Big Data und SQL - das passt!

Philipp Loer
ORDIX AG
Paderborn

Schlüsselworte
Hadoop, Hive, Sqoop, SQL

Einleitung
In diesem Vortrag werden, nach einer kurzen Einführung in Apache Hadoop, die beiden Werkzeuge Hive und Sqoop vorgestellt. Im Anschluss werden in einer Live-Demo zunächst Daten aus einer Oracle-Datenbank mit Sqoop in Hive-Tabellen importiert. Anschließend werden einige Statements in der Hive Query Language abgesetzt.

Hadoop ist ein Big Data Framework für skalierbare, verteilte Software. Es besteht aus den beiden Komponenten Hadoop Distributed File System (HDFS) für eine verteilte Datenspeicherung und MapReduce für eine verteilte Datenverarbeitung.

Apache Hive wurde geschaffen, um Analysten mit guten SQL-Kenntnissen die Möglichkeit zu geben, die Daten zu untersuchen ohne sich zuvor die zur Erstellung eines MapReduce-Jobs erforderlichen Fähigkeiten und Kenntnisse aneignen zu müssen. Hive ist heute ein von vielen Unternehmen eingesetztes, erfolgreiches Apache-Projekt zur Verarbeitung unterschiedlicher Daten mit SQL.
Apache Sqoop ermöglicht es Daten aus relationalen Datenbanken in Hadoop zu importieren. Auch ein Export der Daten aus Hadoop in relationale Datenbanken ist möglich.

Apache Hadoop
Apache Hadoop ist ein Framework, das die verteilte Verarbeitung von großen Datenmengen auf einem Cluster mit einfachen Programmiermodellen ermöglicht. Genutzt werden hierzu aus handelsüblicher Hardware bestehende Knoten. Es wurde entwickelt, um von wenigen Knoten auf tausende nahezu linear skalieren zu können. Hierbei stellt jeder Knoten sowohl Rechenleistung als auch Speicherplatz zur Verfügung. Hadoop besteht im Kern aus zwei Komponenten: MapReduce für die Berechnungen und HDFS für die Speicherung von großen Datenmengen.
Das Hadoop Framework nimmt dem Programmierer Arbeit ab. Dieser muss nur noch die „Lücken“ (die Map- und Reduce-Funktionen, s. u.) füllen. Hadoop organisiert die Verteilung der Berechnung im Cluster und alle Aufgaben, die sich aus der Verteilung von Anwendungen ergeben. Die Berechnung selbst muss jedoch vom Programmierer implementiert werden. Hierbei stellt sich unweigerlich die Frage, welche Vorteile Hadoop bietet, wenn doch der Programmierer den eigentlichen Programmcode immer noch selbst entwerfen muss.
Hauptvorteil von Hadoop ist nicht die nahezu lineare Skalierbarkeit, sondern der vereinfachte Umgang mit großen Datenmengen. Das Hadoop Framework steuert die Daten und die Berechnungen im Cluster. Wenn man all dies selbst programmieren wollte, müsste man sich unter anderem um die folgenden Dinge kümmern:

- Koordination der Berechnungen der einzelnen Knoten
- Koordination der Datenverteilung über die Knoten im Cluster
- Erkennung von Fehlern
- Neustart fehlerhafter Berechnungen

Hadoop bietet eine praxiserprobte, robuste Lösung für all diese Probleme. Es reduziert den Aufwand der Entwicklung auf zwei Aufgaben: Erstens muss das Problem auf kleine Teile herunter gebrochen werden, damit es parallel in einem Cluster bearbeitet werden kann. Zweitens muss der Programmcode für diese Berechnung bereitgestellt werden. HDFS als Dateisystem von Hadoop bietet theoretisch unbegrenzten Speicherplatz. Daher hat sich um Hadoop ein ganzes Ökosystem von Technologien gebildet, um den Umgang mit großen Datenmengen weiter zu vereinfachen

Die Abbildung 1 zeigt die Einordnung der Knoten in einem Cluster. Sie unterteilen sich zunächst in Master- und Slave-Knoten. Slave-Knoten sind die „Arbeitspferde“ des Cluster. Sie speichern die Daten und nehmen die (Teil-)Berechnungen vor. Die Master-Knoten wiederum unterteilen sich nach ihren jeweiligen Aufgaben: Die Steuerung der Datenspeicherung sowie die Steuerung der Berechnungen.

