Oracle NoSQL – eine Alternative für die traditionelle Datenbank?

Gunther Pippèrr

Ist Oracle NoSQL nur ein Hype oder die Zukunft? Oder wird es ein Miteinander mit der traditionellen Datenbank geben?

Die Oracle-NoSQL-Datenbank bietet als klassischer Key-Value-Store über verteilte Knoten Ausfallsicherheit und Performance für die Verarbeitungen von Massendaten.

Im Gegensatz zur Oracle-Datenbank, dem Schweizer Taschenmesser für viele Probleme, ist die Oracle NoSQL mehr ein Präzisionswerkzeug für genau eine gezielte Aufgabe.

Um die NoSQL-Datenbank im Vergleich zur traditionellen Oracle-Datenbank besser einschätzen zu können, stellt der Artikel den prinzipiellen Aufbau der Oracle NoSQL dar, zeigt die Vorteile dieser Architektur gegenüber dem traditionellen RDBMS-Ansatz auf und gibt eine Einschätzung, für welche Art von Anwendungen Oracle NoSQL Vorteile bringt.

Das NoSQL-Konzept und Oracle NoSQL

Der Begriff „NoSQL" ist mehr ein Denkansatz zur Datenverwaltung und Verarbeitung als eine spezielle Technologie. Gemeinsam ist den verschiedenen Implementierungen auf dem Markt aber meist der Versuch, eine hohe Skalierbarkeit durch massive Parallelisierung über viele Rechnerknoten zu erreichen. Die Oracle-NoSQL-Datenbank ist dabei ein Vertreter der Key-Value-Store-Datenbanken. Auf Basis der soliden Berkeley-Datenbank „Java Edition" hat Oracle die bestehenden Replikationsmechanismen der Berkeley-Datenbank optimiert und damit eine neue Datenbank Oracle NoSQL entwickelt. Diese folgt dem aktuellen Trend, eine moderne Datenbank mehr nach der Philosophie hinter dem CAP-Theorem und dem Base-Consistency-Ansatz umzusetzen als das traditionelle ACID (Atomicity, Consistency, Isolation und Durability) einer relationa-

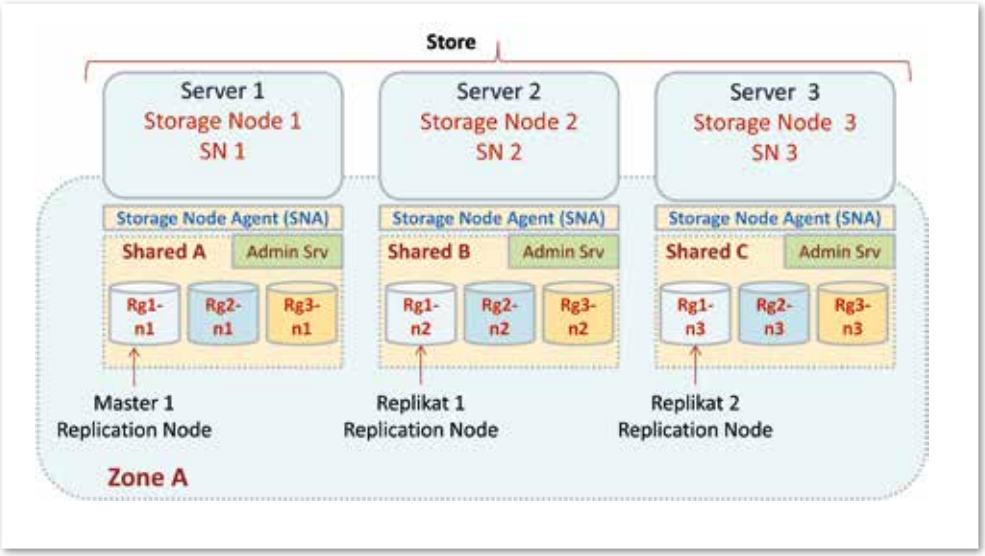

Abbildung 1: Aufbau eines NoSQL Stores mit drei Storage Nodes (SN) auf je einem Server und Replikationsfaktor „drei"

Das CAP-Theorem

Mit dem CAP-Theorem, aufgestellt von Eric Brewer im Jahre 2000, kann bewiesen werden, dass es unmöglich ist, ein verteiltes System zu erstellen, das gleichzeitig die Anforderungen „Konsistenz" (C), „Verfügbarkeit" (A) und „Partitionstoleranz" (P) erfüllt. Es können nur je zwei der Eigenschaften zur selben Zeit erreicht werden:

- *Consistency (Konsistenz)*
  Alle Partner in einen System sehen zur selben Zeit den gleichen Datenbestand
- *Availability (Verfügbarkeit)*
  Alle Anfragen an das Gesamtsystem werden immer komplett beantwortet
- *Partition Tolerance (Partitionstoleranz)*
  Verliert das System einen Teilnehmer, ein Netzsegment oder gehen Nachrichten verloren, arbeitet das Gesamtsystem ungestört weiter.

