Einführung in Oracle NoSQL Database

Ute Middendorf
Metafinanz Informationssysteme GmbH
München

Schlüsselworte
NoSQL, Einführung, Architektur, Installation, Datenzugriffe

Einleitung

Der Begriff "NoSQL" wurde bereits im Jahr 1998 durch Carlo Strozzi (IBM) für seine relationale Datenbank ohne SQL-API geprägt. Erst seit 2009 hat sich die Bedeutung von dem wörtlichen "kein SQL" hin zu "Not only SQL" gewandelt. Seitdem dient NoSQL als Sammelbegriff für Datenbanken mit nicht-relationalem Ansatz.
Im Zusammenhang mit Big Data gewinnen NoSQL Datanbanken immer mehr an Bedeutung, denn herkömmliche relationale Datenbanken sind nicht für die Verarbeitung der riesigen, unstrukturierten Datenmengen ausgelegt.
Auch im Produktportfolio von Oracle befindet sich eine NoSQL Datenbank. Bei der NoSQL Lösung von Oracle handelt es sich um eine verteilte, hochskalierbare Key-Value Datenbank. Doch was genau verbirgt sich dahinter?
In diesem Vortrag wird alles Wissenswerte rund um die Oracle NoSQL Datenbank erläutert: Angefangen beim Konzept und der Architektur, über die Installation hin zu den Datenzugriffen.

Grundlagen

Eine einheitliche Definition für NoSQL Datenbanken gibt es eigentlich nicht. Es gibt vielmehr nur eine Reihe von Kriterien, die NoSQL Datenbanken beschreiben:

- Kein relationales Datenmodell
- Eignung für Systeme mit verteilter und horizontaler Skalierbarkeit
- Open Source
- Schemafrei oder nur schwächere Schemarestriktionen
- Einfache Datenreplikation zur Unterstützung der verteilten Architektur
- Einfache API
- Nicht ACID sondern BASE als Transaktionsmodell

Nicht alle NoSQL Datenbanken erfüllen grundsätzlich alle der gelisteten Kriterien.

In verteilten Datenbanksystemen spielt das CAP-Theorem eine enorme Rolle. Aufgrund der Tatsache, dass sich die zu verarbeitenden Datenmengen und die zu bedienenden Anwenderzahlen nur noch auf der Basis dynamisch skalierbarer Rechnernetze realisieren lassen, stellen verteilte Datenbanksysteme eine Basistechnologie der meisten NoSQL-Datenbanksysteme dar. Somit spielen die Erkenntnisse des CAP-Theorems auch bei NoSQL Systemen eine zentrale Rolle.
Das Akronym CAP steht für die englischsprachigen Begriffe **C**onsistency (Konsistenz), **A**vailability (Verfügbarkeit) und **P**artition Tolerance (Ausfalltoleranz). Hierbei bedeutet Konsistenz, dass nach Abschluss einer Transaktion alle replizierten Daten des manipulierten Datensatzes den gleichen Inhalt haben. Verfügbarkeit besagt, dass Anfragen stets innerhalb einer geforderten maximalen Zeit

beantwortet werden. Unter Ausfalltoleranz wird verstanden, dass das System auch beim Ausfall einzelner Teile weiterhin arbeitet und Antworten liefert.

Gemäß des CAP-Theorems ist es einem verteilten System nicht möglich gleichzeitig die drei Anforderungen Konsistenz (C), Verfügbarkeit (A) und Ausfalltoleranz (P) zu gewährleisten. Zur gleichen Zeit können maximal zwei der drei CAP Anforderungen erfüllt werden.

