Oracle NoSQL Database und MapReduce
- Ja wie geht das denn? -

Jascha Knack, Harm Knolle

Hochschule Bonn-Rhein-Sieg
Sankt Augustin

Schlüsselworte: Oracle NoSQL, MapReduce, Hadoop, Hive, Benchmark

Abstract: Ist die erreichbare Performance von „Key-Value“-Systemen zur Verwaltung schemaloser Daten tatsächlich so viel höher als die traditioneller relationaler Datenbanksysteme? Dieser Frage soll anhand eines speziellen Anwendungsszenarios auf der Grundlage strukturierter und unstrukturierter Daten nachgegangen werden. Im Vergleich treten die Oracle NoSQL Database als „Key-Value-System“ sowie die Oracle Database 12c als relationaler Vertreter gegeneinander an. Neben einer Beschreibung, wie sich „MapReduce“-Anfragen mittels des Hadoop-Frameworks auf Oracle NoSQL ausführen lassen, werden der Benchmark und die Resultate der Messungen erläutert, bewertet und interpretiert.

Einleitung

Die steigende Beliebtheit sogenannter NoSQL-Datenbanken hat mittlerweile zur Polarisierung der Datenbank-Community geführt. Befürworter weisen auf die einfache Administrierbarkeit der performance-fördernden horizontalen Skalierung schemaloser und einfach strukturierter Datenspeicher hin. Die relational denkenden Gegner bemängeln die abgeschwächte Konsistenz und das Fehlen deskriptiver Anfragesprachen, wodurch ein traditioneller „Join“ oft selbst programmiert werden muss. Aber werden hier letztendlich nicht „Äpfel“ mit „Birnen“ verglichen? Die relationalen Systeme lassen sich immer dann gewinnbringend einsetzten, wenn sich Daten anwendungsneutral modellieren und einheitlich normalisiert in fest vorgegebenen strukturierten Relationen speichern lassen. Nehmen jedoch die Geschwindigkeiten permanent einströmender Daten und die Vielfalt unstrukturierter Daten immer weiter zu, lohnt sich dagegen ein Blick in die Speichertechnik von NoSQL-Systemen. Die relationalen Systeme bieten speziell für unstrukturierte Attributwerte den Datentyp LOB (Large Object) an. Da die Werte dieser LOBs jedoch nur bedingt von den relationalen Systemen interpretiert werden können, sinkt die Performance, wenn LOBs im Rahmen der Zugriffe dennoch ausgewertet werden sollen. „Key Value“-Systeme, repräsentieren eine besonders einfache Speichertechnik der NoSQL-Systeme. Sie fokussieren sich auf den Spezialfall solcher Anwendungen, in denen es lediglich LOBs (Values) zu verwalten gilt, deren Inhalte ausschließlich über einen Schlüssel (Key) adressierbar sind. Durch den Wegfall eines relationalen Interpreters und die reine Konzentration auf die Speicherung von „Key Value“-Paaren lassen sich bei gleichzeitiger horizontaler Skalierung enorme Antwortzeiten erzielen.

In diesem Papier soll der Frage nachgegangen werden, wie sich „Key-Value“-Systeme gewinnbringend im Rahmen von MapReduce-Anfragen einsetzen lassen und wie sich diese Technik auf das Laufzeitverhalten im Vergleich zu einem relationalen System auswirken kann. Um die Praxistauglichkeit des Gesamtsystems zu evaluieren, wird ein einfaches E-Commerce-Szenario

entworfen. Zu den besonderen Anforderungen von E-Commerce-Anwendungen gehören Skalierfähigkeit und Hochverfügbarkeit.