Die traditionelle Oracle-Datenbank ist darauf optimiert, Consistency und Availability (CA) zu erfüllen; fallen allerdings einzelne Komponenten aus, ist das System nicht mehr immer verfügbar.

len Datenbank erneut zu implementieren. Allerdings kann der Entwickler bei einer Oracle-NoSQL-Implementierung frei entscheiden, wie er die Consistency je nach Transaction umsetzen möchte. Mit diesem Ansatz sind in einer Applikation – je nach Notwendigkeit – beide Welten in einer Datenbank-Umgebung realisierbar.

Das „P" in CAP steht für „Partition Tolerance". Um dies bei einem Ausfall eines Netz-Segments zu gewährleisten, hat die Oracle-NoSQL-Datenbank ein Replikationskonzept umgesetzt und der Store (entspricht im Denkansatz der RDBMS einer kompletten Datenbank) ist über möglichst viele Server, die Storage Nodes (SN), verteilt. Auf jedem dieser Storage Nodes ist ein zentraler Prozess gestartet, der „Storage Node Agent" (SNA). Er übernimmt eine zentrale Rolle, überwacht den jeweiligen Storage Node und liefert wichtige Informationen an den Admin-Service, der den Store verwaltet. Dieser besteht wiederum aus einer kleinen Datenbank (was im Prinzip dem System Tablespace mit dem Data Dictionary einer relationalen Datenbank entspricht) und dient zur Überwachung und Konfiguration des Stores. Damit bei einem Ausfall diese zentrale Komponente redundant zur Verfügung steht, kann der Admin Service über mehrere SNs gespiegelt werden.

Beim Anlegen des Stores definiert der Administrator mit dem Replikationsfaktor, wie viele Kopien von einem Master über alle verfügbaren Knoten beziehungsweise Storage Nodes verteilt werden sollen. Bei einem Replikationsfaktor von „drei" bedeutet dies zum Beispiel, dass ein Master und zwei Replikate in einer Replication Group (Rg) verwaltet werden. Auf den verfügbaren Storage Nodes sind dann drei Replication Nodes für diese Gruppe angelegt, möglichst je auf einem separaten Server. Jeder Storage Node besteht damit aus einem oder mehreren Replication Nodes, die in einem sogenannten „Shared" organisiert sind *(siehe Abbildung 1)*.

Ebenfalls beim Anlegen des Stores definiert der Administrator die Anzahl der Partitionen, in die der Store unterteilt ist. Jeder Replication Group (Rg) wird die gleiche Anzahl

Das BASE-ACID-Theorem

BASE ist ein alternatives Konsistenzmodell für verteilte Datenbanken und steht für „Basically Available", „Soft state" und „Eventual consistency". Die traditionellen relationalen Datenbank-Systeme arbeiten aber meist nach dem ACID-Ansatz, „Atomarität (Abgeschlossenheit)", „Konsistenzerhaltung", „Isolation (Abgrenzung)" und „Dauerhaftigkeit".

Beim BASE-System wird die unbedingte ACID-Anforderung „Konsistenzerhaltung" der traditionellen Datenbanken zugunsten einer besseren Verfügbarkeit und höheren Performance weniger streng umgesetzt. Bis sich alle Teilnehmer wieder synchronisiert haben, wird toleriert, dass zwischen zwei Endzuständen eines Systems unterschiedliche Stände eines Datensatzes existieren können.

PROLICENSE®
OPTIMIZING SOFTWARE ASSETS
KOMPETENT – UNABHÄNGIG – ERFOLGSBASIERT

ORACLE LIZENZBERATUNG MIT GARANTIE?

Wir sind nur unseren Mandaten verpflichtet.
Garantie: Kosteneinsparungen von mind. 300% bezogen auf unser Honorar!

Über **30 Mill. EUR Einsparungen** in 2013.

Sprechen Sie mit uns oder unseren Mandanten!

Besuchen Sie uns auf der DOAG 2014 Stand 302

ProLicense GmbH
Friedrichstraße 191 | 10117 Berlin
Tel: +49 (0)30 60 98 19 230 | www.prolicense.com

an Partitionen zugeordnet und diese werden damit über mehrere Storage Nodes (SN) mit einem Master/Slave-Konzept repliziert.

Ein Konzept von Zonen erlaubt es, auch Gruppen von Rechnern zu definieren, um beim Ausfall einer kompletten Infrastruktur (zum Beispiel einem kompletten Server-Rack) immer noch genug funktionsfähige Knoten für den Store zur Verfügung zu haben. Der Store verteilt sich dann so, dass möglichst immer in einer Zone ein Master und in der anderen Zone ein Slave der Replication Group existiert.

