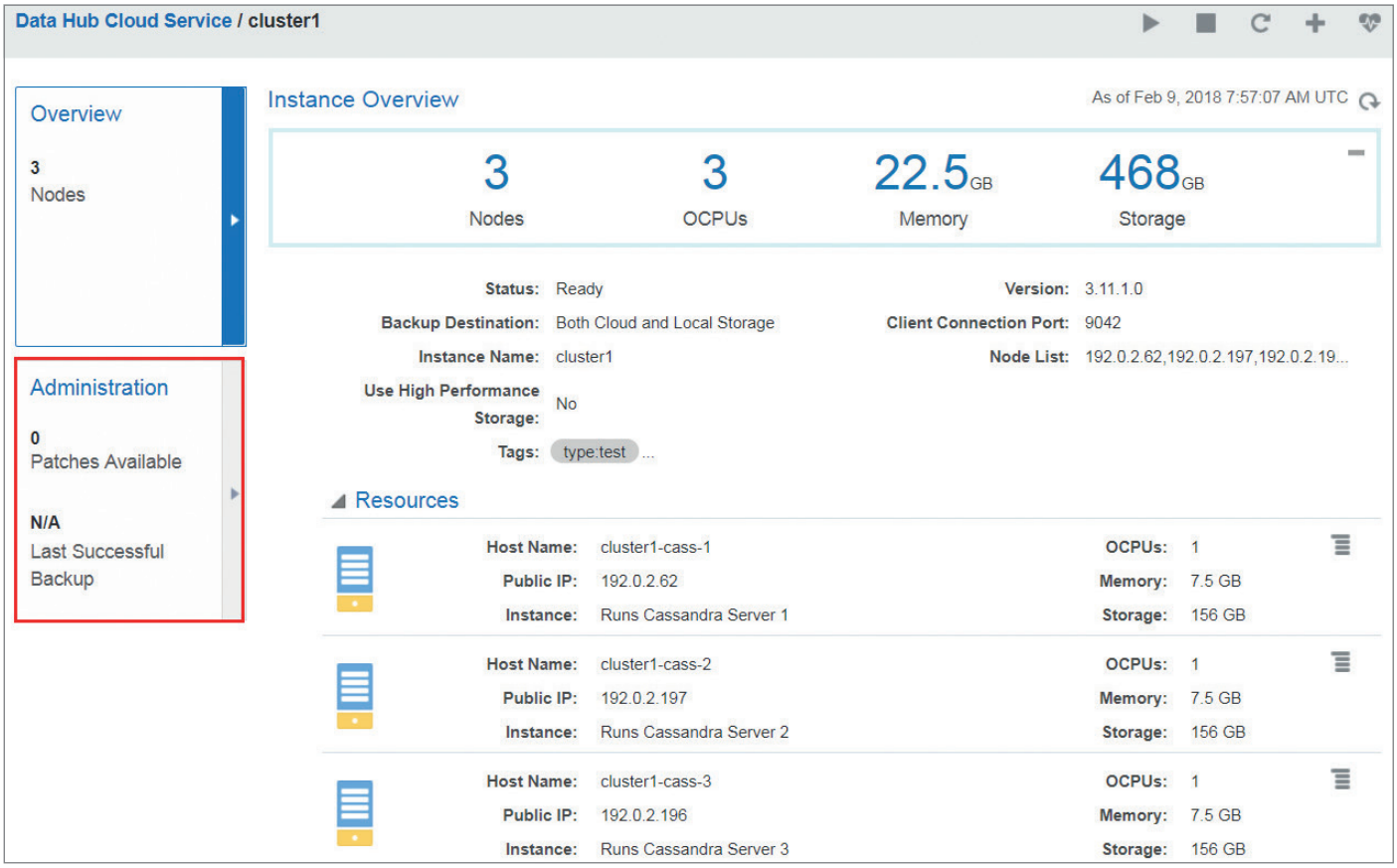

Abbildung 5: Apache Cassandra Cluster im Oracle Data Hub Cloud Service [3]

gen über umfangreiche und ausgereifte Features. Neue Datenbanken haben noch Nachholbedarf, sodass ein benötigtes Security Feature noch fehlen kann. Auch auf die Herausforderungen durch NoSQL-, NewSQL- und Cloud-native Datenbanken gibt es Antworten:

- Flexibles Datenmodell für Schema-on-read durch JSON-Unterstützung (SQL-Standard 2016)
- Nutzung von Oracle OWF zur Speicherung Graph-basierter Daten
- MemOptimized RowStore als Alternative zu Key-Value-Stores
- Autonomous Datenbanken für bestimmte Szenarien zur Verringerung von Tuning-Maßnahmen und betriebliche Tätigkeiten
- Weitere längst bekannte Maßnahmen, etwa durch Informational (Deferred) Constraints, Partitionierung, Parallelisierung etc.

In der Oracle Cloud sind NoSQL-Datenbanken als „Managed Services" verfügbar:

- Oracle NoSQL wird als autonome Datenbank angeboten
- Cassandra kann im Oracle Data Hub Cloud Service konfiguriert werden *(siehe Abbildung 5)*
- Mit Bitnami besteht eine Partnerschaft, um MongoDB in der Oracle Cloud verfügbar zu machen *[4]*
- Apache HBase ist innerhalb des Big Data Cloud Service nutzbar
- Container-Technologien wie Docker und Kubernetes können genutzt werden

Fazit

Im Gegensatz zu den objektorientierten beziehungsweise XML-Datenbanken werden NoSQL-, NewSQL- und Cloud-ready Datenbanken bestehen bleiben. Diese haben sich bereits einen Anteil des Gesamtmarktes gesichert und sind für bestimmte Use Cases sinnvoll.

Referenzen

[1] NewSQL database systems are failing to guarantee consistency, and I blame Spanner: *http://dbmsmusings.blogspot.com/2018/09/newsql-database-systems-are-failing-to.html*

[2] F1, A Distributed SQL Database That Scales: *https://db.disi.unitn.eu//pages/VLDBProgram/pdf/industry/p769-shute.pdf*

[3] *https://www.oracle.com/webfolder/technetwork/tutorials/obe/cloud/dhcs/getstarted/dhcs.html*

[4] *https://bitnami.com/stack/mongodb/cloud/oracle*

Andreas Buckenhofer
andreas.buckenhofer@daimler.com