Karlsruher Institut für Technologie

Institut für Angewandte Informatik und formale Beschreibungssprachen

Optimierung von Analytischen Abfragen über Statistical Linked Data durch Horizontale Skalierung

Diplomarbeit

von

Sébastien Jelsch

im Studiengang Informatik

Referent:	Prof. Dr. Rudi Studer
Betreuender Mitarbeiter:	Dr. Benedikt Kämpgen

Bearbeitungszeit: 14. Juli 2015 – 07. Januar 2016

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt, die wörtlich oder inhaltlich übernommenen Stellen als solche kenntlich gemacht und die Satzung des KIT zur Sicherung guter wissenschaftlicher Praxis in der jeweils gültigen Fassung beachtet habe.

Karlsruhe, 07.01.2016

.....

Ort, Datum (Sébastien Jelsch)

Kurzzusammenfassung

In den letzten Jahren ist die Menge der verfügbaren Linked Data im World Wide Web stetig gestiegen. Immer mehr Provider veröffentlichen ihre statistischen Datensätze nach dem Linked-Data-Prinzip, um diese Daten mit weiteren Informationen aus unterschiedlichen Quellen anreichern zu können. Durch die Verlinkung beliebiger Zusatzinformationen sollen die Daten näher bestimmt und neue Erkenntnisse erlangt werden.

Bevor Analysten jedoch in der Lage sind, solche statistischen Daten vergleichen zu können, verbringen sie unverhältnismäßig viel Zeit mit der Identifizierung, Erfassung und Aufbereitung der relevanten Daten. Zusätzlich führt die zunehmende Größe an verfügbaren RDF-Datensätzen dazu, dass diese nicht mehr effizient auf einem einzelnen Rechner analysiert werden können. Aus diesem Grund sind für Analysen großer Datenmengen Technologien aus dem Big-Data-Umfeld notwendig, die diese Beschränkungen mittels Parallelisierung über viele Rechner hinweg überwinden. In der Folge ist für die Analyse einer beliebig großen RDF-Datenmenge ein Lösungsansatz erforderlich, der diese Datensätze generisch und automatisiert in ein horizontal skalierbares Open Source OLAP-System integriert, die Daten in geeigneter Form aufbereitet und dem Nutzer präsentiert.

In dieser Arbeit wird ein horizontal skalierbarer Extract-, Transform- und Load-Prozess (ETL-Prozess) konzipiert, umgesetzt und im Hinblick auf die Ausführungszeit für die Integration der RDF-Daten in das OLAP-System Apache Kylin evaluiert. Des Weiteren werden die Antwortzeiten analytischer MDX- und SQL-Abfragen untersucht. Die Ergebnisse zeigen, dass der ETL-Prozess erst ab einer größeren Datenmenge einen Vorteil gegenüber Import-Vorgängen in nicht-horizontal skalierenden Systemen bietet. Jedoch werden analytische Abfragen in diesem System aufgrund der horizontalen Skalierung in sehr kurzer Zeit beantwortet.

Schlüsselwörter: Linked Data, Data Cube, Parallelisierung, MapReduce

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Abkürzungsverzeichnis

API Application Programming Interface

AWS Amazon Web Services

BI Business Intelligence

BIBM Business Intelligence Benchmark

CDH Cloudera Distribution Including Apache Hadoop

DSD Data Structure Definiton

EC2 Elastic Compute Cloud

ETL-Prozess Extract-, Transform- und Load-Prozess

FOAF Friend of a Friend

HDFS Hadoop Distributed File System

HiveQL Hive Query Language

HTTP Hypertext Transfer Protocol

JDBC Java Database Connection

MDM Multidimensionales Datenmodell

MDX Multidimensional Expression

MOLAP Multidimensionales OLAP

NoSQL Not only SQL

OLAP On-Line Analytical Processing

QB RDF Data Cube Vocabulary

RDF Resource Description Framework

RDFS RDF Schema

ROLAP Relationales OLAP

SPARQL SPARQL Protocol And RDF Query Language

SQL Structured Query Language

SSB Star Schema Benchmark

URI Uniform Resource Identifier

W3C World Wide Web Consortium

XML Extensible Markup Language

1. Einleitung

Das heutige World Wide Web (kurz: Web) besitzt aufgrund seiner rasanten Entwicklung eine für Menschen unüberschaubare, stetig wachsende Menge an Informationen. Neben den Vorteilen der universellen Verfügbarkeit, der ständigen Aktualität und der einfachen Bereitstellung von Daten überwiegt jedoch ein wesentlicher Nachteil: die Ausrichtung der Informationen orientiert sich an einem menschlichen Betrachter. Im Gegensatz zu einer Maschine kann ein menschlicher Benutzer die Bedeutung von Informationen auf einer Webseite erfassen, zueinander in Beziehung setzen und in eine andere Darstellungsform transformieren, um neues Wissen zu generieren. Die Datenfülle und dezentrale Organisation des Webs führen dazu, dass eine Suche nach gewünschten Informationen erschwert wird. Erstrebenswert ist daher eine inhaltliche und semantische Suche.

Hierzu prägte Tim Berners-Lee 2001 das Konzept Semantic Web. Die Grundidee des Konzepts besteht darin, das herkömmliche Web durch Standards zu erweitern. Im Vordergrund steht dabei die Verwertung der Informationen von Maschinen und der einfache Austausch zwischen diesen. Dazu sind grundlegende Anforderungen notwendig, die neben einer klaren Definition des Standards auch eine flexible Anwendung und mögliche Erweiterungen erlauben sollen.

Aus diesem Grund wurde das Resource Description Framework (RDF) entwickelt. RDF ist eine Modellierungssprache zur Beschreibung strukturierter Informationen und ein Standard des World Wide Web Consortium. Mit dem RDF-Standard soll es Anwendungen ermöglicht werden, Daten über das Web auszutauschen, ohne ihre eigentliche Bedeutung zu verlieren und sie mit zusätzlichen Informationen aus anderen Datenquellen anzureichern. Im Gegensatz zum herkömmlichen Web geht es hierbei nicht nur um eine korrekte Darstellung von Daten oder Dokumenten, sondern vielmehr um die Verknüpfung von Daten aus unterschiedlichen Quellen für die Weiterverarbeitung der Informationen.

1.1. Motivation

In den letzten Jahren ist das Interesse gestiegen, statistische Daten nach dem Linked-Data-Prinzip zu veröffentlichen und die Möglichkeit zu bieten, Daten mit anderen Informationen aus unterschiedlichen Quellen zu kombinieren. Ein Vorteil besteht darin, beliebige Zusatzinformationen zu den statistischen Daten zu verlinken, um die Bedeutung der Daten näher zu bestimmen und neue Erkenntnisse zu erlangen. Beispielsweise können Provenance-Informationen (z. B. Woher stammen die Daten? Welche Qualität besitzen sie?) oder weitergehende Informationen (z. B. Welche Anzahl an Mitarbeitern besitzt die Firma, von denen die Daten handeln?) hinzugefügt werden. Ferner können auch interne Daten (z. B. aus dem Intranet) einer Firma mit den statistischen Daten verlinkt und zur Analyse verwendet werden.

Bevor Analysten jedoch in der Lage sind, Unternehmensleistungen vergleichen zu können, verbringen sie unverhältnismäßig viel Zeit mit der Identifizierung, Erfassung und Aufbereitung der relevanten Daten. Der Aufwand steigt mit der Anzahl heterogener Datenquellen. Damit verbunden sind unterschiedliche Formate oder Bezeichnungen für identische Objekte. Diese Prozesse müssen optimiert und möglichst automatisiert werden. Zusätzlich führt die zunehmende Größe an verfügbaren RDF-Datensätzen[1] dazu, dass diese nicht mehr effizient auf einem einzelnen Rechner verarbeitet und analysiert werden können. Daher sind neue Konzepte zur Auswertung statistischer Datensätze notwendig. Ferner ist ein Lösungsansatz erforderlich, der eine beliebig große RDF-Datenmenge generisch und automatisiert integriert, diese in geeigneter Form aufbereitet und dem Nutzer präsentiert.

1.2. Zielsetzung

Für entscheidungsunterstützende Analysen numerischer Datensätze bietet das Konzept OLAP (**O**n-**L**ine **A**nalytical **P**rocessing) eine multidimensionale Betrachtung des Datenbestands. In einer vorangegangenen Arbeit von Kämpgen und Harth [KH11] wurde hierfür ein Extract-, Transform- und Load-Prozess (ETL-Prozess) vorgestellt, der statistische Linked Data aus unterschiedlichen RDF Stores, unter Anwendung der Abfragesprache SPARQL und dem RDF Data Cube Vocabulary (QB), in ein multidimensionales Datenmodell transformiert. Die Daten wurden für die Analysen in einem relationalen Data Ware-

[1] s. Datensätze, die nach dem Linked-Data-Prinzip veröffentlicht werden: `http://lod-cloud.net/`.

house gespeichert. Auf diese Weise konnte mit der OLAP-to-SQL-Engine Mondrian die Vorteile der multidimensionalen Abfragemöglichkeit und erweiterten Selektierbarkeit von OLAP-Anfragen mit MDX (engl. für **M**ulti**D**imensional E**X**pression) genutzt werden. Ziel des ETL-Prozesses war es, entscheidungsunterstützende Analysen auf RDF-Datensätzen durchzuführen. Dieser Ansatz beinhaltet jedoch vier wesentliche Probleme:

(V1) Die Dauer des ETL-Prozesses bei großen Datensätzen mit vielen Zusatzinformationen ist nicht zufriedenstellend, da innerhalb der RDF-Daten die nötigen Informationen für das multidimensionale Datenmodell (Metadaten und Daten) herausgezogen werden müssen.

(V2) Bei einer Aktualisierung des Datenbestands muss der ETL-Prozess neu durchgeführt werden.

(V3) Bei der Hinzunahme neuer Daten muss der ETL-Prozess ebenfalls neu durchgeführt werden.

(V4) Zusatzinformationen in den Datensätzen werden bei der Erstellung des multidimensionalen Datenmodells gefiltert und können in den analytischen Abfragen nicht berücksichtigt oder als Zusatzinformation abgefragt werden.

Sowohl relationale Datenbanken, RDF Stores als auch OLAP Engines skalieren in der Regel nicht horizontal und besitzen daher eine natürliche Grenze bzgl. ihrer Datenspeicher- und Datenverarbeitungskapazität. Aus diesem Grund sind für Analysen großer Datenmengen Technologien aus dem Big-Data-Umfeld notwendig, die diese Beschränkungen mittels Parallelisierung über viele Rechner hinweg überwinden. Mit Apache Hadoop sind derartige Technologien in einem Open Source Software Stack verfügbar. Bislang wurde nicht erforscht, ob eine enorm große RDF-Datenmenge in einem automatisierten ETL-Prozess durch eine Umsetzung der Architektur von Kämpgen und Harth mit Komponenten aus dem Hadoop-Ökosystem für OLAP-Analysen bereitgestellt werden kann.

Der hier präsentierte Lösungsansatz überführt Kämpgen und Harths Konzept in eine horizontal skalierende Architektur auf der Basis von Apache Hadoop. Die nicht-skalierbaren Komponenten, wie die RDF-Datenbank, die Abfragesprache SPARQL und die relationale Datenbank, werden dabei durch Technologien und Frameworks aus dem Hadoop-Ökosystem ersetzt.

Eine weitere Zielsetzung dieser Arbeit liegt in der generischen und automatisierten Umsetzung des ETL-Prozesses unter Verwendung der Metainformationen des QB-Vokabulars. Ferner soll der ETL-Prozess auf seine Ausführungszeit im Hinblick auf die Anzahl der ver-

wendeten Rechner im Vergleich zur relationalen Datenbank MySQL und dem RDF Store Open Virtuoso evaluiert werden.

1.3. Verwandte Arbeiten

In einer vorherigen Arbeit [KH13] wurden die statistischen Daten in einem RDF Store geladen, um analytische Abfragen mittels der graphenbasierten Sprache SPARQL auszuführen. Es zeigte sich, dass die Ausführung der analytischen Abfragen weniger effizient durchgeführt wurden, als vergleichsweise eine relationale Datenbank mit äquivalenten Daten im Sternschema. Eine weitere Arbeit [CMEF+13] beschäftigte sich mit der Optimierung eines RDF Stores durch horizontale Skalierung. Da NoSQL-Systeme für komplexe OLAP-Operationen weniger geeignet sind, war die Ausführung der analytischen Abfragen nicht effizient genug. In der Arbeit von Abelló et al. [AFR11] wurden analytische Abfragen auf MapReduce-basierten Systemen evaluiert. Dabei wurden die Vorteile von Big-Data-Technologien bei der Generierung eines OLAP Cubes für analytische Abfragen überprüft, jedoch ohne eine horizontale skalierbare OLAP Engine für die Analysen zu verwenden. Zusammenfassend kann festgestellt werden, dass eine Analyse einer beliebig großen Menge an RDF-Daten eine Herausforderung darstellt.

1.4. Gliederung

Die vorliegende Arbeit ist in sechs Kapitel unterteilt. Das erste Kapitel gibt dem interessierten Leser eine kurze Einführung und einen Gesamtüberblick über die geplanten Ziele. Das zweite Kapitel enthält eine ausführliche Beschreibung der Grundlagen und der verwendeten Begriffe, wobei der Fokus auf traditionellen Konzepten für Analysen statistischer Datensätze, der Idee hinter dem Semantic Web und auf die in dieser Arbeit verwendeten Big-Data-Technologien im Verbund mit Apache Hadoop und sein Ökosystem liegt. Der dritte Abschnitt beschreibt das in dieser Arbeit entworfene Konzept und die Gesamtarchitektur. Zudem werden die verwendeten Komponenten aus dem Apache Hadoop-Ökosystem sowie ihr Zusammenspiel in diesem Abschnitt einer näheren Betrachtung unterzogen. Das vierte Kapitel beschreibt die technische Umsetzung des ETL-Prozesses. Auf den Evaluationsaufbau und die erhaltenen Ergebnisse wird im fünften Kapitel eingegangen. Anschließend werden im letzten Kapitel das Fazit der Arbeit und mögliche Erweiterungen besprochen.

2. Grundlagen

In diesem Kapitel werden die in der Abschlussarbeit verwendeten Begriffe und Notationen erläutert. Der erste Abschnitt beschreibt die wesentlichen Anforderungen für die Analyse statistischer Datensätze sowie die daraus entstandenen Konzepte und Prozesse. Im Vordergrund steht dabei die Definition des Begriffs *Business Intelligence*, die grundlegenden Aufgaben der Speicherkomponente in Form eines *Data Warehouses* und das Prinzip hinter dem Konzept *OLAP*. Der zweite Abschnitt behandelt das Konzept *Semantic Web* und bietet eine ausführliche Beschreibung von *RDF*. Die Veröffentlichung statistischer Datensätze nach dem *Linked-Data*-Prinzip und dem *RDF Data Cube Vocabulary* schließen diesen Abschnitt ab. Im letzten Abschnitt wird eine Einführung in das Thema *Big Data* und die grundlegenden Technologien aus dem Apache-Hadoop-Ökosystem behandelt.

2.1. Konzepte und Prozesse zur systematischen Analyse von statistischen Datensätzen

Damit Führungskräfte strategische Entscheidungen treffen können, benötigen sie einen genauen Überblick über ihr Unternehmen. Aus diesem Grund ist es notwendig, den Entscheidungsträgern alle relevanten Informationen und Daten für die Analysen zur Verfügung zu stellen. Die Begriffe und die damit verbundenen Konzepte und Prozesse sind Gegenstand der nächsten Abschnitte.

2.1.1. Der Begriff Business Intelligence

Der Begriff *Business Intelligence* (BI) hat sich in den letzten drei Dekaden sowohl in der Wissenschaft als auch in der Wirtschaft etabliert (vgl. [GK06]). Eine genaue Abgrenzung des Begriffs erweist sich jedoch als schwierig. Bis zum heutigen Zeitpunkt besteht Uneinigkeit in der eindeutigen Definition. Dessen ungeachtet muss eine BI-Anwendung ver-

ständnisunterstützenden Charakter zur Entscheidungsfindung und besseren Einsicht des Unternehmens aufweisen (vgl. [GGD08, S. 30]). Für den weiteren Verlauf dieser Arbeit wird daher als Begriffsverständnis für BI eine Definition in Anlehnung an Strauch und Winter [SW02] gewählt:

> „Der Begriff 'Business Intelligence' [...] umschreibt den IT-gestützten Zugriff auf Informationen sowie die IT-gestützte Analyse und Aufbereitung von Informationen mit dem Ziel der Unterstützung betrieblicher Entscheidungen."

Nach dieser Definition wird Business Intelligence im weiten Begriffsverständnis (vgl. [KWZ98]; [Int99]; [HH02]) in drei Schichten unterteilt. Dieses Schichtenmodell ist Gegenstand des nächsten Abschnitts.

Das Business-Intelligence-Schichtenmodell

Im Folgenden werden die drei Schichten des BI-Schichtenmodells und ihre jeweiligen Aufgaben beschrieben. Zum besseren Verständnis dient die Abbildung 2.1.

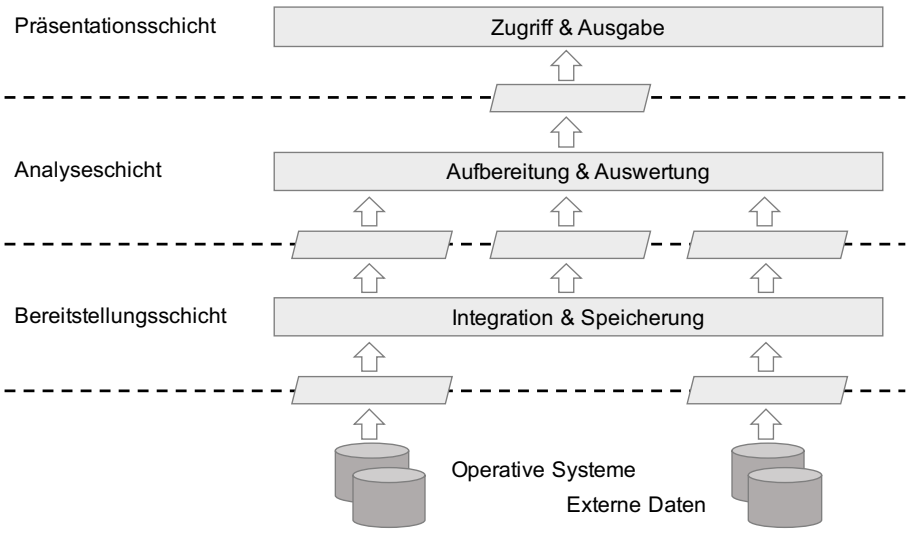

Abbildung 2.1.: BI-Schichtenmodell in Anlehnung an [GGD08, S.109] und [KR13, S. 19].

Schicht 1: Datenbereitstellung

Im Rahmen der Datenbereitstellung werden alle entscheidungsrelevanten Daten aus mehreren und teilweise sehr unterschiedlichen operativen Systemen eines Unternehmens geladen, gesäubert, vereinheitlicht und in ein zentrales, entscheidungsorientiert

aufgebautes Data Warehouse (s. Abschnitt 2.1.2) systematisch zusammengeführt. Auf diese Weise wird zu Informations- und Analysezwecken eine Überführung von vielfältigen und heterogenen Datenquellen in einen gemeinsamen und konsistenten Datenbestand ermöglicht (vgl. [GGD08, S. 109-111]).

Schicht 2: Analyse

Der Datenbestand eines Data Warehouses stellt das Fundament für die Analysen dar. Die aufbereiteten Daten werden in dieser Schicht nach verschiedenen Kriterien und Methoden ausgewertet. Je nach Anwendungsfall unterscheiden sich die benötigten Analysefunktionen erheblich voneinander. Besonders häufig wird eine navigationsorientierte Analysemöglichkeit bereitgestellt, die sich durch die Verdichtung der zu untersuchenden Kennzahlen eines Unternehmens charakterisiert (vgl. [GGD08, S. 111-114]).

Schicht 3: Präsentation

Schließlich beinhaltet die letzte Schicht Funktionen für den Zugriff auf das Datenmaterial. Im Vergleich zur Analyseschicht liegt hier der Schwerpunkt in der Präsentation der relevanten Inhalte. Die Darstellungsformen richten sich nach den Bedürfnissen der Analysten und reichen von der Ausgabe durch eine mehrdimensionalen Tabelle bis hin zu grafischen Abbildungen in Form von Balken-, Säulen-, Linien- und Flächendiagrammen (vgl. [GGD08, S. 114-116]).

Grundsätzlich ist in vielen konkreten Anwendungsfällen eine exakte Trennung zwischen den Schichten nicht möglich. In der Praxis erfolgt der Einsatz von Business Intelligence stets unternehmensspezifisch und orientiert sich dabei an den betriebswirtschaftlichen Anforderungen des Unternehmens (vgl. [KR13, S. 1-5]). Demnach kann der Umfang einzelner Schichten mehr oder weniger stark ausgeprägt sein (vgl. [GK06, S. 108]). Dessen ungeachtet hat sich die BI-Schichtenarchitektur für theoretische Grundlagen als sinnvoll erwiesen.

Die in der Datenbereitstellungsschicht benötigte Speicherkomponente und die damit verbundenen Anforderungen werden im folgenden Abschnitt genauer erläutert.

2.1.2. Data Warehouse als Speicherkomponente

Im Allgemeinen wird ein Data Warehouse als eine Sammlung von Daten aus unterschiedlichen, heterogenen operativen Systemen (wie z. B. Vertrieb, Produktion) angesehen. Zudem bildet ein Data Warehouse das Fundament der unternehmensspezifischen Entscheidungs-

unterstützung für Führungskräfte und Analysten (vgl. [GGD08, S. 117-119]). Sinn und Zweck liegt in der Möglichkeit, Entscheidern eines Unternehmens eine globale Sicht auf heterogen verteilte Datenbestände und somit einen einheitlichen und zentralen Zugriff auf die relevanten Daten zu bieten. Unabhängig davon, an welcher Stelle die Daten ursprünglich gespeichert wurden oder welche Struktur sie aufwiesen.

Im Vergleich zu applikations- und prozessorientierten operativen Systemen wird ein Data Warehouse in der Regel durch folgende Merkmale charakterisiert (vgl. [GGD08, S.119-121]):

Merkmal 1: Themenorientierung

Im Gegensatz zu operativen Systemen liegt der Fokus bei einem Data Warehouse auf das inhaltliche Thema. Operative Daten, die nur bei der Abwicklung eines Prozesses benötigt werden, werden im Allgemeinen nicht in einem Data Warehouse gespeichert. Daher ist vor der Speicherung eine Selektion der relevanten Daten durchzuführen.

Merkmal 2: Vereinheitlichung

Die unterschiedlichen Datenquellen der operativen Systeme führen in der Regel zu einer großen Heterogenität der Daten. Ziel der Vereinheitlichung ist ein konsistenter Datenbestand, bei dem Unstimmigkeiten, fehlende Werte oder unterschiedliche Bezeichnungen für gleiche Objekte korrigiert werden (s. Abschnitt 2.1.3).

Merkmal 3: Zeitorientierung

Im Vergleich zu operativen Anwendungen, bei denen der Zugriff auf aktuelle Daten im Moment des Zugriffs erfolgen muss, wird bei einem Data Warehouse für die Auswertung der Informationen lediglich eine zeitpunktbezogene Korrektheit benötigt (vgl. [GGD08, S. 120]).

Merkmal 4: Beständigkeit

Im Allgemeinen werden für die Analysen Daten benötigt, die einen zeitlichen Verlauf beschreiben. Daher werden diese Daten über einen langen Zeitraum hinweg in einem Data Warehouse gespeichert und nur in Ausnahmefällen aktualisiert (vgl. [MB00, S. 13]).

Zusammenfassend ist zu sagen, dass die Ziele einer Data Warehouse-Lösung darin bestehen, Daten über lange Zeiträume und mit einem konkreten Zeitbezug zu sammeln, aufzubereiten und bedarfsgerecht für die Analysen zur Verfügung zu stellen (vgl. [Inm05, S. 29-39]). Als wesentlicher Bestandteil gelten die Bausteine, die für die Überführung der Daten aus den

Datenquellen in das zentrale Data Warehouse zuständig sind. Der nächste Abschnitt soll diese Bausteine und den damit einhergehenden Prozess genauer erläutern.

2.1.3. Der Extract-, Transform- und Load-Prozess

Die Befüllung der Data Warehouse-Speicherkomponenten ist von zentraler Bedeutung (vgl. [GGD08, S. 133-144]). Ziel ist, Daten aus heterogenen operativen Systemen für den betrieblichen Anwender nutzbar in eine Zieldatenbank abzulegen. Hierzu ist die Extraktion und eine Transformation der Quelldaten in die benötigte Datenstruktur erforderlich. Dieser als Extract, Transform und Load (ETL) bezeichneter Prozess wird in Abbildung 2.2 veranschaulicht. Die einzelnen Phasen werden im Folgenden genauer erläutert.

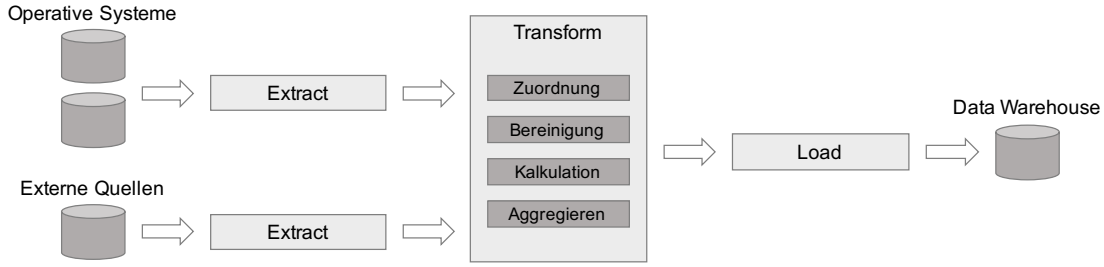

Abbildung 2.2.: ETL-Prozess zur Befüllung eines Data Warehouses in Anlehnung an [GGD08, S. 134].