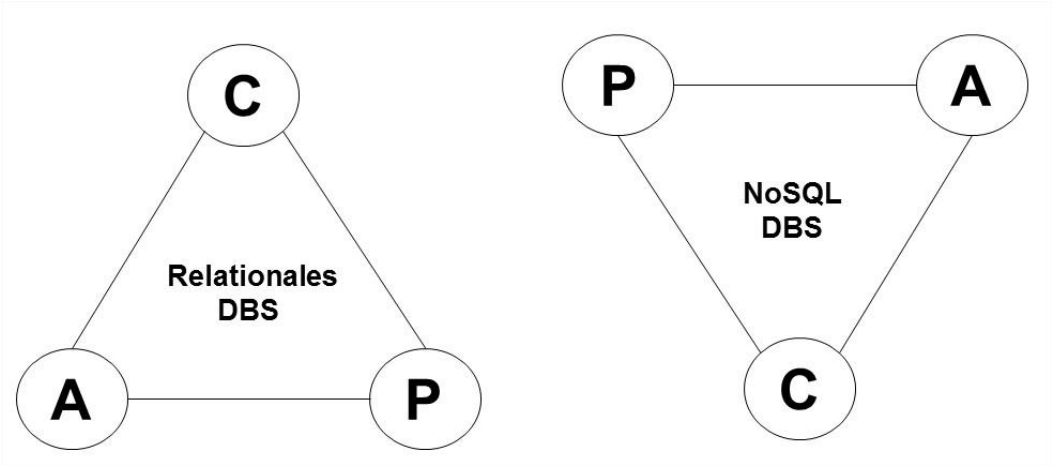

Abb. 1: CAP bei Relationalen DBMS und NoSQL-Datenbank Systemen

Bei relationalen Datenbanksystemen liegt die Priorität auf der Konsistenz der Daten. Zusätzlich erfüllen sie entweder die Anforderung nach Verfügbarkeit oder nach Ausfallsicherheit. Zum Beispiel erfüllt das Oracle RDBMS tatsächlich nur die zwei CAP Anforderungen Konsistenz (C) und Verfügbarkeit (A). Wird ein System, das den CA Anforderungen entspricht, um einen Replikationsmechanismus erweitert, so muss zur Erhaltung der Konsistenz die geänderten Daten auf allen beteiligten Replikationsservern geschrieben werden, und somit sinkt die Verfügbarkeit. Beeinflusst man nun das Verhalten bezüglich des Schreibens der Replikate (z.B.: das Schreiben von 2 der vorhandenen 5 Replikate reicht aus, damit das Schreiben als erfolgreich beendet gilt), so ist die Anforderung der Konsistenz nicht mehr gegeben.
Relationale Datenbanksysteme folgen bezüglich Transaktionen dem ACID Konzept:

- **A**tomicity: Eine Transaktion wird ganz oder gar nicht ausgeführt
- **C**onsistency: Datenkonsistenz ist das Hauptanliegen
- **I**solation: Transaktionen sind voneinander unabhängig
- **D**urability: Eine festgeschriebene Änderung kann immer wiederhergestellt werden

Im Gegensatz zu relationalen Datenbanksystemen liegt das Hauptaugenmerk der NoSQL Datenbanken auf der Ausfallsicherheit (P). Bei den auf dem Markt existierenden NoSQL Datenbanken, gibt es einige, die neben der Ausfallsicherheit Wert auf die Verfügbarkeit (A) legen. Andere wiederum haben zusätzlich Konistenz (C) umgesetzt. Die Oracle NoSQL Datenbank gehört zu den Systemen, die den Entwicklern die Entscheidung überlassen, ob neben der Ausfallsicherheit mehr Wert auf Verfügbarkeit (A) oder Konsistenz (C) gelegt werden soll. Dieser andere Anspruch an die CAP Werte spiegelt sich auf in dem Transaktionsverhalten wieder. Eine Unterstützung von ACID bei einem Hauptaugenmerk auf Verfügbarkeit und Partitionstoleranz wäre ein Widerspruch in sich, da ja gerade die Konsistenz in diesem Konstrukt nicht mehr garantiert wird. NoSQL Datenbanken verwirklichen das dem ACID Modell gegenüberstehende BASE Konzept:

- **B**asically **A**vailable: Prinzipiell ist das System verfügbar, es können jedoch einzelne Teile ausfallen
- **S**oft State: loser Zustand, d.h. es findet ein fließender Wechsel zwischen konsistenten und inkonsistenten Zuständen statt
- **E**ventually consistent : zu einem undefinierten zukünftigen Zeitpunkt ist Konsistenz gegeben

Architektur
Abhängig von der Art und Weise der Datenspeicherung lassen sich NoSQL Datenbanken in vier Kategorien aufteilen:

- dokumentenorientierte Datenbanken ("document stores")
- Key-Value-Datenbanken
- spaltenorientierte Datenbanken
- Graphendatenbanken

Die Oracle NoSQL Datenbank fällt in die Kategorie Key-Value-Datenbanken. Die Datenhaltung in der Oracle NoSQL DB erfolgt in dem sogenannten KVStore (Key-Value Store). Hierbei handelt es sich um Sammlung von Storage Nodes, die wiederum eine Reihe von Replication Nodes beheimatet. Die Daten selber sind über die Replication Nodes verteilt. Der KVStore kann in einer traditionellen Three-Tire-Web Architektur entweder einzeln oder parallel zum herkömmlichen RDBMS betrieben werden.