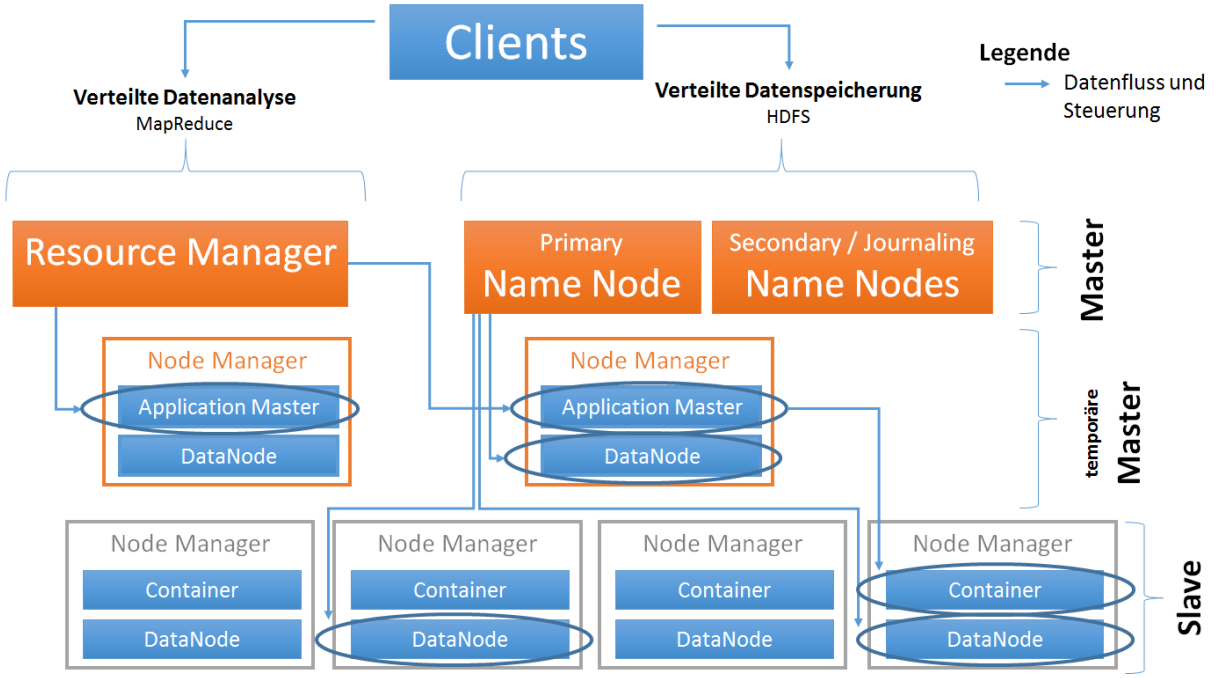

Abb. 1: Hadoop-Architektur

Hadoop Distributed File System

Dateien werden in HDFS in über mehrere Data Nodes verteilten Blöcken gespeichert. In einem Dateisystem bilden Blöcke üblicherweise die kleinste Speichereinheit. Der von einer Datei in einem Dateisystem belegte Speicherplatz ist folglich immer ein ganzzahliges Vielfaches der Blockgröße. Während in „General Purpose“-Dateisystemen die Blockgrößen üblicherweise im niedrigen Kilobyte-Bereich liegen, ist in HDFS die Blockgröße von 128 Megabyte (MB) der aktuelle Standard. Durch diese großen Blöcke werden der Verwaltungsaufwand und die Suchzeit in HDFS für den einzelnen Block reduziert. Durch die Aufgabenverteilung zwischen dem Name Node und den Data Nodes wird zudem eine in verteilten Dateisystemen vorteilhafte Blockabstraktion erreicht. Eine einzelne Datei kann so größer werden als die größte Festplatte des Dateisystems und es wird eine Vereinfachung des Speicher-Subsystems erreicht: Das Subsystem wird aufgrund seiner geringeren Komplexität weniger fehleranfällig. Darüber hinaus müssen keine Berechtigungsinformationen (Metadaten) auf Blockebene gespeichert werden. Diese lassen sich zentral im Data Node verwalten. Die Blockabstraktion vereinfacht zudem die Replikation. Einfache Blockkopien werden verteilt, um Datenverlust durch

