Oracle und Big Data –
Zentrale Techniken des Software-Stacks erklärt

Philipp Krätzig
ARETO Consulting GmbH
Köln

Schlüsselworte
Big Data, NoSQL, Key/Value, parallel, verteilt, Hadoop, HDFS, MapReduce, Fehlertoleranz

Einleitung
Nachdem Big Data Analytics zunehmend als ein entscheidender Wettbewerbsfaktor wahrgenommen wird, sind eine Vielzahl an Software-Tools erschienen, die den Anspruch erheben, speziell auf die Anforderungen von Big Data (Volume, Velocity, Variety, Variability) zugeschnitten zu sein. Aufgrund der unterschiedlichen Anforderungen unterscheidet sich die Infrastruktur, die diese Tools für die Datenerfassung, Datenorganisation und Datenanalyse bereitstellen, deutlich von der im klassischen Data-Warehouse. So wie dort kristallisieren sich allerdings auch bei Big Data einige grundlegende Paradigmen als besonders zweckmäßig heraus, die dann von verschiedenen Software-Produkten aufgegriffen werden, aber grundsätzlich immer wieder auftauchen.

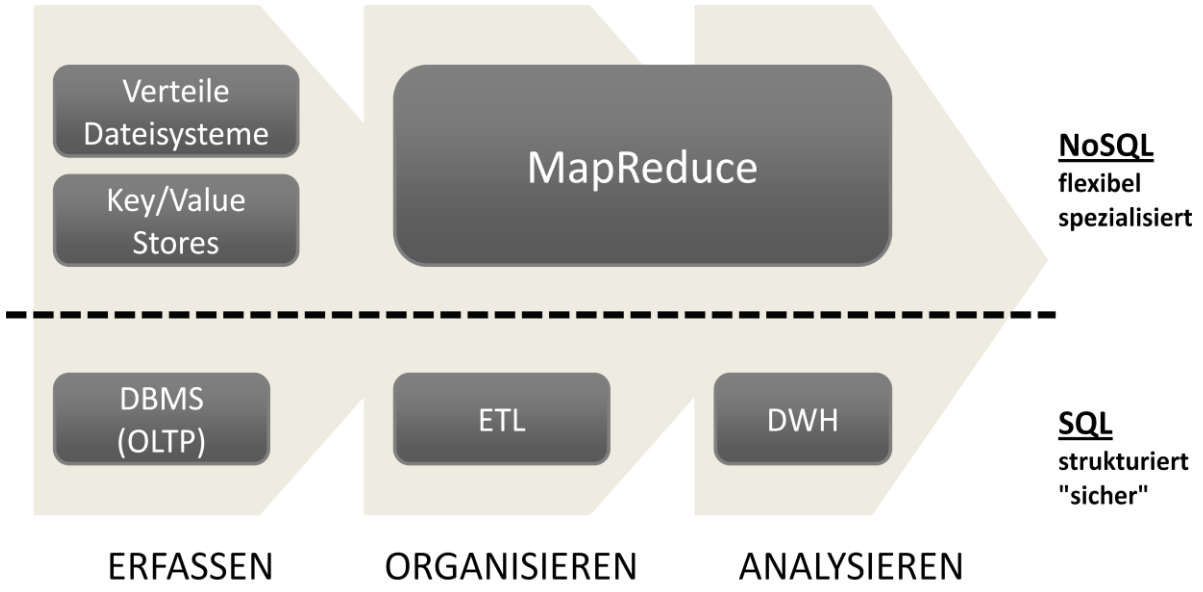

Abb. 1: Infrastruktur-Anforderungen

Im Folgenden wollen wir uns einige dieser zentralen Big Data Techniken innerhalb des Datenverarbeitungsprozesses näher anschauen. Als konkretes Beispiel dazu dient die Big Data Appliance von Oracle.