Phase 1: Extract

Die Extraktion ist der erste Schritt des ETL-Prozesses. In dieser Phase werden die Daten aus den verschiedenen Quellen geladen und für die darauffolgenden Transformationen bereitgestellt. In der Regel werden durch wohldefinierte Filtervorschriften die Quelldaten auf einen im Vorfeld festgelegten, relevanten Umfang reduziert.

Phase 2: Transform

Nachdem die relevanten Daten aus den operativen Systemem extrahiert wurden, erfolgt im nächsten Schritt eine Transformation der Quelldaten. Diese Phase besteht im Allgemeinen aus mehreren Transformationsschritten, um eine Verbesserung der Datenqualität zu erhalten. Bei der Säuberung werden die Daten auf Fehler kontrolliert, aufkommende Konflikte gelöst und ein Verhalten bei fehlenden oder unerlaubten Werten definiert (vgl. [KR13, S. 19-21]). Zusätzlich findet in vielen Fällen eine Duplikateliminierung statt.

Phase 3: Load

Die Zielsetzung der letzten Phase ist das Laden der transformierten Daten aus dem Arbeitsbereich in das Data Warehouse. Besondere Bedeutung erlangt hierbei die Effizienz des Schreibvorgangs. Die Zieldatenbank sollte so wenig wie möglich blockiert werden.

Nur nach erfolgreichem Abschluss dieser Phasen liegen die Daten für die Analyse in der benötigten Struktur und Datenqualität im Data Warehouse vor. Die Güte des ETL-Prozesses ist entscheidend für die gewünschte Qualität der verfügbaren und vorliegenden Daten. Da auf diese Daten fast ausschließlich lesend zugegriffen wird, kann eine auf Abfragen optimierte Modellierung erfolgen (vgl. [Kem11]). Dies ist Gegenstand des nächsten Abschnitts.

2.1.4. On-Line Analytical Processing

Das von Codd et al. [CCS93] geprägte Konzept OLAP (**O**n-**L**ine **A**nalytical **P**rocessing) bietet für entscheidungs- und führungsrelevante Fragestellungen eine multidimensionale Betrachtung des Datenbestands. Auf diese Weise soll für Fach- und Führungskräfte sowie Analysten ein besonders gut zu repräsentierendes Geschäftsverständnis des Unternehmens erreicht werden (vgl. [GGD08, S. 143]).

In dieser Arbeit werden die fünf charakteristischen OLAP-Merkmale von Pendse und Creeth [PC95] mit dem Akronym FASMI (**F**ast **A**nalysis of **S**hared **M**ultidimensional **I**nformation) einer genaueren Betrachtung unterzogen. Das Ziel ist eine klare Definition der OLAP-Eigenschaften. Im Einzelnen bedeutet FASMI:

Geschwindigkeit (Fast)

Das Ergebnis der Analyse soll möglichst schnell zur Verfügung stehen. Um die interaktive Eigenschaft nicht zu verlieren, sind Abfragen im Sekundenbereich vom OLAP-System zu beantworten, auch bei großen Datenmengen.

Analyse (Analysis)

Der Analyseprozess soll die Anforderungen erfüllen, die im jeweiligen Fall benötigt werden. Nach Pendse und Creeth darf der Benutzer nicht mit Programmiertätigkeiten belastet werden.

Gemeinsamer Zugriff (Shared)

Der Zugriff auf den Datenbestand soll für mehrere Anwender gleichzeitig möglich sein.

Für lesende oder schreibende Zugriffsarten sind demzufolge Sicherheitsmechanismen für einen verlässlichen und gültigen Benutzerzugriff notwendig, obwohl letztere Zugriffsart in der Regel nicht von allen Systemen gewährleistet wird.

Mehrdimensionalität (Multidimensional)

Ein weiteres und zentrales Kriterium stellt die konzeptionelle Mehrdimensionalität dar, unabhängig von der eingesetzten Datenbanktechnologie. Ferner sollen Hierarchien in Dimensionen in vollem Umfang unterstützt werden (s. Abschnitt 2.1.4).

Information

Unabhängig der Menge oder Herkunft sollen alle relevanten Daten verarbeitet und aufgenommen werden können.

Ist bei der Anfrage nicht nur ein einzelner Zugriff auf einen Wert oder einen kleinen Datenbereich, sondern ein dynamischer, flexibler und interaktiver Zugriff auf einen großen Datenbereich erforderlich, so gehört sie der Kategorie der OLAP-Anfragen an. OLAP-Anfragen sind häufig in Data Warehouses zu finden, da hier sehr komplexe Fragestellungen wie „Wie hat sich 2015 der Umsatz der Firma am Standort 'Berlin' in der Produktkategorie 'Schuhe' im Vergleich zum Jahr 2014 verändert?“ beantwortet werden sollen. Durch diese Art von Anfragen wird das Datenmodell an die Analyseanforderungen angepasst (vgl. [Kem11]). Aus dieser Anpassung resultiert eine multidimensionale Sichtweise für komplexe Analysen der Daten mit Präsentationsunterstützung (vgl. [BG13, S. 106f.]).

OLAP-Datenmodell

Die Modellierung der Datenstruktur erweist sich als zentrale Aufgabe eines Business-Intelligence-Projektes. Auf der konzeptionellen Ebene wird OLAP im Allgemeinen als multidimensionales Datenmodell (MDM) dargestellt (vgl. [KR13, S. 7-8]; [Kem11]; [KH11]). Obwohl kein einheitlicher Standard für die Darstellung eines MDMs existiert, haben alle die Gemeinsamkeit, Daten in einem n-dimensionalen Würfel abzubilden, dem sogenannten *OLAP Cube* (auch als OLAP-Würfel, Hypercube oder Data-Cube bezeichnet). Dabei werden die zu analysierenden Daten in Fakten (engl. *Facts*) und Dimensionen (engl. *Dimensions*) unterteilt.

Fakten sind einzelne Datenpunkte im Cube. Sie beinhalten die zu analysierenden Kennzahlen (engl. *Measures*). Measures sind ausschließlich numerischer Art und beschreiben z. B. den Umsatz, die Kosten und den Profit eines Unternehmens. Diese Kennzahlen müssen im

Vorfeld der Konzeption als solche deklariert werden. Der Zugriff auf ein Fakt wird über die Instanzen der Attribute, die *Members* genannt werden, ermöglicht.

Die Dimensionen hingegen identifizieren die Achsen des OLAP Cubes und ermöglichen somit den Zugriff auf die Measures. Die Strukturierung nach Dimensionen ist das grundlegende Prinzip von OLAP. Dimensionen ermöglichen verschiedene Sichtweisen auf die Daten und unterstützen die Vorgehensweise der multidimensionalen Analyse. Zum besseren Verständnis des multidimensionalen Modells zeigt Abbildung 2.3 eine häufig gewählte Darstellungsmöglichkeit.

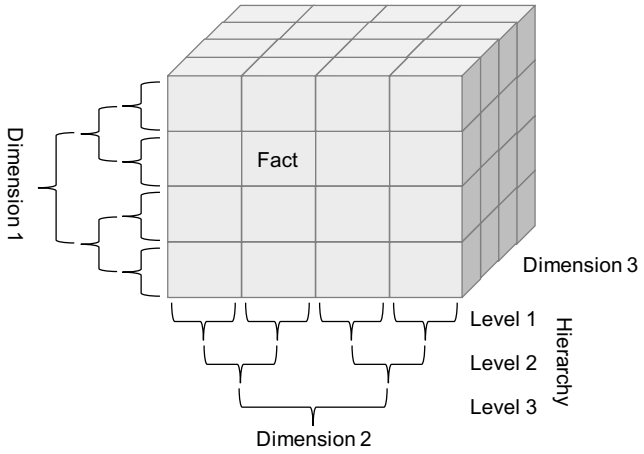

Abbildung 2.3.: Beispiel eines multidimensionalen Modells mit drei Dimensionen in Anlehnung an [KH11].

In der Regel besitzt eine Dimension mehrere Attribute (engl. *Attribute*), die zueinander in Beziehung stehen. Im OLAP-Kontext wird solch eine Beziehung als Hierarchie (engl. *Hierarchy*) bezeichnet, die über mehrere Ebenen (engl. *Levels*) eine Dimensionen charakterisieren. Eine hierarchische Anordnung findet sich in vielen Daten wieder, z. B. in der häufig verwendeten Dimension „Zeit“. Eine Dimension kann zudem mehrere Hierarchien besitzen, z. B. besteht ein Jahr aus 12 Monaten aber auch gleichzeitig aus 52 Wochen.

Neben der hierarchischen Anordnung der Daten besteht eine weitere Eigenschaft von OLAP-Systemen darin, dem Analysten die Möglichkeit zu bieten, sich entlang einer solchen Hierarchie zu navigieren. Durch die Anordnung der Daten ermöglicht ein OLAP-System eine einfache, flexible und schnelle Bereitstellung entscheidungsrelevanter Informationen aus verschiedenen Perspektiven. Hierzu sind Operationen notwendig, die im folgenden Abschnitt erläutert werden.

OLAP-Funktionen

Zentrales Ziel eines OLAP-Systems ist eine anschauliche und verständliche Visualisierung des OLAP Cubes für den Benutzer. Mithilfe von verschiedenen OLAP-Funktionen sollen Analysten in der Lage sein, durch den OLAP Cube zu navigieren und Kennzahlen zu verdichten oder zu filtern. Zusätzlich dient Abbildung 2.4 der Veranschaulichung der vorgestellten Operationen.

Drill-Down

Bei der Drill-Down-Operation findet eine Navigation zu detailliertere Daten statt. Anschaulich kann eine Hierarchie als Graph angesehen werden. Durch die Drill-Down-Funktion wandert der Benutzer entlang eines definierten Pfades von einem höhergelegenden Hierarchieobjekt zu einem tiefergelegenden Hierarchieobjekt. Der Detaillierungsgrad der Daten wird dadurch verfeinert.

Roll-Up

Die Roll-Up-Operation ist die Inverse der Drill-Down-Operation. Ausgehend von einem Hierarchieobjekt wird entlang eines Pfades auf ein höhergelegenes Hierarchieobjekt zugegriffen. Der Detaillierungsgrad der Daten wird dadurch verringert.

Slice

Bei dieser Operation wird bildlich gesehen eine „Scheibe“ aus dem Cube extrahiert (s. oranger Bereich in Abbildung 2.4). Dies entspricht einer Einschränkung des OLAP Cubes in einer bestimmten Dimension.

Dice

Diese Operation entspricht dem Ausschneiden eines Teil-Cubes durch Einschränkungen auf mehreren Dimensionen.

Für die Realisierung des Datenmodells existieren unterschiedliche Ansätze, die in den folgenden Abschnitten genauer betrachtet werden.

Relational OLAP (ROLAP)

Greift ein OLAP-System bei der Analyse auf die Daten einer relationalen Datenbank zu, wird dies als relationales OLAP (ROLAP) bezeichnet. Dabei bilden die in Bezug stehenden Relationen die Dimensionen in einem denormalisiertes Sternschema (engl. *Star Schema*)

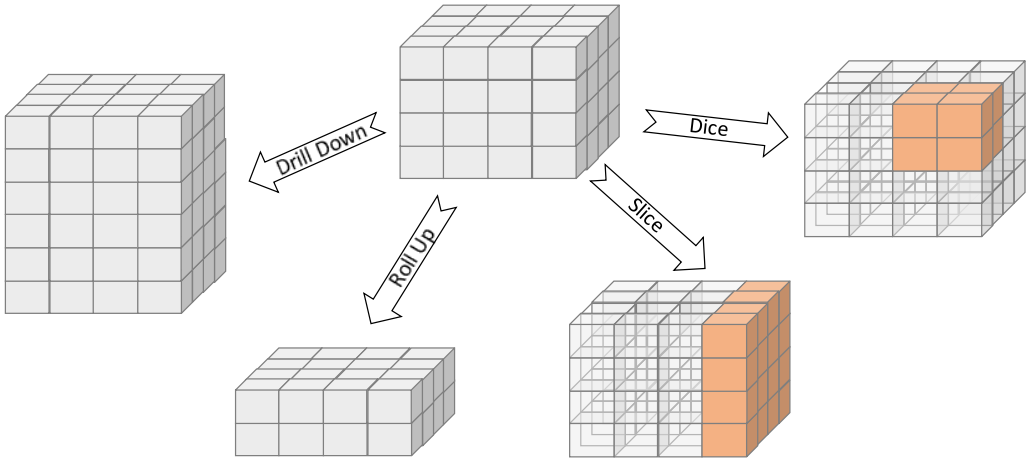

Abbildung 2.4.: OLAP-Cube-Perspektiven mit den vorgestellten OLAP-Operationen Drill Down, Roll Up, Slice und Dice.

ab, dessen Layout sich erkennbar von einem operativen System unterscheidet (s. Abbildung 2.5).

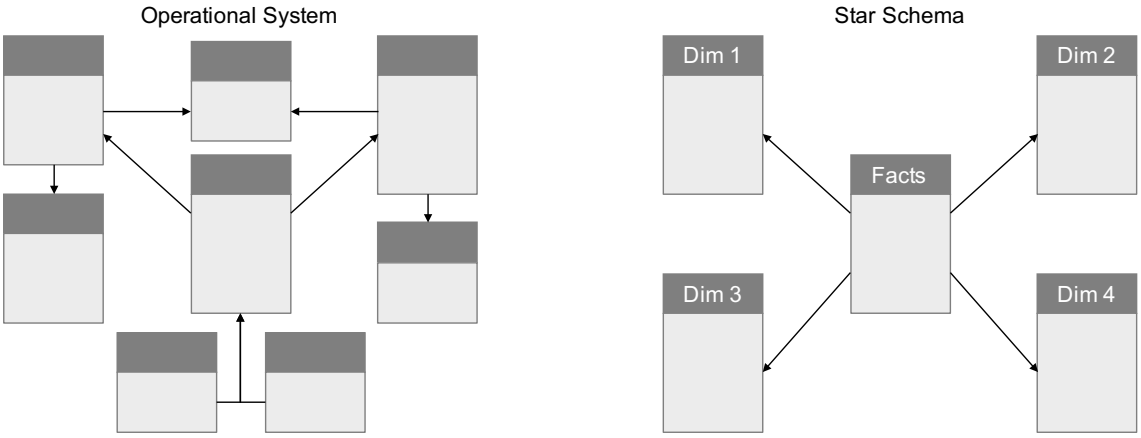

Abbildung 2.5.: Unterschied zwischen der Anordnung der Relationen eines operativen Systems (links) und der Anordnung der Relationen im Sternschema (rechts) in Anlehnung an [Tot00, S. 112].

Das Sternschema Die Bezeichnung Star Schema leitet sich aus der sternförmigen Anordnung der Relationen ab. Wie rechts in Abbildung 2.5 dargestellt, besteht das Datenmodell aus einer zentralen Relation, die sogenannte Faktentabelle, und aus mehreren, sternförmig angeordneten Relationen, die als Dimensionstabellen bezeichnet werden.

Die Faktentabelle repräsentiert alle Beziehungen und Measures der aus Geschäftsprozessen entstehenden Ereignissen eines Unternehmens. In der Regel umfasst die Faktentabelle eine große Anzahl an Einträgen, jedoch eine relativ geringe Anzahl an Spalten.

Die beschreibenden Informationen der Fakten werden in den zugehörigen Dimensionstabellen gespeichert. Diese können beispielsweise Informationen zum Kunden, Produkt oder Ort beinhalten. Jede Dimensionstabelle steht in einer 1:n-Beziehung zur Faktentabelle. Die Dimensionstabellen haben im Allgemeinen einen einzigen Primärschlüssel, der jeden Tupel eindeutig identifiziert. Im Gegensatz dazu besitzt die Faktentabelle den Primärschlüssel jeder zugehörigen Dimensionstabelle als Fremdschlüssel. In der Regel stellt die Menge der Fremdschlüssel den Primärschlüssel der Faktentabelle dar. Dies impliziert, dass jede Kombination von Dimensionen nur einmal vorkommen darf. In der Praxis wird dieser Fall durch eine Zeit-Dimension sichergestellt.

Das Sternschema hat nicht die Normalisierung als Ziel. Stattdessen wird durch die Verletzung der dritten Normalform die Redundanz und damit einhergehend der erhöhte Speicherbedarf für eine bessere Verständlichkeit des Datenmodells in Kauf genommen. Ein weiterer Vorteil dieser Missachtung ist die reduzierte Anzahl der benötigten Join-Anfragen für die gewünschte Analyse. Dieser geringere Aufwand führt zu einer schnelleren Ausführungsgeschwindigkeit der Abfragen (vgl. [Tot00, S. 111f.]).

In der Praxis wird ROLAP aufgrund der weiten Verbreitung von relationalen Datenbanken und der Abfragesprache SQL sehr häufig eingesetzt. Als weiteres Datenmodell wird im nächsten Abschnitt das sogenannte Multidimensional OLAP vorgestellt.

Multidimensional OLAP - MOLAP

Im Vergleich zur Datenspeicherung in einer relationalen Datenbank, die die Daten als Datensätze speichert, werden beim multidimensionalen OLAP (MOLAP) die Daten direkt in eine dafür vorgesehene multidimensionale Datenbank als Datenpunkte gespeichert. Dabei sind Vorberechnungen der Aggregationen im OLAP Cube erforderlich. Dieser Aufwand ermöglicht während der Analyse eine kürzere Ausführungsdauer, führt jedoch gleichzeitig zu einem größeren Speicherplatzverbrauch und einem längeren ETL-Prozess, da die Aggregationen für jede Kombination der Dimensionen vorzuberechnen sind.

Eine einzelne Teilmenge der Dimensionen wird als *Cuboid* bezeichnet. Die Anzahl an Cuboids steigt exponentiell zur Anzahl der Dimensionen an. Bei einem OLAP Cube mit vier

Dimensionen werden bereits 2^4 Cuboids benötigt (s. Abbildung 2.6 links).

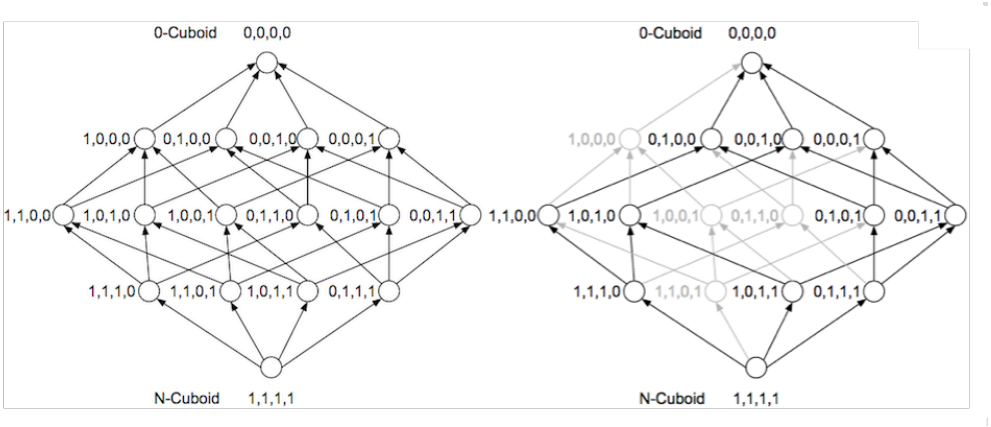

Abbildung 2.6.: Aufbau eines Aggregationsgitters für die Berechnung der Cuboids mit vier Dimensionen. Links Full Cube, rechts ein Beispiel eines Partial Cubes.

Der größtmögliche Cuboid, bei dem die Aggregation nach allen Dimensionen stattfindet (in der Abbildung 2.6 durch eine 1 symbolisiert), wird als *N-Cuboid* bezeichnet. Ausgehend von diesem N-Cuboid werden durch das Entfernen von jeweils einer Dimension die *N-1-Cuboids* berechnet. Dieser Prozess wird solange durchgeführt, bis keine Dimensionen zur Berechnung der Measures vorhanden sind. Dies entspricht dem kleinstmöglichen Cuboid und wird als *0-Cuboid* bezeichnet.

In der Regel werden jedoch mehr als vier Dimensionen für eine interaktive Analyse verwendet. Aus diesem Grund hat sich in der Praxis der *Partial Cube* etabliert (s. Abbildung 2.6 rechts). Bei einem Partial Cube werden nur die Cuboids vorberechnet, die bei der Analyse benötigt werden. Dies führt zu einer Reduzierung der Vorberechnungen und des Speicherplatzverbrauchs. Eine ungünstige Wahl des Partial Cubes kann jedoch dazu führen, dass Anfragen nicht beantwortet werden können, da nicht alle Cuboids vorberechnet wurden. Aus diesem Grund ist der Partial Cube in den Fällen uneingeschränkt sinnvoll, in denen das Analysebedürfnis der Benutzer genau bekannt ist.

2.1.5. Multidimensionale Abfragesprache: MDX

Analog zum relationalen Modell mit der Abfragesprache SQL ist für das multidimensionale Datenmodell die Abfragesprache MDX[1] (engl. für **M**ulti**D**imensional E**X**pression, vgl.

[1] MDX wurde ursprünglich von Microsoft entwickelt und im Jahr 1998 veröffentlicht.

[SHWC05]) entstanden. MDX stellt eine auf das multidimensionale Datenmodell zugeschnittene Menge an Operationen bereit, wie z. B. Drill-Down, Roll-Up, Slicing und Dicing (vgl. Abschnitt 2.1.4). Inzwischen hat sich diese Abfragesprache zu einem Standard[2] entwickelt und findet in unterschiedlichen Produkten Anwendung (vgl. [Kem11]).

Eine SQL-Abfrage stellt das Ergebnis in zweidimensionaler Form (Spalten und Zeilen) bereit. Im Gegensatz dazu liefert eine MDX-Abfrage als Ergebnis wiederum einen multidimensionalen Cube zurück.

Ähnlich wie SQL ist eine MDX-Anfrage in die drei Bereiche „*SELECT*“, „*FROM*“ und „*WHERE*“ unterteilt. Im SELECT-Bereich werden die Achsen des Cubes deklariert. Für jedes Measure im Cube wird durch den Schnitt der Achsen in Kombination mit den Dimensionsangaben in der WHERE-Klausel ein sogenannter Kontexttupel gebildet (vgl. [Kem11]). Für MDX gilt die Besonderheit, dass die Measures als eigene Dimension mit dem Namen *MEASURES* behandelt werden. Schließlich gibt die FROM-Klausel an, aus welchem Cube die Daten für die Anfrage ausgelesen werden sollen.

MDX stellt eine Vielzahl von OLAP-Funktionen für die Navigation durch Hierarchien zur Verfügung. Beispielsweise ist für eine Rückgabe, die alle Monate eines Jahres auflistet, keine explizite Angaben der einzelnen Monate in der MDX-Abfrage notwendig. Diese Eigenschaft wird als erweiterte Selektierbarkeit bezeichnet. Listing 2.1 zeigt ein Beispiel einer MDX-Abfrage.

```
1  SELECT
2    {[Measures].[Profit]}, {[Measures].[Revenue]} ON COLUMNS,
3    {[Dates].[Year].[2015].CHILDREN}, {[Dates].[Year].[2014].CHILDREN} ON ROWS
4  FROM Sales
5  WHERE
6    [Supplier].[Region].[Germany]
```

Listing 2.1: Beispiel einer MDX-Anfrage. Berechne den Profit und Umsatz aus dem Jahr 2015 und 2014 auf Monatsebene für alle Lieferanten aus Deutschland.

Ein Beispiel-Ergebnis der MDX-Abfrage aus Listing 2.1 ist in Tabelle dargestellt.

MDX ist eine mächtige Abfragesprache und wird daher bei komplexen OLAP-Operationen häufig eingesetzt. Neben dem Vorteil der erweiterten Selektierbarkeit besteht eine weite-

[2] s. Spezifikation von MDX unter `https://msdn.microsoft.com/en-us/library/ms145506.aspx`.

		Profit	Revenue
2015	January	2596	11553
	February	3667	13009
	...	...	...
2014	January	1877	9333
	February	2269	10019
	...	...	...

Tabelle 2.1.: Ergebnis der MDX-Abfrage aus Listing 2.1.

re Stärke von MDX darin, mit Roll-Up- und Drill-Down-Operationen durch Hierarchien entlang eines Pfades zu navigieren.

MDX wird bei der Konzeption des ETL-Prozesses eine wichtige Rolle einnehmen. Jedoch kann im Rahmen dieser Abschlussarbeit nicht auf den vollen Funktionsumfang von MDX eingegangen werden. Daher findet der interessierte Leser in andere Quellen weitere Informationen zur multidimensionalen Abfragesprache MDX (vgl. [SHWC05]; [Kem11]; [WZP05]).

2.2. Die Idee hinter dem Konzept Semantic Web

Die Idee hinter dem von Tim Berners-Lee geprägten Konzept Semantic Web besteht darin, das herkömmliche Web durch Standards zu erweitern. Im Vordergrund steht dabei die Verwertung der Informationen von Maschinen und der einfache Austausch zwischen diesen (vgl. [BLHL+01]). Zusätzlich sind grundlegende Anforderungen an diese Standards notwendig, die neben einer klaren Definition auch eine flexible Anwendung und mögliche Erweiterungen erlauben sollen.

Eine geeignete Repräsentation der Informationen ist somit eines der wichtigen Ziele des Semantic Webs. Für den maschinellen Umgang mit diesen Daten sind folglich Methoden für die Kommunikation und den Austausch zwischen Rechnern zu finden und zu definieren (vgl. [HKRS07, S. 12]). Dieser Aufgabe hat sich das World Wide Web Consortium[3] (W3C) verschrieben. Der daraus entwickelte Standard Resource Description Framework ist Gegenstand der folgenden Abschnitte.

[3] s. Webseite des World Wide Web Consortium unter `http://www.w3.org/Consortium/`.