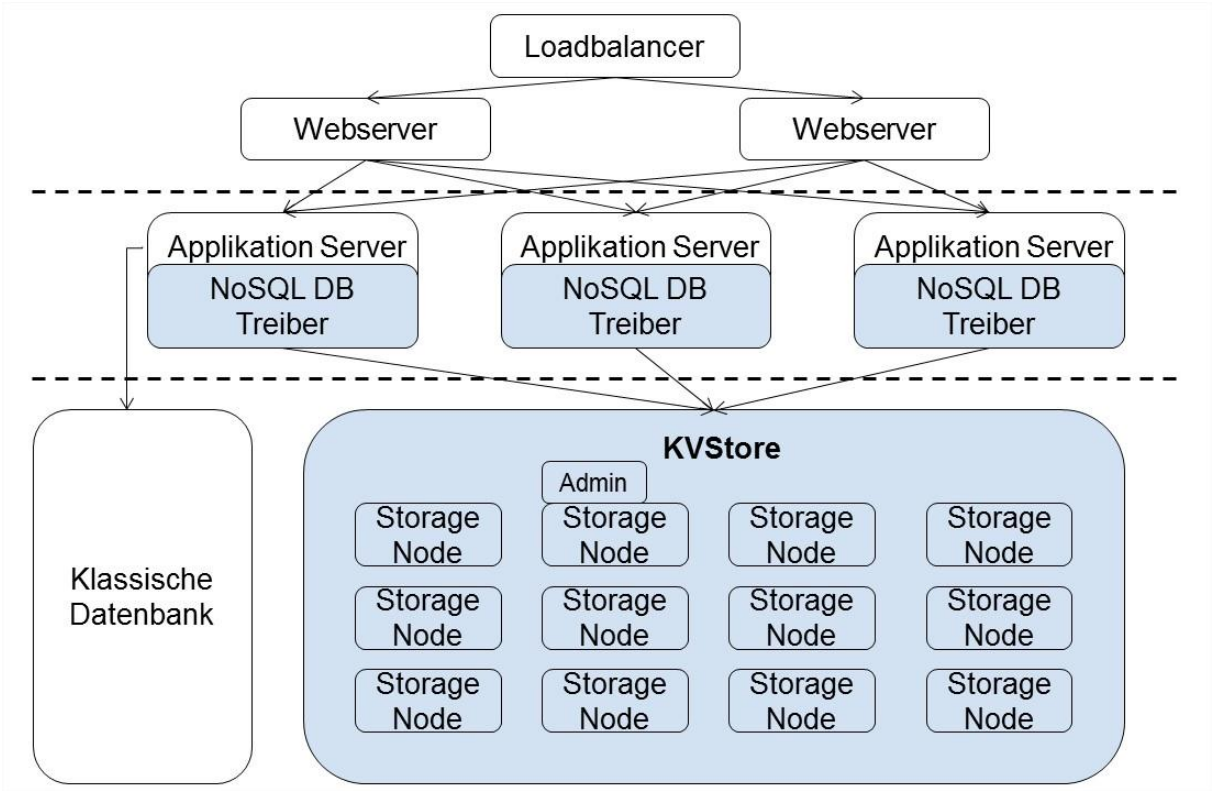

Abb. 2:Key Value Store innerhalb einer typischen Three-Tire-Architektur

Der KVStore besteht aus vielen Storage Nodes, bei denen es sich jeweils um einen eigenen Server (physikalisch oder virtuell) mit eigenem lokalen Storage handelt. Auch wenn es nicht zwingend erforderlich ist, so sollten die einzelnen Storage Nodes identisch sein.

Die Kennzahl Capacity stellt ein grobes Maß für die Hardware Ausstattung der einzelnen Storage Nodes dar. Der Capacity-Wert bestimmt die Anzahl der Replication Nodes, die auf dem jeweiligen Storage Node angesiedelt werden.

Auf einem der Storage Nodes läuft ein Administrations Service, der Informationen über den KVStore sammelt und an das Oracle NoSQL DB Dashboard weiterreicht.