Oracle's Big Data Software-Stack
Betrachtet man die von Oracle angebotene Big Data Software, so sieht man zunächst, dass es zur Datenerfassung bzw. –speicherung mit der Oracle NoSQL Database einen Key-Value-Store einsetzt anstelle der klassischen relationalen Datenbank. Desweiteren wird das hochverfügbare, verteilte Dateisystem HDFS der Hadoop Software verwendet. Das Open Source Projekt "Hadoop" der Apache

Foundation nimmt momentan eine Vorreiterstellung im Big-Data-Bereich ein. Im Rahmen der Cloudera-Distribution setzt es Oracle auch im nächsten Prozess-Schritt (der Datenorganisation bzw. Datenintegration) ein, um eine massivparallele Verarbeitung der Daten durchzuführen. Kern der Software ist dabei der MapReduce-Alorithmus, der später noch genauer erläutert wird.

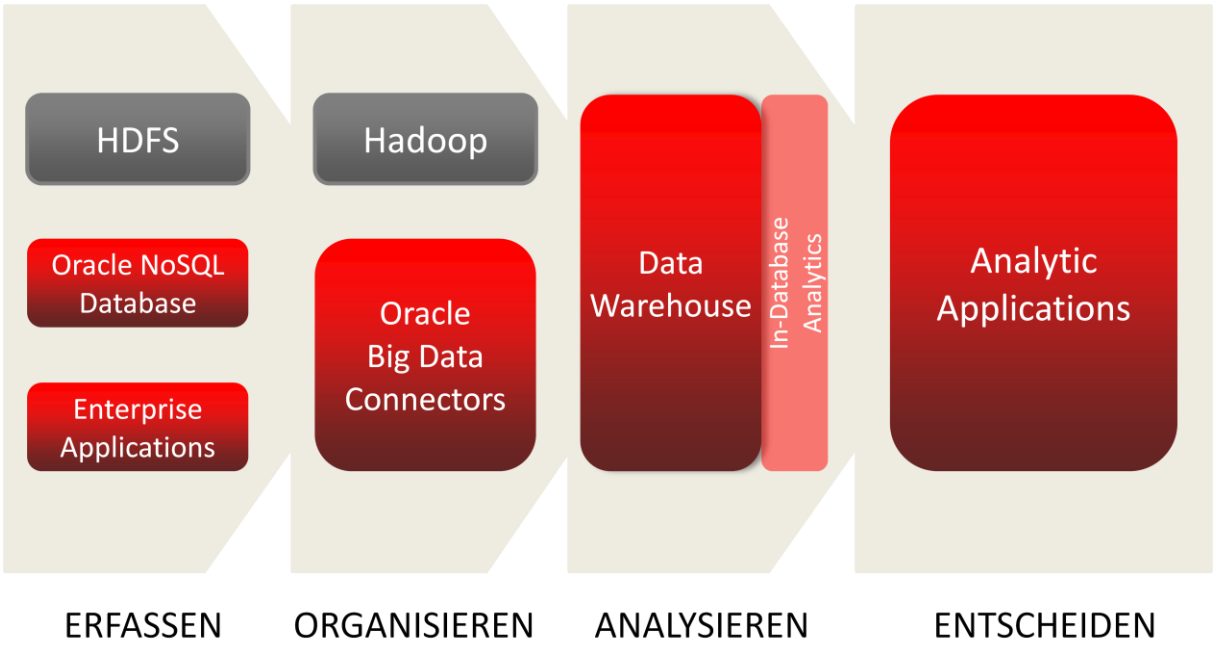

Abb. 2: Oracle's Big Data Lösungen

Nach der Verarbeitung durch Hadoop sollen die Daten zur Analyse in ein klassisches Data Warehouse auf einer Oracle Datenbank geladen werden. Hierzu hat Oracle eine Reihe von Konnektoren entwickelt (Big Data Connectors), die eine entsprechende Anbindung zum Laden der Daten bereitstellen. Ab diesem Punkt lassen sich dann die gewohnten Analyse-Anwendungen einsetzen, wobei sich bei Big Data der Schwerpunkt hin zu statistischer Analyse und Data Mining verschieben dürfte, was gerade in Kombination mit den traditionellen Unternehmensdaten neue Erkenntnisse liefern kann.