2.2.1. Resource Description Framework

Das Resource Description Framework (RDF) ist eine Modellierungssprache für die Beschreibung strukturierter Informationen über Ressourcen im Web und ein Standard des W3C (vgl. [LS99]). Die erste offizielle Spezifikation wurde im Jahr 1999 veröffentlicht, wobei die Konzentration damals auf der Repräsentation von Metainformationen der Web-Ressourcen lag. Heute dient RDF, aufgrund verschiedener Überarbeitungen der Spezifikation, als Grundlage des Semantic Webs und wird häufig als Darstellungsformat gewählt. Zum Zeitpunkt der Abschlussarbeit wurde die aktuelle Version[4] im Juni 2014 veröffentlicht.

2.2.2. RDF-Datenmodell

Die auf elementaren Aussagen aufgebaute Modellierungssprache RDF beschreibt Verbindungen und Beziehungen zwischen Web-Ressourcen. Jede Aussage (engl. *Statement*) wird in RDF in drei grundlegenden Einheiten unterteilt: *Subjekt*, *Prädikat* und *Objekt*. Ein Zusammenschluss dieser drei Einheiten wird als *Triple* bezeichnet. Das Subjekt und das Objekt repräsentieren hierbei Web-Ressourcen, während das Prädikat ein Merkmal und eine Beziehung zwischen dem Subjekt und dem Objekt beschreibt. Im Gegensatz zu einem Subjekt oder einem Prädikat kann ein Objekt entweder eine weitere Web-Ressource oder ein Datenwert (engl. *Literal*) ohne weiterführende Beziehung sein.

Die Menge der Triples in einem RDF-Dokument bildet einen gerichteten Graphen. Ein Graph besitzt eine Menge von Knoten (Subjekte oder Objekte), die durch gerichtete Kanten (Prädikate) verbunden werden. Sowohl Knoten als auch Kanten werden mit eindeutigen Bezeichnungen gekennzeichnet. Abbildung 2.7 zeigt ein einfaches Beispiel des Statements „Alice kennt Bob“ durch einen gerichteten Graphen mit zwei Knoten und einer Kante.

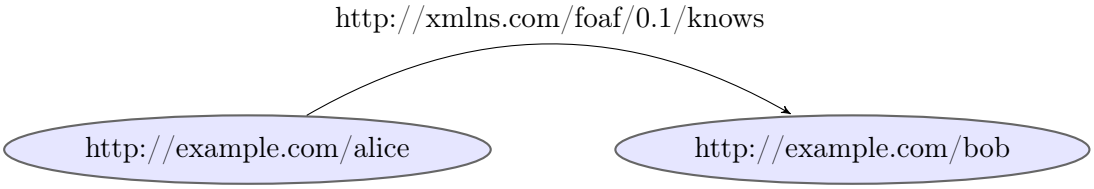

Abbildung 2.7.: Beispiel eines RDF-Graphs zur Beschreibung einer Beziehung zwischen den Web-Ressourcen Alice und Bob.

[4] s. Spezifikation von RDF unter `http://www.w3.org/TR/2014/NOTE-rdf11-primer-20140624/`.

Die dezentrale Verwaltung von Web-Ressourcen führt zu zwei grundlegenden Problemen. Einerseits kann es vorkommen, dass gleiche Web-Ressourcen unterschiedliche Bezeichner erhalten, andererseits kann es vorkommen, dass für unterschiedliche Web-Ressourcen der gleiche Bezeichner verwendet wird. Um dem zweiten Problem entgegenzuwirken, werden im Allgemeinen zur eindeutigen Identifizierung von Web-Ressourcen sogenannte *Uniform Resource Identifiers* (URI, engl. für „einheitlicher Bezeichner für Ressourcen“) als Indikatoren eingesetzt. Eine URI ist eine Zeichenfolge, die abstrakte oder auch physikalische Ressourcen identifiziert (vgl. [BLFM04]).

Wie in Abbildung 2.7 zu sehen ist, werden sowohl Knoten als auch Kanten eines RDF-Graphs mit URIs beschriftet. Bei Literalen sowie sogenannten *Blank Nodes* ist zu berücksichtigen, dass diese Regel nicht angewendet wird (vgl. [HKRS07, S. 56-59]). Literale sind reservierte Bezeichner für RDF-Ressourcen eines bestimmten Datentyps. Im Allgemeinen werden diese Werte durch eine Zeichenkette beschrieben. Der optionale Datentyp gibt dabei an, wie der Datenwert zu interpretieren ist. So beschreiben die Zeichenfolgen „012“ und „12“ dieselbe natürliche Zahl, aber unterschiedliche Zeichenketten. Bei fehlender Angabe eines Datentyps wird das Literal als Zeichenkette interpretiert.

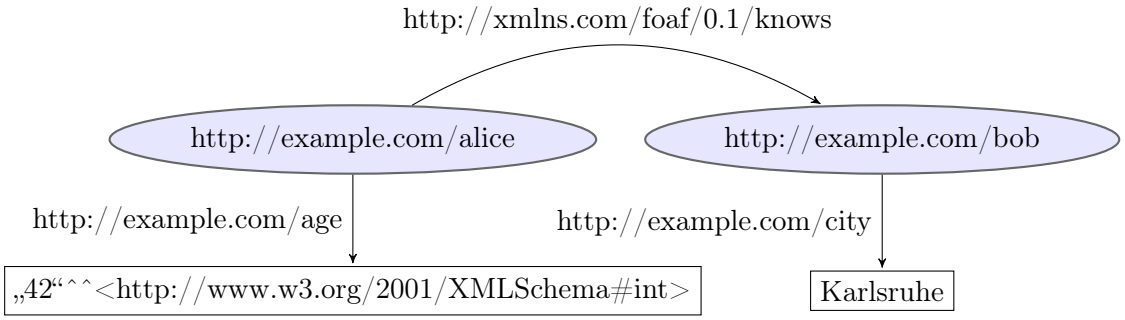

Abbildung 2.8.: Beispiel eines RDF-Graphs mit Literalen und optionalem Datentyp.

Abbildung 2.8 zeigt ein einfaches Beispiel eines RDF-Graphs mit Literalen. Nach gängiger Konvention werden Ressourcen eines RDF-Graphs durch Ellipsen und Literale durch Rechtecke dargestellt.

2.2.3. RDF-Serialisierung

Wie im vorherigen Abschnitt erläutert, handelt es sich bei RDF um ein Datenmodell. Zur Beschreibung eines RDF-Graphs können unterschiedliche Serialisierungen verwendet

werden. Die im Allgemeinen verwendeten Notationen werden in diesem Abschnitt kurz erläutert.

RDF / XML

Die Auszeichnungssprache *XML* (**Ex**tensible **M**arkup **L**anguage) wird häufig für einen plattform- und implementierungsunabhängigen Austausch von Daten und Informationen zwischen verschiedenen Rechnern mit unterschiedlichen Betriebssystemen verwendet (vgl. [BPSM+98]). Listing 2.2 zeigt das Beispiel aus Abbildung 2.8 in der RDF/XML-Notation.

```
1  <?xml version="1.0" encoding="utf-8" ?>
2  <rdf:RDF xmlns:foaf="http://xmlns.com/foaf/0.1/" xmlns:ex="http://example.com/"
3           xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
4    <rdf:Description rdf:about="http://example.com/alice">
5      <foaf:knows>
6        <rdf:Description rdf:about="http://example.com/bob">
7          <ex:city>Karlsruhe</ex:city>
8        </rdf:Description>
9      </foaf:knows>
10     <ex:age rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">
11       42
12     </ex:age>
13   </rdf:Description>
14 </rdf:RDF>
```

Listing 2.2: Beispiel eines RDF-Dokuments.

Turtle

RDF/XML wurde lange Zeit als Standard-Format zur Beschreibung von RDF-Daten angesehen. Aufgrund der Tatsache, dass XML auf einem Baum- und RDF auf einem Graphen-Modell basiert, wurden im Laufe der Zeit weitere Serialisierungen entwickelt. Eine davon ist die häufig verwendete *Turtle*-Notation[5]. Listings 2.3 zeigt das oben dargestellte Beispiel aus Listing 2.2 in der Turtle-Notation. URIs werden dabei in spitzen Klammern dargestellt. Jede Aussage wird durch einen Punkt abgeschlossen. Um nicht bei jedem Triple das Sub-

[5] s. Spezifikation von Turtle unter `http://www.w3.org/TeamSubmission/turtle/`.

jekt angeben zu müssen, können durch Semikola getrennt weitere Prädikate und Objekte angegeben werden. Zusätzlich gibt es die Möglichkeit, URIs durch die *@prefix*-Syntax einen kürzeren Namen zu geben und diesen im Dokument anstelle der URI wiederzuverwenden.

```
1 @prefix ex: <http://example.com/> .
2 @prefix foaf: <http://xmlns.com/foaf/0.1/> .
3
4 ex:alice
5   foaf:knows ex:bob ;
6   ex:age "42"^^<http://www.w3.org/2001/XMLSchema#integer> .
7 ex:bob ex:city "Karlsruhe" .
```

Listing 2.3: Turtle-Notation des Beispiels aus Listing 2.2.

Die kürzere und für den Anwender besser lesbare Turtle-Notation wird im Rahmen dieser Abschlussarbeit in allen RDF-Listings verwendet. Aus Gründen der Übersichtlichkeit werden die verwendeten Präfixe in allen späteren Listings nicht explizit angegeben, dafür aber im Anhang A.2 alphabetisch sortiert aufgelistet.

N-Triples

In diesem Abschnitt wird eine weitere Notation erläutert, die in der Konzeption und Implementierung des später vorgestellten ETL-Prozesses benötigt wird. *N-Triples*[6] ist eine zeilenbasierte Darstellung eines RDF-Graphs. Jede Zeile repräsentiert genau ein Triple der Form „<SubjectURI> <PredicateURI> <ObjektURI> .“. Der Punkt in der Zeile definiert das Ende der jeweiligen Aussage. Das Beispiel aus Listing 2.2 wird in der N-Triples-Notation im Listing 2.4 dargestellt.

```
1 <http://example.com/alice> <http://example.com/age> "42"^^<http://www.w3.org
      /2001/XMLSchema#integer> .
2 <http://example.com/alice> <http://xmlns.com/foaf/0.1/knows> <http://example.com
      /bob> .
3 <http://example.com/bob> <http://example.com/city> "Karlsruhe" .
```

Listing 2.4: N-Triples-Notation des Beispiels aus Listing 2.2.

Wie in Listing 2.4 zu erkennen ist, kann der optionale Datentyp eines Literals nach der

[6] s. Spezifikation von N-Triples unter `http://www.w3.org/TR/n-triples/`.

Zeichenfolge „^^“ angegeben werden. Auch hier gilt: Ist der Datentyp nicht definiert, wird der Wert des Literals als Zeichenkette interpretiert.

Der Vorteil dieser Notation besteht in der einfachen und zeilenbasierten Darstellung der Triples. Jedes Triple wird in einer einzelnen Zeile beschrieben. Wie im Verlauf der Abschlussarbeit zu sehen sein wird, wird diese Notation bei der Umsetzung des ETL-Prozesses eine entscheidende Rolle spielen. Ein Nachteil dieser Serialisierung besteht jedoch in dem benötigten Speicherplatz. Da keine Präfixe oder andere Möglichkeiten einer kürzeren Darstellungsform bestehen, sind in der Regel RDF-Dokumente in der N-Triples-Notation im Vergleich zur Turtle-Notation größer.

2.2.4. RDF-Schema

Allein der Einsatz von URIs in RDF erlaubt noch keine semantisch eindeutige Interpretation aller Informationen (vgl. [HKRS07, S. 47-50]). Wie bereits im Abschnitt 2.2.2 beschrieben, können gleiche URIs für unterschiedliche Ressourcen und unterschiedliche URIs für gleiche Ressourcen verwendet werden. Dieser Umstand erzwingt die Verwendung von wohldefinierten Schemata, sogenannte RDF-*Vokabulare*. Unter einem RDF-Schema (RDFS) wird eine Menge von Bezeichnern mit definierter Bedeutung verstanden, die RDF semantisch erweitert und eine Beschreibung benutzerspezifischer Klassen erlaubt (vgl. [HKRS07, S. 66-68]).

In den vorangegangenen Abschnitten wurde erläutert, wie Aussagen über Web-Ressourcen getätigt werden können, z. B. am konkreten Beispiel aus Listing 2.2: „Alice kennt Bob“. Durch die Namensgebung *Alice* und *Bob* interpretiert der Anwender den Bezug zu Personen. Maschinen können nicht ohne Weiteres diesen Bezug herstellen. Die Verwendung von wohldefinierten Vokabularen soll die Interpretation dieser Aussage auch für Maschinen ermöglichen.

Im Listing 2.2 wird durch die Verwendung des *FOAF*-Vokabulars[7] (Friend of a Friend) mit der URI „foaf:knows “ impliziert, dass es sich hierbei bei Alice und Bob um Personen, also der Klasse „foaf:Person“ handelt. Wie in vielen Programmiersprachen üblich, beginnen Klassen mit einem Großbuchstaben und Merkmale mit Kleinbuchstaben.

Das Thema dieser Arbeit behandelt jedoch keine Beziehungen zwischen Personen, sondern die Analyse von statistischen Datensätzen. In den letzten Jahren hat sich zur Veröffentli-

[7] s. Spezifikation des FOAF-Vokabulars unter `http://xmlns.com/foaf/spec/`.

chung solcher Datensätze nach dem Linked-Data-Prinzip das RDF Data Cube Vocabulary bewährt. Diese werden in den nächsten Abschnitten genauer betrachtet.

2.2.5. Statistical Linked Data

Das Linked-Data-Prinzip beschreibt eine Menge von bewährten Verfahren für das Veröffentlichen und Verlinken von strukturierten Daten im Web (vgl. [BHBL09]). Die 2006 von Tim Berners-Lee veröffentlichte Arbeit „Linked Data: Design Issues“ [BL06] identifiziert vier Prinzipien für die standardisierte Veröffentlichung von Daten:

- Die frei im Web verfügbaren Daten werden mit URIs identifiziert.
- Das *Hypertext Transfer Protocol* (HTTP) dient zum Auffinden dieser Daten über das Web.
- Die Verlinkung dieser Daten soll mit etablierten Standards erfolgen, z. B. mittels RDF und der graphenbasierten Abfragesprache *SPARQL*[8] (rekursives Akronym: **S**PARQL **P**rotocol **A**nd **R**DF **Q**uery **L**anguage) (vgl. [PAG06]).
- Das Hinzufügen von Links zu anderen URIs soll die Möglichkeit bieten, neue Daten und Informationen zu entdecken.

In den letzten Jahren ist das Interesse gestiegen, statistische Daten nach dem Linked-Data-Prinzip zu veröffentlichen und so die Möglichkeit zu bieten, Daten mit anderen Informationen aus unterschiedlichen Quellen und RDF Stores zu kombinieren. Ein Vorteil besteht hierbei darin, beliebige Zusatzinformationen mit den numerischen Daten verlinken zu können, um die Bedeutung der Daten näher zu bestimmen. Wie in Abschnitt 1.1 bereits erwähnt können z. B. Provenance-Informationen oder weitergehende Informationen hinzugefügt werden. Des Weiteren können auch interne Daten mit den numerischen Daten verlinkt und für die Analyse verwendet werden. Diese verlinkten Daten und Informationen werden als *Statistical Linked Data* bezeichnet.

Es existieren verschiedene Vokabulare, die für die Veröffentlichung statistischer Linked Data in Form von multidimensionalen Cubes verwendet werden können (vgl. [HHR+09]; [VLH+10]; [EV12]). Das RDF Data Cube Vocabulary hat sich jedoch weitestgehend durchgesetzt.

[8] s. Spezifikation von SPARQL unter `http://www.w3.org/TR/2013/REC-sparql11-overview-20130321/`.

2.2.6. Das RDF Data Cube Vocabulary

Das *RDF Data Cube Vocabulary*[9] (QB) ist ein RDF-Vokabular und ein W3C-Standard zur Beschreibung statistischer Datensätze in Form von multidimensionalen Cubes (vgl. [ZPdMS+13]). Die Veröffentlichung von statistischen Daten nach dem Linked-Data-Prinzip sowie die Repräsentation eines multidimensionalen Datenmodells mit RDF waren die wichtigsten Voraussetzungen bei der Entwicklung des QB-Vokabulars.

Wie im Abschnitt 2.1.4 bereits erläutert, bietet das multidimensionale Datenmodell (MDM) eine leicht zu interpretierende Anordnung der statistischer Daten, sodass Analysten einfach und effizient durch den Datenraum navigieren können. Hauptaugenmerk liegt in der Unterstützung der betrieblichen Entscheidungsfindung.

Statistische Daten können im Bereich Semantic Web mit dem QB-Vokabular beschrieben werden. Abbildung 2.9 veranschaulicht einen Auszug der relevanten Klassen und Beziehungen des Schemata.

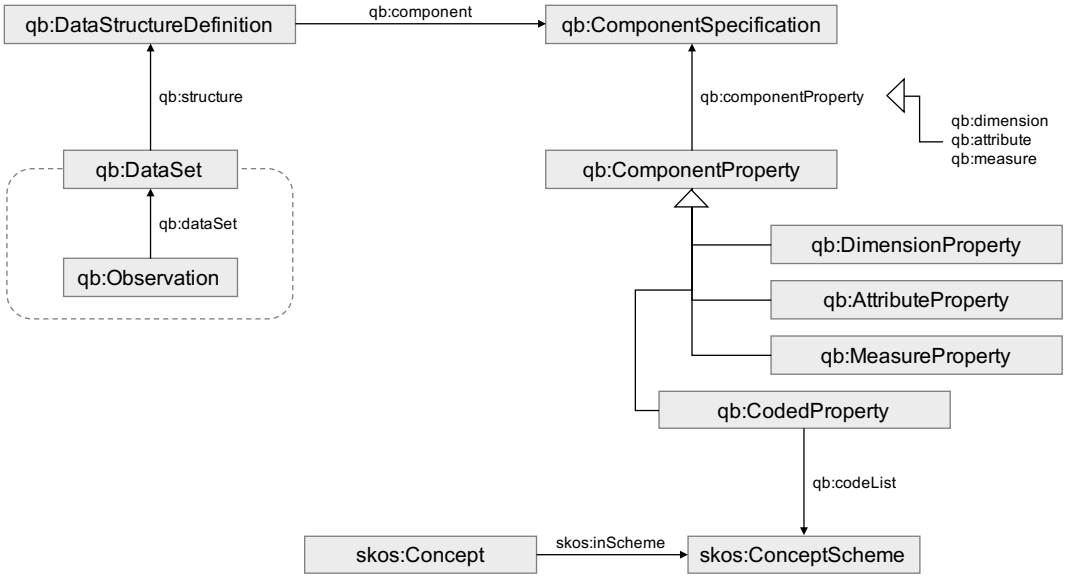

Abbildung 2.9.: Wichtigste Klassen und Beziehungen des RDF Data Cube Vokabulars. Quelle: `http://www.w3.org/TR/2014/REC-vocab-data-cube-20140116/`.

Der Datensatz (engl. *Dataset*) umfasst die Dimensionen, Measures sowie die Beobachtungen (engl. *Observations*) und Operationen, die mit den Daten ausgeführt werden können.

[9] s. Spezifikation von QB unter `http://www.w3.org/TR/2014/REC-vocab-data-cube-20140116/`.

Die wichtigsten Klassen und Beziehungen werden in der folgenden Auflistung näher erläutert.

- Die Struktur des Cubes wird durch die Klasse *qb:DataStructureDefiniton* (DSD) definiert. Analog zum OLAP-Konzept, bei dem die Daten in Fakten und Dimensionen unterteilt werden, findet bei QB eine Unterteilung der Daten in zwei Bereichen statt. Die erste Unterteilung durch die Property *qb:structure* beschreibt den zugehörigen Datensatz des Cubes. Die zweite Unterteilung findet durch die Property *qb:component* statt, welche für die Spezifikation der Dimensionen und Measures im Datensatz verantwortlich ist.
- *Observations* sind Instanzen von *qb:Observation*. Sie stellen einzelne Beobachtung des Cubes dar und können aus einem oder mehreren Dimensionen und Measures bestehen. Analog zum OLAP-Konzept beschreibt eine Observation einen Fakt.
- Die Klasse *qb:DataSet* stellt die Gesamtmenge der Observations dar. Durch die Verbindung *qb:structure* wird der Bezug zur Datenstruktur definiert.
- *qb:DimensionProperty*, *qb:AttributeProperty* und *qb:MeasureProperty* sind vererbte Klassen der *qb:ComponentProperty*-Klasse und repräsentieren die Dimensionen, Attribute und Measures des Cubes. Analog zum OLAP-Konzept beschreibt die Klasse qb:DimensionProperty die Dimensionen, qb:MeasureProperty die Measures und qb:AttributeProperty zusätzliche Attribute des Cubes.
- Im OLAP-Kontext besitzt jeder Fakt in der Regel einen Bezug zu einer konkreten Instanzen einer Dimensionen. Diese als Member bezeichnete Beziehung kann im QB-Vokabular entweder explizit, durch die Eigenschaft *qb:codeList*, oder implizit durch das Vorkommen in den Fakten beschrieben werden.
- In QB werden Hierarchien durch die Klasse *skos:ConceptScheme* und die Levels durch *skos:Concept* repräsentiert. Der Bezug eines Levels zu einer Hierarchie wird durch die Property *skos:inScheme* definiert. Der Bezug zu einer konkreten Instanz einer Dimension wird durch die Property *skos:member* bestimmt. Die Vater-Kind-Beziehung *skos:narrower* ermöglicht die Navigation zwischen den Levels (vgl. [KH11]).

In einem OLAP Cube wird in der Regel für jedes Measure eine Aggregationsfunktion definiert, wie z. B. *sum*, *min*, *max*, *avg* und *count*. In QB gibt es keine Beschreibung dieser Aggregationsfunktionen für Measures. Daher wird in dieser Arbeit zur Beschreibung

der Measures das QB4O-Vokabular[10] mit der Property *qb4o:aggregateFunction* verwendet. Dies ermöglicht den Einsatz der URIs für die Aggregationsfunktionen *qb4o:sum*, *qb4o:min*, *qb4o:max*, *qb4o:avg* und *qb4o:count*.

Für die Analyse einer enorm großen Datenmenge sind Technologien aus dem Big-Data-Bereich notwendig. Diese werden im nächsten Kapitel einer Beschreibung unterzogen.

2.3. Big Data

Es existieren unzählige Definitionen von Big Data (vgl. [KH14, S. 34-37]). Die anerkannteste Definition charakterisiert Big Data anhand von drei bis vier Kriterien (vgl. [Lan01]; [KH14, S. 35]; [Dor15, S. 7-8]): „Volume“, „Velocity“, „Variety“ und gegebenenfalls „Veracity“.

Volumen (Volumne)

Die große Menge an Daten ist der wesentlichste Aspekt von Big Data. Das anfallende Datenvolumen der zu verarbeitenden Informationen steigt stetig an. Ein Grund hierfür sind z. B. kleine und immer leistungsfähigere Computerchips, die in viele Lebensbereiche vordringen und neue Daten generieren. Dies führt zu einer anwachsenden Datenmenge, die bei der Analyse genutzt werden soll.

Geschwindigkeit (Velocity)

Dieses Merkmal wird nicht immer eindeutig interpretiert. Zum einen wird hierunter die Geschwindigkeit verstanden, mit der neue Daten entstehen (vgl. [Dor15, S. 7]; [Fas14]). Andererseits kann Velocity auch die Geschwindigkeit beschreiben, mit der Daten verarbeitet werden müssen (vgl. [KH14, S. 35]).

Vielfalt (Variety)

Das Merkmal Variety kennzeichnet die Heterogenität der Daten. Die zunehmende Anzahl an Datenquellen, die für die Analyse herangezogen werden, kann zu stark variierenden oder gar unbekannten Datenstrukturen führen. Ferner sind für Big Data viele der relevanten Daten unstrukturiert. Auch in solchen Fällen sollen Analysen möglich sein und zu neuen Erkenntnissen führen.

Richtigkeit (Veracity)

Dieses Merkmal wird in einigen Big-Data-Definitionen als viertes Merkmal angesehen

[10] s. Spezifikation von QB4O unter `http://purl.org/qb4olap/cubes#`.

(vgl. [BWBN14]; [KH14, S. 35]). Diese Eigenschaft bezieht sich auf die Qualität der Daten bezogen auf die Richtigkeit und Vollständigkeit. Bevor betriebliche Entscheidungen getroffen werden können, muss die Korrektheit und die Relevanz der Daten zum Zeitpunkt der Analyse bekannt sein.

In der Regel übersteigen eine solche Datenmenge und die für die Analyse verwendeten Prozesse die Speicher- und Verarbeitungskapazität eines einzelnen Rechners. Aus diesem Grund hat sich in den letzten Jahren die Verwendung von *Rechner-Clusters*[11] etabliert. Eine solche Architektur wurde dazu optimiert, eine Berechnung, eine Analyse oder ein Programm verteilt auf verschiedenen Rechnern parallel auszuführen. Die Eigenschaften einer solchen Architektur können in drei Bereiche zusammengefasst werden (vgl. [Dor15, S. 279]; [RSS15, S. 65-66]):

Parallele Verarbeitung der Daten

Die Sicherstellung einer konstanten Verarbeitungszeit bei steigendem Datenvolumen durch verteilte Parallelverarbeitung muss gewährleistet sein. Durch die Speicherung der Daten auf lokalen Festplatten vieler Rechnerknoten soll eine Verarbeitung der Daten jeweils auf den Knoten durchgeführt werden, auf denen die Daten gespeichert wurden. Dies erlaubt einen gleichzeitig stattfindenden und unabhängigen Datenzugriff auf die Daten einzelner Rechner. Diese als *Shared Nothing* bezeichnete Architektur soll eine lineare Skalierbarkeit garantieren, die praktisch unbegrenzt ist (vgl. [RSS15, S. 55-58]).