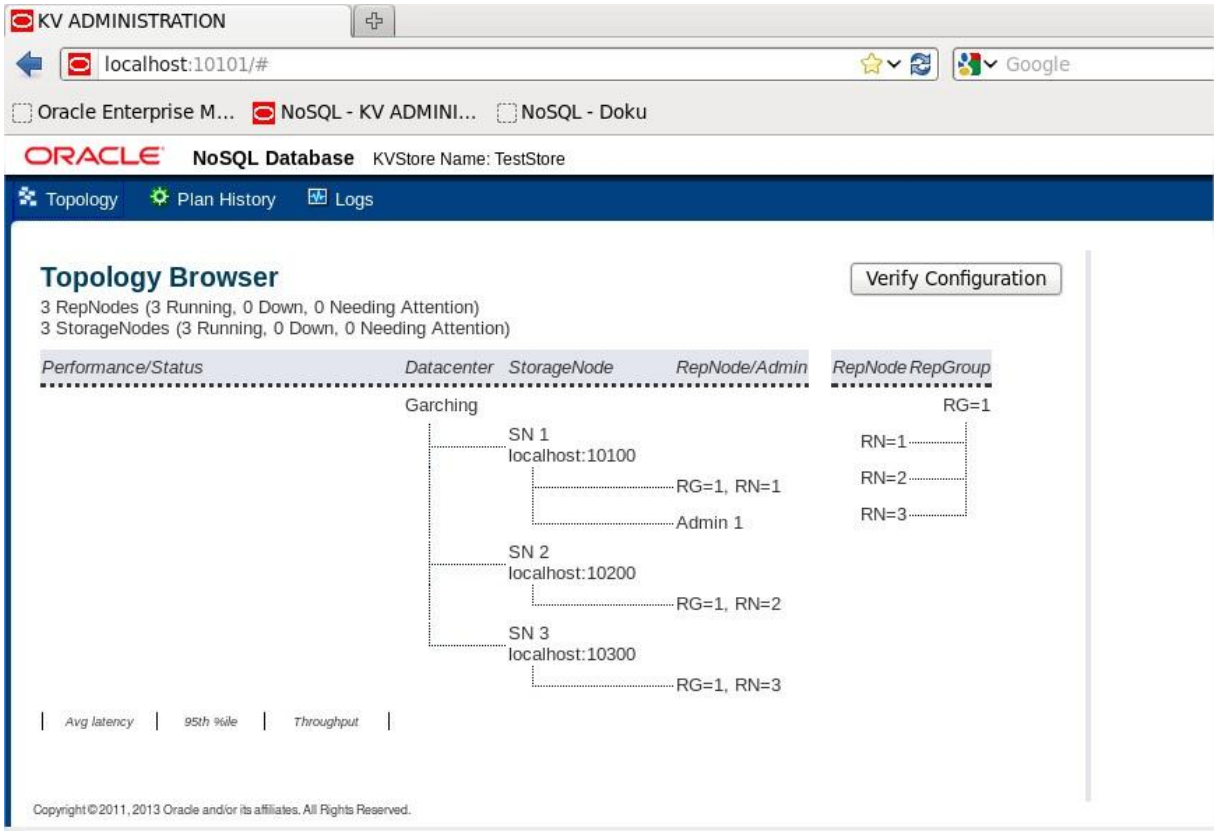

Abb. 3:Oracle NoSQL DB Dashboard

Die Zustandsinformationen eines jeden Storage Nodes werden durch den Storage Node Agent (SNA), der auf jedem Storage Node läuft, beobachtet und auf Anfrage an den Administrations Service gemeldet. Darüber hinaus verteilt der Administration Service noch Konfigurationen an die SNA.

Wie bereits erwähnt, kann ein Storage Node ein oder mehrere Replication Nodes beherbergen. Jeder Replication Node gehört einer Replikationsgruppe (Shard) an. Pro Shard übernimmt ein Replication Node die Aufgabe des Masters. Dieser ist dafür zuständig alle modifizierende Zugriffe (create, update,

delete) zu managen. Alle anderen Replication Nodes haben die Rolle eines Read-Only Replikats. Ein Replikationsfaktor von drei, d.h. ein Master und zwei Read-Only Replikate, ist ein typische Installationsart. So ist sichergestellt, dass Leseanfragen beantwortet werden können selbst wenn zwei Server ausfallen. Im Falle eines Ausfalls des Masters übernimmt einer der Read-Only Replikaten die Aufgabe des Masters.

Um Festzulegen, in welchem Shard die Daten abgelegt werden, wird für jeden Schlüssel eines Key-Value-Paares mit Hilfe einer Hash-Funktion eine Partition ermittelt. Die einzelnen Partitionen sind fest den jeweiligen Shards zugeordnet. Aufgrund der Nutzung von einer Hash-Funktion werden die Daten gleichmäßig über die einzelnen Storage Nodes verteilt. Es kann jedoch sinnvoll sein, gewisse Daten, die häufig zusammen abgefragt werden, wie zum Beispiel Name und Adresse eines Kunden nicht verstreut voneinander zu speichern. Die Oracle NoSQL DB gibt dem Entwickler die Möglichkeit zusammengehörige Daten in lokaler Nähe voneinander abzulegen, indem der Schlüssel in zwei Teile, einem Major und einem Minor Key, unterteilt ist. Anstelle der Schlüssel KundenName und KundenAdresse kann der Entwickler den Major Key KundenNummer verwenden und als Minor Key dann Name und Adresse. Die Hash-Funktion zur Ermittlung der Partition wird auf den Major Key angewandt. Somit stellt die Oracle NoSQL DB sicher, dass alle Daten, die zu einem Major Key gehören der gleichen Replikationsgruppe zugeordnet und auf den gleichen Servern gespeichert werden.

Die Anzahl der Storage Nodes lassen sich bei Bedarf anpassen. In dem Fall einer Erweiterung können zusätzliche Replikationsgruppen im KVStore konfiguriert werden. Die Oracle NoSQL DB balanciert dann die vorhandenen Partitionen zwischen den alten und neuen Shards aus. Die Anzahl der verwendeten Partitionen wird während der Installation festgelegt und kann im Nachhinein nicht mehr verändert werden.