Oracle NoSQL Database
Die Oracle NoSQL Datenbank ist eine verteilte, hochgradig skalierbare, Key-/Value-basierte Datenbank basierend auf der 2006 von Oracle aufgekauften Berkeley DB.
Im Vergleich zu einem relationalen Datenbanksystem ist in einer Key-/Value-Datenbank kein komplexes Datenschema definiert. Stattdessen werden Datensätze lediglich durch einen Schlüssel identifiziert, der auf einen Container mit den relevanten Daten verweist. Durch diese einfache Struktur ist man sehr flexibel bei Änderungen. Außerdem müssen eingehende Daten nicht interpretiert und in ein Schema gepresst werden, weswegen die Erfassung neuer Daten sehr performant geschieht. Durch diese Eigenschaften ist der Ansatz für Big Data sehr gut geeignet, da die hohe Transaktionszahl bzw. Datenmenge sowie die hohe Variabilität der Daten adressiert werden. Für sich alleine genommen bietet die NoSQL-Datenbank allerdings noch nicht viel Mehrwert, da auf der einfachen Datenstruktur nur simple Abfragen möglich sind. Die Stärke ergibt sich erst im Zusammenspiel mit den weiteren Elementen des Software-Stacks wie dem MapReduce-Algorithmus in Hadoop.
Aber schauen wir uns zunächst die Architektur der Oracle NoSQL Datenbank genauer an: Wie bereits erwähnt werden die Daten hier anhand eines Schlüssels gespeichert. Dieser kann beliebig definiert werden und auch aus mehreren Teilen zusammengesetzt sein (Major Key Path/Minor Key Path), ist im

einfachsten Fall aber beispielsweise eine Zeichenkette. Zusammen mit dem Schlüssel können dann folgende Operationen zum Lesen und Schreiben von Daten verwendet werden:

```
get(key)
put(key, newValue)
putIfVersion(key, newValue, requiredVersion)
putIfPresent(key, newValue)
putIfAbsent(key, newValue)
delete(key)
deleteIfVersion(key, requiredVersion)
```

Mit der *get*-Methode werden die entsprechenden Daten gelesen, mit *delete* gelöscht. *putIfAbsent* entspricht einem *Insert*, *putIfPresent* einem *Update*, während ein einfaches *put* mit einem *Merge* vergleichbar ist. Eine Besonderheit ist, dass für jeden Datensatz zusätzlich eine Versionsnummer gespeichert wird, die bei jeder Änderung hochgezählt wird. Die Methoden *putIfVersion* und *deleteIfVersion* werden nur ausgeführt, wenn die angeforderte Versionsnummer vorliegt. Auf diese Weise können "Lesen-Ändern-Schreiben"-Operationen durchgeführt werden, ohne dass Konsistenzfehler auftreten. Neben diesen Einzel-Operationen steht auch noch eine API für Massenverarbeitungen (Bulk operations) bereit.