Hashing und Partitionierung der Daten

In der Architektur des Oracle-NoSQL-Stores nimmt der Client-Treiber die zentrale Rolle ein. Die gesamte Logik der Datenverteilung über den Store und das Transaktionsverhalten erfolgt auf der Client-Seite. Die Verteilung der Daten auf die Partitionen des Stores erfolgt durch die Keys (Schlüssel auf die Daten). Der Client-Treiber kennt die Struktur des Stores, ermittelt über einen MD5-Hash auf den Key den passenden Master für den Datensatz und überträgt die Daten zum Schreiben auf den Storage Node, der den Master mit der entsprechenden Partition hält *(siehe Abbildung 2)*. Die Daten sind binär im Store abgelegt, der Zugriff erfolgt ausschließlich über den Schlüssel. Beim Lesen kann der Client jedoch vom Master und von allen Replikaten die Daten erhalten; dies optimiert entscheidend die Performance.

Im Prinzip sind die Daten zu einem Key immer ein binärer Datencontainer (etwa ein serialisiertes Java-Objekt) und damit nicht selbstbeschreibend. Dies hat zwar im ersten Ansatz einige Vorteile bei der schnellen Entwicklung, denn der Entwickler muss sich nicht mit dem lästigen Thema der Datenmodellierung beschäftigen und kann sich frei in den Daten der Datenbank bewegen. Es hat allerdings den Nachteil, dass über den Lebenszyklus einer Applikation sehr viel Sorgfalt in der Pflege notwendig ist.

Interessanterweise führen aber genau diese hohen Freiheitsgrade in letzter Zeit wieder vermehrt dazu, dass alte Ansätze nun doch in den neuen NoSQL-Datenbank-Varianten auftauchen. So unterstützt Oracle NoSQL ab der Version 2 das AFRO-Serialisierungs-Format. Ab der Version 3 wird das Konstrukt der Tabelle zur

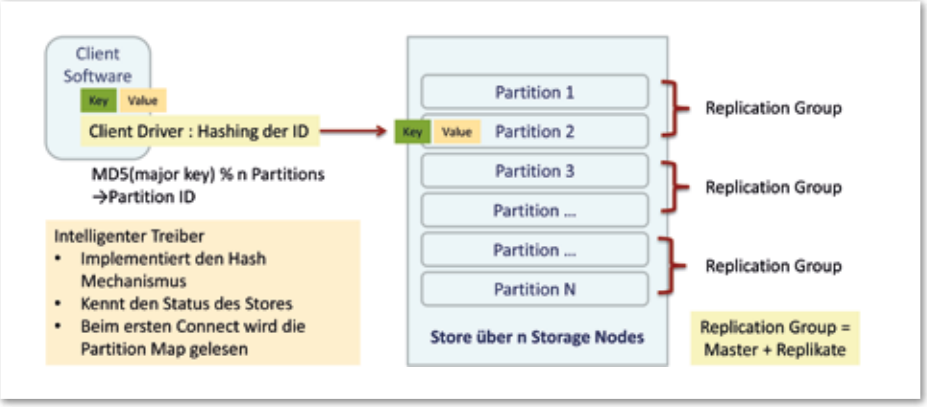

Abbildung 2: Hashing und Partitionierung der Daten der Oracle-NoSQL-Datenbank

```
// Key mit Minor und Mayor Komponente anlegen
Key k = Key.createKey("MAIN_KEY", "SLAVE_KEY");
// Wert als Byte Array anlegen
byte[] b = ("Wert").getBytes();
// Wert in den Store schreiben
kvstore.put(k, Value.createValue(b));
```

Listing 1: Beispiel für das Schreiben in Java

```
// Key erzeugen
Key datakey = Key.createKey("ABCDEFG");
// Datensatz aus dem Store mit dem Key wieder lesen
ValueVersion vv = kvstore.get(datakey);
// Daten aus dem Satz extrahieren
Value vdata = vv.getValue();
// Nutzdaten wieder herstellen (entserialiseren)
String data = new String(vdata.getValue());
// Daten ausgeben
System.out.println(data);
```