Hardwarefehler zu vermeiden. Standardmäßig wird jeder Block auf drei verschiedenen Data Nodes gespeichert.

MapReduce-Programmiermodell

MapReduce Jobs bestehen aus drei Phasen: Der Map-Phase, der Shuffle-Phase und der Reduce Phase. Zweck der Aufteilung in diese drei Phasen ist die Ermöglichung der Verteilung der Berechnung auf mehrere Rechner für eine parallele Verarbeitung. Während der Map-Phase werden die Eingabedaten auf mehrere Map-Tasks aufgeteilt. Diese verwenden den vom Entwickler bereitgestellten Programmcode und laufen parallel ab. Die Map-Funktion verarbeitet eine Menge von Key/Value-Paaren und erzeugt hieraus eine Reihe. Diese Paare stellen ein Zwischenergebnis dar.

Die Zwischenergebnisse werden in der Shuffle-Phase je nach Zweck der Aufgabe in einem gemeinsamen oder in mehreren Zwischenspeichern abgelegt. Alle Key/Value-Paare mit gleichem Schlüsselwert werden immer in demselben Zwischenspeicher abgelegt. Die Anzahl der Zwischenspeicher ist durch die Zahl der im nachfolgenden Schritt zu startenden Reduce-Prozesse begrenzt. Für jeden der zu startenden Reduce-Prozesse wird ein Zwischenspeicher angelegt.

Während der Reduce-Phase wird für jeden Zwischenspeicher ein Reduce-Task gestartet. Jeder Reduce-Task berechnet aus den Zwischenergebnissen eine Liste von Endergebniswerten, die in der Ausgabeliste als Key/Value-Paare gesammelt werden.

Alle Map-Berechnungen sind untereinander unabhängig und können parallel in einem Cluster ausgeführt werden. Die Reduce-Berechnungen hängen zwar von den Ergebnissen der Map-Phase ab, sind aber untereinander ebenfalls unabhängig. Sind alle Zwischenergebnisse aus der Map-Phase berechnet, können sie ebenfalls parallel auf verschiedenen Knoten berechnet werden.

Apache Hive

Hive wurde ursprünglich 2007 von Facebook entwickelt, um folgendes Problem zu lösen: Die enormen Datenmengen, welche bei Facebook jeden Tag anfallen, sollten in Hadoop gespeichert werden. Leider bot Hadoop zum damaligen Zeitpunkt keine Unterstützung für SQL. Daher wurde Hive geschaffen, um Analysten mit guten SQL-Kenntnissen die Möglichkeit zu geben, die Daten zu analysieren, ohne sich zuvor die zur Erstellung eines MapReduce-Jobs erforderlichen Fähigkeiten und Kenntnisse aneignen zu müssen.

SQL ist nicht unbedingt die ideale Lösung für jedes Big-Data-Problem. So ist es beispielsweise für machine learning nicht geeignet. Für viele Analysen, Aggregationen und Abfragen ist es hingegen eine sehr gute Lösung. Der größte Vorteil von SQL ist jedoch seine Verbreitung in der IT und bei den Business-Intelligence-Werkzeugen. Durch die Verwendbarkeit von SQL kann Hadoop gut mit diesen Produkten interagieren. Der von Hive verwendete SQL-Dialekt heißt Hive Query Language (HQL)

Hive ist nicht für OLTP-Workloads bestimmt, es bietet keine Unterstützung für Transaktionen, Echtzeitabfragen oder Constraints. Das Haupteinsatzgebiet sind riesige Datenmengen, bei denen nur Daten hinzugefügt werden.

