In der letzten Ausgabe der DOAG *News* wurden unter der Überschrift „SQL oder NoSQL DB: Das ist die Frage" die Oracle NoSQL Datenbank und (kurz) das quelloffene Hadoop Distributed Filesystem (HDFS) als Systeme zum Speichern großer Mengen un- oder schwach strukturierter Daten mit geringer Informationsdichte vorgestellt. Dieser Artikel baut darauf auf und beschreibt, wie mit den so gespeicherten Daten sinnvoll gearbeitet werden kann.

Einführung für RDBMS-Kenner: Hadoop, MapReduce, Oracle Loader for Hadoop und mehr

Carsten Czarski, ORACLE Deutschland B.V. & Co. KG

Da sowohl NoSQL-Datenbanken als auch das HDFS keine Datenstrukturen kennen und folgerichtig keine Abfragesprache anbieten, ist eine andere Vorgehensweise erforderlich. Abbildung 1 zeigt stark vereinfacht die Einordnung von Big Data in eine IT-Landschaft. Der graue Bereich unten zeigt die traditionelle Vorgehensweise mit strukturierten Daten und hoher Informationsdichte: Daten werden in OLTP-Systemen erfasst, mit ETL-Prozessen in ein Data Warehouse geladen und stehen dann zum Reporting oder für Analyseprozesse zur Verfügung. Für Big Data, also für große Datenmengen mit geringer Informationsdichte, kommen diese Methoden jedoch nicht infrage – allein das Erstellen eines vernünftigen Tabellenmodells für solche Daten wäre schon ein extrem schwieriges Vorhaben (siehe Abbildung 1).

Daher werden diese Daten zunächst in Systemen abgelegt, die ohne Datenmodell auskommen und massiv parallel ausgelegt werden können (siehe Artikel in der letzten Ausgabe). Allerdings bieten diese Systeme folgerichtig keine Abfrage- und Analyse-Möglichkeiten, wie man sie von einem RDBMS her kennt. Zur Auswertung der Daten muss daher prinzipiell der gesamte Datenbestand durchgearbeitet werden – und wegen der riesigen Menge sind hier verteilte Systeme erforderlich.

Verteilte Verarbeitung mit Hadoop

Hadoop ist, kurz gesagt, ein Framework zur verteilten Datenspeicherung und -verarbeitung. Hadoop-Cluster können sehr groß werden. Die größten Installationen haben eine vierstellige Anzahl an Rechnerknoten. Hadoop besteht im Wesentlichen aus zwei Komponenten. Das Hadoop Distributed Filesystem (HDFS) stellt ein über die Rechnerknoten verteiltes Dateisystem bereit. Wie schon im Artikel in der letzten Ausgabe erwähnt, wird dieses zur dateiorientierten Speicherung von Big Data verwendet. Ist dagegen eher die satzorientierte Speicherung gefragt, sind NoSQL-Datenbanken besser geeignet.

Wie ein normales Dateisystem auf einem PC sieht auch HDFS eine Art „File Allocation Table" vor. Diese Datenstruktur enthält die Information, auf welchem Knoten im Cluster die einzelnen Blöcke einer Datei abgelegt sind, und wird auf einem besonderen Rechnerknoten, dem „Name Node", im Hauptspeicher gehalten. Ein Block im HDFS ist allerdings größer als auf einem PC – 256 MB sind der Normalfall. Der „Name Node" weiß, auf welchen „Data Nodes" die Blöcke einer HDFS-Datei liegen. Da in einem verteilten System immer einzelne Knoten ausfallen können, werden die Daten redundant gespeichert. Normalerweise arbeitet man mit einem Replikationsfaktor von drei – jeder Block ist im Cluster also dreimal vorhanden.

Der zweite Teil des Hadoop-Frameworks heißt „MapReduce" und ist ein Programmier-Framework zur ver-