Listing 2: Beispiel für das Lesen in Java

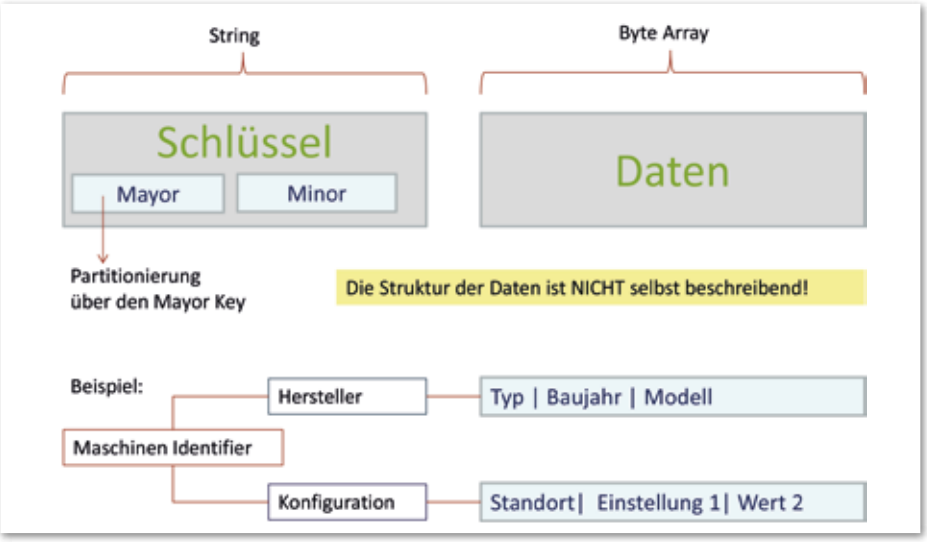

Abbildung 3: Das Mayor/Minor-Schlüssel-Konzept der Oracle-NoSQL-Datenbank

Definition der Datenstruktur der Values im Key Store eingeführt, um eine gewisse Wartbarkeit des Codes sicherzustellen.

Neben der Logik der Partitionierung der Daten stellt der Entwickler der Applikation auch über den Treiber ein, ob er nach dem CAP-Theorem mehr eine „CA“ (Consistency und Availability) oder „AP“ (Availability und Partition Tolerance) anstrebt. Der Entwickler entscheidet, wie beim Schreiben und Lesen vorgegangen werden soll: Ob zum Beispiel schnell das Schreiben in den Cache eines Master Nodes ausreicht oder ob sicherheitshalber der Datensatz doch auch auf allen Replikaten erfolgreich geschrieben werden soll. Das Gleiche gilt auch für das Lesen: Ist der erste einigermaßen aktuelle Datensatz ausreichend (Eventually Consistent) oder muss sichergestellt sein, dass hier auch der aktuellste Datensatz beziehungsweise die letzte Version gelesen wurde.

Daten einlesen und auslesen

Eine eigentliche SQL-Syntax für das Arbeiten mit dem Store steht zurzeit nicht zur Verfügung. Das Auslesen der Daten erfolgt mit „PUT“- und „GET“-Methoden des Java- oder C-API *(siehe Listing 1 und 2)*.

Innerhalb einer Gruppe von Aktionen auf demselben „Mayor Key Path“ lässt sich diese Gruppe von Aktionen auch zu einer Schreib-Transaktion zusammenfassen. Diese atomare Transaktion stellt sicher, dass entweder alles erfolgreich oder gar nicht abgearbeitet wird.

Auf die Daten wird immer über den Schlüssel zugegriffen, dazu ist der Key in zwei Komponenten unterteilt, den Mayor- und den Minor-Key. Der Mayor-Key definiert (über das Ergebnis des „MD5 Hash % Anzahl der Partitionen“), in welcher Partition die Daten laden. Der Minor-Key dient der Datenmodellierung, um Daten logisch zu gruppieren *(siehe Abbildung 3)*. Da die Daten immer auf dem Client ausgewertet werden, ist eine performante Netzwerk-Anbindung an den Store eine unbedingte Voraussetzung für gute Performance.

Diese Architektur ist für das Lesen eines Datensatzes mit einem festen Schlüssel optimiert, ein Single Read kann mit einer hohen Wahrscheinlichkeit sehr schnell erfolgen. Aber bereits der Versuch, alle Datensätze eines Stores zu zählen, zeigt das Problem dieses Ansatzes: Alle Schlüssel müssen über das Netz zum Client übertragen werden, um festzustellen, wie viele es zum Schluss gibt. Ab der Version 3 wird das Problem etwas entschärft. Man hat die Option, einen sekundären Index auf den Daten aufzubauen, falls die Datensätze eines Key-Values-Paares mit der Tabellen-Option im Store angelegt wurden.

Einsatz-Szenarien

Das schnelle Lesen eines Schlüssels aus dem Datenbestand prädestiniert die Oracle-NoSQL-Datenbank für Anwendungsfälle, bei denen es auf die schnelle Bereitstellung eines eindeutigen Datensatzes ankommt. Dieses „Persitent Cache“- beziehungsweise „Stream Processing“-Verhalten taucht immer öfter als Problem mit der traditionellen Datenbank auf, denn diese ist per se keine Queue oder kann nicht wirklich performant als reiner Cache dienen. Dazu ist das immer unbedingt notwendige ACID-Verhalten einer relationalen Datenbank mit zu viel Aufwand verbunden.

Aufgrund genau dieser Anforderung kommt bei einem namhaften Kunden mit einem sehr großen Webportal die NoSQL DB zum Einsatz. Auf der Website und in der dazugehörigen App müssen Security Token und Personalisierungen für einzelne Anwender schnell bereitstehen. Dazu werden die personalisierten Bonus-Gutscheine für jeden Kunden als ein Record mit der Kundennummer als Schlüssel wöchentlich in die NoSQL DB geladen und stehen dann mit einer berechenbaren Wahrscheinlichkeit innerhalb weniger Millisekunden für die Applikation zur Verfügung. Zuvor konnten diese Daten zwar auch aus einer traditionellen Datenbank ermittelt werden, je nach Kunde aber mit unterschiedlichen Antwortzeiten, abhängig davon, wie stark der Kunde personalisiert ist und wie komplex die Regeln dazu sind.