Horizontale Skalierung

Bei Bedarf kann durch Hinzufügen von neuen Rechnern die Kapazität des Clusters erweitert werden. Dieser Vorgang wird als horizontale Skalierung bezeichnet. Zur Vermeidung hoher Hardware-Kosten soll die Möglichkeit bestehen, das Cluster durch Rechner mit gewöhnlicher Hardware (engl. *Commodity Hardware*) bereitzustellen und zu erweitern.

Redundante Speicherung

Zum Zweck der Fehlertoleranz und für eine Verbesserung der Antwortzeiten sollen die Daten auf mehrere Rechnerknoten repliziert und redundant gespeichert werden. Bei Ausfall eines Rechners kann der entsprechende Prozess auf einem anderen Rechnerknoten fortgesetzt werden. Zur Vermeidung eines inkonsistenten Systems müssen

[11] Ein Cluster wird als Verbund von mehreren vernetzten Computern bezeichnet.

daher Datenänderungen auf alle Replikate übertragen werden. Ein weiterer Vorteil wird in der Ausführung eines Prozesses deutlich, indem ein Rechner mit hoher Last durch einen Rechner mit niedriger Last ersetzt werden kann.

Sowohl relationale Datenbanken, RDF Stores als auch OLAP Engines skalieren in der Regel nicht horizontal. Sie besitzen eine natürliche Grenze bzgl. ihrer Datenspeicher- und Datenverarbeitungskapazität (vgl. [Dor15, S. 260-262]). Bei der Analyse großer Datenmengen sind daher Big-Data-Technologien mit den oben definierten Eigenschaften notwendig. Mit Apache Hadoop sind derartige Technologien in einem Open Source Software Stack verfügbar.

2.3.1. Das Apache-Hadoop-Framework

Für die verteilte Parallelverarbeitung großer Datenmengen hat sich Apache Hadoop[12] als Standard etabliert[13]. Apache Hadoop ist ein quelloffenes und in Java entwickeltes Top-Level-Projekt der Apache Software Foundation.

Die Hauptkomponenten von Apache Hadoop bestehen aus dem Hadoop Distributed File System und dem Programmiermodell MapReduce. In den nachfolgenden Abschnitten werden diese Komponenten einer näheren Betrachtung unterzogen.

2.3.2. Das verteilte Dateisystem von Apache Hadoop: HDFS

Das *Hadoop Distributes Files System* (HDFS) ist ein verteiltes Dateisystem, welches von Googles 2003 vorgestellte *Googles File System* [GGL03] inspiriert wurde (vgl. [SRC10]). Die wichtigsten Ziele des Entwurfs lagen in der Fehlertoleranz bei Commodity-Hardware-Umgebungen, in der Ausrichtung großer Datenmengen und in der parallelen Batch-Verarbeitung dieser Daten. Letzteres bezweckt einen hohen Lesedurchsatz anstelle von niedrigen Latenzen. Zusätzlich basiert die Umsetzung von HDFS auf der Annahme, dass Daten oft gelesen, aber selten bis gar nicht verändert werden (das sogenannte *write-once-read-many*-Prinzip, vgl. [RSS15, S. 66-69]).

HDFS baut auf einer Master-Slave-Architektur auf, bestehend aus dem Masterknoten *NameNode* und mehreren *DataNodes*. Der NameNode ist für die Verwaltung des Datei-

[12] s. Webseite des Apache Hadoop Projekts unter `http://hadoop.apache.org/`.
[13] s. Apache Hadoop PowerdBy unter `http://wiki.apache.org/hadoop/PoweredBy`.

systems sowie für den Dateizugriff des Clients verantwortlich. Der NameNode unterstützt eine Vielzahl von Operationen, wie z. B. das Öffnen, Schließen und Umbenennen von Dateien und Verzeichnissen. Die DataNodes dagegen sind für die Speicherung und Verwaltung der Daten zuständig. Bei der Speicherung der Daten im HDFS findet zunächst eine Aufteilung der Dateien in Datenblöcke statt. Diese Blöcke werden über die verschiedenen DataNodes horizontal verteilt. Die Größe des Blocks kann bei jedem Upload in das HDFS und pro Datei einzeln festgelegt werden (Standardwert ist 128 MB). Ferner kann pro Datei ein Replikationsfaktor definiert werden. Dieser Wert gibt die Anzahl der Replikationen von jedem einzelnen Block im HDFS an. Standardgemäß werden bei Apache Hadoop drei Replikationen angelegt. Die Zuordnung zwischen DataNode und Block erfolgt durch den NameNode. Zur Veranschaulichung der HDFS-Architektur dient die Abbildung 2.10.

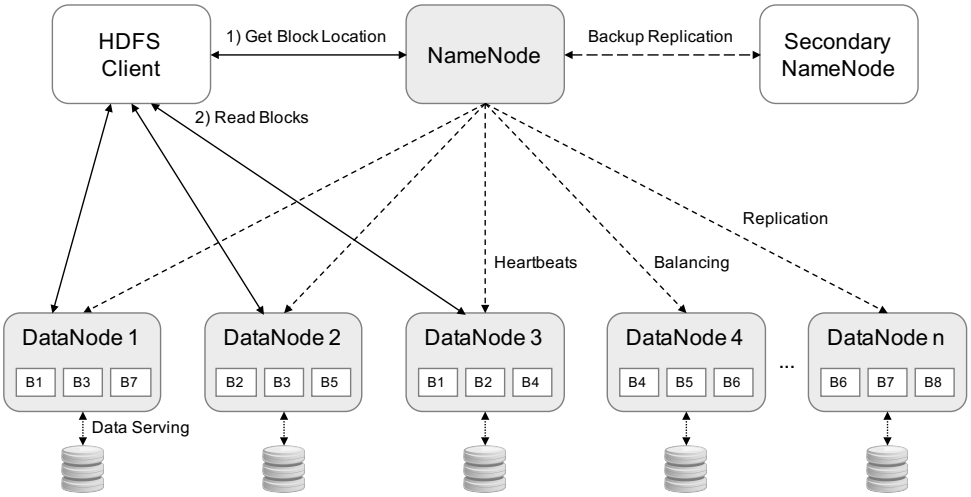

Abbildung 2.10.: Architektur von HDFS und Beispiel einer Leseoperation in Anlehnung an [RSS15, S. 67] und [Dor15, S. 280].

Die Matainformationen zur Verzeichnisstruktur und weitergehende Informationen (z. B. welcher Block wurde auf welchem DataNode gespeichert?) werden in dem lokalen Hauptspeicher des NameNodes gespeichert (vgl. [RSS15, S. 67]). Ferner werden alle Änderungsvorgänge in das lokale Dateisystem übertragen, um jederzeit einen nachvollziehbaren Ablauf der ausgeführten Operationen zu gewährleisten.

Die Registrierung eines neuen DataNodes erfolgt über eine *Heartbeat*-Message (vgl. [SRC10]). Zudem steht der NameNode in ständigem Kontakt mit allen Rechnern des Clusters. So wird gewährleistet, dass ein DataNode seiner Funktion nachkommen kann. Wird diese Nachricht in einer bestimmten Zeitspanne nicht übermittelt, geht der NameNode von einem Ausfall

des Rechners aus und führt Maßnahmen zur Neuverteilung der verlorenen Blöcke auf die restlichen DataNodes durch.

Da ein NameNode einen *Single-Point-of-Failure* darstellt, ist es üblich, bei Ausfall des NameNodes die Verfügbarkeit des Dateisystems durch einen zweiten NameNode sicherzustellen, dem sogenannten *Secondary NameNode* (s. Abbildung 2.10).

Der Ablauf von Operationen ist bei HDFS fest vordefiniert. Wie in Abbildung 2.10 dargestellt, kontaktiert der Client, z. B. bei einer Lese-Operation, zuerst den NameNode. Dieser liefert eine Liste mit allen DataNodes, die eine Replikation der benötigten Blöcke aufweisen. Anschließend kontaktiert der Client die DataNodes direkt an, um die Blöcke der benötigten Dateien anzufordern.

Die Replikation der Datenblöcke im HDFS dient nicht nur dem Zweck der Fehlertoleranz und der Ausfallsicherheit, wie der nächste Abschnitt zeigen soll.

2.3.3. Parallele Ausführung mit dem Programmiermodell MapReduce

MapReduce ist ein von Google 2004 vorgestelltes Programmiermodell für die parallele Verarbeitung großer, unstrukturierter oder semi-strukturierter verteilter Datensätze in Rechner-Clustern (vgl. [DG04]). Aufgrund der Einfachheit, Ausfallsicherheit und hohen Flexibilität hat das MapReduce-Modell eine weite Verbreitung erfahren, vor allem durch die frei verfügbare Implementierung in Apache Hadoop.

Das Hauptziel einer MapReduce-Funktion liegt darin, ein definiertes Problem in mehrere Teilaufgaben, sogenannte *Map-Task*, zu zerlegen, diese über die Rechner eines Clusters für die parallele Berechnung zu verteilen und die Zwischenergebnisse innerhalb des Clusters auszutauschen. Nach Beendigung der Berechnungen werden die Zwischenergebnisse durch sogenannte *Reduce-Tasks* aggregiert und zu einem Endergebnis zusammengefasst (vgl. [Dor15, S. 280-281]). Eine parallele Ausführung ist möglich, da die Prozesse zur Berechnung der Ergebnisse zu den verarbeitenden Daten bewegt werden.

Ähnlich wie HDFS basiert auch die MapReduce-Engine von Apache Hadoop auf der Master-Slave-Architektur. Ein auszuführender MapReduce-Prozess, ein sogenannter *Job*, wird vom Master-Knoten, dem sogenannten *JobTracker*-Knoten, in mehrere *Tasks* zerlegt. Der JobTracker dient als Koordination- und Kontroll-Komponente eines MapReduce-Jobs und weist, nach der Zerlegung des Jobs in mehrere Tasks, diese den sogenannten *Worker-*

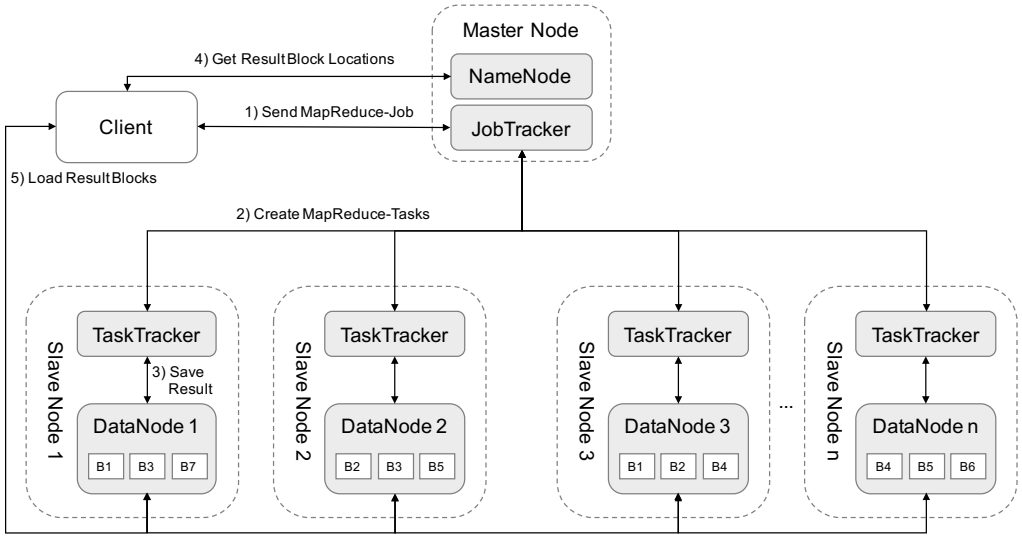

Abbildung 2.11.: Gesamtablauf eines MapReduce-Jobs in Apache Hadoop, angelehnt an [RSS15, S. 69].

Knoten zu. Zudem steht der JobTracker-Knoten in ständigem Kontakt mit den Worker-Knoten. So soll sichergestellt werden, dass abgebrochene Tasks erneut ausgeführt werden. Abbildung 2.11 veranschaulicht den Gesamtablauf eines MapReduce-Prozesses in Hadoop.

Grundlegendes Ziel bei der Ausführung eines MapReduce-Jobs ist die Zuweisung eines Task an einen Worker-Knoten, der im HDFS den für den Prozess notwendigen Block gespeichert hat. Dies führt zu einem lokalen Lesezugriff. Wurde der benötigte Datenblock zuvor nicht lokal gespeichert, muss dieser zuerst über HDFS angefordert und lokal gespeichert werden.

Die Aufgabe einer MapReduce-Funktion besteht darin, eine große Datenmenge von *Key-Value*-Paaren zusammenzufassen und auf eine kleinere Menge von Key-Value-Paaren zu reduzieren. Hierbei bilden die zwei Funktionen *Map* und *Reduce* die Hauptkomponenten der Berechnung.

$$map(key_{in}, value_{in}) \rightarrow [(key^1_{tmp}, value^1_{tmp}), \ldots, (key^n_{tmp}, value^n_{tmp})]$$
$$reduce([(key^s_{tmp}, [value^t_{tmp}, \ldots, value^u_{tmp}])]) \rightarrow [(key^q_{out}, value^q_{out}), \ldots, (key^p_{out}, value^p_{out})]$$

Ein wesentlicher Schritt eines MapReduce-Jobs ist die *Shuffle*-Phase. Voraussetzung für die Ausführung eines Reduce-Tasks ist die Sortierung der Ergebnisse eines Map-Tasks nach ihrem Key und dem Zusammenfassen der berechneten Values in eine Liste (vgl. [RSS15, S. 70]).

Nachdem in der Shuffle-Phase das Ergebnis des Map-Tasks sortiert und auf der Festplatte des Workers gespeichert wurde, wird der Reduce-Task angestoßen. Dabei findet anhand der Keys eine Aggregierung der Values statt. Anschließend wird das Ergebnis in eine HDFS-Datei gespeichert. Da es sich bei einem Worker-Knoten auch gleichzeitig um ein DataNode handelt, wird das Ergebnis entsprechend der Erläuterung im vorangegangenen Abschnitt 2.3.2 horizontal verteilt und repliziert.

MapReduce - Beispiel

In diesem Abschnitt soll das MapReduce-Programmiermodell anhand eines Beispiels erläutert werden. Analog zu HelloWorld-Programmen gibt es bei MapReduce das *Word-Count*-Beispiel. Hierbei sollen in einem Text die Anzahl der Wörter ermittelt werden. Abbildung 2.12 dient dabei der Veranschaulichung.

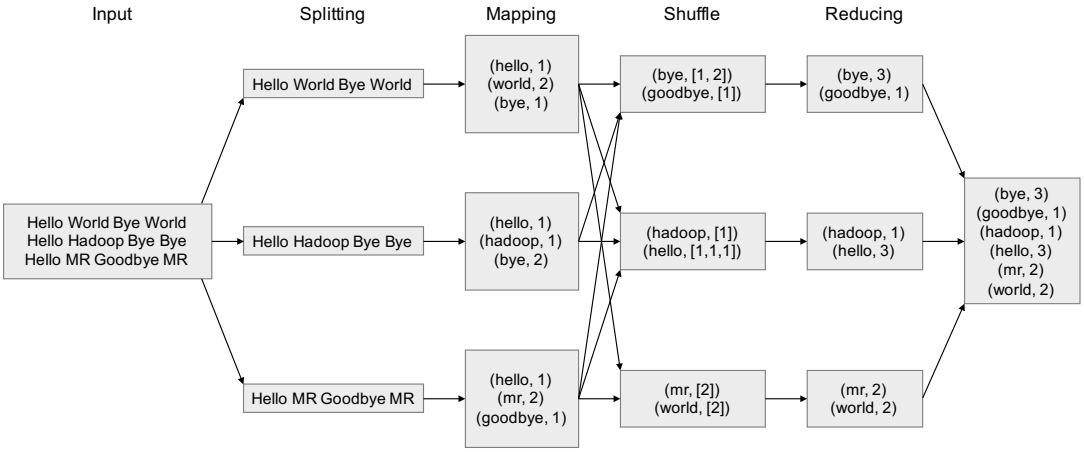

Abbildung 2.12.: Word-Count-Beispiel eines MapReduce-Jobs mit drei Map- und drei Reduce-Tasks, eigene Darstellung.

Die Eingabe besteht aus einer Text-Datei mit einem beliebigen Inhalt. Durch die Speicherung im HDFS wird die Datei in drei Datenblöcke unterteilt und auf die Rechner des Cluster verteilt. Der Map-Task transformiert jedes Wort in ein Key-Value-Paar. Beispielsweise könnte die Map-Phase die Eingabe wie folgt umwandeln:

$$map(sentence_1, \text{„Hello World Bye World“}) \rightarrow [(\text{hello}, 1), (\text{world}, 2), (\text{bye}, 1)]$$
$$map(sentence_2, \text{„Hello Hadoop Bye Bye“}) \rightarrow [(\text{hello}, 1), (\text{hadoop}, 1), (\text{bye}, 2)]$$
$$map(sentence_3, \text{„Hello MR Goodbye MR“}) \rightarrow [(\text{hello}, 1), (\text{mr}, 2), (\text{goodbye}, 1)]$$

Die Shuffle-Phase sortiert die berechneten Paare nach ihrem Key in alphabetischer Reihenfolge und fasst die berechneten Values in einer Liste zusammen. Dieses Ergebnis dient als Eingabe der Reduce-Funktionen. Der Reduce-Job hat die Aufgabe, die Elemente in der Liste zu zählen und in einem Wert zu aggregieren.

$$reduce([(\text{bye}, [1, 2]), (\text{goodbye}, [1])]) \rightarrow [(\text{bye}, 3), (\text{goodbye}, 1)]$$
$$reduce([(\text{hadoop}, [1]), (\text{hello}, [1, 1, 1])]) \rightarrow [(\text{hadoop}, 1), (\text{hello}, 3)]$$
$$reduce([(\text{mr}, [2]), (\text{world}, [2])]) \rightarrow [(\text{mr}, 2), (\text{world}, 2)]$$

Die Ausgabe der Reduce-Funktionen wird zu einem Endergebnis zusammengefasst und in einer Datei im HDFS gespeichert.

Im diesem Abschnitt wurde das Konzept des MapReduce-Programmiermodells an einem Beispiel erläutert. Die Entwicklung von MapReduce-Jobs erfordert für unternehmensspezifische Analysen zur Entscheidungsfindung jedoch spezialisierte Software-Entwickler. Diese Prozesse sind bei Änderungen schwierig zu pflegen. Ferner ist ein Umzug eines bestehenden MapReduce-Jobs in einen neuen oder ähnlichen Kontext in der Regel nicht leicht umzusetzen. Aus diesem Grund wird im nächsten Abschnitt Apache Hive vorgestellt.

2.3.4. Apache Hadoops Data Warehouse: Apache Hive

Apache *Hive*[14] ist eine 2009 von Facebook veröffentlichte Open Source Data-Warehousing-Lösung für Apache Hadoop (vgl. [TSJ+09], s. Abbildung 2.13). Ähnlich wie bei relationalen Datenbanken werden die Daten in tabellarischer Form mit Spalten und Zeilen dargestellt, wobei die Daten im HDFS gespeichert und horizontal verteilt werden. Diese Daten lassen sich mit der zugehörigen SQL-ähnlichen Abfragesprache *HiveQL* (Hive Query Language) abfragen. HiveQL unterstützt neben primitiven Datentypen wie z. B. Strings, Integer und Boolean auch Mengen (engl. *Collections*) wie Arrays, Maps und verschachtelte Kombinationen beider Datenstrukturen (vgl. [TSJ+10]). Eine HiveQL-Abfrage wird bei der Ausführung automatisch in ein oder mehreren MapReduce-Jobs kompiliert und auf Apache Hadoop ausgeführt. Zusätzlich ist es möglich, benutzerspezifische MapReduce-Skripte in die Abfragen zu integrieren.

Die zugrundeliegende I/O-Bibliotheken in Hive erlauben unterschiedliche Datenstrukturen. So ist es möglich, zeilenbasierte Textdateien wie CSV-Dateien oder komprimierte Datei-

[14] s. Webeite von Apache Hive unter `https://hive.apache.org/`.

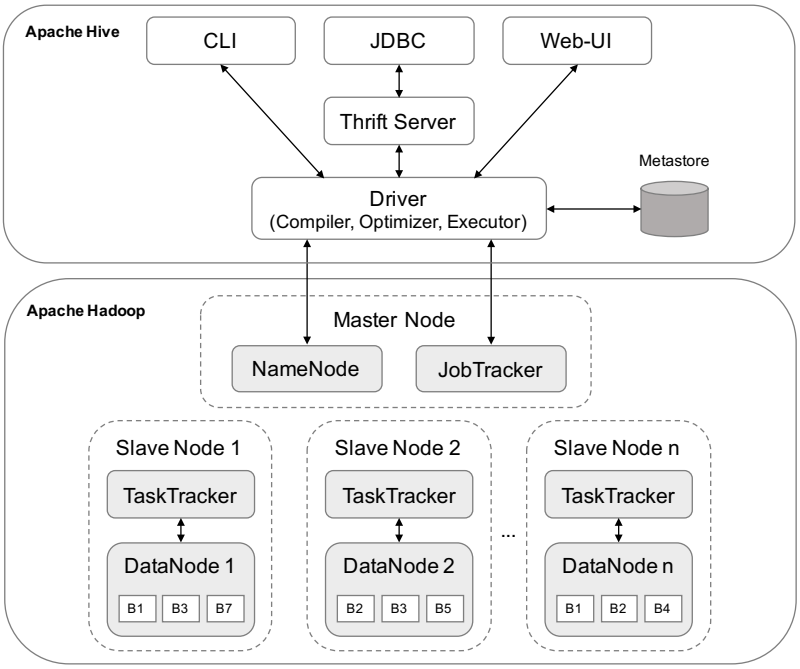

Abbildung 2.13.: Architektur von Apache Hive auf Basis von Apache Hadoop in Anlehnung an [TSJ+09].

en, wie z. B. im Avro[15]- oder Parquet[16]-Format, im HDFS abzulegen und mit HiveQL abzufragen.

Für die Speicherung der Metainformationen von Hive-Tabellen ist der sogenannte *Hive-Metastore* zuständig. In der Regel wird hierfür eine relationale Datenbank wie MySQL verwendet. Dieser Katalog speichert neben dem Hive-Schemata (Welche Hive-Tabellen existieren? Welche Spalten enthalten die Hive-Tabellen? Welchen Datentyp haben die Spalten? Welches Datenformat wird verwendet?) auch zusätzlich statistische Datenwerte (Wie viele Einträge besitzt die Hive-Tabelle? Wie viel Speicherplatz wird verbraucht?), die bei der Abfrage-Optimierung und Generierung der MapReduce-Jobs benötigt werden (vgl. [TSJ+09]; [TSJ+10]).

Ein großer Vorteil von Apache Hive ist die mögliche Anbindung mittels eines JDBC[17]-Treibers, welcher in dieser Abschlussarbeit bei der Umsetzung des ETL-Prozesses eine

[15] s. Apache Avro Dokumentation unter `http://avro.apache.org/docs/current/`.
[16] s. Apache Parquet Dokumentation unter `https://parquet.apache.org/documentation/latest/`.
[17] JDBC steht für Java Database Connection. Sie definiert eine einheitliche Schnittstelle (API) zu Datenbanken unterschiedlichster Hersteller.

wesentliche Rollen spielen wird. Zusätzlich wird in der Konzeption der Wide Column Store Apache HBase eine wichtige Funktion haben. Dies ist Gegenstand des nächsten Abschnitts.

2.3.5. Apache Hadoops Datenbank: Apache HBase

Apache *HBase*[18] ist eine von Googles *BigTable* [CDG+06] inspirierte, spaltenorientierte, fehlertolerante und horizontal skalierbare Datenbank auf Basis von HDFS. Sie zählt zu den sogenannten *NoSQL*-Datenbanksystemen[19] (vgl. [Geo11, S. 457]).

Ähnlich wie bei relationalen Datenbanken basiert das Datenmodell von HBase auf Tabellen. Eine HBase-Tabelle besteht aus Zeilen und Spalten. Eine Zeile beschreibt einen Datensatz, während eine Spalte ein Attribut repräsentiert. Entsprechend einem Primärschlüssel in relationalen Datenbanken erfolgt jeder Zugriff auf eine Zeile in HBase durch einen nicht veränderbaren und eindeutigen Schlüssel (engl. *Row Key*). HBase verwendet für den Row Key ein Byte-Array, wodurch dem Entwickler bei der Definition eines Schlüssels viel Spielraum geboten wird (vgl. [RW12, S. 66]).

Spalten können zur Laufzeit hinzugefügt werden. Leere Zeilen existieren in HBase nicht. Das Anlegen einer Zeile findet nur dann statt, wenn sie einen Wert besitzt. Zusätzlich werden die Zeilen versioniert. Aus diesem Grund fügt HBase bei Einfügeoperationen automatisch einen Zeitstempel (engl. *Timestamp*) hinzu.

Ein wichtiges Unterscheidungsmerkmal von HBase im Vergleich zu relationalen Datenbanken ist das Datenschema. Das Datenschema in HBase wird durch eine Tabelle, eine Spaltenfamilie (engl. *Column Family*) und deren Eigenschaft festgelegt. Die Column Family ist ein von Googles BigTable übernommenes Konzept (vgl. [CDG+06]; [RW12, S. 66]). Die Daten einer Column Family werden physisch zusammenhängend gespeichert. Dies führt bei Abfragen zu kürzeren Ausführungszeiten. Aus diesem Grund ermöglicht HBase zufällige Lese- und Schreiboperationen in Echtzeit für große Datenmengen (vgl. [RW12]).

Der Zugriff auf eine Spalte erfolgt über den Verbund zwischen der Column Family und dem Bezeichner der Spalte in Form von $[ColumnFamily] : [Spaltenbezeichner]$. Eine Column Family muss vorab als Teil des Schemata einer HBase-Tabelle definiert sein. Spalten können zu jedem Zeitpunkt zu einer Column Family hinzugefügt werden, solange diese existiert.

[18] s. Webseite von Apache HBase unter `http://hbase.apache.org/`.