Das NoSQL-Datenbanksystem Oracle NoSQL ist besonders für den verteilten Betrieb in einem Computer-Cluster konzipiert [4] und scheint für diesen Anwendungszweck prädestiniert zu sein. Das implementierte „Key-Value“-Datenmodell lässt jedoch nur einfache Selektionen auf das Schlüsselattribut zu [1]. In der NoSQL-Welt wird häufig ein MapReduce-Ansatz angewendet, falls sich die in einer Anfrage spezifizierten Werte nicht jeweils vollständig aus jeweils einem Datensatz ermitteln lassen, sondern aus mehreren Datensätzen abgeleitet bzw. berechnet werden müssen, die sich zudem auch noch in unterschiedlichen Tabellen befinden können. Dies ist beispielsweise der Fall, wenn normalisierte Daten im Rahmen einer Anfrage über Fremdschlüssel denormalisiert werden. Relationale Systeme lösen diese Aufgabenstellung u.a. mit Verbundoperation. Ein anderes Beispiel sind Aggregationen wie z.B. min oder max über unterschiedliche Datensätze einer Tabelle. In einem „Key-Value“-Umfeld „verstecken“ sich die benötigten Referenzen bzw. die zu aggregierenden Attributwerte allerdings im Value. Oracle NoSQL stellt jedoch kein explizites MapReduce-Framework zur Verfügung. Daher soll zunächst der Frage nachgegangen werden, inwieweit sich komplexere Anfragen wie Aggregationen und Verbundoperationen mit Hilfe von Apache Hadoop sowie dem dort enthaltenen MapReduce-Framework realisieren lassen. Durch die Integration von Apache Hive ließe sich drüber hinaus die Komplexität von MapReduce-Abfragen reduzieren und die Abfrageschnittstelle vereinheitlichen.

Die Leistungsbewertung erfolgt im Rahmen einer Gegenüberstellung mit dem relationalen Datenbanksystem Oracle 12c, das seine Stärken bei Verbund- und Aggregationsoperationen unmittelbar ausspielen kann, jedoch nur traditionell zentralisiert zum Einsatz kommen wird.

Oracle NoSQL

Das Oracle NoSQL-Datenbanksystem stellt eine „Key-Value“-Datenbank sowie ein Managementsystem mit entsprechenden Zugriffs- und Manipulationsmöglichkeiten für die Daten bereit. Bei der Entwicklung dieses Systems wurde ein besonderer Fokus auf Zuverlässigkeit, Skalierbarkeit und Verfügbarkeit gelegt [4]. Diese Eigenschaften werden im wesentlichen durch Verteilung und Replikation der Daten über ein Computer-Cluster erreicht. Ein solches Cluster muss dabei keine räumliche Lokalität aufweisen und kann mittels Standard-Hardware aufgebaut werden. Da das „Key-Value“-Datenmodell in der Basisvariante ausschließlich Zugriffe über den Schlüssel unterstützt, muss der Anwendung bekannt sein, wie dieser lautet. Eine direkte Selektion auf den Wertebereich ist nicht möglich und hätte eine Art „Full-Table-Scan“ mit anschließender Selektion durch die Anwendung zur Folge. Um bei Zugriffen dennoch die interne Struktur der Values berücksichtigen zu können, bietet Oracle NoSQL zusätzliche Strukturierungsmöglichkeiten an: JSON und Tabellen.

JSON-Unterstützung

Ab Version 2.x kann der Wert der “Key-Value“-Paare, durch Unterstützung des Serialisierungs-Frameworks Apache Avro, strukturiert in JSON abgelegt werden [5]. Die Strukturdefinition erfolgt durch spezielle Schemata, die dem Datenbanksystem sowie der Client-Anwendung bekannt gemacht werden müssen. Soll die Strukturdefinition im Betrieb erweitert werden, bietet Apache Avro die Möglichkeit der Schemaevolution. Anschließend können Client-Anwendungen mit verschiedenen Versionsständen der Schemata auf die identische Datenbank ohne Verluste oder Konflikte zugreifen.

D*ie Table-API*

In Version 3.x wurde mit der Table-API eine Möglichkeit eingeführt, Daten innerhalb von Oracle NoSQL tabellarisch abzulegen [6]. Die Table-API wird von Oracle empfohlen, da sie wichtige Funktionalitäten wie z.B. Sekundärindizes implementiert. Hierbei werden die Tabellen mittels einer SQL-ähnlichen DDL formuliert. Der Primärschlüssel muss mindestens eine Spalte der Tabelle enthalten. Der Key entspricht dann der Konkatenation aus Tabellenname und Primärschlüssel:

```
Key: TABLE / PRIMARY KEY                    Value: ROW CONTENT
```