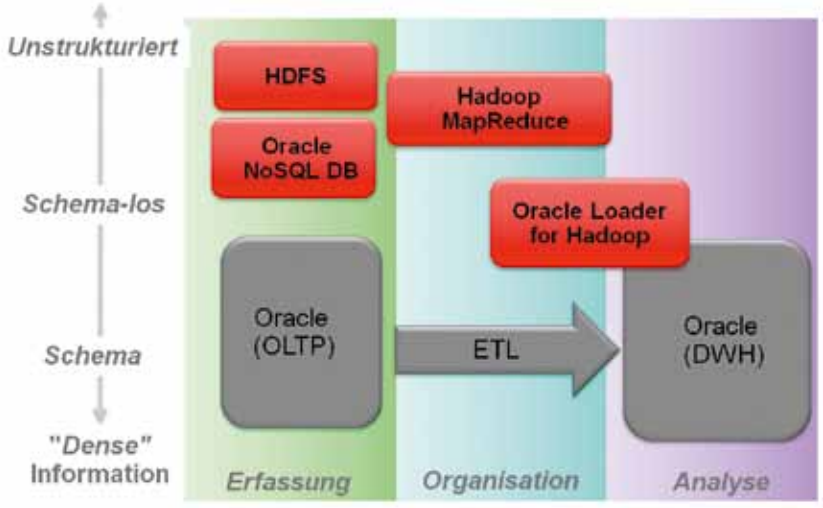

Abbildung 1: Neue Datenformen (Big Data) im Zusammenspiel mit dem Data Warehouse

teilten Verarbeitung. In einem MapReduce-Job sind Code und Datenfluss für die verteilte Verarbeitung optimiert. Der Entwickler eines MapReduce-Jobs kann sich allein auf die Geschäftslogik konzentrieren und muss keinerlei speziellen Code für die verteilte Verarbeitung schreiben.

Abbildung 2 zeigt das Zusammenspiel der Rechnerknoten in einem Hadoop-Cluster. Ein MapReduce-Job wird von einem Client-Rechner an den „Job Tracker" übermittelt. Dieser sorgt dann für die parallele Verarbeitung auf den einzelnen „Data Nodes". Wenn der Job mit Daten im HDFS arbeitet, weist der „Job Tracker" den „Data Nodes" die Teilaufgaben so zu, dass sie – soweit möglich – mit den Datenpaketen arbeiten können, die sie selbst halten. Zwischenergebnisse und temporäre Dateien werden wiederum ins HDFS geschrieben, sodass jeder Rechnerknoten in einem eventuell nachgelagerten Job darauf zugreifen kann.

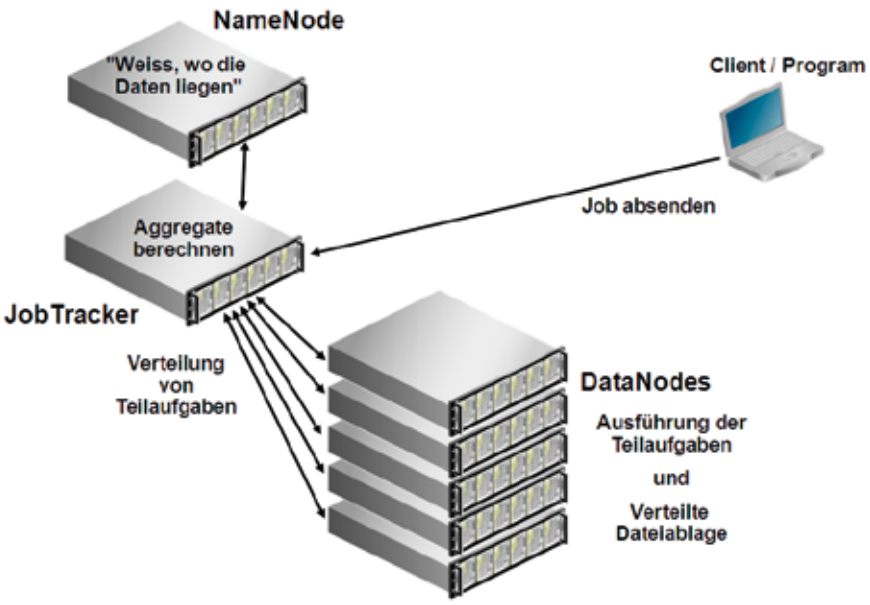

Abbildung 2: Schematische Architektur eines Hadoop-Clusters

MapReduce

MapReduce ist, wie bereits erwähnt, ein Programmier-Framework: Jede Art von Aufgabe kann als MapReduce-Job implementiert und dann verteilt ausgeführt werden. MapReduce schreibt nicht vor, was der Job zu tun hat, sondern vielmehr, wie dieser zu implementieren beziehungsweise wie der Code zu organisieren ist.

Technisch stellt sich das so dar, dass Interfaces vorgegeben sind, die dann vom Entwickler ausprogrammiert werden. Wie immer bei einem Interface sind die Signaturen, also die Ein- und Ausgabeparameter der Funktionen beziehungsweise Java-Methoden, vorgegeben.