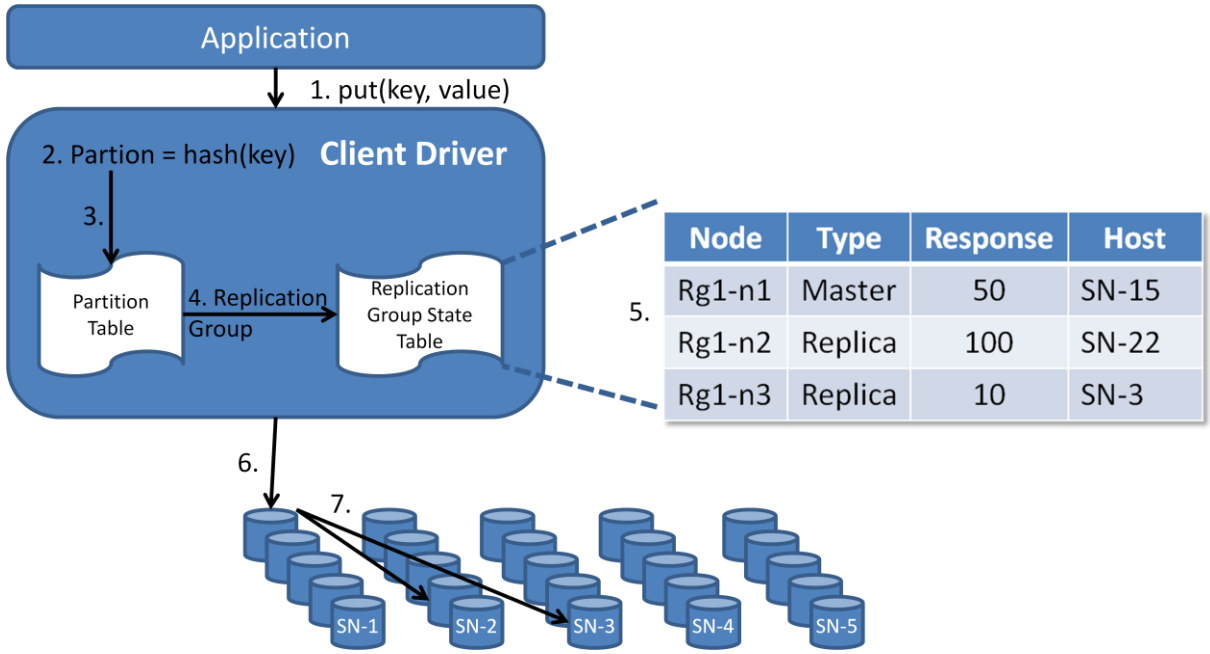

Abb. 3: Oracle NoSQL Datenbank Architektur

Am Beispiel einer *putIfAbsent*-Operation lässt sich die Architektur der Datenbank gut nachvollziehen: Zunächst wird durch den Client-Treiber (Java) auf den Schlüssel oder einen Teil davon (Major Key Path) eine Hash-Funktion angewendet. Das Ergebnis ist eine Partitions-Nummer. Die Anzahl an Partitionen wird bei der Konfiguration festgelegt und sollte um Größenordnungen höher sein, als die Anzahl an *Storage-Nodes* (z.B. 25000 Partitionen bei 25 Storage-Knoten).
Mit der Partitions-Nummer kann in der Partitions-Tabelle die zugehörige *Replikationsgruppe* ermittelt werden. Jede Replikationsgruppe besteht aus der gleichen (konfigurierbaren) Anzahl von *Replikationsknoten*. Die Anzahl an Knoten in einer Replikationsgruppe bestimmt die Fehlertoleranz des Systems. Die Informationen über die Knoten in der Gruppe erhält der Treiber über die sog. *Replication Group State Table* (RGST). Hier ist u. A. gespeichert welcher Knoten (momentan) der

Master der Gruppe ist und welche Antwortzeiten die Nodes haben. Basierend auf diesen Informationen entscheidet der Client an welchen Knoten die Anfrage weitergeleitet wird. Während wir bei einem Lese-Zugriff z.B. auf dem Storage-Node mit der geringsten Last gelandet wären, muss die Schreib-Anfrage zunächst zum Master-Knoten gehen.
Der Master-Knoten prüft (wg. *putIfAbsent*) zunächst ob der Schlüssel bereits existiert und gibt ggf. eine Fehlermeldung zurück. Im anderen Fall wird die Operation ausgeführt und die Daten werden eingefügt. Danach werden die Änderungen an die anderen Nodes in der Replikationsgruppe propagiert. Ein Storage-Node entspricht dabei typischerweise einem physischen Server mit eigenem lokalen Festspeicher, Hauptspeicher, CPU und einer IP-Adresse.