Hier wird auch deutlich, dass die Datenstruktur keineswegs verändert wird, sondern weiterhin eine „Key-Value“-orientierte Datenbank im Hintergrund arbeitet. Das Managementsystem übernimmt die Abbildung mit Hilfe der Tabellendefinition. Sekundärindizes werden analog implementiert. Keys entsprechen dann einer Konkatenation aus Tabellen- und Spaltenname und dem gewünschten Attributwert. Als Values fungieren dann die gesuchten eindeutige Key-Werte:

```
Key: TABLE / COLUMN / COLUMNVALUE        Value: PRIMARY KEY
```

Wenn eine Selektion über eine Spalte erfolgt, die nicht Teil des Primärschlüssels ist, fragt das Datenbanksystem zunächst den oder die internen Primärschlüssel der gewünschten Spalte ab, um anschließend einen Direktzugriff auf die entsprechenden Datensätze zu ermöglichen.

Analog zum Avro Schema unterstützt auch die Table-API eine Evolution der Datenstruktur. Dies ist eingängig, da zusätzliche Spalten von der Client-Anwendung einfach ignoriert werden können. Die Abfragen werden in der Client-Anwendung über die Oracle NoSQL Table-API formuliert. Eine DML analog SQL existiert nicht. Darüber hinaus bietet Oracle NoSQL eine Schnittstelle zur Anbindung der Table-API an das Apache Hadoop Framework.

Oracle NoSQL und MapReduce

Um exponentiell wachsende Datenmengen über verteilte Systeme hinweg verarbeiten zu können, sind Verfahren nötig, die sich effizient parallelisieren lassen. Eines dieser Verfahren ist MapReduce. Das MapReduce-Programmiermodell ist bekannt aus funktionalen Programmiersprachen und bietet die Möglichkeit, massiv parallele Abfragen auf große Datenmengen auszuführen. Hierzu werden zwei Funktionen definiert: Map und Reduce. Die Map-Funktion wird dabei auf einen vorselektierten oder den gesamten Datenbestand ausgeführt und liefert eine Menge von Ergebnissen. Jeder, durch die Map-Funktion analysierte Datensatz, liefert dabei ein unabhängiges Teilergebnis zurück. Im Anschluss an die Map-Funktionen werden die jeweiligen Ergebnismengen mittels Reduce-Funktionen aggregiert und deren Ergebnisse zurückgeliefert. Die Anzahl aufeinander folgender Reduce-Prozesse ist nicht begrenzt. So können die zurückgelieferten Aggregate in weiteren Prozessschritten weiter verdichtet werden.

Dieses Programmiermodell bietet sich hervorragend zur parallelen und verteilten Berechnung komplexer Datenbankanfragen in einem horizontal skalierten Datenbankumfeld an. Wie einleitend bereits festgestellt, bietet Oracle NoSQL kein explizites MapReduce-Framework. Im Folgenden wird erläutert, wie sich MapReduce mit Unterstützung von Apache Hadoop und Apache Hive auch in einem Oracle-NoSQL-Cluster sinnvoll anwenden lässt. Die Abbildung 1 verdeutlicht das Zusammenspiel der Komponenten.

Apache Hadoop

Apache Hadoop ist ein Big-Data-Framework zur verteilten Verarbeitung großer Datenmengen [2, 7]. Das speziell für Hadoop entwickelte Dateisystem, Hadoop Distributed File System (HDFS), bietet Cluster-Funktionalität in Form von Verteilung und Replikation der Daten. Hadoop implementiert den MapReduce-Algorithmus. Dieser bietet durch sein funktionales Paradigma Schutz vor Deadlocks sowie Race Conditions, da nur mit unveränderlichen Variablen gearbeitet wird. Daher kann ein zeitaufwendiges Sperren der Datensätze entfallen, wodurch das Verfahren besonders für verteilte Systeme geeignet ist.