In MapReduce-Jobs werden die Daten in Form von Key-Value-Paaren ausgetauscht. Der Entwickler bekommt diese als Eingabe in seinen Job und muss Key-Value-Paare wieder ausgeben. Dabei durchlaufen die Daten stets die folgenden Phasen:

1. Input in den MapReduce-Job
2. Mapper-Phase
3. Shuffle & Sort
4. Reducer-Phase
5. Ausgabe aus dem Job

Zur Veranschaulichung ein Beispiel: Angenommen, in der Oracle NoSQL Datenbank befinden sich 100.000 Key-Value-Paare. Als Schlüssel (Key) dient jeweils die hochgezählte Zahl zwischen 1 und 100.000, als Wert (Value) eine ganzzahlige Zufallszahl zwischen 1 und 100. Die Aufgabe des MapReduce-Jobs ist es nun, für jede Zahl zwischen 1 und 100 deren Vorkommen zu zählen.

```
Key             Value
--------------- -------------
1               67
2               12
3               1
:               :
99999           12
100000          56
```

Listing 1

```
public static class Map
extends Mapper <Text, Text, Text, IntWritable> {
  Text                              key =   new Text();
  private final static IntWritable value = new IntWritable(1);

  @Override
  public void map(Text keyArg, Text valueArg, Context context)
  throws IOException, InterruptedException {
    key.set(valueArg.toString());
    context.write(key, value);
```

Listing 2

```
Input                           Output
Key             Value           Key         Value
--------------- -------------   ----------- -----------
1               67              67          1
2               12              12          1
3               1               1           1
:               :               :           :
99999           12              12          1
100000          56              56          1
```

Listing 3

Eingabe in den MapReduce-Job

Alles beginnt mit der Übergabe der Quelldaten an den MapReduce-Job als Key-Value-Paare. Der Java-Entwickler verwendet hierfür die InputFormat-Klassen. Die vom Hadoop-Framework mitgelieferte Klasse „TextInputFormat" wandelt Dateien im HDFS in Key-Value-Paare um – der Entwickler bekommt dann ein Text-Fragment als „Value" und die Position in der Datei als „Schlüssel". Die Oracle NoSQL Datenbank liefert die Java-Klasse „KVInputFormat" mit, welche die Key-Value-Paare der NoSQL-Datenbank an den MapReduce-Job durchreicht. Für andere Quellsysteme kann man sich eigene InputFormat-Klassen schreiben. Im Beispiel liefert die Oracle NoSQL Datenbank folgende Key-Value-Paare (siehe Listing 1).

```
Key             Value
--------------- -------------
67              {1,1,1,1,...}
12              {1,1,1}
:               :
56              {1,1,1,1...}
```

Listing 4

```
public static class Reduce extends Reducer
  <Text, IntWritable, Text, IntWritable> {
  private IntWritable result = new IntWritable();

  public void reduce(
    Text key, Iterable<IntWritable> values, Context context
  ) throws IOException, InterruptedException {
    int sum = 0;
    for (IntWritable val : values) {sum += val.get();}
    result.set(sum);
    context.write(key, result);
  }
```

Listing 5

```
Input                          Output
Key             Value          Key         Value
--------------- -------------  ----------- -----------
67              {1,1,1,1,...}  67          617
12              {1,1,1}        12          3
1               {1,1,1,1,...}  1           132
:               :              :           :
56              {1,1,1,1...}   56          872
```

Listing 6

Mapper

Der Mapper ist eine eigene Java-Methode (genauer: eine eigene Java-Klasse) mit definierter Ein- und Ausgabeschnittstelle. Hier führt der Entwickler ein „Mapping" durch, indem er die empfangenen Key-Value-Paare auf neue Key-Value-Paare abbildet (siehe Listing 2).

Wichtig ist, dass der Entwickler zur gleichen Zeit stets nur ein Key-Value-Paar sieht. Somit kann ein Zusammenfassen zu diesem Zeitpunkt noch nicht erfolgen, wohl aber kann (und muss) der Entwickler dieses hier vorbereiten, indem er die Key-Value-Paare, die zusammengefasst werden sollen, auf neue Key-Value-Paare mit gleichem Schlüssel abbildet. Im Beispiel würde der Mapper die empfangenen Key-Value-Paare also wie in Listing 3 zeigen.