Oracle selbst sieht in den aktuellen Präsentationen allerdings die Oracle NoSQL DB eher als ein Vorsystem zum Sammeln der Daten. Die eigentliche Auswertung erfolgt dann wieder klassisch im Data Warehouse beziehungsweise über eine Integration in das Hadoop-Ökosystem für das Veredeln der Daten. Ob sich jedoch ein Key-Value-Store wirklich für das Sammeln unstrukturierter Daten eignet, sei bezweifelt. Das Kernproblem jeder Datenbank-Entwicklung steht dem entgegen: Vor dem Sammeln muss exakt klar sein, welche Analysen am Ende gefahren werden sollen. Das definiert dann, wie die Daten später benötigt werden, also wie sie vorher abzulegen sind – insbesondere dann, wenn nur über den Schlüssel auf die Daten zugegriffen werden kann.

Durchaus sinnvolle Szenarien ergeben sich damit aus dem Einsatz als schneller Cache und als zusätzliche Datenquelle in Hadoop-Umgebungen. Hierzu lassen sich die Map/Reduce-Abfragen aus Hadoop an

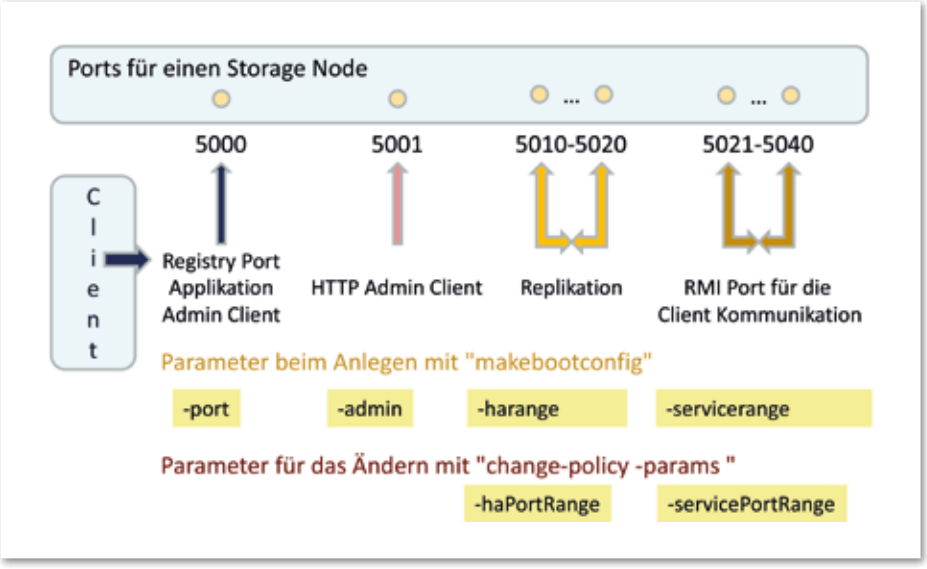

Abbildung 4: Die notwendigen Netzwerk-Ports der Oracle-NoSQL-Datenbank

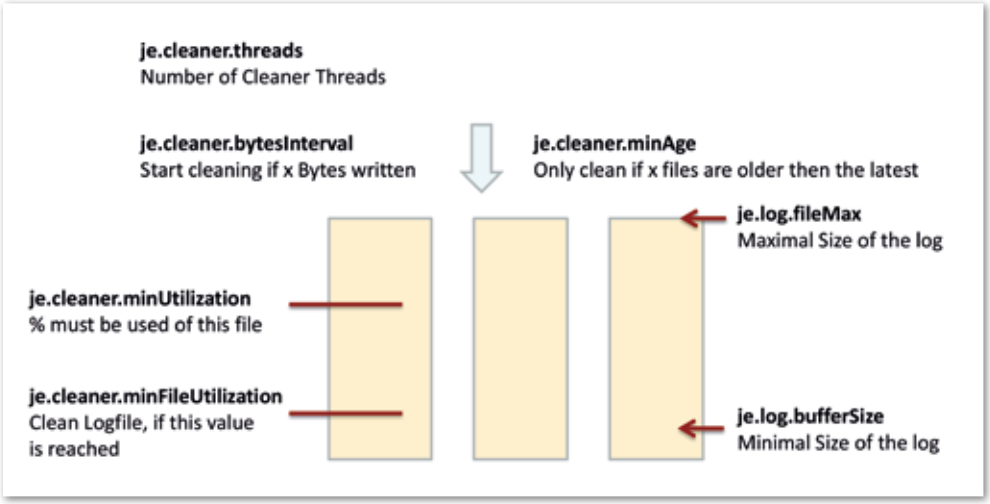

Abbildung 5: JE-Parameter für das Storage-Verhalten der Berkeley-Java-Datenbank

```
#Class Path auf die Libraries
export KVLIB=/opt/oracle/produkt/11.2.0/kv-2.1.8/lib
export KVCLASS=$KVLIB/kvclient.jar:$KVLIB/kvstore.jar:$KVLIB/je.jar
#Pfad zu den Datendateien
export JEENV=/opt/oracle/kvdata/GPIDB/sn1/rg1-rn1/env/
#Aufruf der JE Klasse
java -classpath $KVCLASS com.sleepycat.je.util.DbSpace -h $JEENV
          File Size (KB)  % Used
--------  ---------       ------
00000000  429058          44
TOTALS    429058          44
```