Die komplette Beschreibung eines KVStores, also die vorhandenen StorageNodes, Shards, Replication Nodes und der Administration Service, wird als Topologie bezeichnet.

Installation

Bevor mit der Installation gestartet wird, müssen die Voraussetzungen geprüft werden. Als Betriebssystem auf den einzelnen Knoten wird Linux und Solaris 10 – am besten nicht in einer Virtual Machine – unterstützt. Darüber hinaus muss eine passende Java Version (für NoSQL DB 11.2 ist dies mindestens Java SE 6 (JDK 1.6.0 u25)) zur Verfügung stehen. Darüber hinaus muss noch sichergestellt sein, dass eine Zeitsynchronisation zwischen den einzelnen Knoten, z.B. mittels ntp, eingerichtet ist, um eine zeitliche Abweichung von weniger als einer halben Sekunden zu gewährleisten.

Die für die Installation benötigte Software kann im OTN heruntergeladen werden:
http://www.oracle.com/technetwork/products/nosqldb/downloads/index.html
Hierbei hat man die Wahl zwischen der Community Edition (kostenlos und mit OpenSource-Lizenz) oder der Enterprise Edition (kommerzielle Lizenz mit der Möglichkeit eines Wartungsvertrages). Derzeit besteht der Hauptunterschied dieser beiden Editionen nur in der Monitoring-Funktionalität und der Möglichkeit NoSQL-Daten direkt mittels einer externen Table in einer Oracle RDMS Datenbank zu verarbeiten.

In einer produktiven Umgebung, wird man sich an die verteilte Architektur halten, die im vorangegangenen Abschnitt beschrieben wurde. Zu Testzwecken ist es jedoch möglich, eine Oracle NoSQL Datenbank auf nur einem einzigen Knoten zu installieren. Im Folgenden werden die einzelnen Schritte beschrieben, die für eine Installation der Oracle NoSQL DB Community Edition 11gR2

(11.2.0.39) auf einen CDH4-Knoten (CDH4 = Cloudera Distribution for Hadoop Version 4) notwendig sind.

Im ersten Schritt muss die Software auf alle (hier also nur auf den einen) Knoten verteilt werden. Am besten wählt man dazu auf allen Knoten das gleiche Verzeichnis, hier: /opt/oracle/NoSQL. Es darf sich allerdings nicht um ein geteiltes Filesystem handeln. Jeder Knoten benötigt die Software auf einem eigenen Filesystem.

1a. Software unter /opt/oracle entpacken
```
cd /opt/oracle
tar –xvf /Downloads/kv-cd-2.0.39.tar
```

1.b Softlink auf das Verzeichnis mit Versionsnummer erstellen
```
ln –s kv-2.0.39 NoSQL
```

Im Home-Verzeichnis wurde die folgende Verzeichnisstruktur angelegt:

```
|-- src
|-- doc
| |-- AdminGuide
| |-- examples
| |-- GettingStartedGuide
| |-- javadoc
| |-- misc
| |-- RDFGraph
|-- examples
| |-- avro
| |-- externaltables
| |-- hadoop
| |-- hello
| |-- schema
|-- lib
|-- src
| |-- oracle
```

Als nächstes wird eine Umgebungsvariable für das Software-Home-Verzeichnis des KeyValue Stores erstellt:

```
KVHOME=/opt/oracle/NoSQL
```

Bei der Einrichtung der Oracle NoSQL Datenbank müssen eine Reihe von TCP/IP-Ports angegeben werden. Zum einen wird ein Listener-Port für die Kommunikation der Anwendung mit dem KVStore benötigt, zum anderen ein Port für die Admin-Konsole. Darüber hinaus wird noch eine Reihe von Ports benötigt, über die die interne Kommunikation der einzelnen Knoten miteinander stattfinden kann. Neben den Ports muss noch die Entscheidung getroffen werden, in welchem Verzeichnis ($KVROOT) die Key Value Stores gespeichert werden sollen. Für die Beispiel-Installation werden die Werte der folgenden Tabelle verwendet.

	Storage Node 1	Storage Node 2	Storage Node 3
$KVROOT = /opt/oracle/…	**KVROOT1**	**KVROOT2**	**KVROOT3**
Listener Port	**10100**	**10200**	**10300**
Admin-Konsole	**10101**	**10201**	**10301**
Interne Kommunikation	**10110 bis 10120**	**10210 bis 10220**	**10310 bis 10320**

2. Anlegen des Root-Verzeichnisses für die KVStores.

```
mkdir /opt/oracle/KVROOT1
mkdir /opt/oracle/KVROOT2
mkdir /opt/oracle/KVROOT3
```