Da gezählt werden soll, wie oft jede Zufallszahl vorkommt, müssen die Key-Value-Paare anhand der Zufallszahl zusammengefasst werden – der Mapper legt also die Zufallszahl als neuen Schlüssel fest. Als Wert dient die Zahl „1" – denn die Zufallszahl ist „einmal vorgekommen". Wie schon gesagt, sieht der Mapper nur ein Key-Value-Paar auf einmal. Das ist die Grundlage für die verteilte Verarbeitung. Die eigentliche Zählung erfolgt später im Reducer.

Shuffle & Sort

Diese Funktion wird vom Hadoop-Framework übernommen – der Entwickler muss nichts machen. Es werden die vom Mapper ausgegebenen Key-Value-Paare nach Schlüsseln sortiert, zusammengefasst und dann an den ebenfalls vom Entwickler implementierten Reducer übergeben. Im Beispiel sieht das aus wie in Listing 4.

Shuffle & Sort fasst die 100.000 Key-Value-Paare zusammen, die der Mapper ausgegeben hat, sortiert sie und ordnet sie Schlüsseln zu. Ein Schlüssel (die Zufallszahl) kommt nun nur noch einmal vor und enthält alle Werte (im Beispiel nur 1er) als Array. In dieser Form gehen die Key-Value-Paare an den Reducer.

Reducer

Wie der Mapper ist auch der Reducer eine eigene Java-Methode mit einer definierten Eingabe- und Ausgabe-Schnittstelle. Der Entwickler bekommt die Schlüssel und Werte-Arrays aus der „Shuffle & Sort"-Phase übergeben und kann diese nun weiterverarbeiten. Dabei erzeugt er wiederum Key-Value-Paare aus der Ausgabe. „Zählen" bedeutet dabei das Durcharbeiten des Arrays und gleichzeitige Hochzählen einer Variable wie bei „cnt = cnt + 1". Für das Beispiel sieht der Code des Reducer wie in Listing 5 aus. Dieser Code wandelt die Key-Value-Paare um (siehe Listing 6).

Ausgabe aus dem MapReduce-Job

Der letzte Schritt ist wiederum die Ausgabe der vom Reducer generierten Key-Value-Paare. Verwendet der Entwickler die vom Hadoop-Framework mitgelieferte Klasse „TextOutputFormat", so werden die Key-Value-Paare als Textdatei im HDFS abgelegt. Die ebenfalls mitgelieferte Klasse „SequenceFileOutputFormat" legt die Paare im Binärformat im HDFS ab. Das ist sinnvoll, wenn ein nachgelagerter MapReduce-Job damit weiterarbeiten soll. Und na-

türlich kann ein Entwickler auch eigene OutputFormat-Klassen schreiben. Für gängige Aufgaben haben sich in der OpenSource Community mittlerweile Standard-Implementierungen durchgesetzt – man muss also nicht wirklich alles selbst machen. Einige Implementierungen sind nachfolgend kurz vorgestellt.

Hive

Hive ist ein sehr mächtiger, generischer MapReduce-Job, der es erlaubt, SQL-Abfragen auf Dateien im HDFS auszuführen. Auf den ersten Blick fragt man sich, wie das sein kann, denn es wurde ja schon mehrfach gesagt, dass NoSQL-Datenbanken und das HDFS eben keine SQL-Abfragesprache anbieten und das wegen fehlender Strukturen auch gar nicht können. Hive erlaubt die Definition einer Tabelle auf Basis einer Datei im HDFS – diese Tabelle wird ähnlich zu einer externen Tabelle in der Oracle-Datenbank definiert und funktioniert auch ganz ähnlich. Ist die Tabelle definiert, so kann sie mit SQL abgefragt werden – und Hive arbeitet hier wiederum ganz ähnlich wie die Oracle-Datenbank mit externen Tabellen. Die SQL-Abfrage wird geparst, es wird eine Art „Ausführungsplan" erstellt und die Ausführung erfolgt als MapReduce-Job, der die HDFS-Datei Zeile für Zeile durcharbeitet und dabei die in der SQL-Abfrage ausgedrückte Logik abarbeitet. Die Abbildung der Textdatei auf Zeilen und Spalten sowie das Filtern anhand der WHERE-Klausel erfolgt in der Mapper-Phase, Aggregats-Funktionen werden im Reducer abgebildet.