Listing 3

```
#Class Path auf die Libraries
export KVLIB=/opt/oracle/produkt/11.2.0/kv-2.1.8/lib
export KVCLASS=$KVLIB/kvclient.jar:$KVLIB/kvstore.jar:$KVLIB/je.jar
#Pfad zu den Datendateien
export JEENV=/opt/oracle/kvdata/GPIDB/sn1/rg1-rn1/env/
#Aufruf der JE Klasse
java -classpath $KVCLASS com.sleepycat.je.util.DbFilterStats -p Cleanin
g:cleanerBackLog,Cleaning:nCleanerDeletions $JEENV/je.stat.csv
3,0
3,1
…
```

Listing 4

die Oracle NoSQL DB dirigieren und die Ergebnisse werden wiederum in einem Hadoop-Warehouse abgelegt.

Planung und Aufsetzen einer Oracle-NoSQL-Umgebung

Ein wichtiger Punkt für die Performance ist das Netzwerk. Wird auf höchste Performance Wert gelegt, lohnt sich durchaus der Einsatz von InfiniBand für die Kommunikation der Knoten untereinander. Jeder Datensatz muss schnellstmöglich auf die Replikate übertragen werden, um das Zeitfenster der „Eventual Consistency" möglichst klein zu halten. Eine exakt gleiche Systemzeit aller Knoten des Stores ist sehr wichtig; der NTP-Service auf jeden Knoten ist mit größter Sorgfalt einzurichten, um Ausfälle im Store zu vermeiden.

Oracle NoSQL DB benötigt für die Kommunikation der Storage Nodes untereinander und mit einem Client eine recht hohe Anzahl von Ports *(siehe Abbildung 4)*. Soll vor der NoSQL-DB-Umgebung eine Firewall für erweiterte Sicherheit sorgen, ist darauf zu achten, eine Portrange auch für die Client-Kommunikation zu reservieren (Parameter „servicePortRange" beim Anlegen des Stores), damit auch die für die RMI-Kommunikation (Remote Method Invocation) notwendigen Ports zwischen Client und Datenbank-Knoten in der Firewall freigeschaltet werden können. Bei mehr als einem Store empfiehlt es sich, diese Portranges zu standardisieren, um die Wartung und den Betrieb erheblich zu erleichtern. Um den I/O-Stack der Maschinen zu entlasten, kann zwar ein Cache-Bereich im Memory definiert werden, die Storage Locations der Master und Replikate sollten aber auf eigenen Platten beziehungsweise LUNs liegen.

Da die Berkeley-Datenbank alle Redo-Daten mit in die Datendateien schreibt und bei einem Update oder Delete die vorherigen Daten nie überschrieben werden, wachsen die Datendateien permanent. Ein Cleaner-Prozess im Hintergrund räumt dann gelegentlich (getriggert über eingestellte Thresholds) die Dateien auf und verdichtet die Datenbank dann wieder. Bei großen Lösch- und Einfüge-Operationen kann es allerdings vorkommen, dass unerwartet viel Plattenplatz verbraucht wird, bis der Cleaner im Hintergrund mit dem Aufräumen hinterherkommt. Neben der Definition einheitlicher Portranges für die Umgebungen sollte auch ein Namenskonzept für den Store nicht fehlen, um diese eindeutig zu unterscheiden.

Die Eigenschaften der Berkeley-Datenbank abfragen und anpassen

Unter der Oracle NoSQL ist die Java-Version der Oracle-Berkeley-Datenbank im Einsatz. Ein sehr bedeutender Unterschied zum gewohnten Verhalten des traditionellen RDBMS-Systems stellt dabei das Log-Verhalten der Berkeley-Datenbank dar. Oracle NoSQL unterscheidet nicht zwischen einem Online Redo Log und den eigentlichen Datendateien. Auch für die „Before Images" eines Datensat-

zes existiert kein Undo Tablespace; alles wird über die Datendateien abgewickelt. Jede Aktion auf den Daten führt zu einem Eintrag in die Datendateien, auch das Löschen. Das führt dazu, dass die Datenbank im ersten Schritt scheinbar immer größer wird. Erst ab bestimmten Schwellwerten wird im Hintergrund ein „Cleaner Thread" gestartet, der die Datendateien optimiert. Der Plattenplatz für die Storage Nodes sollte daher nicht zu knapp bemessen sein, zumal der Cleaner mit niedriger Priorität im System läuft. In der Kundenumgebung des Autors werden beispielsweise regelmäßig deutlich mehr als zehn Millionen Datensätze gelöscht und neu geladen, was im ersten Moment zu einem sprunghaften Wachstum der Datenbank-Dateien führt und entsprechenden Platz im Filesystem benötigt.