Apache Hive

Apache Hive ist eine Data-Warehouse Software, die Abfrage und Verwaltung großer Datenmengen unterstützt [2, 8]. Hive kann externe Tabellen anbinden, so auch solche, die mit der Table-API von Oracle NoSQL definiert wurden. Die an SQL angelehnte Abfragesprache HiveQL wird zur Laufzeit interpretiert und je nach Komplexität in MapReduce-Programme überführt. Diese werden anschließend an Hadoop übergeben. Somit kann Hive als Bindeglied zwischen Hadoop und Oracle NoSQL mit der Client-Anwendung angesehen werden, um Zugriffe auf den Datenbestand zu vereinheitlichen und zu erleichtern. Die Komplexität von zu implementierenden Map- und Reduce-Funktionen wird hinter der leicht verständlichen Abfragesprache HiveQL gekapselt. Zusätzlich optimiert der intelligente Cache von Hive wiederkehrende Abfragen. Um von Client-Anwendungen auf die Funktionalitäten von Hive zugreifen zu können, bietet der HiveServer2 eine JDBC Schnittstelle an. Diese kann in der bevorzugten Programmiersprache, z.B. Java, adressiert und mit SQL Anweisungen genutzt werden.

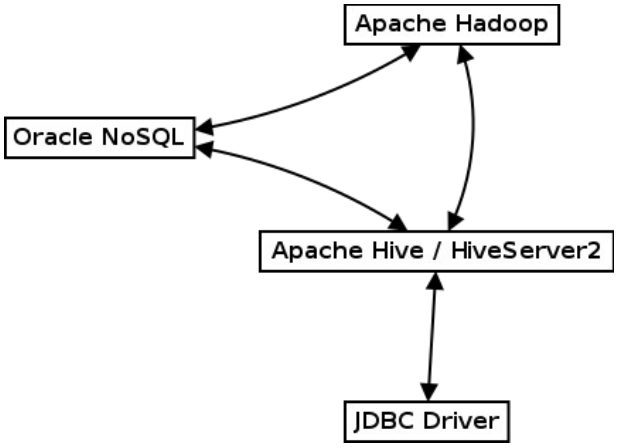

Abbildung 1: Zusammenspiel der Komponenten

Anwendungsbeispiel: Die E-Commerce-Plattform

In der Fallstudie wird der vereinfachte Ausschnitt einer E-Commerce-Plattform betrachtet. Das in Abbildung 2 dargestellte Schema beschreibt einen Artikelstamm mit einer nicht weiter strukturierten Artikelbeschreibung, einem Kundenstamm sowie die besuchten und gekauften Artikel.

Die beschriebene Datenstruktur wird auf dem relationalen System Oracle 12c direkt und auf dem nicht-relationalen Oracle NoSQL-Datenbanksystem mittels der Table-API im Rahmen einer Apache Hadoop und Hive Umgebung umgesetzt.

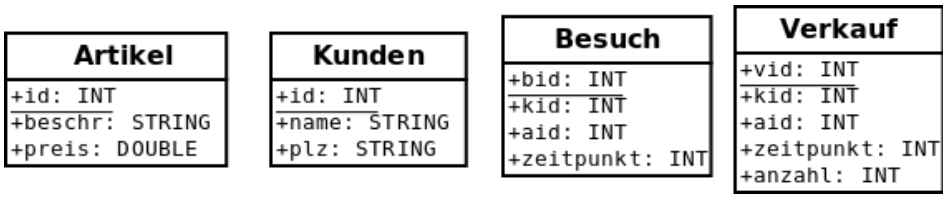

Abbildung 2: Datenschema E-Commerce

Installation

Als Testsystem fungiert ein Computer-Cluster, bestehend aus fünf identischen Rechnern (Abbildung 3). Auf vier Systemen ist jeweils Oracle NoSQL und Apache Hadoop installiert. Das fünfte System ist für Oracle 12c bestimmt.

Die Installation von Oracle NoSQL (Version: 3.4.7) beinhaltet das Entpacken des Archivs und die Cluster-Einrichtung. Der Zeitaufwand hierfür beträgt pro Cluster-Knoten ca. 10 Minuten. Die Installation von Apache Hadoop (Version 2.7.1) auf den vier NoSQL-Knoten erfordert neben dem Entpacken der Archivdatei die Anpassung diverser Konfigurationsdateien. Der Aufwand für den ersten Cluster-Knoten beträgt ca. 30 Minuten, während sich die folgenden Knoten innerhalb ca. 10 Minuten Aufwand konfigurieren lassen. Die Installation von Apache Hive (Version 1.0.1) erfolgt auf einem Knoten und benötigt ca. 10 Minuten (entpacken des Archivs und Anpassung der Konfigurationsdatei).