[19] NoSQL (Not only SQL) bezeichnet Datenbanken, die einen nicht-relationalen Ansatz verfolgen und in der Regel nicht mit der Abfragesprache SQL abgefragt werden können.

Das zugrundeliegende Datenmodell von HBase ist ein assoziatives Array[20]. Die Zeilen können wiederum auch als ein assoziatives Array gesehen werden (Row Key → Column Family). Die Werte der Column Family werden ebenfalls als assoziatives Array betrachtet. Aus diesem Grund wird das Datenmodell von HBase als ein mehrdimensionales assoziatives Array bezeichnet.

Ähnlich wie Apache Hadoop baut Apache HBase auf einer Master-Slave-Architektur auf. Der zentrale Master-Knoten, der sogenannte *MasterServer*, überwacht die Slave-Knoten, die in HBase als *RegionServer* bezeichnet werden. Der MasterServer übernimmt die Verteilung der Daten auf die verschiedenen RegionServer. Diese wiederum stellen den Datenzugriff sicher und übernehmen die Speicherung der Daten ins HDFS. Abbildung 2.14 zeigt ein Beispiel eines Clusters mit mehreren RegionServer.

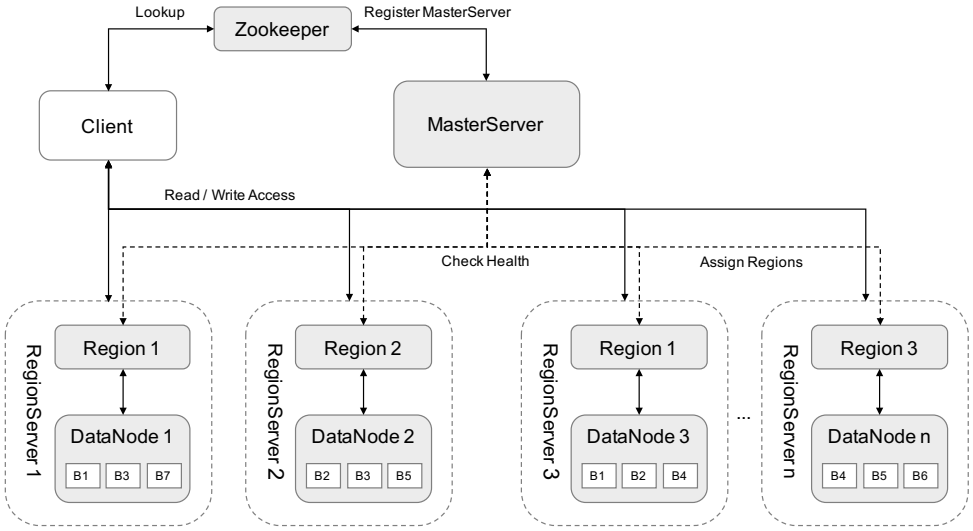

Abbildung 2.14.: Architektur von HBase, angelehnt an [RW12, S. 79].

Zu Beginn werden die Daten einer HBase-Tabelle in einer einzelnen *Region* gespeichert. Überschreitet die Datenmenge einen definierbaren Schwellwert, wird die Region durch den MasterServer automatisch in zwei neue Regionen mit gleicher Größe geteilt und auf die verfügbaren RegionServer übertragen. Aufgrund der Sortierung der Datensätze nach dem Row Key kann zu jedem Zeitpunkt der RegionServer ermittelt werden, der die benötigten Daten lokal gespeichert hat.

[20] Ein assoziatives Array wird auch als *Map* oder *Dictionary* bezeichnet.

Im Gegensatz zum Masterknoten in HDFS und MapReduce übernimmt Apache Zookeeper[21] die Funktionen und Aufgaben wie die Synchronisation, die Konfiguration und die Ausfallsicherheit von HBase (vgl. [HKJR10]). Zookeeper ist eine Koordinierungsstelle für verteilte Systeme und vereinfacht die Umsetzung und Überwachung von verteilten Anwendungen.

HBase bietet einen wahlfreien Zugriff auf extrem große Datenmengen und eignet sich besonders gut als horizontal skalierende Datenbank zur Datenhaltung mehrerer Milliarden Datensätze (vgl. [RW12, S. 80]). Aus diesem Grund wird HBase im Rahmen dieser Abschlussarbeit von besonderer Bedeutung sein.

[21] s. Apache Zookeeper Webseite unter `https://zookeeper.apache.org/`.

3. Konzeption

In den letzten Jahren ist das Interesse gestiegen, statistische Daten nach dem Linked-Data-Prinzip zu veröffentlichen[1]. Dieses Konzept bietet die Möglichkeit, Daten mit zusätzlichen Informationen aus unterschiedlichen Quellen im Web zu verknüpfen.

Für die Analayse statistscher Datensätze werden häufig Konzepte aus der Business Intelligence eingesetzt. Dabei werden die zu analysierenden Daten aus unterschiedlichen, heterogenen operativen Systemen mithilfe eines ETL-Prozesses bereinigt, Konflikte aufgelöst und in ein Data Warehouse gespeichert. Zur Unterstützung der betrieblichen Entscheidungsfindung unterstützt das Konzept OLAP die Analyse der konsolidierten Daten mit verschiedenen Operationen, die eine interaktive Navigation durch den Datenraum ermöglichen.

Die Analyse von Statistical Linked Data mit OLAP scheint ein vielversprechender Ansatz zur Unterstützung der Entscheidungsfindung zu sein. In einer früheren Publikation „Transforming Statistical Linked Data for Use in OLAP Systems“ [KH11] von Kämpgen und Harth wurde ein ETL-Prozess vorgestellt, der Statistical Linked Data im QB-Vokabular aus einer RDF-Datenbank in ein multidimensionales Datenmodell transformiert. Aufgrund der Metainformationen im QB-Vokabular war es möglich, die RDF-Daten automatisiert in eine relationale Datenbank im Sternschema zu speichern und für OLAP-Abfragen bereitzustellen.

Bei diesem Ansatz stellt sich jedoch die Frage, wie eine enorm große Menge an Statistical Linked Data effizient analysiert werden kann? Sowohl die relationale als auch die RDF-Datenbank skalieren in diesem ETL-Prozess nicht horizontal und besitzen daher eine natürliche Grenze bzgl. ihrer Datenspeicher- und Datenverarbeitungskapazität. Dies resultiert in einer sehr langen Ausführungsdauer des ETL-Prozesses zur Befüllung der relationalen Datenbank im Sternschema. Zusätzlich ist die Ausführung interaktiver Analysen von großen Datenmengen bei relationalen Datenbanken nicht effizient genug, da bereits ein

[1] s. Linked Data Webseite unter `http://linkeddata.org/`.

einfaches Scannen der Daten zu einer hohen zeitlichen Latenz führt.

Für Analysen großer Datenmengen sind daher Technologien aus dem Big-Data-Umfeld notwendig, die die Beschränkungen klassischer Systeme mittels Parallelisierung über viele Rechner hinweg überwinden. Mit Apache Hadoop sind derartige Technologien in einem Open Source Software Stack verfügbar.

In diesem Kapitel werden die verwendeten Technologien und deren Zusammenspiel, sowie die Idee, das Konzept und die geplante Architektur genauer betrachtet. Ziel ist es, einen Lösungsansatz zu präsentieren, der das von Kämpgen und Harth [KH11] vorgestellte Konzept in eine horizontal skalierende Architektur auf der Basis von Apache Hadoop überführt.

3.1. Verwendete Technologien

Bevor auf die einzelnen Komponenten und deren Zusammenspiel in der Architektur eingegangen wird, sollen die in der Konzeption verwendeten Technologien genauer beschrieben werden. Ähnlich zur Herangehensweise bei der Definition des Begriffs „Business Intelligence“ in Kapitel 2.1 wird im ersten Abschnitt 3.1.1 ein Projekt aus der Datenbereitstellungsschicht in Form von Apache Kylin vorgestellt. In Abschnitt 3.1.2 wird das Open-Source-Projekt Mondrian aus der Analyseschicht einer näheren Betrachtung unterzogen.

3.1.1. Apache Kylin

Vor der Veröffentlichung von *Apache Kylin*[2] im Jahr 2014 war es im Open-Source-Bereich nicht ohne weiteres möglich, auf Basis von Apache Hadoop das OLAP-Konzept für interaktive Abfragen auf Grundlage einer beliebig großen Datenmenge effizient umzusetzen. Es wurden zwar einige Arbeiten zum Thema *OLAP-on-Hadoop* veröffentlicht, die sich weder in der Praxis noch in der Wissenschaft etablieren konnten (vgl. [CEMK+15]; [WDS13]; [ZW13]; [AFR11]; [AKB13]).

Die von eBay im Oktober 2014 veröffentlichte OLAP-Engine Apache Kylin zeigt einen interessanten Ansatz, um die in der klassischen Business Intelligence seit vielen Jahren etablierten OLAP Cubes mit Unterstützung der Hadoop-Plattform in die Big-Data-Welt zu übertragen. Der Einsatz von Kylin wird durch etablierte Open-Source-Projekte aus dem

[2] s. Apache Kylin Webseite unter `http://kylin.incubator.apache.org/`.

Hadoop-Ökosystem begünstigt, die seit vielen Jahren von einer Vielzahl von Unternehmen produktiv eingesetzt werden.

Die folgende Abbildung 3.1 stellt das Zusammenspiel der Hadoop-Komponenten in Kylins Architektur dar. Dabei ist zwischen einem *Offline*- und *Online*-Datenfluss zu unterscheiden.

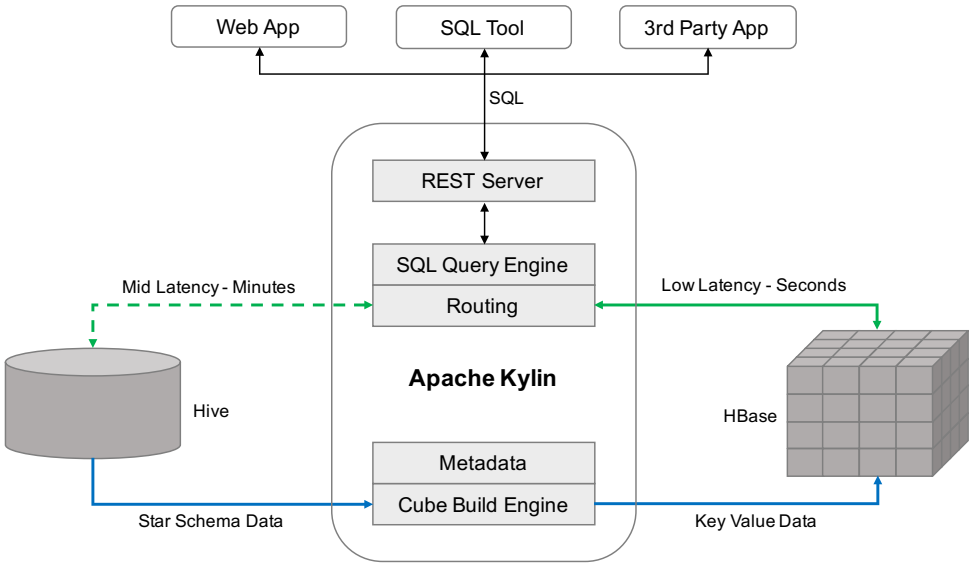

Abbildung 3.1.: Kylin-Architektur mit Komponenten aus dem Hadoop-Ökosystem, in Anlehnung an `http://kylin.apache.org/`.

Offline-Datenfluss

Die Generierung des OLAP Cubes setzt die Speicherung der Daten im HDFS und die Modellierung dieser Daten in Apache Hive als Sternschema voraus. Zusätzlich werden Metainformationen benötigt, welche die Cube-Struktur und die später möglichen OLAP-Abfragen beschreiben. Im Offline-Datenfluss (blauer Pfad in Abbildung 3.1) greift die *Cube Build Engine* auf diese Metainformationen zu und generiert mit mehreren, hintereinander ausgeführten HiveQL-Abfragen die Cuboids. Durch die Übersetzung der HiveQL-Abfragen in MapReduce-Jobs findet eine parallele Verarbeitung statt (s. Abschnitt 2.3.4).

Für die Speicherung der Cuboids wird die NoSQL-Datenbank HBase verwendet. Im nicht-relationalen, spaltenorientierten und verteilten Wide Column Store werden die vorberechneten Aggregationen der verschiedenen Cuboids gespeichert und über das Cluster hinweg horizontal verteilt (s. vorangehender Abschnitt HBase 2.3.5).

Kylin bietet die Möglichkeit, den Offline-Pfad für neu hinzukommende Daten inkrementell auszuführen. Dies kann in beliebigen Zeitabschnitten erfolgen, wie z. B. jede Stunde, einmal am Tag oder einmal im Monat. Mit dieser Eigenschaft wird im Rahmen der Abschlussarbeit das in der Einleitung vorgestellte Problem (V3) untersucht.

Online-Datenfluss

Nach der erfolgreichen Generierung des OLAP Cubes stehen die Daten für die Analysen zur Verfügung. Der Online-Datenfluss (grüner Pfad in Abbildung 3.1) beschreibt die Interaktion mit Kylin. Dabei werden SQL-Anfragen entweder direkt über die REST-Schnittstelle oder mithilfe der mitgelieferten Treiber und einem SQL-basierten BI-Tool an den REST[3]-Server gesendet.

In der SQL-Query-Engine schreibt Apache Calcite[4] die SQL-Abfrage in HBase Requests um. Wurden die angefragten Daten im HBase Cube vorberechnet, können die SQL-Abfragen, durch die horizontale Speicherung der Cuboids mit einer Ausführungszeitn im (Sub-)Sekundenbereich beantwortet werden. Das entspricht dem *Low-Latency-Pfad* in Abbildung 3.1 (grüner, durchgezogener Pfad).

Sind die angeforderten Daten der SQL-Abfrage nicht vorberechnet, kann das SQL-Statement an Apache Hive weitergeleitet werden. Hierbei wandelt Hive die Query in MapReduce-Jobs um, deren Abarbeitung mit einem entsprechenden Overhead und Latenzen verbunden ist (s. MapReduce Abschnitt 2.3.3). Das entspricht der Ausführung der SQL-Anfrage im *Mid-Latency-Pfad* (grüner, gestrichelter Pfad). Je nach Datenmenge eignet sich der Vorgang nur eingeschränkt für interaktive Analysen.

Das wesentliche Ziel von Kylin besteht darin, für möglichst viele Abfragen des Nutzers den Low-Latency-Pfad bereitzustellen. Aus diesem Grund sind während der Definition des OLAP Cubes die Anforderungen der Analysten für die betriebliche Entscheidungsfindung bestmöglich zu berücksichtigen.

Definition des OLAP-Datenmodells in Kylin

Kylin setzt für die Generierung des OLAP Cubes die Abbildung der Daten in Apache Hive im Sternschema voraus. Bevor mit der eigentlichen Daten-Modellierung des OLAP Cubes

[3] REST steht für *Representational State Transfer* und beschreibt ein zustandsloses Client-Server-Protokoll über HTTP.

[4] s. Apache Calcite Webseite `http://calcite.apache.org/`

begonnen werden kann, sind die Tabellen aus dem Hive Metastore mit Kylin zu synchronisieren. Dabei werden im Hintergrund Metainformationen zu den Hive-Tabellen erstellt und in eine eigens dafür vorgesehene HBase-Tabelle gespeichert. Dadurch stehen der Modellierung des OLAP Cubes alle notwendigen Informationen zur Verfügung, wie beispielsweise die Spaltennamen der Dimensionstabellen oder die Datentypen der Measures. Erst nach dieser Synchronisierung kann die Daten-Modellierung des Cubes Schritt für Schritt interaktiv in der Weboberfläche oder mit der dafür vorgesehenen REST-Schnittstelle mit einem JSON Request aufgebaut werden.

Der erste Schritt besteht in der Definition des OLAP-Datenmodells. Nach Auswahl der Faktentabelle sind die Dimensionstabellen zu definieren und um Angaben der *Primary-* und *Foreign Keys* zu ergänzen.

Die Beschreibung der Dimensionen findet anhand von drei unterschiedlichen Typen statt: *Normal*, *Hierarchy* und *Derived*. Bei der Auswahl „Normal“ wird die Dimension ohne jede Besonderheit zum Cube hinzugefügt. Beim Typ „Hierarchy“ bietet Kylin eine hierarchische Anordnung der Attribute einer Dimension an. Wie bereits in Abschnitt 2.1.4 beschrieben, spielt diese Art der Anordnung der Daten eine wichtige Rolle für die Drill-down- und Roll-up-Navigation durch den Cube. Die letzte Möglichkeit, eine Dimension hinzuzufügen, besteht im Typ „Derived“. Hierbei handelt es sich um Attribute einer Dimension, die keinem hierarchischen Aufbau entsprechen und sich eindeutig durch einen Primary Key ableiten lassen. Die Berechnung der Measures basiert folglich lediglich auf diese einzelnen Keys. Das führt zu einer Reduktion der Kombinationsmöglichkeiten der Dimensionen, da nicht jede mögliche Zusammenstellung der Spaltenwerte berücksichtig werden.

Zusätzlich zu den Dimensionen müssen die Measures des OLAP Cubes definiert werden. Die Auswahl der Aggregationsfunktionen ist zum Zeitpunkt der Abschlussarbeit auf *SUM*, *MIN*, *MAX*, *COUNT* und *COUNT_DISTINCT* beschränkt.

Optional können sogenannte *Refresh Settings* definiert werden. Dies ermöglicht OLAP Cubes mit neu hinzukommenden Daten zu generieren und mit bestehenden Cubes zu vereinen. Hierbei ist eine Date-Spalte im Format „YYYY-MM-DD“ auszuwählen und ein Startdatum anzugeben.

Kylin bietet zudem die Möglichkeit, erweiterte Einstellungen zu definieren, die eine Optimierung des Cubes ermöglichen. Bei einer großen Anzahl an Dimensionen ist es nicht sinnvoll, jedes Cuboid zu berechnen. Beispielsweise wären bei 30 Dimensionen 2^{30} (etwas

mehr als eine Milliarde) Cuboids zu erstellen. Mit *Aggregation Groups* kann eine Unterteilung dieser 30 Dimensionen in Gruppen durchgeführt und so ein Partial Cube definiert werden (s. Abschnitt 2.1.4). Statt 2^{30} Cuboids zu generieren, kann der Cube beispielsweise in drei Gruppen à 10 Dimensionen aufgeteilt werden. Die Anzahl der Cuboids reduziert sich dadurch auf $2^{10} + 2^{10} + 2^{10}$ (= 3072 Cuboids). Sowohl die Berechnungszeit der Aggregationen als auch der Speicherverbrauch des OLAP Cubes werden hierdurch deutlich reduziert. Bei ungünstiger Wahl der Aggregation Groups kann jedoch ein fehlendes Cuboid die Ausführungszeit der Analyse stark beeinträchtigen (s. Mid-Latency-Pfad in Abbildung 3.1).

Der Cube Build Process

Nach erfolgreicher Modellierung des OLAP Cubes ist die Ausführung des *Cube-Build*-Prozesses möglich. Anhand der Metainformationen und der Datenmodellierung werden die Aggregate in den verschiedenen Kombinationsmöglichkeiten der Dimensionen berechnet. Abbildung 3.2 stellt den Workflow dar.

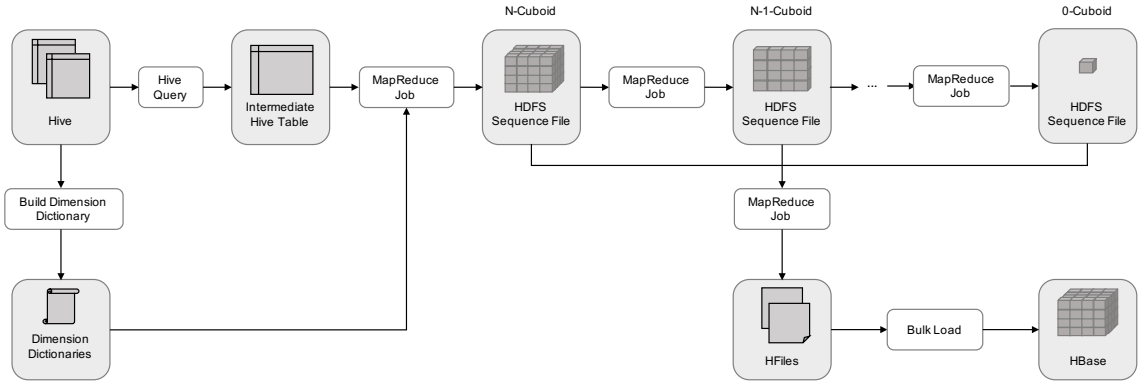

Abbildung 3.2.: Kylins Cube-Build-Prozess als Workflow.

Im ersten Teil des Prozesses wird ein *Dictionary* angelegt. Das Dictionary wird bei Dimensionen verwendet, die keine große Kardinalität besitzen. In solch einem Fall speichert Kylin bei der Generierung des OLAP Cubes nicht den eigentlichen Wert in die HBase-Datenbank, sondern eine Referenz zum Dictionary. Diese Einstellung kann pro Spalte der Dimensionstabellen individuell angegeben werden.

Im nächsten Abschnitt des Prozesses werden alle verwendeten Hive-Tabellen des OLAP Cubes zunächst in einer *Intermediate Hive Table* zusammengefasst. Dabei wird in einer ein-

zelnen HiveQL-Abfrage die Faktentabelle mit allen Dimensionstabellen in der Intermediate Hive-Tabelle vereint und gespeichert. Diese Tabelle dient als Grundlage für die Generierung des größtmöglichen N-Cuboids (vgl. Abbildung 2.6 in Abschnitt 2.1.4). Durch verschiedene Gruppierungen werden mithilfe mehrerer hintereinander ausgeführter MapReduce-Jobs die immer kleiner werdenden Cuboids berechnet, bis der 0-Cuboid ermittelt wird. Ein ausreichend großer Speicherplatz im HDFS ist zwingend erforderlich, da alle Zwischenergebnisse als *HDFS Sequence Files* abgelegt werden.

Sind die Cuboids berechnet, werden die Zwischenergebnisse in einem letzten MapReduce-Job in eine *HFile* transformiert. HFiles sind Dateien im HDFS, die HBase für die Datenspeicherung verwendet. Der letzte Schritt besteht im Importieren dieser HFile in HBase in Form eines *Bulk-Load*-Prozesses.

Je nach Größe der Datenmenge und der Anzahl der Knoten des Clusters kann der Build-Prozess einige Zeit beanspruchen. Anschließend sind die Vorbereitungen für die Verarbeitung analytischer OLAP Queries abgeschlossen.

Vor- und Nachteile von Kylin

Die Vorteile von Apache Kylin sind vielfältig. Es handelt sich um den ersten erfolgreichen Versuch, OLAP-Funktionalitäten auf Apache Hadoop aufzusetzen. Zudem basiert das Open-Source-Projekt aufgrund der Verwendung von HDFS, Hive, MapReduce, HBase und Calcite auf Systemen aus dem Hadoop-Ökosystem, die sich in den letzten Jahren etabliert haben. Der produktive Einsatz von Kylin bei eBay zeigt das Potenzial des Projektes. Die Möglichkeit der horizontalen Skalierung im Cube-Build-Prozess sowie die horizontale Verteilung der Cuboids durch HBase führt dazu, dass Kylin enorm große Datensätze verarbeiten und die vorberechneten Daten für Analysen effizient bereitstellen kann.

Ein wesentlicher Nachteil besteht jedoch in der eingeschränkten Abfragemöglichkeit: Kylin kann ausschließlich SQL-Abfragen interpretieren. Aus diesem Grund wird im nächsten Abschnitt Pentahos MDX-to-SQL Engine Mondrian vorgestellt.

3.1.2. Pentaho Mondrian

MDX ist eine Abfragesprache, welche häufig bei komplexen OLAP-Operationen gewählt wird (s. Abschnitt 2.1.5). Aus diesem Grund setzen viele OLAP Client MDX als Standard-

sprache ein. Neben dem Vorteil der erweiterten Selektierbarkeit besteht eine weitere Stärke von MDX darin, mit Roll-Up- und Drill-Down-Operationen durch Hierarchien entlang eines Pfades zu navigieren. Jedoch ist die Anzahl der Datenbanken, die MDX interpretieren und verarbeiten können, gering. Infolgedessen wurde bereits 2001 das Projekt Mondrian[5] begonnen - ein quelloffener, in Java entwickelter OLAP Server (vgl. [BGH13, S. 3]). Auf der einen Seite wollen Analysten MDX-Abfragen für ihre Analysen nutzen, auf der anderen Seite jedoch nicht auf die einfache Anwendung und Nutzung von relationalen Datenbanken verzichten. Um auf die Daten einer relationalen Datenbank mit MDX zuzugreifen, muss eine Umwandlung der MDX-Abfragen in SQL stattfinden. Diese Transformation war eines der wichtigen Ziele bei der Entwicklung von Mondrian.

Ein weiteres Ziel bei der Entwicklung von Mondrian bestand darin, interaktive Analysen von größere Datenmengen zu ermöglichen. Die relationalen Daten müssen daher im Sternschema angeordnet sein. Zusätzlich wurde in Mondrian ein Cache implementiert, der aus bereits ausgeführten SQL-Abfragen einen multidimensionalen OLAP Cube erstellt, um nachfolgende MDX-Abfragen direkt über den Cube im Cache beantworten oder das Ergebnis ableiten zu können. Abbildung 3.3 zeigt ein typisches Ablaufdiagramm in Mondrian.