Enthält das HDFS Dateien von wohlbekannter Struktur (Log-Dateien), auf der man eine externe Tabelle definieren kann, so erlaubt Hive die Auswertung derselben mit SQL – was wesentlich einfacher und schneller ist als das Selbst-Schreiben eines MapReduce-Jobs. Allerdings wird nach wie vor der ganze Datenbestand durchgearbeitet. Auf Daten in der NoSQL-Datenbank kann Hive (noch) nicht arbeiten.

Sqoop

Diese Funktion bietet das Laden einer Datei im HDFS in eine relationale Datenbank per JDBC. Wie für Hive muss die HDFS-Datei auch für Sqoop eine gewisse Struktur aufweisen und es ist gegebenenfalls ein Mapping auf Spalten-Namen in einer Tabelle nötig. Sqoop parst die Datei, generiert die Spalten und Zeilen in der Mapper-Phase und schreibt sie in der Reducer-Phase per JDBC in die Datenbank. Analog dazu ist auch der umgekehrte Weg, also das Auslesen einer Datenbank-Tabelle per JDBC und die Ablage der Daten als Textdatei im HDFS, möglich.

Oracle Loader for Hadoop

Als Alternative zu Sqoop bietet Oracle als Teil der „Big Data Connectors" den „Oracle Loader for Hadoop" an. Im Gegensatz zum generischen Sqoop ist dieser für die Oracle-Datenbank optimiert und eignet sich besonders für das Laden großer Datenmengen in diese. Oracle Loader for Hadoop ist ebenfalls als MapReduce-Job implementiert und erlaubt zusätzlich zum Laden per JDBC auch das Batch-orientierte Laden in die Datenbank. Dafür kann beispielsweise eine Data-Pump-Datei erzeugt werden, die dann als externe Tabelle (Data-Pump-Format) in die Oracle-Datenbank eingebunden werden kann. Die Übernahme in die Zieltabellen kann dann mit den Mitteln der Datenbank (INSERT ... SELECT, Multi-Table INSERT oder Pipelined Functions) erfolgen. Gerade bei großen Datenmengen ist dieses Verfahren wesentlich effizienter.

Fazit

Letztlich hat sich der Kreis geschlossen: Big Data sieht zunächst vor, dass große Mengen un- oder nur schwach strukturierter Daten als Dateien ins HDFS oder als Key-Value-Paare in die Oracle NoSQL Datenbank gespeichert werden. Diese verteilten Systeme sind in der Lage, auch größte Datenmengen mit kurzen Antwortzeiten aufzunehmen und ständig zu wachsen.

Da hier keine Abfragesprache existiert, muss für jede Auswertung der gesamte Datenbestand durchgearbeitet werden. Diese Aufgabe übernimmt MapReduce – ebenfalls massiv parallel. Die gewünschten Auswertungen beziehungsweise Aggregationen werden als MapReduce-Jobs implementiert und im Hadoop-Cluster ausgeführt. Als letzter Schritt einer solchen Jobkette kann schließlich der Oracle Loader for Hadoop ins Spiel kommen, der die gefundenen Aggregate in die Oracle-Datenbank lädt, wo sie Teil des Data Warehouse und damit zur Basis für Reporting, Business Intelligence und weitere Analyse werden können.

Big Data Appliance

An dieser Stelle bietet sich die Einordnung der „Oracle Big Data Appliance" an. Dieses Engineered-System ist speziell für die Anforderungen eines Hadoop-Clusters oder der NoSQL-Datenbank ausgelegt. Im Gegensatz zur sehr aufwändigen Einrichtung eines Hadoop-Clusters mit Standard-Hardware findet der Setup einer Big Data Appliance skriptgesteuert in kürzester Zeit statt. Damit ist dieses Engineered-System hochinteressant, wenn man Big-Data-Technologien nutzen möchte, die Installation und den Betrieb eines Hadoop- oder NoSQL-Datenbank-Clusters auf „Commodity-Hardware" jedoch vermeiden möchte.

Weitere Informationen

[1] Oracle Dojo, Eine Einführung in Big Data: http://www.oracle.com/webfolder/technetwork/de/community/dojo/index.html
[2] Whitepaper Big data Overview: http://www.oracle.com/technetwork/server-storage/engineered-systems/bigdata-appliance/overview/wp-bigdatawithoracle-1453236.pdf
[3] Apache Hadoop (Map Reduce und HDFS):http://hadoop.apache.org
[4] Oracle NoSQL Datenbank im OTN: http://www.oracle.com/technetwork/products/nosqldb/overview/index.html

Carsten Czarski
carsten.czarski@oracle.com
http://twitter.com/cczarski
http://sql-plsql-de.blogspot.com