Im Prinzip lässt sich jedoch das Verhalten der Datenbank auch vom Administrator optimieren. Es stehen Dutzende Parameter zur Verfügung, um das System an eigene Anforderungen anzupassen. Mit der CC-Edition der Oracle NoSQL kann dies über eigene Anpassungen im Sourcecode erfolgen. Einfacher lassen sich die Eigenschaften allerdings über die Berkeley-JE-Parameter-Datei „je.properties" setzen.

In der Berkeley-JE-Dokumentation ist von „<environment home>/je.properties" die Rede; ein Test hat ergeben, dass die Datei immer dann gefunden wird, wenn „je.properties" parallel zu den Datendateien liegt (zum Beispiel in „$kvroot/kvstore/sn1/rg1-rn1/env/"). Soll ein ganzer Store konfiguriert werden, muss natürlich in jedem Daten-Verzeichnis „je.properties" hinterlegt sein. Allerdings sind die von Oracle per Default gewählten Einstellungen in den meisten Fällen ausreichend *(siehe Abbildung 5)*.

Für die weitere Analyse oder auch nur für das tiefere Verständnis der Datenbank kann man auch die nativen Berkeley-Methoden aufrufen. Dazu müssen nur der Java-Class-Patch gesetzt und die „com.sleepycat.je"-Klassen direkt auf den Datendateien aufgerufen werden. *Listing 3* zeigt ein Beispiel für das Auslesen des Füllgrads der Datendateien auf einen Store Node. Allerdings gelten diese Werte dann nur je für die aktuellen Datendateien in diesem Verzeichnis.

Um das Laufzeitverhalten der Datenbank besser zu verstehen, sind die Statistikdaten in der Datenbank in der Datei „je.stat.csv" im jeweiligen „env"-Verzeichnis eines Store Nodes *(etwa unter „/opt/oracle/kvdata/GPIDB/sn1/rg1-rn1/env/je.stat.csv")* hilfreich.

Mit dem Beispiel in *Listing 4* für das Cleaner-Verhalten lassen sich Statistiken auch über die JE-Klassen lesen. Bei den NoSQL-Umgebungen können die „alert.log"-Dateien „je.info.<n>" der Datenbank lästig werden. Sie liegen ebenfalls parallel zu den Datendateien und sind zurzeit noch von Hand zu löschen.

Lizenzierung

Erfreulicherweise hat sich Oracle entschieden, den Hauptteil der Software als Com-

WER, WIE, WAS, WIESO, WESHALB, WARUM …
ORACLE APEX-WORKSHOP BEI ORBIT

Weitere Informationen unter
www.orbit.de/workshops

Wir laden Sie zu unserem kostenfreien Workshop zum Thema **Oracle Application Express (APEX)** ein.
Als Gastredner begrüßen wir u.a. Experte **Denes Kubicek**, Oracle Developer of the Year und Oracle ACE Director.

» APEX Überblick
» APEX Reporting
» APEX Plug-ins
» APEX MS Office-Integration

Wo: **ORBIT, Mildred-Scheel-Str. 1, 53175 Bonn**
Wann: **05. November 2014, 09.00 Uhr – 12.30 Uhr**

Anmeldungen an: **mailing@orbit.de**

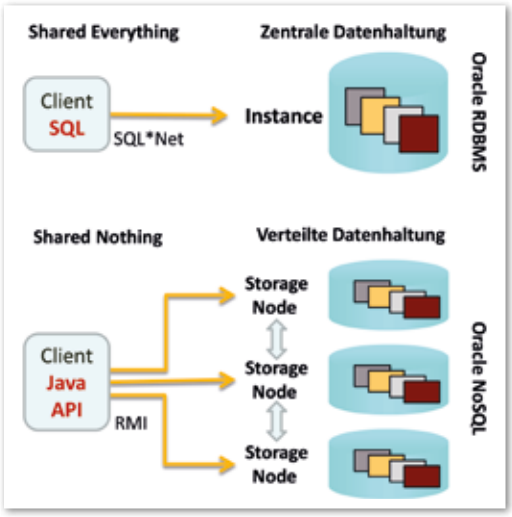

Abbildung 6: Die Architekturen im direkten Vergleich

munity Edition (CC, lizenziert nach der „GNU Affero General Public License") zur relativ freien Verwendung zu lizenzieren. Für diese CC-Edition kann laut dem Oracle Store seit Kurzem auch ein Support-Vertrag abgeschlossen werden.

Sollen weitere Sicherheitsfeatures wie Oracle Wallet oder eine Integration mit „External"-Table in die RDBMS umgesetzt werden, ist die Enterprise Edition erforderlich. Die Lizenzkosten der EE NoSQL Edition bewegen sich dabei in den gleichen Welten wie bei der Oracle-Datenbank.