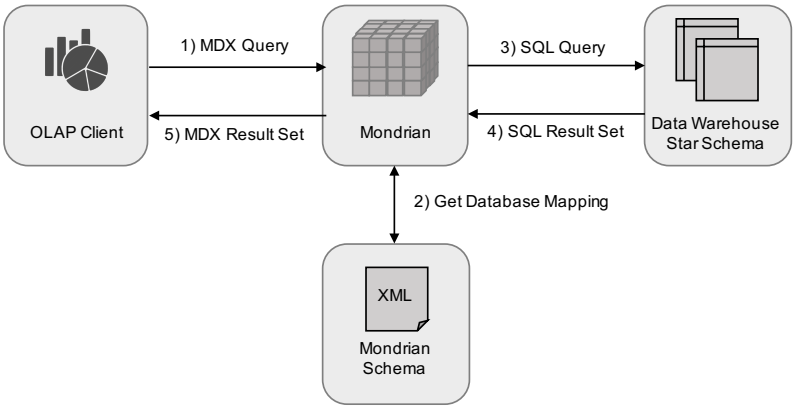

Abbildung 3.3.: Ablaufdiagramm von Mondrian bei der Ausführung einer MDX-Abfrage, in Anlehnung an [BGH13, S. 12].

Die MDX-Abfragen können entweder über einen OLAP Client wie Saiku[6] oder direkt über einen API Call, z. B. mit der Java-Bibliothek OLAP4J[7], an den Mondrian-Server gesendet werden. Nach Validierung der MDX-Abfrage wird überprüft, ob das Ergebnis mit Hilfe von

[5] s. Pentaho Mondrian Webseite unter `http://community.pentaho.com/projects/mondrian/`.
[6] s. Saiku Webseite `http://www.meteorite.bi/products/saiku`.
[7] s. OLAP4J Webseite `http://www.olap4j.org/`.

zuvor ausgeführten MDX-Abfragen und durch Speicherung der Ergebnisse als OLAP Cube im Cache beantwortet werden kann. Falls dies möglich ist, wird das Ergebnis direkt aus dem Cache abgeleitet und als MDX Result Set zurück an den OLAP Client gesendet. Kann die Abfrage nicht aus den Daten im Cache abgeleitet werden, werden unter Zuhilfenahme des Mondrian Schema eine Folge von SQL-Anfragen generiert und an das Data Warehouse gesendet. Die Einzelresultate werden zu einem Ergebnisse aggregiert, zusammengefasst und im Cache in einer multidimensionalen Struktur als Cube gespeichert, sodass dieses Ergebnis bei späteren Abfragen wiederverwendet werden kann. Der letzte Schritt besteht im Zurücksenden des von Mondrian ermittelten Resultats als MDX Result Set an den OLAP Client.

Mondrian operiert in der Regel mit klassischen relationalen Datenbanken wie MySQL oder PostgreSQL. Die Beziehungen zwischen den relationalen und den multidimensionalen Strukturen werden im sogenannten *Mondrian Schema* definiert. Diese in XML beschriebene Konfigurationsdatei gibt die Tabellen und Spalten des relationalen Datenbankschemas an und definiert die Faktentabelle als auch die Dimensionstabellen des multidimensionalen OLAP Cubes. Zusätzlich werden alle Measures des Cubes, die verwendeten Aggregationsfunktionen, Hierarchien in den Dimensionen mit ihren Levels und Attributen im Mondrian Schema deklariert.

Ferner kann Mondrian durch *SQL Dialects* erweitert werden, um die Kommunikation mit anderen Datenbanken zu ermöglichen, die SQL-Abfragen oder zumindest eine Teilmenge davon interpretieren können. Ein weiteres Ziel der vorliegenden Arbeit wird daher in der Implementierung eines Kylin Dialects bestehen, die die Beantwortung von MDX-Abfragen in Apache Kylin realisieren soll.

3.2. Idee und Aufbau der Architektur

Der eingangs in Abschnitt 1.2 beschriebene ETL-Prozess von Kämpgen und Harth [KH11] transformiert die Statistical Linked Data einer RDF-Datenbank durch mehrere, hintereinander ausgeführten SPARQL-Abfragen in ein multidimensionales Datenmodell. Aufgrund der Metainformationen im QB-Vokabular ist es möglich, die RDF-Daten automatisiert in eine relationale Datenbank im Sternschema zu speichern und für OLAP-Abfragen bereitzustellen.

Wie bereits erläutert, sind Technologien aus dem Big-Data-Umfeld für die Analyse einer

enorm großen Datenmenge notwendig. Der hier präsentierte Lösungsansatz überführt daher Kämpgen und Harths ETL-Prozess in eine horizontal skalierende Architektur auf der Basis von Apache Hadoop. Die nicht-skalierbaren Komponenten, wie die RDF-Datenbank, die Abfragesprache SPARQL und die relationale Datenbank werden dabei durch Technologien und Frameworks aus dem Hadoop-Ökosystem ersetzt. Abbildung 3.4 veranschaulicht die neue Gesamtarchitektur. Die Hauptziele bestehen darin, die Ausführungsdauer zur Beladung des Data Warehouses sowie die Antwortzeiten analytischer OLAP-Abfragen auch bei einem beliebig großen Datenvolumen zu reduzieren.

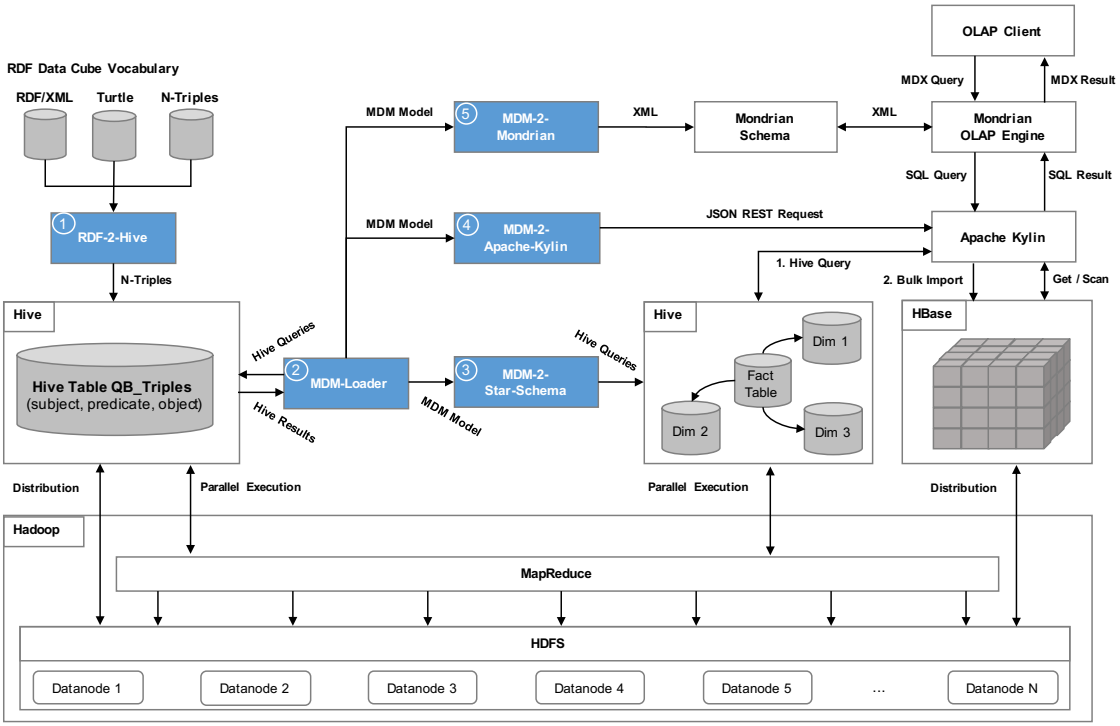

Abbildung 3.4.: Parallelisierungsarchitektur des ETL-Prozesses mit MapReduce zur Bewirtschaftung der RDF-Daten im QB-Vokabular in Apache Kylin.

Für die Umsetzung der Architektur ist neben einem funktionierenden Apache Hadoop Cluster auch eine fehlerfreie Installation von Apache Kylin auf einem Cluster-Knoten notwendig. Im Rahmen dieser Arbeit wird Kylin auf dem Master-Knoten installiert.

Die Umsetzung des ETL-Prozesses besteht aus fünf Komponenten, wie in Abbildung 3.4 dargestellt wird. Die Konzeptionen der einzelnen Komponenten sowie deren Aufgaben werden in den nächsten Abschnitten beschrieben.

3.2.1. Komponente 1: Umzug der RDF-Daten nach Hive

Wie bereits im Abschnitt 2.2.6 beschrieben, beinhalten RDF-Daten, die durch das QB-Vokabular beschrieben werden, neben den eigentlichen statistischen Daten zusätzliche Metainformationen, die die Struktur des OLAP Cubes beschreiben. Die statistischen Linked Data müssen für die später stattfindende parallele Ausführung des ETL-Prozesses in das Hadoop-Ökosystem umgezogen werden.

Vor dem Umzug der RDF-Daten ins HDFS muss eine Transformation stattfinden. Die durch verschiedene Syntaxen beschriebenen RDF-Daten werden in das zeilenbasierte N-Triples-Format umgewandelt. Dies hat folgende Gründe:

- Das einfache N-Triples-Format enthält pro Zeile genau ein Triple der Form Subjekt, Prädikat und Objekt. Durch diese einfache Struktur ist es möglich, die Hive-Tabelle QB_Triples mit drei Spalten (*subject*, *predicate*, *object*) zu generieren.
- Aufgrund der Speicherung im HDFS werden die Daten in Datenblöcke mit einer bestimmten Größe aufgeteilt, horizontal über die verschiedenen Datanodes des Clusters verteilt und repliziert (s. Abschnitt 2.3.2). Ein nicht-zeilenbasiertes Format könnte dazu führen, dass ein Triple über mehrere Zeilen in zwei unterschiedliche Datenblöcke getrennt wird. Dies würde zu fehlenden Triples und einem fehlerhaften Verhalten beim Auslesen der Triples führen.

Folglich ist es notwendig, die RDF-Daten vor dem Umzug in das HDFS in das zeilenbasiertes N-Triples-Format umzuwandeln. Nach dieser Transformation und dem Umzug ins HDFS kann die Hive-Tabelle *QB_ Triples* durch eine *Create-HiveQ*L-Abfrage erstellt werden. Ziel der RDF-2-Hive-Komponenten ist es, der nächsten Komponente *MDM-Loader* die Möglichkeit zu bieten, die Struktur des OLAP Cubes aus den RDF-Daten im QB-Vokabular mit mehreren, hintereinander ausgeführten HiveQL-Abfragen aus der Hive-Tabelle QB_Triples auszulesen.

3.2.2. Komponente 2: Auslesen des multidimensionalen Models

Nach Generierung der Hive-Tabelle QB_Triples werden mithilfe von verschiedenen, hintereinander ausgeführten HiveQL-Abfragen die Metainformationen aus dem RDF im QB-Vokabular ausgelesen. Diese Aufgabe übernimmt die zweite Komponente *MDM-Loader*. Die Abfragen haben das Ziel, alle Cube-Informationen mit effizienten HiveQL-Abfragen auszu-

lesen. Die Definition geeigneter HiveQL-Abfragen und der daraus entstehenden MapReduce-Jobs führt zur parallelen Verarbeitung der Abfragen über die Cluster-Knoten. Dabei werden die Measures, Dimensionen, Hierarchien, Levels, Attribute und Fact Members des OLAP Cubes aus einer beliebig großen Menge an RDF-Daten ermittelt.

Grundlegendes Ziel ist eine geeignete Repräsentation des Datenmodells. Zudem werden in dieser Komponente bereits Metainformationen definiert und Vorbereitungen für die darauffolgenden Schritte getätigt. Wie bereits im Abschnitt 3.1.1 dargestellt, müssen die Daten für Kylin in Hive im Sternschema vorliegen. Folglich werden bereits anhand der ausgelesenen Informationen aus dem MDM die Tabellen- und Spaltennamen der Faktentabelle und der Dimensionstabellen generisch bestimmt. Zudem beinhaltet das MDM alle erforderlichen Metainformationen wie die Datentypen sowie Primary-Key- und Foreign-Key-Zuweisungen des Sternschemas.

Ausgehend vom MDM und den ausgelesenen Metainformationen aus den RDF-Daten im QB-Vokabular werden die restlichen drei Komponenten ausgeführt.

3.2.3. Komponente 3: Generierung des Sternschemas in Hive

Alle relevanten Daten müssen vor der Generierung des OLAP Cubes in Hive-Tabellen im Sternschema modelliert werden. Demnach ist eine Transformation der Hive-Tabelle QB_Triples in Hive-Tabellen des Sternschemas notwendig. Hive bietet die Möglichkeit, eine neue Tabelle aus bereits bestehenden Tabellen durch einen HiveQL-Statement zu generieren. Solche Abfragen werden im Hive-Kontext als *CTAS*-Statements (Create Table As Select) bezeichnet. Für die Generierung des Sternschemas werden durch die zuvor ausgelesenen Informationen aus der QB_Triples-Tabelle die Fakten- und alle Dimensionstabellen erstellt. Die Schwierigkeit liegt darin, das zeilenbasierten Format der QB_Triples-Tabelle (jede Zeile repräsentiert ein Triple, eine konkrete Instanz einer Dimension kann mehrere Attribute und dadurch mehrere Zeilen lang sein) in ein spaltenorientiertes Format umzuwandeln (jede Zeile repräsentiert ein Objekt, die Spalten stellen die Attribute dar).

3.2.4. Komponente 4: Metadata Modell und Cube Build in Kylin

Der REST-Server von Kylin bietet, neben der Möglichkeit SQL-Abfragen auszuführen, noch weitere Funktionen an. Die dritte Komponente *MDM-2-Kylin* nutzt diese Möglichkeit,

um folgende REST Requests auszuführen:

Optional: Neues Projekt erstellen

Bevor der OLAP Cube generiert werden kann, ist optional ein neues Projekt in Kylin anzulegen. Neben dem Namen des Projekts ist auch die Angabe einer Beschreibung möglich.

Synchronisation

Die Synchronisation der Hive-Tabellen in Kylin erfolgt durch einen einfachen REST Request. Ähnlich zum Hive-Metastore werden hierbei Informationen über die synchronisierten Tabellen generiert und in einer dafür speziell vorgesehene HBase-Tabelle gespeichert.

OLAP-Datenmodell definieren

Nach der Synchronisierung werden die Metainformationen des zu generierenden OLAP-Datenmodells mit einem weiteren REST Request an Kylin übermittelt. Aus diesem Grund ist eine Transformation des zuvor ausgelesenen MDMs in ein JSON-Format notwendig. Kylin benötigt folgende Metainformationen für den Cube Designer:

- Deklaration der Faktentabelle in Hive.
- Festlegung der Hive-Tabellen, die die Dimensionstabellen des OLAP Cubes repräsentieren. Zusätzliche Informationen beinhalten den Join-Typ (*Inner*-, *Left*-, *Right*-Join) und die Join-Bedingung (*Foreign-Key*- und *Primary-Key*-Zuweisung).
- Informationen über den Aufbau der einzelnen Dimensionen: Hierarchien, Levels, und Attribute. Zudem sind Dimensionen, die durch eine Spalte in der Faktentabelle definiert werden, ebenfalls anzugeben.
- Deklaration der Measures des OLAP Cubes durch Angabe des Datentyps, der Aggregationsfunktion und der zugehörigen Spalte der Faktentabelle.
- Optionale Definition von Refresh-Settings und Aggregation Groups (s. Erläuterung in Abschnitt 3.1.1).

Cube-Build-Prozess anstoßen

Die dritte Aufgabe der MDM-2-Kylin-Komponente besteht darin, den Cube-Build-Prozess über den REST-Server zu starten. Aufgrund der Dauer des Prozesses soll der aktuelle Stand in einem definierten Intervall abgefragt und dem Benutzer mitgeteilt werden.

3.2.5. Komponente 5: Mondrian Schema definieren

Die letzte Komponente hat die Aufgabe, aus den Informationen des MDMs ein Mondrian Schema zu generieren. Diese XML-Konfigurationsdatei wird in Mondrian benötigt, um die MDX-Abfragen des OLAP Clients in ein für Kylin verständliches SQL umzuwandeln (s. Abschnitt 3.1.2). Hierbei findet eine Transformation des MDMs in das XML-Format statt. Nach diesem Schritt ist der ETL-Prozess beendet.

3.3. Ziele der Architektur

Erst nach erfolgreichem Abschluss der fünf Komponenten ist es möglich, die Daten entweder mit MDX-Abfragen über einen OLAP Client wie Saiku mit Mondrian, über die SQL REST-Schnittstelle in Kylin oder über ein SQL-basiertes BI-Tool wie Tableau[8] und dem JDBC-Treiber abzufragen.

Die Dauer des ETL-Prozesses richtet sich nach der Menge der zu verarbeitenden Daten und der Anzahl der Cluster-Knoten. Eine Hypothese dieser Abschlussarbeit liegt in der Überprüfung, ob die parallele Ausführung durch die hintereinander ausgeführten HiveQL-Abfragen und der Generierung von MapReduce-Jobs einen Vorteil gegenüber nicht horizontal skalierenden ETL-Prozessen mit sich bringt?

Ein weiteres Ziel liegt in der Reduzierung der Ausführungsdauer bei analytischen OLAP-Abfragen. In Abhängigkeit der Clustergröße soll durch die horizontale Speicherung der Cuboids in HBase die Abfragedauer auch bei einer beliebig großen Datenmenge im Sekundenbereich liegen.

Nach der Definition des Konzepts wird im nächsten Kapitel die Implementierung, der technische Aufwand, die dabei entstandenen Probleme sowie die verwendeten HiveQL-Abfragen einer genaueren Betrachtung unterzogen.

[8] s. Webseite von Tableau unter `http://www.tableau.com/de-de/business-intelligence`.

4. Implementierung

In diesem Kapitel wird auf die technische Implementierung sowie auf die Anforderungen des ETL-Prozesses detailliert eingegangen. Zunächst wird in Abschnitt 4.1 die entwickelte Kommunikationsmöglichkeit zwischen Apache Kylin und Pentaho Mondrian durch einen SQL-Dialekt für die Ausführung von MDX-Abfragen behandelt. Anschließend werden die im vorherigen Kapitel definierten Komponenten des ETL-Prozesses auf technischer Ebene anhand eines abstrakten Beispiels verdeutlicht. Die Anforderungen, die entstandenen Probleme mit ihren Lösungsansätzen sowie weitergehende Optimierungsmöglichkeiten werden in den folgenden Abschnitten beschrieben.

4.1. Kommunikation zwischen Apache Kylin und Mondrian

Mondrian übersetzt MDX-Abfragen in SQL-Statements für unterschiedliche relationale sowie nicht-relationale Datenbanken. Zum Zeitpunkt der Abschlussarbeit existiert jedoch kein SQL-Dialekt für die Generierung von SQL-Abfragen für Apache Kylin. Wie bereits im Abschnitt 3.1.1 beschrieben, kann Kylin durch die Integration von Apache Calcite eine ANSI-SQL-Teilmenge interpretieren. Daher werden viele SQL-Abfragen erfolgreich in HBase Requests umgesetzt, einige wenige wie z. B. implizit angegebenen Joins bleiben jedoch nicht ausführbar. Listing 4.1 zeigt ein einfaches Beispiel einer SQL-Abfrage, die von Kylin nicht ausgeführt werden kann.

```
1 -- Join-Anfrage, die von Kylin nicht unterstützt wird
2 SELECT *
3 FROM Fact, Dim1
4 WHERE Fact.dim1_id = Dim1.id;
```

Listing 4.1: Beispiel einer SQL-Anfrage mit impliziten Join.

Demnach müssen Joins durch das SQL-Schlüsselwort *JOIN* und der direkt darauffolgenden

ON-Bedingung deklariert werden (s. Listing 4.2).

```
1 -- Join-Anfrage, die von Kylin unterstützt wird
2 SELECT *
3 FROM Fact
4 JOIN Dim1 ON Fact.dim1_id = Dim1.id;
```

Listing 4.2: Beispiel einer SQL-Anfrage mit expliziten Join.

Zudem unterstützt Kylin keine *Distinct Counts*. Die Integration von Mondrian in Kylin setzt daher ein SQL-Dialekt voraus, welcher für Kylin verständliche SQL-Abfragen generiert.

4.1.1. Kylin SQL-Dialekt in Mondrian erstellen

Mondrian stellt Methoden bereit, um mit einem SQL-Dialekt das gewünschte SQL-Statement zu generieren. Das in Listing 4.1 vorgestellte Implicit-Join-Problem kann durch die Methode aus Listing 4.3 gelöst werden.

```
1 public boolean allowsJoinOn() {
2   return true;
3 }
```

Listing 4.3: Generierung von expliziten SQL-Joins durch die Methode *allowsJoinOn()*.

Der Einsatz des Kylin Dialekts erwies sich bei der Umsetzung in Mondrian 3 als fehlerhaft. Die Methode *allowsJoinOn()* wurde bei der Generierung der SQL-Abfragen nicht berücksichtigt. Dieser Fehler wurde in Mondrian 4 behoben. Folglich wurde für den weiteren Verlauf der Abschlussarbeit Mondrian 4 verwendet.

```
1 public boolean allowsCountDistinct() {
2   return false;
3 }
```

Listing 4.4: Deaktivierung der *DISTINCT COUNT*-Methode im Kylin-SQL-Dialekt durch die Methode *allowsCountDistinct()*.

Die nächste zu implementierenden Methode in Kylins Dialect ist *allowsCountDistinct()*,

die *false* als Rückgabewert hat (s. Listing 4.4), da Kylin zum Zeitpunkt der Arbeit solch eine Count-Abfrage nicht unterstützt.

4.2. Implementierung der ETL-Komponenten

Für die Entwicklung des ETL-Prozesses wurde die Programmiersprache Java gewählt. Mithilfe des JDBC-Treibers von Apache Hive wird der Zugriff auf die QB_Triples-Tabelle und die Generierung der Hive-Tabellen im Sternschema ermöglicht.

Zum besseren Verständnis wird dieser technische Abschnitt anhand eines abstrakten Beispiels beschrieben. Zur Veranschaulichung dient die Abbildung 4.1. Die zu analysierenden RDF-Daten im QB-Vokabular beschreiben in diesem Beispiel einen OLAP Cube mit zwei Measures: *measure1* (Gleitkommazahl vom Typ „double“ und der Aggregationsfunktion „sum“) und *measure2* (Ganzzahl vom Typ „int“ und der Aggregationsfunktion „max“). Zusätzlich stellen die RDF-Daten zwei Dimensionen *dim1* und *dim2* dar. Die erste Dimension *dim1* besitzt zwei Hierarchien: *hier1* mit drei Levels (*level1_1*, *level1_2* und *level1_3*) sowie *hier2* mit zwei Levels (*level2_1* und *level2_2*). Die zweite Dimension *dim2* verfügt über eine Hierarchie *hier3* mit zwei Levels *level3_1* und *level3_2*. Ein Fakt enthält neben den Measures noch Verlinkungen zu konkreten Instanzen der Dimensionen.

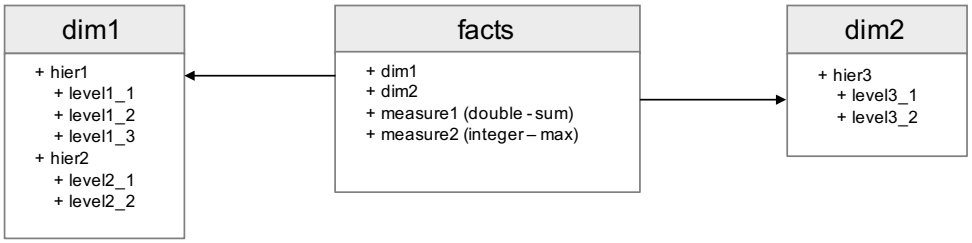

Abbildung 4.1.: Veranschaulichung des verwendeten Beispiels zur Beschreibung der Umsetzung der technischen Komponenten des ETL-Prozesses.

4.2.1. Komponente 1: Umzug der RDF-Daten nach Hive

Vor dem Umzug der RDF-Daten ins HDFS findet eine Transformation der RDF-Daten in das N-Triples-Format statt. Hierfür wird bei der Umsetzung auf Apache Jena zurückgegriffen. Die Jena-Komponente *RIOT* (RDF I/O Technology) transformiert eine Eingabedatei

mit beliebigen RDF-Format in das N-Triples-Format. Der dazugehörige Code-Abschnitt ist in Listing 4.5 zu finden.

```
./bin/riot --output=nt rdf_file_1.xml rdf_file_2.ttl rdf_file_3.nt > all_triples.nt
```

Listing 4.5: Umwandlung der RDF-Daten in das N-Triples-Format mit Apache Jena.

Der Umzug der RDF-Daten ins HDFS findet über den in Listing 4.6 dargestellten Befehl statt.

```
hadoop fs -copyFromLocal /path/on/disk /path/on/hdfs
```

Listing 4.6: Umzug der Daten ins HDFS in durch einen Befehl in der HDFS Shell.

Im Abschnitt 2.3.4 wurde beschrieben, dass die zugrundeliegenden I/O-Bibliotheken HiveQL-Abfragen auf unterschiedliche Datenstrukturen erlauben. Aus diesem Grund ist nach dem Umzug der RDF-Daten ins HDFS die Generierung einer sogenannten *External Hive Table* durch ein einzelnes HiveQL-Statement möglich. Eine External Hive Table ist eine Hive-Tabelle, die durch Angabe einer *Location* eine Tabelle mit Daten aus einem existierenden HDFS-Ordner erstellt. Die HiveQL-Abfrage erstellt die Tabelle *QB_ Triples* mit drei Spalten (*subject*, *predicate*, *object*) über die RDF-Daten im N-Triples-Format. Listing 4.7 zeigt die verwendete HiveQL-Abfrage für die Erstellung der External Hive Table.

```
1 CREATE EXTERNAL TABLE QB_Triples ( subject STRING, predicate
      STRING, object STRING )
2 ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.RegexSerDe'
3 WITH SERDEPROPERTIES ( "input.regex" = "([^ ]*) ([^ ]*) (.*) \\.", "output.
      format.string" = "%1$s %2$s %3$s" )
4 STORED AS TEXTFILE LOCATION '/path/on/hdfs';
```

Listing 4.7: Generierung einer externen Hive-Tabelle durch eine HiveQL-Abfrage.