3. Anlegen der Strukturen für die KVStores

```
java -jar $KVHOME/lib/kvstore.jar makebootconfig -root /opt/oracle/KVROOT1
-host localhost -harange 10110,10120 -port 10100 -admin 10101 -capacity 1 -
num_cpus 0 -memory_mb 0

java -jar $KVHOME/lib/kvstore.jar makebootconfig -root /opt/oracle/KVROOT2
-host localhost -harange 10210,10220 -port 10200 -admin 10101 -capacity 1 -
num_cpus 0 -memory_mb 0

java -jar $KVHOME/lib/kvstore.jar makebootconfig -root /opt/oracle/KVROOT3
-host localhost -harange 10310,10320 -port 10300 -admin 10101 -capacity 1 -
num_cpus 0 -memory_mb 0
```

4. Starten des Storage Node Agents

```
nohup java -jar $KVHOME/lib/kvstore.jar start -root /opt/oracle/KVROOT1 &
nohup java -jar $KVHOME/lib/kvstore.jar start -root /opt/oracle/KVROOT2 &
nohup java -jar $KVHOME/lib/kvstore.jar start -root /opt/oracle/KVROOT3 &
```

5. Starten des Administrations CLI auf einem Knoten

```
java -jar $KVHOME/lib/kvstore.jar runadmin -port 10100 -host localhost
```

Dieser 5. Schritt startet das Administration Command Line Interface (Prompt: kv->), das nun genutzt wird, um die NoSQL DB zu konfigurieren. Das Admin CLI wird mittels exit verlassen.

6. Name des KVStores festlegen

```
configure -name TestStore
Store configured: TestStore
```

7. Rechenzentrum konfigurieren

```
plan deploy-datacenter -name "Garching" -rf 3 –wait
Executed plan 1, waiting for completion...
Plan 1 ended successfully
```

8. Ersten Storage Node hinzufügen
```
plan deploy-sn -dc dc1 -host localhost -port 10100 –wait
Executed plan 2, waiting for completion...
Plan 2 ended successfully
```

9. Starten des Admin-Prozesses
```
plan deploy-admin -sn sn1 -port 10101
Started plan 3. Use show plan -id 3 to check status.
      To wait for completion, use plan wait -id 3
```

Informationen über den Step 3 anzeigen
```
show plan -id 3
Plan Deploy Admin Service (3)
State:                  SUCCEEDED
Attempt number:         1
Started:                2013-05-24 14:54:38 UTC
Ended:                  2013-05-24 14:54:38 UTC
Total tasks:            1
 Successful:            1
```

10. Einrichten des Storage Node Pools
```
pool create -name TestPool
```

Aktuelle Topologie anzeigen
```
show topology
store=TestStore  numPartitions=0 sequence=2
  dc=[dc1] name=Garching repFactor=3

  sn=[sn1]  dc=dc1 localhost:10100 capacity=1 RUNNING
```

11. Storage Node zum Pool hinzufügen
```
pool join -name TestPool -sn sn1
Added Storage Node(s) [sn1] to pool TestPool
```

12. Hinzufügen der weiteren Storage Nodes
```
plan deploy-sn -dc dc1 -host localhost -port 10200 –wait
plan deploy-sn -dc dc1 -host localhost -port 10300 -wait
```

Aktuelle Topologie anzeigen
```
show topology
store=TestStore  numPartitions=0 sequence=4
  dc=[dc1] name=Garching repFactor=3

  sn=[sn1]  dc=dc1 localhost:10100 capacity=1 RUNNING

  sn=[sn2]  dc=dc1 localhost:10200 capacity=1 RUNNING

  sn=[sn3]  dc=dc1 localhost:10300 capacity=1 RUNNING
```

13. Weitere Storage Nodes zum Pool hinzufügen
```
pool join -name TestPool -sn sn2 -sn sn3
Added Storage Node(s) [sn2, sn3] to pool TestPool
```

14. Topologie erstellen. Anzahl Partitionen bewusst wählen!
```
topology create -name topo -pool TestPool -partitions 10
Created: topo
```

15. Topologie deployen – Das eigentliche Anlegen des Key Value Stores

```
plan deploy-topology -name topo –wait
Executed plan 6, waiting for completion...
Plan 6 ended successfully
```

Übersicht der ausgeführten Konfigurations-Steps

```
show plans
     1 Deploy Datacenter (1)    SUCCEEDED
     2 Deploy Storage Node (2)  SUCCEEDED
     3 Deploy Admin Service (3) SUCCEEDED
     4 Deploy Storage Node (4)  SUCCEEDED
     5 Deploy Storage Node (5)  SUCCEEDED
     6 Deploy Topo (6)          SUCCEEDED
```