Das von den Nutzern erstellte SQL-Statement wird durch den Hive Interpreter im Map Reduce Jobs umgewandelt. Die Erstellung von Joins oder Gruppierungen ist somit deutlich einfacher als die Erstellung von normalem MapReduce-Programmcode. Traditionelle Datenbanken prüfen beim Hinzufügen von Daten, ob diese zur Tabellendefinition passen (schema on write). Passen die Daten nicht, so werden sie abgelehnt. Hive hingegen prüft die Daten nicht zum Zeitpunkt des Einfügens, sondern erst wenn diese gelesen werden. Diese Vorgehensweise wird „schema on read“ genannt. Dadurch müssen die Daten nicht vor dem Speichern überprüft werden und können schneller gespeichert werden. Das Hinzufügen von Datensätzen ist daher nur ein kopieren oder verschieben. Zudem biete diese Vorgehensweise den Vorteil Daten speichern zu können, deren Struktur zum Zeitpunkt der Speicherung noch unbekannt ist.

Die Architektur von Hive ist in Abbildung 2 dargestellt. Hive bietet ein Command Line Interface (beeline), eine web-basierte Benutzeroberfläche sowie einen eine Thrift-API. Zur Ausführung von

HQL können alle drei Schnittstellen genutzt werden. Verbindungen über JDBC, ODBC oder programmiersprachenindividuelle Schnittstellen können nur über Thrift hergestellt werden.
Das HQL-Statement wird zur Ausführung von der Schnittstelle an den Hive Driver übergeben. Daraufhin erstellt der Compiler auf Anforderung des Hive Driver einen Ausführungsplan. Hierzu wird der Metastore nach den Metadaten (s . u.) befragt. Der vom Compiler erstellte Ausführungsplan wird anschließend vom Hive Driver an die Execution Engine übermittelt. Diese erstellt daraus einen oder mehrere MapReduce-Jobs, welche an den Hadoop Cluster zur Ausführung übermittelt werden. Die Ergebnisse gelangen dann über die Execution Engine und den Hive Driver an die Benutzerschnittstelle. Mit der Hive CLI (bzw. seit neuestem beeline) bietet Hive eine Kommandozeile für die Ausführung von HQL-Statements. Der Aufruf erfolgt über den Befehl hive bzw. beeline.
Eine Hive-Datenbank entspricht dem, was in vielen relationalen Datenbanken unter einem Schema verstanden wird. Innerhalb einer Datenbank können Tabellen angelegt werden. Die Standard-Datenbank liegt im Hive-Warehouse-Verzeichnis im HDFS (/user/hive/warehouse/). Alle anderen Datenbanken liegen ebenfalls hier, jedoch in separaten Unterverzeichnissen. Tabellen einer Datenbank liegen in jeweils eigenen Unterverzeichnissen.
Der Hive Metastore speichert die Zuordnung in HDFS gespeicherter Dateien zu den Hive-Tabellen. Darüber hinaus enthält er alle Informationen, die für eine De-/Serialisierung der Daten benötigt werden. Dieser Systemkatalog vergrößert somit die Flexibilität und ermöglicht einen gewissen Abstraktionsgrad. Der Nutzer muss nur den Tabellennamen wissen, nicht aber in welchen Dateien die Daten liegen, welches Speicherformat verwendet wird und wie die interne Struktur der Dateien aufgebaut ist. Standardeinstellung für den Hive Metastore ist eine Apache-Derby-Datenbank auf dem Client. Wird Hive von mehreren Benutzern verwendet, empfiehlt sich die Einrichtung eines gemeinsamen Metastore. In der Regel wird hierzu MySQL oder eine andere relationale Datenbank verwendet.