Die zentralisierte Installation von Oracle 12c auf einem Konten ist mit einem deutlich höheren Zeitaufwand von ca. 1 Stunde sowie weiteren Systemeingriffen wie z.B. Parameteranpassungen in der Systemdatei sysctl.conf verbunden. Allerdings müssen hier keine weiteren Systeme installiert werden.

Prozessor	Intel Core i7-4790K, 4.0 – 4.4 GHz
Arbeitsspeicher	32 GB DDR3-1866
Festplatte	Hitachi DeskStar
Netzwerk	1 GBit/s
Betriebssystem	CentOS 7

Abbildung 3: Cluster bestehend aus Standard-Hardware

Konfiguration von Hadoop und Hive im Zusammenspiel mit Oracle NoSQL

Zunächst müssen die entsprechenden Datenstrukturen für die E-Commerce-Plattform in der Oracle NoSQL-Datenbank angelegt werden. Anschließend wird das verteilte Dateisystem HDFS von Hadoop initialisiert, um so die Oracle NoSQL Datenstrukturen Hive bekannt machen zu können. Erst danach kann mittels Hive und der Abfragesprache HiveQL die Datenbank relational abgefragt werden.

Die Definition der Datenstrukturen erfolgt für Oracle NoSQL innerhalb des administrativen CLI [3] mittels der DDL der Table-API. Dieses wird exemplarisch für die Tabelle „Kunde“ gezeigt (die Erzeugung der weiteren Tabellen erfolgt analog):

```
EXEC "CREATE TABLE kunde
         (id INTEGER, name STRING, plz STRING, PRIMARY KEY (id))"
```

Vor dem ersten Einsatz zur Ablage der Daten im Computer-Cluster benötigt das verteilte Dateisystem von Hadoop eine Initialisierung:

```
$HADOOP_HOME/bin/hadoop namenode -format
```

Anschließend werden die Testdaten mittels einer, für dieses Szenario entworfenen, Applikation eingefügt. Die Einfügeoperationen erfolgen ebenfalls mittels der Table-API direkt gegen die Oracle NoSQL Datenbank.

Über eine Definition von externen Tabellen wird Apache Hive die in Oracle NoSQL definierte Datenstruktur bekannt gemacht. Dieses folgt exemplarisch für die Tabelle „Kunde“ (die Anlage der weiteren Tabellen findet analog statt):

```
CREATE EXTERNAL TABLE kunde( id INT, name STRING, plz STRING )
   STORED BY 'oracle.kv.hadoop.hive.table.TableStorageHandler'
   TBLPROPERTIES ( "oracle.kv.kvstore"="mystore",
                   "oracle.kv.hosts"="debian:5000",
                   "oracle.kv.hadoop.hosts"="debian",
                   "oracle.kv.tableName"="kunde" );
```

Nach diesem Schritt kann die installierte Umgebung einem ersten Funktionstest unterzogen werden. Hierzu wird im Hive CLI eine Abfrage an die Datenbank gestellt und deren Antwort ausgewertet. Das folgende Protokoll wurde stark gekürzt und enthält nur noch die wichtigsten Ausgaben von Hive:

```
hive> SELECT name, aid FROM kunde, verkauf
      WHERE plz < 10000 AND kid=id;
2015-11-13 11:45:46
   Starting to launch local task to process map join;
   maximum memory = 477626368
2015-11-13 11:45:49
   Dump the side-table for tag: 0
   with group count: 47
   into file: file:/[... path ...]/MapJoin-mapfile00--.hashtable
MapredLocal task succeeded
Launching Job 1 out of 1
Number of reduce tasks is set to 0 since there's no reduce operator
Hadoop job information for Stage-3:
   number of mappers: 1; number of reducers: 0
2015-11-13 11:45:58,843 Stage-3
   map = 0%,  reduce = 0%
2015-11-13 11:46:08,418 Stage-3
   map = 100%,  reduce = 0%, Cumulative CPU 10.1 sec
MapReduce Jobs Launched:
Stage-Stage-3: Map: 1   Cumulative CPU: 10.1 sec   HDFS Read: 554
OK
1XZVHLA3O0C8MD843XRUE8    120684
J651CY67S4X6WGZDNHYLL5    485216
Time taken: 27.649 seconds, Fetched: 2 row(s)
```