Die Location gibt den absoluten HDFS-Pfad des Ordners an, in dem sich die RDF-Daten befinden. Die Anzahl der RDF-Dateien ist irrelevant für die Generierung der Hive-Tabelle.

Der reguläre Ausdruck gibt die zu interpretierende Datenstruktur für die Generierung der Hive-Tabelle an. Wie in Abschnitt 2.2.3 beschrieben, definiert N-Triples ein zeilenbasiertes Format der Form „<SubjectURI> <PredicateURI> <ObjektURI> .“. Mit dem regulären Ausdruck „([^]*) ([^]*) (.*) \\.“ werden die drei Spalten für die Hive-Tabelle QB_Triples

deklariert. Die *Stored-As*-Anweisung gibt den Datentyp der Daten im HDFS-Ordner an.

Von diesem Zeitpunkt an können die RDF-Daten der Hive-Tabelle QB_Triples mit HiveQL-Abfragen ausgelesen werden. Jedoch eignet sich die Speicherung der Daten im HDFS als reine Textdateien nur bedingt für die Ausführung von MapReduce-Jobs. Aus diesem Grund können bereits in dieser Komponente Optimierungen durchgeführt werden, die eine Reduzierung der Ausführungszeiten der HiveQL-Abfragen zur Folge haben.

Optimierungen bei der Generierung der Hive-Tabelle

Die Speicherung der RDF-Daten als reine Textdateien im HDFS und die Generierung einer External Hive Table führt nicht zu effizienten Ausführungszeiten der HiveQL-Abfragen. Durch die unkomprimierte Speicherung der RDF-Daten können die Dateien im N-Triples-Format sehr groß werden. Dies führt zu langen Ausführungszeiten der MapReduce-Jobs. Grund hierfür ist die Aufteilung der Daten in Blöcke bei der Speicherung im HDFS (s. Abschnitt 2.3.2). Standardgemäß wird eine Datei in 128MB großen Datenblöcke aufgeteilt und horizontal über das Cluster verteilt. Wenn beispielsweise die Größe der N-Triples-Datei 10GB beträgt, sind bereits knapp 80 MapReduce-Jobs für die Berechnung der einzelnen Datenblöcken mit jeweils 128MB notwendig. Zwar kann der Datenblock-Wert bei jedem Upload ins HDFS individuell festgelegt, jedoch im Nachhinein nicht ohne ein erneutes Hochladen der Dateien geändert werden.

Optimierung 1: Dateigröße durch komprimierendes Datenformat reduzieren

Zur Reduzierung der Dateigrößen kann nach der Generierung der External Hive Table eine neue Hive-Tabelle erstellt werden, die die Daten aus der External Hive Table ausliest und in einem bestimmten, komprimierten Format im HDFS speichert. Listing 4.8 zeigt die Generierung der Hive-Tabelle *QB_Triples_comp* im *PARQUET*-Format.

```
1 CREATE TABLE QB_Triples_comp ( subject STRING, object STRING, predicate
    STRING ) STORED AS PARQUET;
```

Listing 4.8: Generierung einer Hive-Tabelle durch eine HiveQL-Create-Abfrage unter Anwendung des komprimierten PARQUET-Datenformats.

Das Einfügen der Daten aus der External Hive Table QB_Triples erfolgt durch das HiveQL-Statement aus Listing 4.9.

```
1 INSERT INTO TABLE QB_Triples_comp
2 SELECT subject, object, predicate FROM QB_Triples;
```

Listing 4.9: HiveQL Statement zum Einfügen der Daten aus der Hive-Tabelle QB_Triples in die neue Tabelle QB_Triples_comp.

Nach Ausführung dieser HiveQL-Statements sind die RDF-Daten in der neuen Tabelle QB_Triples_comp vorhanden und können mit HiveQL-Abfragen effizienter abgefragt werden. Im HDFS werden die Dateien im komprimierten PARQUET-Datenformat gespeichert. Dies reduziert die Größe der Dateien und gleichermaßen die Anzahl der benötigten MapReduce-Jobs bei der Ausführung der HiveQL-Abfragen.

Optimierung 2: Schnellere Ausführung der HiveQL-Abfragen durch Partitionierung
Gemäß der Standardeinstellungen legt eine Hive-Tabelle ihre Daten in einem bestimmten HDFS-Ordner ab. Ungeachtet der Verwendung eines komprimierten Datenformats wie PARQUET können HiveQL-Abfragen von enorm große Datenmengen weiterhin zu langen Ausführungszeiten führen.

Eine weitere Optimierung bei der Ausführungszeit kann durch das sogenannte *Partition*-Prinzip erzielt werden. In Hive führt eine Partitionierung dazu, dass die Daten anhand eines festgelegten Spalten-Werts in Unterordnern gespeichert werden.

Bei drei Spalten führt dies zur Überlegung, welche der Spalten (*subject*, *predicate*, *object*) als geeigneter Kandidat für die Partition ausgewählt werden kann. Aufgrund der Tatsache, dass die Anzahl an URIs für *subjects* und *objects* prinzipiell sehr hoch ist, handelt es sich bei diesen beiden Spalten um keine geeigneten Kandidaten. Eine zu hohe Anzahl an Unterordnern führt zu dem Effekt, dass viele kleine Dateien durch eine Vielzahl von MapReduce-Jobs geladen und gelesen werden müssen. Wie später im Kapitel 5 zu sehen ist, wird sich die Spalte *predicates* als geeigneter Kandidat für die Partitionierung erweisen.

```
1 CREATE TABLE QB_Triples_comp ( subject STRING, object STRING )
2 PARTITIONED BY (predicate STRING) STORED AS PARQUET;
```

Listing 4.10: HiveQL-Statement zur Generierung einer Hive-Tabelle unter Anwendung des PARQUET-Datenformats und der *predicate*-Partitionierung.

Für den Einsatz einer Partition werden die HiveQL-Statements aus Listing 4.8 und Listing

4.9 angepasst. Die Resultate sind in Listing 4.10 und 4.11 zu finden.

```
1 INSERT INTO TABLE QB_Triples_comp PARTITION (predicate)
2 SELECT subject, object, predicate FROM QB_Triples;
```

Listing 4.11: HiveQL-Statement zum Einfügen der Daten in eine Hive-Tabelle mit Partition nach der Spalte *predicate*.

Die vorgestellten Optimierungen haben jedoch den Nachteil, dass eine weitere Tabelle generiert werden muss. Dieser Prozess benötigt weiteren Speicherplatz im HDFS und zusätzliche Ausführungszeit bei der Generierung der Hive-Tabelle QB_Triples_comp. Die Dauer der Generierung dieser Hive-Tabelle wird in der Evaluation im Hinblick auf die Ausführungszeit des ETL-Prozesses sowie auf die HiveQL-Abfragen zur Ermittlung der Metainformationen untersucht.

4.2.2. Komponente 2: Multidimensionales Model auslesen

Nach der Speicherung der Triples in der Hive-Tabelle QB_Triples[1] werden die Metainformationen aus dem RDF Data Cube Vocabulary mithilfe von verschiedenen, hintereinander ausgeführten HiveQL-Abfragen ausgelesen und in ein in Java modelliertes, multidimensionales Datenmodell gespeichert. Diese Aufgabe übernimmt die zweite Komponente *MDM-Loader*. Abbildung 4.2 zeigt die konzeptionelle Umsetzung des multidimensionalen Datenmodells.

Die HiveQL-Abfragen sollen die im folgenden aufgelisteten Cube-Informationen mit effizienten HiveQL-Abfragen auslesen.

QB Measures

Measures im QB-Vokabular (*qb:MeasureProperty*) werden durch die Property *qb:measure* beschrieben (s. Beispiel-Graph in Abbildung 4.3). Zusätzliche Informationen, die bei jedem einzelnen Measure ausgelesen werden, sind im Folgenden beschrieben.

1. Jedes Measure kann optional ein Literal mit der Property *rdfs:label* enthalten. Diese Zeichenkette dient als Name des Measures und wird später im Mondrian Schema

[1] Der Name der Hive-Tabelle QB_Triples ist in diesem und in den kommenden Abschnitten unabhängig davon gewählt, ob die Optimierungen aus dem Abschnitt 4.2.1 angewendet wurden oder nicht.

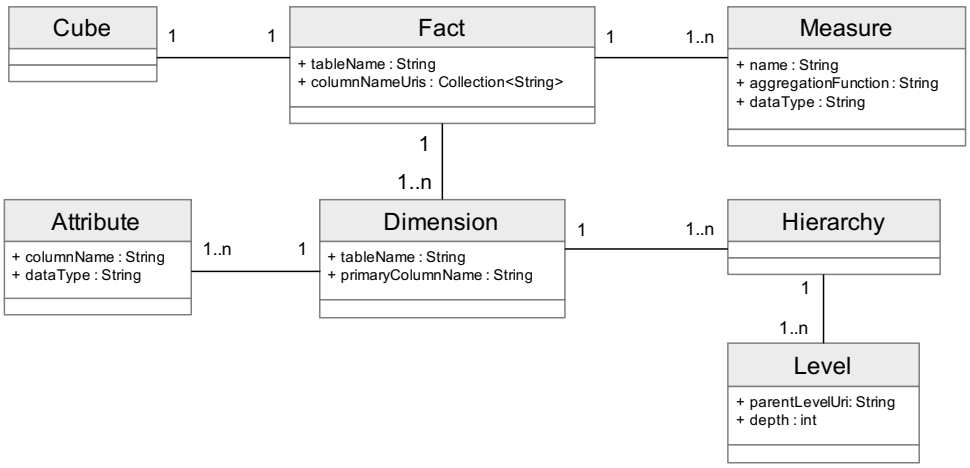

Abbildung 4.2.: Konzeptionelle Umsetzung des MDM-Modells.

angegeben. Fehlt diese Angabe, wird die Bezeichnung der URI verwendet.

2. Jedes Measure besitzt optional einen Datentyp, beschrieben durch die Property *rdfs:range*. Datentypen können z. B. durch die URIs *xsd:int*, *xsd:long*, *xsd:float* oder *xsd:double* repräsentiert werden. Besitzt ein Measure keinen Datentyp, wird die URI „xsd:int“ als Standardwert gewählt.

3. Jedes Measure kann optional eine Aggregationsfunktion besitzen, beschrieben durch die Property *qb4o:aggregateFunction*. Aggregationsfunktionen können z. B. durch die URIs *qb4o:sum*, *qb4o:min*, *qb4o:max*, *qb4o:count* oder *qb4o:avg* dargestellt werden. Bei fehlender Angabe wird die URI „qb4o:sum“ verwendet.

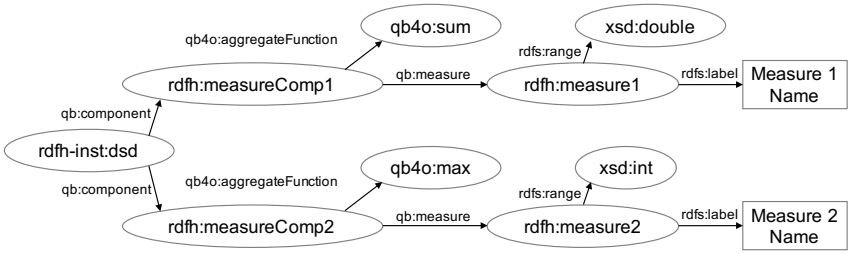

Abbildung 4.3.: Beispiel eines RDF-Graphs zur Beschreibung der Measures im QB-Vokabular.

Wie in Abbildung 4.3 am ersten Measure zu erkennen ist, sind die auszulesenden Informationen entweder am Subjekt „rdfh:measureComp1“ oder am Subjekt „rdfh:measure1“

angehängt. Zum Auslesen der benötigten Informationen werden drei HiveQL-Abfragen benötigt. Diese sind in Listing 4.12 dargestellt.

```
1  SELECT * FROM QB_Triples WHERE
2    predicate = "<http://purl.org/linked-data/cube#measure>";
3
4  SELECT * FROM QB_Triples WHERE
5    predicate = "<http://purl.org/qb4olap/cubes#aggregateFunction>";
6
7  SELECT * FROM QB_Triples WHERE
8    predicate IN ( "<http://www.w3.org/2000/01/rdf-schema#range>",
9      "<http://www.w3.org/2000/01/rdf-schema#label>" )
10   AND subject IN ( "<http://lod2.eu/schemas/rdfh#measure1>",
11     "<http://lod2.eu/schemas/rdfh#measure2>" );
```

Listing 4.12: HiveQL-Statement zum Auslesen der QB-Measure-Informationen.

Aus dem Ergebnis der ersten HiveQL-Abfrage wird für jedes gefundene *qb:measure* ein Java-Objekt *Measure* erstellt. Die zweite Abfrage liest die zugehörigen Aggregationsfunktionen der Measures aus. Die dritte HiveQL-Abfrage führt dazu, dass die Labels und die Ranges ausgelesen werden.

QB Dimensions

Die Dimensionen (*qb:DimensionProperty*) des OLAP Cubes werden im QB-Vokabular durch die Property *qb:dimension* definiert. Wie im Graph in Abbildung 4.4 an der ersten Dimension *rdfh:dim1* zu erkennen ist, sind die auszulesenden Informationen am Subjekt „rdfh:dim1“ angehängt. Das HiveQL-Statement zum Auslesen der Informationen der QB Dimensions wird im Listing 4.13 dargestellt. Aus dem Ergebnis der HiveQL-Abfrage wird für jedes gefundene qb:dimension ein Java-Objekt *Dimension* erstellt. Die Hierarchien und die jeweiligen Levels werden in gesonderten HiveQL-Abfragen ausgelesen.

```
1  SELECT * FROM QB_Triples WHERE
2    predicate = "<http://purl.org/linked-data/cube#dimension>";
```

Listing 4.13: HiveQL-Statement zum Auslesen der Dimensionen im QB-Vokabular.

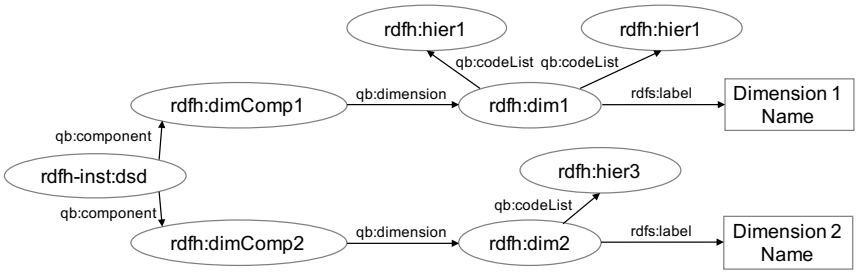

Abbildung 4.4.: Beispiel eines RDF-Graphs zur Beschreibung der Dimensionen im QB-Vokabular.

QB Hierarchies

Hierarchien im QB-Vokabular (*qb:CodedProperty*) werden durch die Property *qb:codeList* bestimmt. Wie im vorangegangenen Abschnitt in Abbildung 4.4 an der ersten Dimension *rdfh:dim1* zu erkennen ist, sind die auszulesenden Informationen am Subjekt „rdfh:dim1“ angehängt. Das HiveQL-Statement zum Auslesen der Informationen der QB Hierarchies wird im Listing 4.14 dargestellt.

```
1 SELECT * FROM QB_Triples WHERE
2   predicate = "<http://purl.org/linked-data/cube#codeList>";
```

Listing 4.14: HiveQL-Statement zum Auslesen der Hierarchien im QB-Vokabular.

Analog zu den Dimensionen wird für jede gefundene Hierarchie ein Java-Objekt *Hierarchy* erstellt. Ferner wird die Hierarchie an die jeweils zugehörige Dimension in einer Java-Collection hinzugefügt. Die weiteren Informationen, wie beispielsweise welche Levels die ermittelten Hierarchien besitzen, werden in der nächsten HiveQL-Abfrage ausgelesen.

QB Levels

In einer Hierarchie werden die Levels (*skos:Concept*) durch die Property *skos:inScheme* bestimmt. Als zusätzliche Information wird bei jedem einzelnen Level die „Tiefe“ (engl. *depth*) ausgelesen. Die Tiefe gibt eine explizite Darstellung der Reihenfolge der verschiedenen Levels einer Hierarchie an.

Wie in Abbildung 4.5 zu erkennen ist, sind die Levels am Subjekt „rdfh:hier1“ bzw. „rdfh:hier2“ angehängt. Die zusätzliche Information ist jedoch direkt mit dem Level verbun-

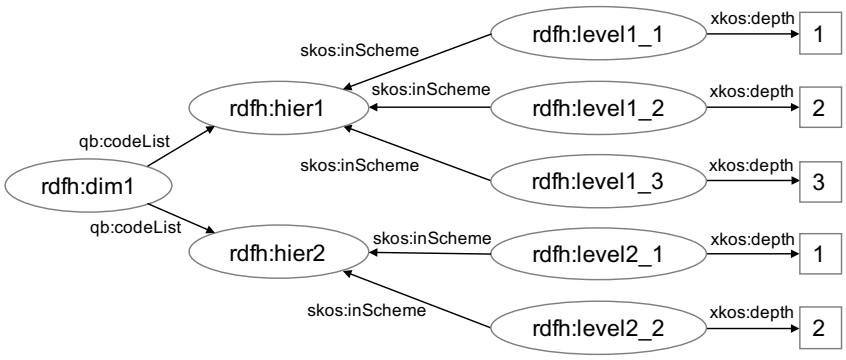

Abbildung 4.5.: Beispiel eines RDF-Graphs zur Beschreibung der Levels von Hierarchien einer Dimension im QB-Vokabular.

den. Infolgedessen ist bei der Abfrage dieser Informationen ein Join über die QB_Triples-Tabelle notwendig. Das HiveQL-Statement zum Auslesen der Informationen der QB Levels wird in Listing 4.15 dargestellt.

```
1 SELECT qbTbl1.subject, qbTbl1.object, qbTbl2.object as depth
2 FROM QB_Triples as qbTbl1
3 JOIN QB_Triples as qbTbl2 ON (qbTbl1.subject = qbTbl2.subject)
4 WHERE
5   qbTbl1.predicate = "<http://www.w3.org/2004/02/skos/core#inScheme>"
6   AND qbTbl2.predicate = "<http://purl.org/linked-data/xkos#depth>";
```

Listing 4.15: HiveQL-Statement für das Auslesen der Levels einer Hierarchie im QB-Vokabular.

Die bereits in der vorherigen HiveQL-Abfrage generierten MDM-Hierarchien (s. Listing 4.13) werden um die ausgelesenen Levels erweitert. Analog zur Speicherung der Hierarchien in den Dimensionen, werden die Levels einer Hierarchie in einer Java-Collection gespeichert.

QB Attributes

Das Ziel bei der Umsetzung des ETL-Prozesses ist die Generierung des kleinstmöglichen OLAP Cubes in Kylin. Mit den vorgestellten HiveQL-Abfragen wird ein multidimensionales Datenmodell definiert, das alle notwendigen Informationen beinhaltet, um Daten mit hierarchischem Aufbau analysieren zu können. Sollen jedoch Analysen auf Daten durch-

geführt werden, die keinem hierarchischem Aufbau entsprechen, muss in Kylin ein OLAP Cube definiert werden, der alle Attribute in den Vorberechnungen der Cuboids einbezieht.

Der Graph in Abbildung 4.6 zeigt ein einfaches Beispiel, wie im QB-Vokabular die Attribute, die keinem hierarchischem Aufbau entsprechen, dargestellt werden können.

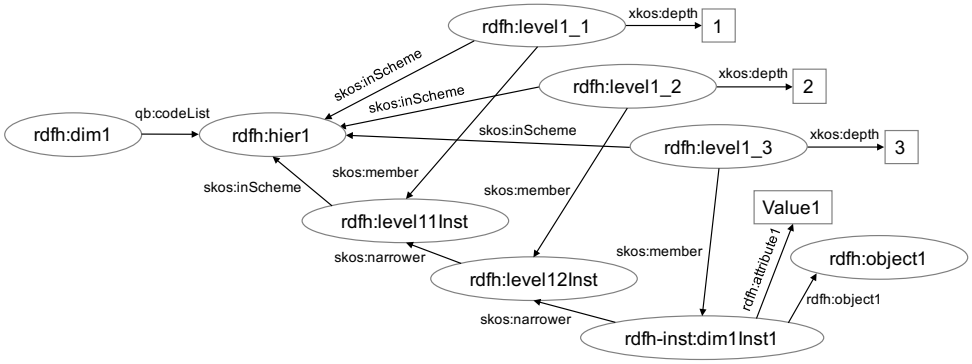

Abbildung 4.6.: Beispiel eines RDF-Graphs zur Beschreibung der Attribute einer Dimensions-Instanz im QB-Vokabular.

Wie in dieser Abbildung zu erkennen ist, sind die auszulesenden Informationen am Subjekt „rdfh:dim1Inst1“ angehängt. Aus diesem Grund ist bei der Abfrage dieser Informationen ein Join über die QB_Triples-Tabelle anhand der Property *skos:member* notwendig. Das HiveQL-Statement zum Auslesen der Informationen wird in Listing 4.16 dargestellt.

```
1  SELECT qbTbl1.subject, qbTbl2.predicate
2  FROM QB_Triples AS qbTbl1
3  JOIN QB_Triples AS qbTbl2 ON (qbTbl2.subject = qbTbl1.object)
4  WHERE
5    qbTbl1.subject IN (
6      "<http://lod2.eu/schemas/rdfh#level1_3>",
7      "<http://lod2.eu/schemas/rdfh#level2_2>",
8      "<http://lod2.eu/schemas/rdfh#level3_2>", ...
9    )
10   AND qbTbl1.predicate = "<http://www.w3.org/2004/02/skos/core#member>"
11 GROUP BY qbTbl1.subject, qbTbl2.predicate
```

Listing 4.16: Relevanter Ausschnitt des HiveQL-Statements zum Auslesen der Attribute der Dimensionen im QB-Vokabular.

Für jedes Attribut wird ein Java-Objekt *Attribute* erstellt und an die zugehörige Dimension

in einer Java-Collection gespeichert. Nach dieser letzten Abfrage enthält das multidimensionale Datenmodell allen notwendigen Metainformationen aus dem QB-Vokabular.

4.2.3. Komponente 3: Sternschema in Hive generieren

Nach der Generierung des MDMs ist eine Transformation der Hive-Tabelle QB_Triples in das Sternschema notwendig. Die Hauptaufgabe der dritten Komponente *MDM-2-Starschema* besteht darin, mit so wenigen CTAS-Abfragen die zeilenbasierten Daten aus der QB_Triples-Tabelle in ein spaltenbasiertes Format zu transformieren.

Dimensionstabellen in Hive generieren

Für jede Dimension aus dem MDM soll eine Dimensionstabelle in Hive generiert werden, die alle Hierarchien und Levels als Spalten enthält. Der Primary-Key wird bei jeder Dimension in der Spalte *key* gespeichert.

Für alle Levels einer Hierarchie wird eine eigene Spalte angelegt. Besitzt die Dimension nur eine Hierarchie, kann in einer einzelnen CTAS-HiveQL-Abfrage die Dimensionstabelle generiert werden. Anderensfalls ist für jede Hierarchie ein CTAS-HiveQL-Statement zu definieren, welches die Dimensionstabelle um die Levels der nächsten Hierarchie durch das Hinzufügen von neuen Spalten erweitert. Listing 4.17 zeigt den relevanten Teil der CTAS-HiveQL-Abfrage.

```
1 CREATE TABLE dim2 STORED AS PARQUET AS
2 SELECT qbTbl1.object AS key, qbTbl2.subject AS level3_1, ... , qbTbl3.object AS
      attribute1, ...
3 FROM QB_Triples qbTbl1
4 LEFT JOIN QB_Triples qbTbl2 ON (qbTbl2.object = qbTbl1.object
5   AND qbTbl2.predicate = "<http://www.w3.org/2004/02/skos/core#narrower>") ...
6 LEFT JOIN QB_Triples qbTbl3 ON (qbTbl3.subject = qbTbl1.object
7   AND qbTbl3.predicate = "<http://lod2.eu/schemas/rdfh#attribute1>") ...
8 WHERE qbTbl1.subject = "<http://lod2.eu/schemas/rdfh#leve3_2>"
9   AND qbTbl1.predicate = "<http://www.w3.org/2004/02/skos/core#member>";
```

Listing 4.17: CTAS-HiveQL-Statement zur Generierung einer Dimensionstabelle anhand der Informationen aus dem MDM.

Die Navigation durch die Levels einer Hierarchie wird durch die Properties *skos:narrower* und *skos:member* ermöglicht. Bei mehr als zwei Levels findet durch die Angaben von *LEFT*-Joins eine Navigation entlang der *narrower*-Property bis zum letzten Level statt.

Die Faktentabelle in Hive generieren

Die Informationen zur Faktentabelle werden aus dem MDM ausgelesen. Für jedes Measure gilt es, eine Spalte in Hive zu definieren. Der Datentyp der Spalte wird aus dem Wert im MDM ausgelesen. Ferner wird für jeden Fremdschlüssel eine Spalte mit der Bezeichnung der Dimension und dem Datentyp String angelegt. Listing 4.18 zeigt den relevanten Ausschnitt der CTAS-HiveQL-Abfrage, mit dem alle Fakten ausgelesen und in einer Tabelle mit dem Titel *facts* gespeichert werden.

```
1  CREATE TABLE facts STORED AS PARQUET AS
2  SELECT
3    qbTbl1.subject AS key,
4    qbTbl2.object AS dim1, ...
5    cast(qbTbl3.object AS DOUBLE) measure1, ...
6  FROM QB_Triples qbTbl1
7  LEFT JOIN QB_Triples qbTbl2 ON ( qbTbl2.subject = qbTbl1.subject
8    AND qbTbl2.predicate = "<http://lod2.eu/schemas/rdfh#dim1>" ) ...
9  LEFT JOIN QB_Triples qbTbl3 ON ( qbTbl3.subject = qbTbl1.subject
10   AND qbTbl3.predicate = "<http://lod2.eu/schemas/rdfh#measure1>" ) ...
11 WHERE
12   qbTbl1.object = "<http://purl.org/linked-data/cube#Observation>"
13   AND qbTbl1.predicate = "<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>";
```

Listing 4.18: Relevanter Ausschnitt des CTAS-HiveQL-Statements zur Generierung der Faktentabelle anhand der Informationen aus dem MDM.