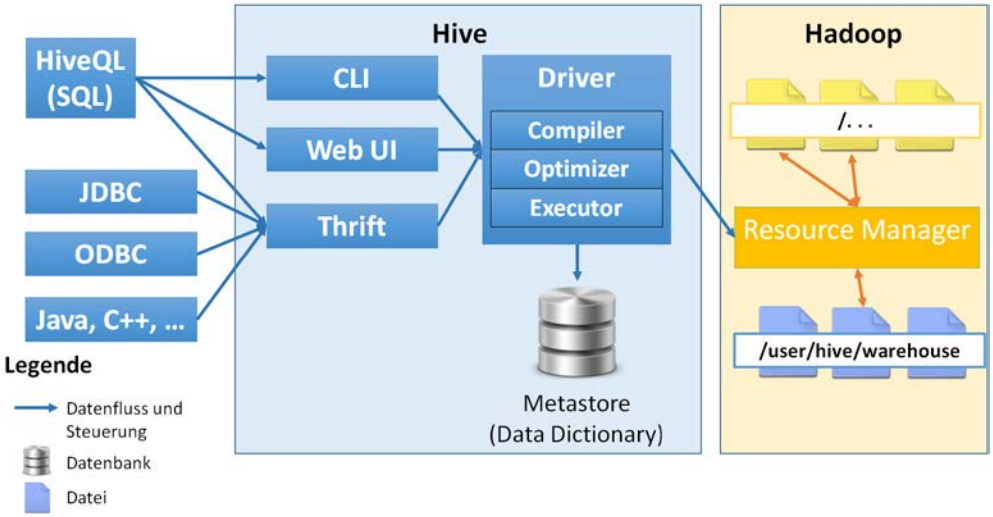

Abb. 2: Hive-Architektur

Apache Sqoop
Sqoop ermöglicht es Nutzern Daten aus Datenbanken in Hadoop zu importieren. Auch ein Export der Daten aus Hadoop in Datenbanken ist möglich. Sqoop ist ein client-basiertes Tool. Eine Installation im Hadoop Cluster ist daher nicht erforderlich. Es wird lediglich ein Client mit Zugang zum Cluster benötigt.

Der **Import** von Daten kann mit dem Befehl `sqoop import` gestartet werden. Hierbei muss angegeben werden, mit welcher Datenbank eine Verbindung aufgebaut und welche Tabelle aus dieser Datenbank importiert werden soll. Optional kann eine `where`-Bedingung zur Einschränkung der Menge an Datensätzen sowie die Anzahl der zu startenden Map-Tasks spezifiziert werden. Da bei einem Import der Daten diese lediglich 1:1 nach Hadoop übernommen werden sind keine Reduce-Tasks erforderlich. Der MapReduce-Job für den Import (und auch für den Export, s. u.) besteht daher nur aus einzelnen Map-Tasks. Standardmäßig werden vier Map-Tasks für den Import gestartet. Es ist aber optional möglich, die Zahl der zu startenden Tasks vorzugeben. Werden mehrere Map-Tasks für einen parallelen Import gestartet, so wird zunächst auf einer passenden Spalte der zu importierenden Tabelle der minimale und maximale Wert bestimmt. Falls eine Primärschlüsselspalte existiert wird diese genommen, ansonsten eine beliebige andere mit möglichst vielen unterschiedlichen Werten. Dieser Wertebereich wird in mehrere, gleich große Abschnitte unterteilt und dann jedem Map-Task ein eigener Abschnitt für den Datenimport zugewiesen. Andere in Datenbanken verfügbare Informationen, wie z. B. Histogramme und andere Statistiken, werden für die Festlegung eines solchen Abschnitts nicht herangezogen. Daher kann es zu einer ungleichen Verteilung der zu importierenden Datenmenge zwischen den einzelnen Map-Tasks kommen, wenn die Werte der gewählten Spalte nicht gleichmäßig verteilt sind.