Betrachtet man die Ausgabe im Detail, so wird der Prozess deutlich: Zunächst wird die Anfrage von Hive interpretiert und bei Bedarf ein oder mehrere MapReduce Prozesse generiert. Diese werden an

Hadoop übergeben, welches anschließend die NoSQL-Datenbank adressiert. Komplexere Abfragen erzeugen Zwischenergebnisse. Diese werden temporär im HDFS abgelegt, um sie in den MapReduce-Prozessen verwenden zu können. Schließlich werden die Ergebnisse der MapReduce-Prozesse an Hive zurückgeliefert und über das CLI ausgegeben.

Die Leistungsbewertung

Für die Leistungsbewertung wurde eine spezielle auf YCSB basierende Benchmark-Anwendung in der Programmiersprache Java entwickelt. In den einzelnen Versuchen wird die Zeit gemessen, die ausgewählte Anfragen für die gesamte gesamte Bearbeitung benötigen Hierbei handelt es sich um die jeweiligen Datenbanksystemzugriffe ergänzt um die Zeit für Hadoop- und Hive-Operationen für Oracle NoSQL bzw. ergänzt um die Zeit für externe Auswertung unstrukturierter Daten durch die Anwendung für Oracle 12c. Die Datenbankzugriffe erfolgen mittels JDBC und werden intern, abhängig vom JDBC-Adapter bzw. dem Datenbanksystem, in relationale oder MapReduce Anfragen überführt. Für den Build-Prozess und zur Auflösung von Abhängigkeiten wird Maven verwendet (siehe Abbildung 4). Das folgende Listing zeigt die Nutzung der JDBC-Schnittstelle zur Anbindung an Hive.

```
import java.sql.*;
public class HelloHive {
   public static void main(String[] args)
      throws SQLException, ClassNotFoundException {
         Class.forName("org.apache.hive.jdbc.HiveDriver");
         Connection con = DriverManager.getConnection
            ("jdbc:hive2://debian:10000/default");
         ResultSet set = con.createStatement().executeQuery
            ("SELECT * FROM kunde LIMIT 5");
         while (set.next()) {
            System.out.println(String.valueOf(set.getInt(1))
               + '\t' + String.valueOf(set.getString(2)));
         }
      }
}
```

Benchmark 1 – Analyse von unstrukturierten Artikelbeschreibungen

In diesem Benchmark werden die Artikelbeschreibungen auf gleiche Wörter untersucht und die entsprechenden Vorkommen über alle Texte hinweg gezählt. Da hierbei nicht-standard Operationen genutzt werden, muss die Abfrage für Oracle12c und Apache Hive unterschiedlich gestaltet werden. In Apache Hive ist es mittels HiveQL möglich, die Anfrage komplett in das Datenbanksystem auszulagern. Aufgrund des in Hadoop nutzbaren MapReduce-Frameworks ist innerhalb der

Group Id	Artifact Id	Version
org.apache.hive	hive-jdbc	1.0.1
org.apache.hadoop	hadoop-common	2.7.1

Abbildung 4: Maven Abhängigkeiten

Anwendung keine weitere Auswertung erforderlich. Dieses führt zu einer deutlichen Entlastung der Anwendung. Die Abfrage in HiveQL ist im Folgenden dargestellt:

```
SELECT tupel.wort, count(1) AS anzahl
FROM (SELECT explode(split(beschr, '\s')) AS wort
      FROM artikel) tupel
GROUP BY tupel.wort ORDER BY tupel.wort;
```

Dagegen kann im relationalen Datenbanksystem Oracle 12c die Analyse des intern unstrukturierten Attributs der Artikelbeschreibung nicht weiter im Sinne der Anfrage interpretiert werden. Somit muss die entsprechende Anfrage über einen Full-Table-Scan erfolgen. Die Analyse der Artikelbeschreibung muss anschließend aufwendig von der Anwendung übernommen werden. Die analoge Abfrage lautet in SQL wie folgt:

```
SELECT beschr FROM artikel;
```

Wie aus Abbildung 5 ersichtlich wird, kann Oracle 12c trotz einem Full-Table-Scan und der externen Verarbeitung der Tupel durch die Anwendung einen deutlichen Vorteil gegenüber dem NoSQL-System erzielen.