Hadoop Distributed File System (HDFS)
HDFS ist das hochverfügbare, verteilte Dateisystem der Hadoop Software. Durch die Verteilung will man dem hohen Datenvolumen von Big Data gerecht werden. Durch das einfache Hinzufügen von neuen Knoten zu einem HDFS Cluster ist dieses hochgradig skalierbar. Ein weiteres Designziel war Fehlertoleranz gegenüber Ausfällen, da so preiswerte Standard-Hardware bei den einzelnen Knoten verwendet werden kann.
Die Architektur von HDFS basiert auf dem Master-Slave-Prinzip, wobei der Master in diesem Falle *NameNode* genannt wird. Dieser zentrale Knoten speichert alle für die Verwaltung des Dateisystems notwendigen Metadaten wie Verzeichnisstrukturen, Dateinamen, etc. Da der NameNode in frühen Versionen von Hadoop als ein Single-Point-of-Failure fungierte wurde diesem mittlerweile ein zweiter NameNode anbeigestellt, der sich in einem passiven *hot standby* Modus befindet, um Hochverfügbarkeit zu gewährleisten. Die Rolle der Slaves nehmen die sog. *DataNodes* ein, die die Nutzdaten auf den einzelnen Clusterknoten verwalten.

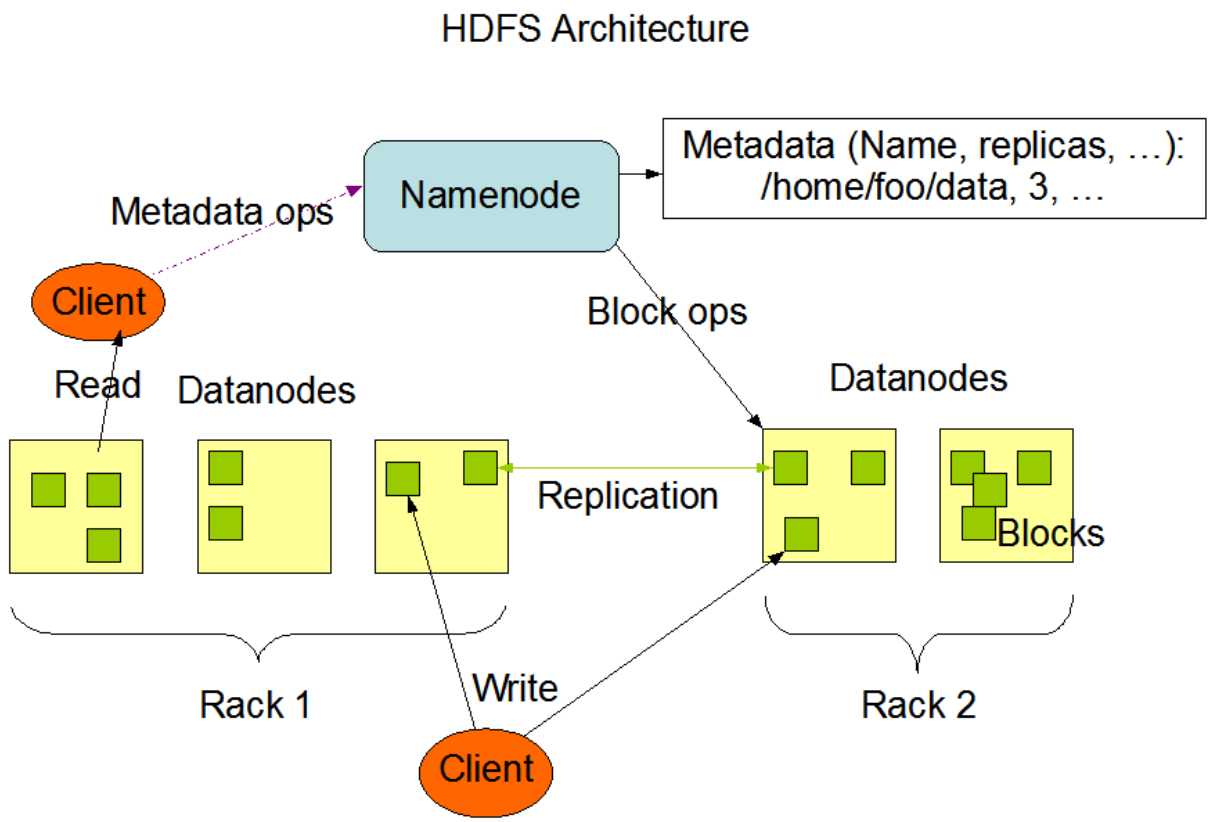

Abb. 4: HDFS-Architektur (Quelle: http://hadoop.apache.org/docs/hdfs/current/hdfs_design.html)

Auch wenn HDFS vom Benutzerstandpunkt aus, wie ein herkömmliches hierarchisches Dateisystem aussieht, bietet es doch einige auf Big Data zugeschnittene Besonderheiten. Beispielsweise wird davon ausgegangen, dass Lese-Operationen effizienter sein müssen als Schreib-Operationen (*write once, read many*). Bytegenaue Zugriff-Operationen werden gar nicht unterstützt, dafür erfolgt die Speicherung der Daten blockweise. Zur Fehlertoleranz und zur Verhinderung von Datenverlust werden die Datenblöcke auf unterschiedliche