Der genaue **Ablauf eines Datenimports** ist in Abbildung 3 dargestellt. Zunächst nutzt Sqoop den JDBC-Konnektor, um das zu importierende Objekt zu analysieren: Die Spaltenliste und die Datentypen der Spalten werden ermittelt. Sodann werden die SQL-Datentypen auf Java-Datentypen gemappt (z. B. `VARCHAR2` zu String). Diese Informationen werden dann genutzt, um eine Record-Container-Klasse zu generieren. Sqoop nutzt diese Java-Klasse zur Deserialisierung der importierten Daten bevor sie im HDFS abgelegt werden können. Diese generierte Klasse kann eine Zeile aus der Datenbanktabelle aufnehmen und mit MapReduce bearbeiten. Der Name der Klasse ist der Tabellenname, sofern beim Aufruf des Import-Befehls nichts anderes angegeben wurde. Sie liefert `get`-Methoden für jedes Feld der importierten Zeile und stellt Methoden zur Verfügung, welche die Serialisierungsmethoden aus dem DBWriteable-Interface implementieren, um mit JDBC interagieren zu können. Das JDBC-Result-Set-Interface stellt einen Cursor zur Verfügung, der die Zeilen der Abfrage liefert. Anschließend sendet Sqoop den Job an den Hadoop Cluster. Die Map-Tasks führen die Abfrage aus, deserialisieren die Zeilen aus dem Result Set in Instanzen der generierten Klasse, um diese entweder direkt in SequenceFiles oder nach einer erneuten Serialisierung in einer Comma Separated Values (CSV)-Datei in HDFS zu speichern.

Sobald die Daten geladen sind kann mit ihnen z. B. mit MapReduce-Jobs gearbeitet werden. Um mit einzelnen Feldern einer importierten Zeile arbeiten zu können, muss diese nach den Feldtrennzeichen geparst werden. Die einzelnen Werte müssen gelesen und in passende Datentypen konvertiert werden, da in den Dateien grundsätzlich jeder Wert als String abgelegt ist. Die beim Import generierte Java-Klasse ist in der Lage diesen Ablauf zu automatisieren. Sie bietet mehrere überladene Methoden mit dem Namen `parse()`, die auf dieser textuellen Repräsentation der Daten arbeiten und die Suche nach dem passenden Feld inkl. der Umwandlung in einen Java-Datentyp übernehmen können.

Mit Sqoop importierte Daten stammen aus einem relationalen Speicher. Daher macht es Sinn mit Werkzeugen wie Hive zu arbeiten, welche auf die Verarbeitung relational gespeicherter Daten ausgelegt sind, Joins unterstützen und als Sprache SQL verwenden. Hierzu gibt es drei Möglichkeiten: Entweder die Daten werden mit Sqoop direkt in eine Hive-Tabelle geladen. Werden sie außerhalb von Hive in HDFS abgelegt, kann auf die Daten durch Anlage einer external Table in Hive zugegriffen oder die Daten müssen nach Hive importiert werden.