Benchmark 2 – Aggregation und Verbund

Der Aggregations- und Verbund-Benchmark zählt die Anzahl der gekauften Artikel pro Kunde. Die Abfrage ist für HiveQL und SQL identisch:

```
SELECT SUM(anzahl), name
FROM verkauf, kunde WHERE kid=id GROUP BY name;
```

Abbildung 5 zeigt, dass auch in diesem Szenario die relationale Datenbank Oracle 12c deutlich in Führung geht.

Benchmark 3 - „Einfache“ Abfrage ohne MapReduce

Hier wird ein Full-Table-Scan auf die Kundentabelle ausgeführt. Die Abfrage ist für HiveQL und SQL ebenfalls identisch:

```
SELECT * FROM kunde;
```

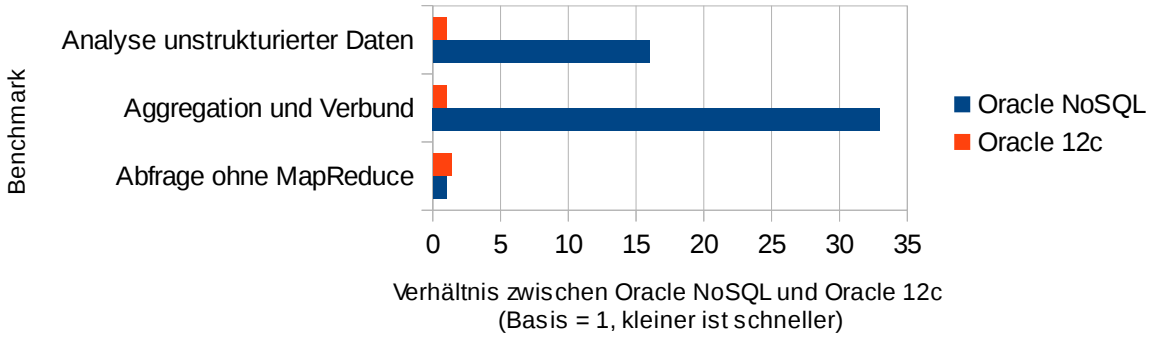

Abbildung 5: Zusammenfassung der Benchmarkergebnisse

In diesem Szenario kann Oracle NoSQL punkten und sich gegen den relationalen Vertreter behaupten. Der Unterschied ist knapp, jedoch in verschiedenen Messreihen reproduzierbar, wie die Abbildung 5 zeigt.

Auswertung der Leistungsbewertung

Aus der Leistungsbewertung geht offenbar hervor, dass sich Hadoop in der vorliegenden Konfiguration mit Oracle NoSQL nur bedingt für Echtzeitauswertungen eignet. Vielmehr bietet sich Oracle NoSQL mit Hadoop als Stapelverarbeitungssystem an, welches enorme Datenmengen verteilt verarbeiten kann. Dies zeigt sich insbesondere durch die verhältnismäßig hohen Antwortzeiten bei „kleinen“ Abfragen, in denen ein MapReduce-Prozess involviert ist. Auch größere Abfragen mit komplexen Verbund- und Gruppierungsoperationen werden ebenfalls eindeutig von der relationalen Datenbank dominiert. Das relationale Modell ist bestens für Abfragen dieser Art geeignet und es ist somit keine Überraschung, das Oracle 12c klar in Führung geht.

Auch bei der Analyse der Artikeltexte liegt Oracle 12c vorne, wenn auch nicht so deutlich. Obwohl der relationale Vertreter für diese Abfrage ausschließlich auf einen Full-Table- zurückgreift, bleibt die Oracle NoSQL Datenbank zurück. Der Overhead des Clusters sowie des Hadoop-Frameworks fallen hier offenbar negativ ins Gewicht.