Erst ab der Version 3 erfüllt auch die Oracle-NoSQL-Datenbank grundlegende Sicherheitsanforderungen. In den vorgehenden Versionen waren die Netzwerk-Administratoren und der Entwickler in der vollen Verantwortung. Ab der Version 3 lassen sich das Datenprotokoll verschlüsseln und eine Benutzerverwaltung in Grundzügen realisieren.

Erfahrungen aus der Administration

Für die Administration und Überwachung des Stores stehen eine Admin-Konsole und eine Art erstes einfaches „SQL*Plus" zur Verfügung. Eine einfache, leider in der Version 2 nicht Passwort-geschützte Web-Oberfläche erlaubt es, den Status des Stores remote per Browser abzufragen. Im Detail lässt sich das Laufzeitverhalten der Store Nodes im Store mit JMX überwachen, zum Beispiel mit Java Mission Control. Schnell zeigt es sich aber, dass eigene Skripte zur Verwaltung notwendig sind, besonders wenn die Zahl der Store Nodes relativ hoch wird. Gegenüber der relationalen Datenbank ist allerdings, auch aufgrund der deutlich kleineren Anzahl an Funktionen, die Verwaltung sehr einfach. Läuft das System einmal stabil, ist eine aufwändige Wartung kaum noch erforderlich. Leider steht noch kein eigenes Plug-in für das Monitoring der Umgebung mit dem Oracle OEM zur Verfügung. In der genannten Umgebung wurde daher für die Überwachung ein eigenes Enterprise-Manager-12*c*-Plug-in realisiert.

In der NoSQL-Welt wird das Thema „Backup" mit der Begründung „Bei genügend Server-Knoten kann ja bei einem Ausfall nichts passieren" meist sehr stiefmütterlich behandelt. Das Problem mit logischen Fehlern wird dagegen kaum beachtet. Mit der Oracle-NoSQL-Datenbank lassen sich jedoch Snapshots des gesamten Stores erzeugen. Auf Basis dieser über alle Knoten konsistenten Snapshots ist ein echtes Backup-Konzept realisierbar. Die Daten in einem Snapshot lassen sich auch in einen anderen Store importieren, etwa um eine Testumgebung aus den Produktionsdaten aufzusetzen.

Fazit

Die Oracle-NoSQL-Datenbank kann den großen Bruder, die traditionelle Oracle-Datenbank, nur in wenigen und sehr speziellen Anwendungsfällen ersetzen *(siehe Abbildung 6)*. Das sehr schnelle Auffinden eines Datensatzes in einer sehr großen Datenmenge ist durch den Hashing-Ansatz und die Möglichkeit einer hohen, ausfallsicheren Parallelisierung sehr gut implementierbar. Allerdings muss beachtet werden, dass der Store nur die Daten hält und dann dem Client über den Key zur Verfügung stellt. Ein eigenständiger Server-Prozess, der Daten vorab aggregiert, steht nicht zur Verfügung. Sollen Daten logisch verknüpft werden, muss der Entwickler im Client diesen Join auf eigene Art und Weise implementieren. Die Datenbank unterstützt ihn dabei nicht.

Weitere wichtige Features, die für eine durchgängige und vollständige Datenbank-Entwicklung notwendig sind, wie ein erster Security-Ansatz, wurden in der V3 eingeführt. Viele weitere Features werden wohl in der nächsten Version folgen. Leider sind die Java-Klassen der Version 2 nicht vollständig kompatibel zur Version 3. Soll die Class Library der Version 3 eingesetzt werden, ist der Code um das Login-Kontext-Objekt „LoginManager" zu erweitern, das bei einigen Aufrufen als dritter Parameter in der V3 erwartet wird. Da dieses auch einfach „null" sein kann, hätte sich das wohl auch als weiterer Konstrukteur für die jeweiligen Klassen realisieren lassen können.

Gegenüber den bereits durchaus mächtigeren und teils viel weiter entwickelten NoSQL-Lösungen aus dem Open-Source-Markt spricht für die Oracle-Lösung die Zusage des Herstellers, das Produkt strategisch weiterzuentwickeln und zu pflegen, sodass auch für die normalen Projektzyklen eines Projekts von zwei bis fünf Jahren ein wartbares und fehlerarmes Produkt zur Verfügung stehen wird. Gerade im Umfeld des Anwendungsfalls „Stream Processing" beziehungsweise „persistent Cache" ist die Oracle-NoSQL-Datenbank ein interessantes Werkzeug, um eine performante Lösung bereitzustellen.

Weitere Informationen

1. http://docs.oracle.com/cd/NOSQL/html/index.html
2. https://community.oracle.com/community/developer/english/big_data/nosql_database
3. http://www.oracle.com/webfolder/technetwork/de/community/dojo/index.html
4. http://www.oracle.com/technetwork/products/berkeleydb/learnmore/bdb-je-architecture-whitepaper-366830.pdf
5. http://www.pipperr.de/dokuwiki/doku.php?id=nosql_datenbank

Gunther Pippèrr
gunther@pipperr.de