Verlassen des Admin CLI

```
exit
```

Den aktuellen Status der Oracle NoSQL Datenbank kann mit Hilfe des folgenden Java-Befehls angezeigt werden.

```
java -jar $KVHOME/lib/kvstore.jar ping -port 10100 -host localhost

Pinging components of store TestStore based upon topology sequence #18

TestStore comprises 10 partitions and 3 Storage Nodes

Storage Node [sn1] on localhost:10100    Datacenter: Garching [dc1]
Status: RUNNING   Ver: 11gR2.2.0.39 2013-04-23 08:28:13 UTC  Build id:
b205fb13eb4e

      Rep Node [rg1-rn1]      Status: RUNNING,MASTER at sequence number: 41
haPort: 10111

Storage Node [sn2] on localhost:10200    Datacenter: Garching [dc1]
Status: RUNNING   Ver: 11gR2.2.0.39 2013-04-23 08:28:13 UTC  Build id:
b205fb13eb4e

      Rep Node [rg1-rn2]      Status: RUNNING,REPLICA at sequence number: 41
haPort: 10210

Storage Node [sn3] on localhost:10300    Datacenter: Garching [dc1]
Status: RUNNING   Ver: 11gR2.2.0.39 2013-04-23 08:28:13 UTC  Build id:
b205fb13eb4e

      Rep Node [rg1-rn3]      Status: RUNNING,REPLICA at sequence number: 41
haPort: 10310

[mfhadoop@mfHadoopOracle oracle]$ java -jar $KVHOME/lib/kvstore.jar ping -
port 10200 -host localhost

Pinging components of store TestStore based upon topology sequence #18

TestStore comprises 10 partitions and 3 Storage Nodes

Storage Node [sn1] on localhost:10100    Datacenter: Garching [dc1]
Status: RUNNING   Ver: 11gR2.2.0.39 2013-04-23 08:28:13 UTC  Build id:
b205fb13eb4e
```

```
      Rep Node [rg1-rn1]      Status: RUNNING,MASTER at sequence number: 41
haPort: 10111

Storage Node [sn2] on localhost:10200    Datacenter: Garching [dc1]
Status: RUNNING   Ver: 11gR2.2.0.39 2013-04-23 08:28:13 UTC  Build id:
b205fb13eb4e

      Rep Node [rg1-rn2]      Status: RUNNING,REPLICA at sequence number: 41
haPort: 10210

Storage Node [sn3] on localhost:10300    Datacenter: Garching [dc1]
Status: RUNNING   Ver: 11gR2.2.0.39 2013-04-23 08:28:13 UTC  Build id:
b205fb13eb4e

      Rep Node [rg1-rn3]      Status: RUNNING,REPLICA at sequence number: 41
haPort: 10310
```

Alternativ kann der Status auch im Web-Browser angeguckt werden. Siehe hierzu *Abb. 3:Oracle NoSQL DB Dashboard*

Datenzugriff

Für den Zugriff auf die Oracle NoSQL Datenbank gibt es keine Abfragesrache wie SQL. Stattdessen erfolgt der Zugriff mittels API-Aufrufen. In der aktuellen Version der Oracle NoSQL Datenbank sind Programmier-APIs für JAVA verfügbar. In Planung sind C-APIs.
Die drei grundlegenden Operationen für das Managen von Key-Value Paaren sind

- PUT – Speichern eines Key-Value Paares
- GET – Abrufen eines Key-Value Paares
- DELETE – Löschen eines Key-Value Paares

Unabhängig von der Operation, die in dem Java Programm ausgeführt werden soll, muss sich das Programm zuerst mit der NoSQL Datenbank verbinden. Die Erzeugung eines Schlüssels erfolgt mit einer Instanz der Klasse oracle.kv.Key während eine Instanz der Klasse oracle.kv.Value für die Erzeugung eines Wertes verwendet wird. Anschliessend kann mit dem Aufruf von oracle.kv.KVStore.Put() das Key-Value Paar gespeichert werden.

Beispiel Code NoSQL_PUT.java:

```
import oracle.kv.*;
public class KV_Store {
   public static void main(String args[]) {
     KVStore store = KVStoreFactory.getStore (
       new KVStoreConfig(“TestStore”,
                         new String[] {“storagenode1:10100”,
                                       “storagenode3:10300”}
                         )
     );
     store.put(
        Key.createKey(“TestKey”),
        Value.createValue(new String(“Test Wert”).getBytes())
     );
     store.close();
```

```
    }
  }
```

Bei der Kompilierung mittels javac ist darauf zu achten, dass sich das Java-Archiv kvclient.jar aus dem Verzeichnis $KVHOME/lib im CLASSPATH befindet. Im ersten Schritt verbindet sich das Java-Programm mittels KVStoreFactory.getStore() mit der NoSQL Datenbank. Als Argument werden der Name des KVStores und mindestens einer der beteiligten Rechnerknoten übergeben. Der NoSQL Treiber arbeitet die Liste der übergebenen Rechnerknoten ab und verbindet sich mit dem ersten erreichbaren, um sich von dort die Informationen zur Topologie der NoSQL Datenbank zu holen. Im zweiten Schritt store.put wird mit Key.createKey eine Instanz der Klasse oracle.kv.Key erzeugt. Die Werte werden im KVStore als Bytes abgelegt. Daher muss der zu speichernde String zuerst mit getBytes() in ein Byte-Array umgewandelt werden. Mit Value.createValue wird aus dem Byte-Array eine Instanz der klasse oracle.kv.Value erzeugt. Im letzten Schritt store.close wird die Verbindung zur NoSQL-Datenbank geschlossen.