Abfragen ohne Aggregation- oder Verbundoperationen beantwortet Hive ohne MapReduce-Prozesse. In diesen Fällen sind die Antwortzeiten vergleichbar. Dies ist einsichtig, da das unterliegende „Key-Value“-Datenmodell besonders für lesende Zugriffe mit bekanntem Schlüssel geeignet ist. Hier ist Oracle NoSQL klar im Vorteil.

Zusammenfassung und Ausblick

Es ist nicht einfach, schemalose „Key Value“-Datenbanken über MapReduce abzufragen, da vielfach doch eine gewisse Struktur der Daten vorausgesetzt werden muss. Der Widerspruch zeigt sich deutlich bei Verbundoperationen über fremdschlüsselartige Beziehungen. Hier bieten andere NoSQL-Kategorien wie z.B. die “Document Stores“ durch ihre modelgegebenen Strukturierungsmöglichkeiten mehr Potenzial. Außerdem ist bei solchen Systemen ein „Map Reduce“-Framework oft bereits integriert. Zur Analyse von enormen Datenbeständen bietet aber auch das analysierte „Key-Value“-System Oracle NoSQL in Kombination mit Hadoop und Hive u.a. interessante MapReduce-Möglichkeiten, sofern die Auswertung nicht in Echtzeit erfolgen soll. Das Apache Spark Projekt hat sich dieser Schwachstelle von Hadoop angenommen und bietet laut eigener Aussage eine bis zu 100-fache Geschwindigkeitssteigerung gegenüber Hadoop. Dies wird vornehmlich über „In-Memory“-Techniken erreicht, wohingegen Hadoop und das HDFS auf Sekundärspeicher zurückgreifen. Zur Durchführung von Analysen im operativen Tagesgeschäft werden jedoch auch die traditionellen und praktisch unverwüstbaren relationalen Datenbanksysteme weiterhin eine große Rolle spielen.

Literatur

[1] Edlich, Stefan; Friedland, Achim; Hampe, Jens; Brauer, Benjamin: NoSQL – Einstieg in die Welt nichtrelationaler Web 2.0 Datenbanken. München : Carl Hanser Verlag, 2010

[2] Freiknecht, Jonas: Big Data in der Praxis – Lösungen mit Hadoop, HBase und Hive. München : Carl Hanser Verlag, 2014

[3] Oracle: NoSQL Database Administrator's Guide – 12c Release 1. https://docs.oracle.com/cd/NOSQL/html/AdminGuide/Oracle-NoSQLDB-Admin.pdf, 2015

[4] Oracle: NoSQL Database Concepts Manual – 12c Release 1. https://docs.oracle.com/cd/NOSQL/html/ConceptsManual/Oracle-NoSQLDB-Concepts.pdf, 2015

[5] Oracle: Getting Started with NoSQL Database Key/Value API – 12c Release 1 https://docs.oracle.com/cd/NOSQL/html/GettingStartedGuide/Oracle-NoSQLDB-GSG.pdf, 2015

[6] Oracle: Getting Started with NoSQL Database Table API – 12c Release 1 https://docs.oracle.com/cd/NOSQL/html/GettingStartedGuideTables/Oracle-NoSQLDB-GSG-Tables.pdf, 2015

[7] The Apache Software Foundation: Hadoop Documentation https://hadoop.apache.org/docs/stable/, 2015

[8] The Apache Software Foundation: LanguageManual – Apache Hive https://cwiki.apache.org/confluence/display/Hive/LanguageManual, 2015

Kontaktadresse:

Jascha Knack, Prof. Dr. Harm Knolle
Hochschule Bonn Rhein Sieg
Fachbereich Informatik
Grantham-Allee 20
53757 Sankt Augustin

Telefon: +49 (0) 2241 865 201
Fax: +49 (0) 2241 865 8253
E-Mail jascha.knack@smail.inf.h-brs.de, harm.knolle@h-brs.de
Internet: http://www.inf.h-brs.de