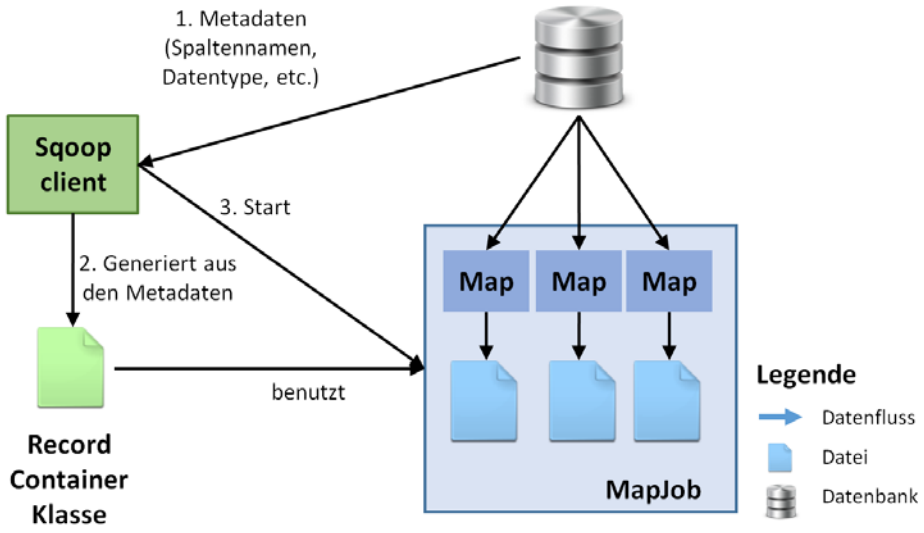

Abb. 3: Sqoop-Import

Die Speicherung der importierten Daten erfolgt standardmäßig in einer CSV-Datei. Das zu verwendende Trennzeichen (delimiter) kann bei dem Import mitgegeben werden. Ebenso welche Maskierungszeichen verwendet werden sollen, um das als Feldtrennzeichen verwendete Zeichen auch innerhalb des Feldes zu ermöglichen. Vorteilhaft an der Speicherung in Textdateien ist, dass ihr Inhalt lesbar ist. Sie können jedoch keine BLOBs speichern. Neben der Speicherung in Textdateien ist auch eine Speicherung in SequenceFiles, Avro oder Parquet möglich. Hier erfolgt die Speicherung binär und diese Formate erlauben die Speicherung von BLOBs. Darüber hinaus ermöglichen sie trotz Komprimierung eine parallele Verarbeitung durch MapReduce-Jobs. Ein direktes Laden in Hive ist nur für Parquet möglich. Im Avro-Format gespeicherte Daten können nicht direkt in Hive geladen werden, ein nachträglicher Import ist jedoch möglich.

Ein **Sqoop-Export** ist das Gegenteil des Imports: Gemeint ist hier der Transfer von Daten aus Hadoop in eine Datenbank. Vor dem Start des Exports muss zunächst die Datenbank vorbereitet werden: Die Tabelle, in welche die Daten geschrieben werden sollen, muss inklusive aller Spalten und mit den passenden Datentypen angelegt werden. Beim Import in eine Hive-Tabelle kann Sqoop das Anlegen der Tabelle übernehmen. Für den Export funktioniert dies leider nicht, da SQL-Datentypen wesentlich präziser sind als Java-Datentypen. Sqoop kann nicht wissen, ob aus einem String in Java eine `CHAR`- oder `VARCHAR`-Spalte werden soll und wie viele Zeichen oder Bytes diese aufnehmen können muss. Daher muss die Tabelle zwingend vor dem Export der Daten angelegt werden. Gestartet wird der Export über den Befehl `sqoop export`. Hierbei muss zusätzlich die Verbindung zur Datenbank, in die der Export erfolgen soll, sowie der Name der Tabelle in dieser Datenbank mit angegeben werden.

Sqoop 2 ist eine komplette Neuimplementierung von Sqoop, um die architektonischen Begrenzungen von Sqoop 1 zu umgehen. Sqoop 1 ist ein kommandozeilenbasiertes Werkzeug und besitzt keine Java-API. Daher ist eine Integration in andere Programme schwierig. Des Weiteren muss in Sqoop 1 jeder Konnektor jedes der verschiedenen Output-Formate unterstützen. Die Entwicklung eines neuen Konnektors ist daher mit großem Aufwand verbunden. Mit Sqoop 2 wird ein eigener Sqoop Server eingeführt, der die Jobs überwacht und mehrere Clients unterstützt: Ein Kommandozeilen-Interface, eine Java-API, eine Representational State Transfer (REST)-API sowie eine web-basierte Benutzeroberfläche. Sqoop 2 befindet sich aktuell noch in der Entwicklung und hat momentan noch nicht alle Merkmale die Sqoop 1 bietet. So gibt es noch keine Unterstützung für Hive oder einen Fast

Connector für Datenbanken wie Oracle oder MySQL. Ein produktiver Einsatz wird daher von Apache noch nicht empfohlen (Stand August 2015, gültig für Versionsnummer 1.99.6).

Kontaktadresse:
Philipp Loer
ORDIX AG
Westernmauer 12-16
D-33098 Paderborn

Telefon:	+49 (0) 5251 / 1063-0
Fax:	+49 (0) 1801 / 67439 - 0
E-Mail	info@ordix.de
Internet:	www.ordix.de