Die Abfrage eines Key-Value Paares erfolgt in ähnlicher Weise wie das Speichern. Zuerst muss wiederum eine Verbindung zur NoSQL Datenbank aufgebaut werden, dann wird wieder eine Instanz von oracle.kv.Key erzeugt. Anschließend wird mit der Methode oracle.kv.KVStore.get() der Wert abgefragt.

Beispiel Code NoSQL_GET.java:

```
import oracle.kv.*;
public class KV_Store {
   public static void main(String args[]) {
     KVStore store = KVStoreFactory.getStore (
       new KVStoreConfig("TestStore",
                         new String[] {"storagenode1:10100",
                                       "storagenode3:10300"}
                         )
     );
     ValueVersion vv store.get(
        Key.createKey("TestKey")
     );
     Value v = vv.getValue();
     System.out.println(new String(v.getValue()));
     store.close();
     }
  }
```

Auch bei dem GET-Beispielprogramm wird zunächst eine Verbindung zu der NoSQL Datenbank aufgebaut. Analog zum Speichern wird ein Schlüssel als Instanz der Klasse oracle.kv.Key erzeugt. Anschliessend wird das Key-Value Paar abgerufen. Hierbei ist zu beachten, dass die Methode get() eine Instanz von oracle.kv.ValueVersion zurück liefert. Neben dem eigentlichen Wert wird noch zusätzlich Informationen über die Version des Wertes zurück geliefert. Anhand dieser Versions-Information kann erkannt werden, ob gegebenenfalls ein veralteter Wert von einer Read-Only Replika gelesen wurde. Mit der Methode get.Value wird der konkrete Wert aus dem ValueVersion-Objekt gelesen. Da es sich hierbei um ein Byte-Array handelt, muss der Wert vor der Ausgabe noch in ein String umgewandelt werden. Der letzte Schritt ist wiederum das Schließen der Verbindung zur NoSQL Datenbank.

Wie im Abschnitt über das CAP-Theorem erwähnt überlässt die Oracle NoSQL Datenbank dem Entwickler die Steuerung der Datenkonsistenz (C) und Verfügbarkeit(A). Das Festlegen der Schreibkonsistenz einer Transaktion erfolgt mit einer Instanz von oracle.kv.Durability. Die Schreibkonsistenz kann entweder beim Aufbau der Verbindung zur NoSQL DB konfiguriert werden oder alternativ bei jedem Aufruf der Methode PUT. Sowohl für den Master als auch für die Replikas wird die Commit-Policy festgelgt. Als drittes wird im Durability Objekt die Replica Acknowledgement Policy konfiguriert. Mögliche Werte für Commit-Policy sind:

- SYNC – Warten bis der Storage Node die Transaktion in die Log-Datei geschrieben hat und diese auf die Platte synchronisiert hat
- WRITE_NO_SYNC – Warten bis der Storage Node die Transaktion in die Log-Datei geschrieben hat. Auf die Plattensynchronisation wird nicht gewartet.
- NO_SYNC – kein Warten

Für die Replica Acknowledgement Policy kann der Entwickler die folgenden Werte setzen:

- ALL – Warten bis alle Replikas das Schreiben der Transaktion bestätigt haben
- SIMPLE_MAJORITY – Warten bis eine einfache Mehrheit der Replikas die Transaktion geschrieben hat
- NONE – kein Warten auf die Replikas

Analog kann der Entwickler auch bei einem lesenden Zugriff steuern, ob die Datenkonsistenz oder die Performance wichtig ist. Hierzu steht dem Entwickler die Klasse oracle.kv.Consistency zur Verfügung. Mögliche Werte für die Consistency-Policy sind:

- ABSOLUTE – Es wird jeweils vom Master gelesen, damit sichergestellt ist, dass der aktuellte Wert gelesen wird
- TIME – Es kann von einer Replika gelesen werden, allerdings darf der Wert nicht älter sein als der Parameter für Time angibt
- VERSION – Es kann von einer Replika gelesen werden, allerdings darf der Wert nicht mehr als x Versionen alt sein.
- NONE_REQUIRED – Es kann ohne Berücksichtigung des Alters des Wertes von einer Replika gelesen werden

Kontaktadresse:
Ute Middendorf
Metafinanz-Informationssysteme GmbH
Leopoldstrasse 146
D-80404 München

Telefon: +49 (0) 89-360 531 0
Fax: +49 (0) 89-360 531 15
E-Mail ute.middendorf@metafinanz.de
Internet: www.metafinanz.de