Knoten im Cluster repliziert. Standardmäßig speichert HDFS einen Datenblock dreifach: einmal das Original, dann eine zweite Kopie auf einem anderen Node im gleichen Rack und eine dritte Kopie in einem weiter entfernteren Node (*rack-awareness*). Jeder DataNode schickt in bestimmten (einstellbaren) Abständen ein Heartbeat-Signal an den NameNode. Bleibt aufgrund eines Hardware-Ausfalls der Heartbeat eines Node längere Zeit aus, erkennt dies der NameNode und markiert diesen als inaktiv. Da nun die Mindestanzahl an Kopien der Blöcke des ausgefallen Nodes unterschritten wurde werden die DataNodes mit den verbleibenden Kopien angewiesen diese auf andere DataNodes zu replizieren.
Der Zugriff auf das HDFS erfolgt über den bereitgestellten Kommandozeilen-Client oder direkt über die Java API (z.B. innerhalb von Hadoop). In seiner Big Data Appliance liefert Oracle zudem einen Konnektor, um HDFS-Daten direkt in die herkömmliche Oracle Datenbank zu laden.

MapReduce mit Hadoop
Bei MapReduce handelt es sich um ein von Google propagiertes Programmiermodell für die hochparallele Datenverarbeitung in Clustern über große Datenmengen. Die Namensgebung lehnt sich dabei an die Funktionen *map* und *reduce* der Programmiersprache Lisp an. Während *map* auf eine Liste von Eingabewerten, je Element eine bestimmte Funktion anwendet und wiederum eine Liste mit den Ausgabewerten zurückgibt, führt *reduce* im zweiten Schritt alle Werte einer Liste zusammen und gibt ein einzelnes Endergebnis aus. Die beiden Verarbeitungsteile werden dann parallel auf mehreren Rechnern eines Clusters ausgeführt.
Die *Hadoop* Software der Apache Foundation ist nichts anderes als ein auf diesem Konzept basierendes Java Framework.

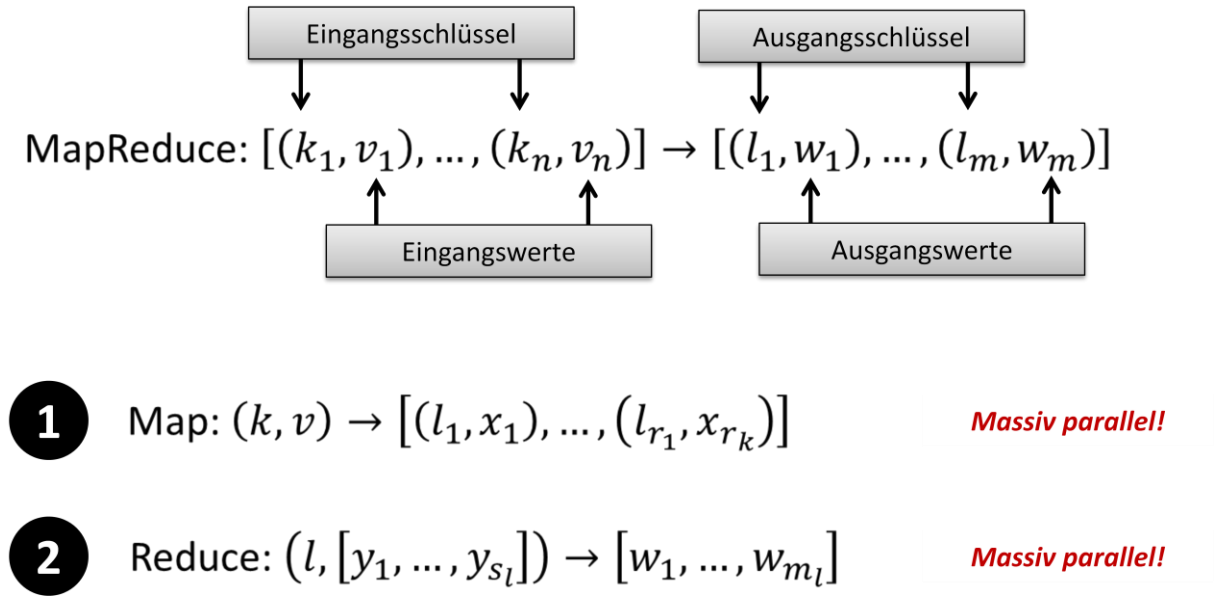

Abb. 5: MapReduce

Die Implementierung der jeweiligen Map- und Reduce-Funktion obliegt dem Programmierer und ist abhängig von der zu lösenden Problemstellung (z.B. Zählen von Wörtern in einem Text). Das Framework unterteilt dann die Eingangsdaten, eine Reihe von Schlüssel-Werte-Paaren, in mehrere

Teile, genannt *Splits*. Diese werden dann einer bestimmten Anzahl von Rechnern im Cluster, den *Workern*, zugewiesen. Jeder Worker führt die Map-Funktion auf dem ihm zugewiesenen Split aus. Die daraus entstandenen Zwischenergebnisse werden dann wiederum auf die Worker aufgeteilt, die die Reduce-Funktion ausführen. Dabei wird sichergestellt, dass Zwischenergebnisse mit dem gleichen Schlüsselwert auch immer dem gleichen Worker übergeben werden. Die Koordination der gesamten Job-Verteilung übernimmt ein zentraler, *JobTracker* genannter Knoten.

Fazit

Das Beispiel Hadoop zeigt, dass die Herausforderung Big Data momentan hauptsächlich durch das verteilen der Auswertungsaufgabe - und damit der Last - auf eine möglichst große Anzahl von Rechnern angegangen wird. Das heißt aber auch, dass sich dieses Modell in erster Linie für Analysen eignet, die sich gut in mehrere Teilaufgaben zerlegen lassen wie beispielsweise das Dursuchen von längeren Texten. Gerade solche unstrukturierten Daten lassen sich aufgrund der flexiblen Architektur mit dem Framework gut verarbeiten. Anhand solcher und ähnlicher Kriterien kann man also gut einordnen für welche Fragestellung sich der Einsatz der Big Data Konzepte anbietet und für welche weiterhin der klassische Weg geeigneter ist.

Kontaktadresse:
Philipp Krätzig
ARETO Consulting GmbH
Julius-Bau-Straße 2
D-51063 Köln

Telefon:	+49 (0) 221-66 95 75-0
Fax:	+49 (0) 221-66 95 75-99
E-Mail	philipp.kraetzig@areto-consulting.de
Internet:	www.areto-consulting.de