Durch die *Cast*-Methode findet bei jedem Measure eine Typumwandlung vom String-Wert in den zuvor ausgelesenen Datentyp des jeweiligen Measures statt. Das Ergebnis dieser CTAS-Abfrage hat die Generierung der Faktentabelle mit allen Observations zur Folge. Tabelle 4.1 zeigt einen möglichen Ausschnitt der Faktentabelle.

Die Vorbereitungen für die Generierung des OLAP Cubes in Kylin sind nach diesem Schritt abgeschlossen. Die MDM-2-Starschema-Komponente hat aus der QB_Triples-Tabelle die

key	dim1	dim2	measure1	measure2
rdfh-inst:fact1	rdfh-inst:dim1Inst1	rdfh-inst:dim2Inst1	1.99	29
rdfh-inst:fact2	rdfh-inst:dim1Inst7	rdfh-inst:dim2Inst1	13.37	1986

Tabelle 4.1.: Tabellarische Darstellung der generierten Faktentabelle nach Ausführung der CTAS-HiveQL-Abfrage aus Listing 4.18.

Hive-Tabellen des Sternschemas generiert; die Grundlage der nächsten Komponente.

4.2.4. Komponente 4: Metadata Modell und Cube Build in Kylin

Ist die Transformation der QB_Triples-Tabelle in Hive-Tabellen im Sternschema abgeschlossen, werden in der vierten Komponente *MDM-2-Kylin* alle notwendigen Schritte für die Generierung des OLAP Cubes in Kylin ausgeführt. Die Vorbereitungen bestehen aus vier Schritten: Dem optionalen Anlegen eines Kylin-Projektes, der Synchronisation der Hive-Tabellen, der Definition des Cube-Modells sowie aus dem Start des Cube-Build-Prozesses. Die Ausführung und die Beschreibungen der Teilkomponenten sind Gegenstand der nächsten Abschnitte.

Optional: Generierung eines Projektes

Die Generierung eines neuen Projektes in Apache Kylin erfolgt durch einen REST Request. Die Informationen beinhalten den Titel und eine kurze Beschreibung des Projektes. Die Angabe dieser Informationen kann vor Ausführung des ETL-Prozesses individuell festgelegt werden. Listing 4.19 zeigt den REST Request. Es ist darauf zu achten, dass der Titel des Projektes in Kylin nur einmalig vorkommen darf.

```
1 {"description": "Project for RDF QB Data.", "name": "project_rdf_qb" }
```

Listing 4.19: JSON REST Request zur Generierung eines neuen Projektes in Kylin.

Synchronisation der Hive-Tabellen

Wurde das Projekt angelegt, ist eine Synchronisation der Hive-Tabellen in Kylin notwendig (s. Abschnitt 3.1.1). Anhand der ausgelesenen Informationen des MDMs ist bekannt,

welchen Bezeichnung die Fakten- und welche Bezeichnung die Dimensionstabellen besitzen. Die Übermittlung der Hive-Tabellen findet durch die kommaseparierte Auflistung der relevanten Tabellen in einem REST Request statt. Zusätzlich muss der Titel des Projektes für die Synchronisation übermittelt werden.

Generierung des Cube Data Models

Nach der Synchronisation der Hive-Tabellen werden die zuvor ausgelesenen Informationen aus dem MDM in ein JSON-Modell umgewandelt und an Kylin gesendet. Der JSON Request wird in zwei Teilen untergegliedert.

modelDescData

In diesem Bereich werden alle Informationen zum Sternschema in Hive definiert. Dabei sind Angaben zur Faktentabelle und zu allen Dimensionstabellen mit Primary- und Foreign-Key-Zuweisung notwendig. Listing 4.20 zeigt den relevanten Ausschnitt des JSON Requests.

```
1 "modelDescData": {
2   "fact_table": "DEFAULT.FACTS",
3   "lookups": [{
4     "join": {"foreign_key":["DIM1"], "primary_key":["KEY"], "type": "INNER"},
5     "table": "DEFAULT.DIM1"
6   }]}
```

Listing 4.20: Ausschnitt des JSON Requests zur Definition der Fakten- und Dimensionstabellen.

Eine notwendige Bedingung in Kylin ist die Großschreibung aller Tabellen und Spalten. Die Definition der Faktentabelle wird durch den Wert unter *fact_table* angegeben. Für jede Dimension des MDMs wird ein Eintrag im Array *lookups* generiert.

cubeDescData

In diesem Request-Bereich werden alle Informationen des OLAP Cubes definiert. Measures werden durch die Angabe der Aggregationsfunktion, des Rückgabewertes und des Spaltennamens definiert. Alle Dimensionen als auch entsprechend alle Hierarchien und Levels müssen zusätzlich angegeben werden. Die Attribute einer Dimension werden als Kylin-Dimension vom Typ „Normal“ definiert. Listing 4.21 zeigt

den relevanten Ausschnitt des JSON Requests mit einer Hierarchie.

```
"cubeDescData": {
  "measures": [{
      "function": {
        "expression": "sum", "returntype": "decimal",
        "parameter": {"type": "column", "value": "MEASURE1" }
      },
      "name": "Measure 1"
  }],
  "dimensions": [{
      "column": ["LEVEL1_1","LEVEL1_2","LEVEL1_3"],
      "hierarchy": true, "name": "Hierarchy 1", "table": "DEFAULT.DIM1"
  }]
}
```

Listing 4.21: Ausschnitt des JSON Requests zur Definition des OLAP Cubes in Kylin.

Analog zur Definition des Sternschemas sind die Tabellen und Spalten in Großbuchstaben anzugeben. Ferner muss bei einer Hierarchie auf die Reihenfolge der Spalten geachtet werden, damit die Hierarchie korrekt im OLAP Cube generiert wird. Dies wird anhand des *depth*-Wertes im QB-Vokabular sichergestellt.

Cube Build Process anstoßen

Im letzten Schritt der Komponente MDM-2-Kylin wird der Cube-Build-Prozess gestartet. Dies wird mit einem einfachen REST Request durchgeführt. Zusätzlich soll diese Komponente den Prozess überwachen und alle 10 Sekunden den aktuellen Stand des Vorgangs abfragen. Je nach Datenmenge und Größe des Clusters kann dieser Vorgang einige Zeit beanspruchen.

4.2.5. Komponente 5: Mondrian Schema definieren

Die letzte Komponente *MDM-2-Mondrian* transformiert die Informationen aus dem MDM in ein Mondrian 4 Schema. Diese XML-Konfigurationsdatei wird in Mondrian bei der Umwandlung der MDX-Abfragen in SQL benötigt. Listing 4.22 stellt den relevanten Ausschnitt für die Measures und Listing 4.23 für die Dimensionen des OLAP Cubes in Kylin dar.

```
1  <MeasureGroups>
2    <MeasureGroup name="Facts" table="FACTS">
3      <Measures>
4        <Measure aggregator="sum" column="MEASURE1" name="Measure 1"/>
5        <Measure aggregator="max" column="MEASURE2" name="Measure 2"/>
6      </Measures>
7      <DimensionLinks>
8        <ForeignKeyLink dimension="DIM1" foreignKeyColumn="DIM2"/>
9        <ForeignKeyLink dimension="DIM2" foreignKeyColumn="DIM2"/>
10     </DimensionLinks>
11   </MeasureGroup>
12 </MeasureGroups>
```

Listing 4.22: Relevanter Ausschnitt des Mondrian Schema für die Definition der Measures aus dem MDM.

```
1  <Dimensions>
2    <Dimension key="KEY" name="DIM1" table="DIM1">
3      <Attributes>
4        <Attribute keyColumn="SUBJECT" name="SUBJECT"/>
5        <Attribute keyColumn="LEVEL1_1" name="Level 1 1"/> ...
6      </Attributes>
7      <Hierarchies>
8        <Hierarchy allMemberName="All Hier1" name="Hier1">
9          <Level attribute="Level 1 1"/> ...
10       </Hierarchy>
11     </Hierarchies>
12   </Dimension>
13 </Dimensions>
```

Listing 4.23: Relevanter Ausschnitt des Mondrian Schema für die Definition der Dimensionen aus dem MDM.

Die Definition der Measures im Mondrian Schema beinhalten Informationen wie die Bezeichnung der Spalte und die Aggregationsfunktion. Im XML-Knoten *Dimensions* ist für jede Dimension des OLAP Cubes ein Kinderelement *Dimension* zu definieren. Des Weiteren muss für jede Hierarchie innerhalb einer Dimension ein XML-Element *Hierarchy* mit

den Level-Attributen erstellt werden. Analog zur Definition der Levels des Cube-Modells in Kylin ist an dieser Stelle ebenfalls auf die Reihenfolge zu achten.

Nach Abschluss dieser Komponente ist der ETL-Prozess beendet. Den Analysten ist es nun möglich, OLAP-Abfragen entweder durch MDX-Statements über Mondrian oder durch SQL-Statements direkt an Kylin zu senden.

Die Dauer des ETL-Prozesses richtet sich nach der Menge der zu verarbeitenden Daten und der Anzahl der Rechnerknoten im Cluster. Im folgenden Kapitel wird die Hypothese überprüft, ob die parallele Ausführung der aus den HiveQL-Abfragen generierten MapReduce-Jobs einen Vorteil gegenüber nicht horizontal skalierenden ETL-Prozessen bietet. Ein weiteres Augenmerk liegt auf der Ausführungsdauer analytischer Abfragen durch die horizontale Speicherung der Cuboids in der HBase-Datenbank.

5. Evaluation

In diesem Kapitel wird der entwickelte ETL-Prozess experimentell untersucht. Die Evaluation soll zeigen, ob die skalierbare Architektur für sehr große RDF-Datenmengen geeignet ist. Im ersten Abschnitt 5.1 werden die Ziele der Evaluation aufgelistet. Abschnitt 5.2 behandelt die Cluster-Umgebung des Experiments und Abschnitt 5.3 erläutert das Datenmodell des Star Schema Benchmarks. Im darauffolgenden Abschnitt 5.4 wird der ETL-Prozess auf einem Cluster mit unterschiedlicher Anzahl an DataNodes und unterschiedlichen Datenmengen untersucht. Der vorletzte Abschnitt 5.5 behandelt die Auswirkung der verschiedenen Clustergrößen auf die Ausführungsdauer der analytischen Abfragen bei unterschiedlich großen Datenmengen.

5.1. Ziel der Evaluation

Das Ziel der Abschlussarbeit ist die Bewertung des entwickelten ETL-Prozesses. Im Folgenden werden einige wichtige Fragestellungen zum ETL-Prozess beschrieben, die im Rahmen der Evaluation beantwortet werden:

- Hat die horizontale Skalierung des ETL-Prozesses bei verschieden großen Datenmengen einen messbaren Effekt?
- Wird der ETL-Prozess auf einem kleinen Cluster und mit kleiner Datenmenge effizient ausgeführt?
- Wird der ETL-Prozess in kürzerer Zeit ausgeführt als ein Import äquivalenter Daten in eine relationale Datenbank bzw. als ein Import in einem RDF Store?

Zusätzlich sollen die im Folgenden aufgelisteten Fragestellungen zu den analytischen Abfragen beantwortet werden:

- Wird die Ausführungsdauer der analytischer Abfragen durch das Hinzufügen von DataNodes beeinflusst?

- Werden die analytischen Abfragen mit deaktiviertem Mondrian-Cache interaktiv beantwortet?
- Gibt es einen Unterschied bezüglich der Antwortzeiten zwischen SQL- und MDX-Abfragen?

Bevor jedoch auf die Ausführungszeiten des ETL-Prozesses und der analytischen Abfragen eingegangen werden kann, soll die Cluster-Umgebung des Experiments beschrieben werden.

5.2. Cluster-Umgebung

Zum Zweck einer reproduzierbaren und vergleichbaren Evaluation wurden die Experimente auf einem Amazon Web Services (AWS) Rechnercluster durchgeführt. Als Distribution wurde CDH (Cloudera Distribution Including Apache Hadoop) in der Version 5.3.8 gewählt. Die Installation erfolgte durch Cloudera Director[1] in der Version 1.5.2. Cloudera Director bietet die Möglichkeit, mit wenig Aufwand ein Cluster bei AWS mit bis zu 1000 Knoten zu erstellen. Außerdem kann so die Größe des Testsystems flexibel angepasst werden, um horizontale Skalierungseffekte zu untersuchen. Zusätzlich installiert Cloudera Director das Monitoring-Programm Cloudera Manager[2], was eine individuelle Konfiguration des Clusters ermöglicht.

Bei AWS stehen verschiedene Instanztypen zur Verfügung. Für den MasterNode sowie für jeden DataNode wurde jeweils eine Amazon EC2-Instanz (Elastic Compute Cloud) vom Typ *m4.xlarge* gewählt. Jeder DataNode dieses Typs verfügt somit über einen Intel Xeon E5-2676 v3 Prozessor mit vier vCPUs (dabei entspricht eine vCPU einem Hyperthread), der mit 2,4 GHz getaktet ist. Zudem weist eine m4.xlarge-Instanz über 16 GiB Arbeitspeicher auf. Als Betriebsystem wurde Red Hat Enterprise Linux in der Version 6.6 (Amazon Maschine Image ami-cf3b47b8) verwendet.

Die in der Evaluation verwendete Konfiguration des Apache Hadoop Clusters ist in Listing 5.1 dargestellt. Alle anderen Werte des Hadoop-Ökosystems wurden bei der Evaluation nicht verändert.

[1] s. Cloudera Director Webseite unter `http://www.cloudera.com/content/www/en-us/products/cloudera-director.html`.

[2] s. Cloudera Manager Webseite unter `https://www.cloudera.com/content/www/en-us/products/cloudera-manager.html`.

```
1 mapreduce.map.memory.mb = 3072
2 mapreduce.map.java.opt = 2048
3
4 mapreduce.reduce.memory−mb = 5120
5 mapreduce.reduce.java.opt = 4096
```

Listing 5.1: Durchgeführte Konfiguration des Apache Hadoop Clusters.

Diese Hadoop-Konfigurationen waren für die Durchführung der Experimente notwendig, da Apache Kylin beim Cube-Build-Prozess bei größer werdender Datenmenge mehr Speicherzuweisung benötigte, als mit den Standardwerten bei der Einrichtung des Clusters mit Cloudera Director erzielt werden konnten.

5.3. Datengenerierung mit dem Star Schema Benchmark

Im Rahmen der Evaluation wurde für die Datengenerierung der Star Schema Benchmark verwendet (vgl. [OOC09]). Im ersten Abschnitt wird zunächst auf das Datenmodell näher eingegangen. Der zweite Abschnitt erläutert die Vorgehensweise bei der Generierung der RDF-Daten im QB-Vokabular.

5.3.1. Das SSB-Datenmodell

Der Star Schema Benchmark (SSB) beschreibt einen OLAP Cube mit den Dimensionen *Customer*, *Date*, *Part*, *Supplier*, *Quantity* und *Discount*. Die *Date*-Dimension enthält zwei Hierarchien. Die anderen Dimensionen des OLAP Cubes besitzen dagegen nur eine einzelne Hierarchie. Die Dimensionen *Quantity* und *Discount* haben keine Attribute und werden im SSB in der Faktentabelle gespeichert. Abbildung 5.1 veranschaulicht das SSB-Datenmodell. Die Hierarchien und damit einhergend die Levels der jeweiligen Dimensionen werden in Abbildung 5.2 aufgeführt. Zudem beinhaltet die Faktentabelle sieben Measures und Aggregationsfunktionen:

- discount (avg),
- extendedprice (sum),
- quantity (sum),

- lo_revenue (sum),
- supplycost (sum),
- sum_revenue (sum(extendedprice * discount)),
- sum_profit (sum(lo_revenue - supplycost)).

Ferner stellt der Benchmark 13 unterschiedliche analytische OLAP-Abfragen bereit (vgl. [OOC09]; [KH13]). Aus Gründen der Übersichtlichkeit ist eine Auflistung dieser OLAP-Anfragen an dieser Stelle nicht sinnvoll. Daher wurde bei Github eine Projektseite[3] angelegt, die neben einer Auflistung auch eine Beschreibung der OLAP Queries und die Ergebnisse der Evaluation beinhaltet.

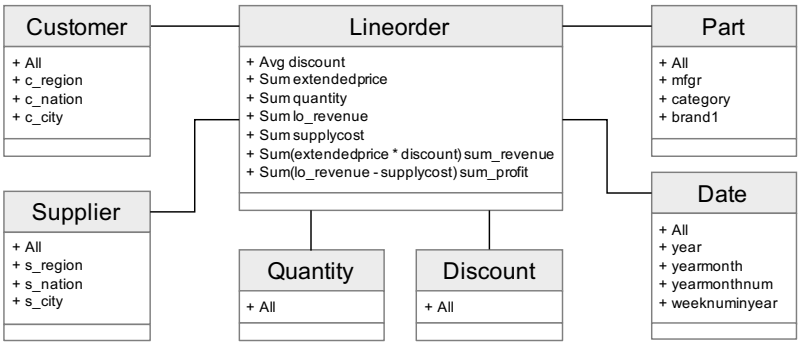

Abbildung 5.1.: SSB-Datenmodell mit der Faktentabelle *lineorder* und den sechs Dimensionstabellen in Anlehnung an [OOC09].

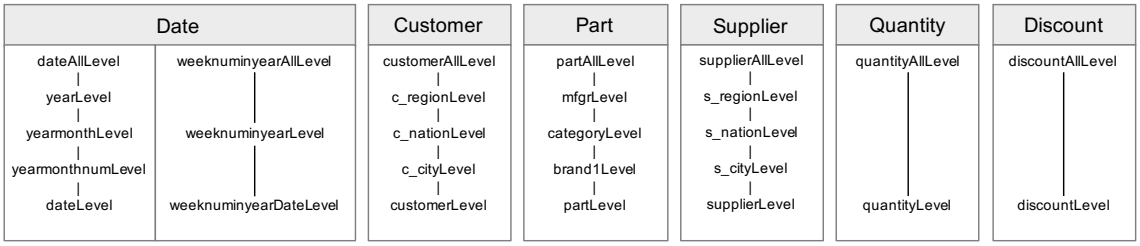

Abbildung 5.2.: Hierarchien und Levels der Dimensionen des SSB-Datenmodells.

[3] s. Projektseite unter `https://github.com/sjelsch/etl-evaluation`.

5.3.2. Generierung der RDF-Daten im QB-Vokabular

Mit dem Star Schema Benchmark werden tabellarische Daten (TBL) generiert. Der Skalierungsfaktor bestimmt die Datenmenge. Dieser Parameter wird bei der Evaluation drei Mal variiert und der ETL-Prozess sowie die Antwortzeiten der analytischen Abfragen mit Skalierung 1 (S1), Skalierung 10 (S2) und Skalierung 20 (S20) untersucht.

Die erstellten TBL-Daten können ohne großen Aufwand in relationale Datenbanken imporiert werden. Für die Generierung der RDF-Daten im QB-Vokabular sind jedoch weitere Schritte notwendig. Hierfür wird das Business Intelligence Benchmark[4] (BIBM) in der Version 0.7.8 verwendet. Mithilfe einer BIBM-Konfigurationsdatei[5] werden die generierten TBL-Daten in RDF-Daten umgewandelt. Nach dieser Transformation sind jedoch noch zwei zusätzliche Schritte notwendig, um die Daten im QB-Vokabular zu erhalten.

- Die Fakten müssen als Observations (*qb:Observation*) deklariert werden und einen Link mit der Property *qb:dataSet* zum Datensatz *rdfh-inst:ds* aufweisen (vgl. Abschnitt 2.2.6).
- Der zweite Schritt beinhaltet die Generierung der Triples für die Levels der Hierarchien. Dazu werden die RDF-Daten in den Triples Store *Open Virtuoso* geladen. Anschließend werden mit vier hintereinander ausgeführten SPARQL Insert-Abfragen die Levels der Hierarchien der Dimensionen *Customer*, *Date*, *Part* und *Supplier* erstellt.

Im Ergebnis wird eine RDF-Datenmenge im QB-Vokabular erstellt. Die Definition des OLAP Cubes im QB-Vokabular wird in der Datei dsd.ttl festgelegt, welche ebenfalls auf der Github-Projektseite zu finden ist.

Die Ausführungszeiten bei der Generierung der RDF-Datenmenge mit den Skalierungen S1, S10 und S20 werden in Tabelle 5.1 aufgelistet. Gemessen wurde die benötigte Zeit für die Generierung der TBL-Daten mit dem Star Schema Benchmark, die Transformation der Daten mit der BIBM-Konfigurationsdatei, der Import der Dimensionen in den Triples Store, die Generierung der Levels, die Transformation der RDF-Daten vom Turtle- in das N-Triples-Format sowie die benötigte Zeit für die gZip-Kompression.

Die Anzahl der Triples bei den Skalierungen S1, S10 und S20 ist in Tabelle 5.2 aufgelistet.

[4] s. Webseite unter `http://sourceforge.net/projects/bibm/`.
[5] s. Datei *ttl/01_ schema.json* auf der Projektseite.

Die Tabelle 5.3 enthält die Größen der N-Triples-Dateien in MB.

	Skalierung 1	**Skalierung 10**	**Skalierung 20**
TBL-Daten-Generierung	16s	161s	323s
TBL-2-TTL	41s	577s	1 164s
Bulk Import in Open Virtuoso	12s	52s	79s
TTL-Levels-Generierung	32s	183s	228s
TTL-2-N-Triples	358s	3 766s	7 567s
gZip-Kompression	205s	2 078s	4 116s
Gesamt	**664s**	**6 817s**	**13 477s**

Tabelle 5.1.: Ausführungsdauer in Sekunden für die Generierung der RDF-Datenmenge.

	Skalierung 1	**Skalierung 10**	**Skalierung 20**
dsd.nt	111	111	111
Date.nt	46 008	46 008	46 008
Customer.nt	270 000	2 700 000	5 400 000
Part.nt	2 000 000	8 000 000	10 000 000
Supplier.nt	16 000	160 000	320 000
Lineorder.nt	120 023 420	1 199 724 280	2 399 894 920
Levels.nt	717 351	3 381 351	4 941 351
Gesamt	**123 072 890**	**1 214 011 750**	**2 420 602 390**

Tabelle 5.2.: Anzahl Triples bei den Skalierungen S1, S10 und S20.

	Skalierung 1	**Skalierung 10**	**Skalierung 20**
Date.nt	5 MB	5 MB	5 MB
Customer.nt	29 MB	295 MB	591 MB
Part.nt	211 MB	849 MB	1 100 MB
Supplier.nt	2 MB	18 MB	35 MB
Lineorder.nt	16 300 MB	167 000 MB	330 000 MB
Levels.nt	101 MB	478 MB	702 MB
Gesamt	**16 648 MB**	**168 645 MB**	**332 433 MB**

Tabelle 5.3.: Größe der N-Triples-Dateien bei den Skalierungen S1, S10 und S20 in MB.

5.4. Ausführung des ETL-Prozesses

In diesem Abschnitt wird die Ausführung des ETL-Prozesses untersucht. Vorab soll gezeigt werden, welche Auswirkungen die im Abschnitt 4.2.1 vorgestellten Optimierungen haben. Anschließend wird die Evaluation mit den Skalierungen S1, S10 und S20 sowie bei einer Clustergröße von 3, 6 und 9 DateNodes untersucht.

5.4.1. Auswirkungen der Optimierungen

Dieses Experiment soll zeigen, welche Auswirkungen die in Abschnitt 4.2.1 vorgestellten Optimierungen auf die Ausführungsdauer des ETL-Prozesses haben. Dabei wurde der ETL-Prozess mit und ohne Optimierungen auf einem Cluster mit 3 DataNodes und der Datenmenge S1 durchgeführt.

Tabelle 5.4 listet die durchschnittlich benötigte Zeit in Sekunden für die Komponenten RDF-2-Hive, MDM-Loader und MDM-2-Star-Schema auf. Für die Ermittlung der Ausführungsdauer wurde der ETL-Prozess jeweils drei Mal ausgeführt.

	Ohne	**PARQUET**	**Partition**	**Kombination**
QB_Triples_opt	-	146s	470s	474s
MDM-Loader - Measures	289s	196s	12s	13s
MDM-Loader - Dimensions	94s	62s	1s	1s
MDM-Loader - Hierarchies	95s	62s	1s	1s
MDM-Loader - Levels	103s	86s	15s	14s
MDM-2-StarSchema - Customer	309s	266s	53s	52s
MDM-2-StarSchema - Date	618s	518s	104s	105s
MDM-2-StarSchema - Part	311s	273s	58s	59s
MDM-2-StarSchema - Supplier	308s	267s	53s	54s
MDM-2-StarSchema - Lineorder	306s	200s	182s	162s
Gesamt	**2433s**	**2076s**	**949s**	**935s**

Tabelle 5.4.: Ausführungsdauer der Komponenten in Sekunden bei Skalierung 1 und 3 DataNodes ohne Optimierungen, mit PARQUET, mit Partitionierung und mit beiden Optimierungen in Kombination.

Die zwei Komponenten MDM-2-Mondrian und MDM-2-Apache-Kylin werden von diesen Optimierungen nicht beeinflusst. Die XML-Konfigurationsdatei für Mondrian wird anhand

der ausgelesenen und im Java-Programm gespeicherten MDM-Informationen aus der zweiten Komponente MDM-Loader generiert. Wie in Abschnitt 3.1.1 erläutert, besteht der erste Schritt des Cube-Build-Prozesses von Apache Kylin darin, eine Intermediate Hive-Tabelle zu generieren. Die restlichen Berechnungen der Cuboids basieren entweder auf dieser Hive-Tabelle oder auf die zuvor berechneten Cuboids.

Ohne Optimierungen wird keine Zeit für die Generierung der Hive-Tabelle *QB_ Triples_ opt* benötigt, da die Daten direkt aus den N-Triples-Dateien im HDFS gelesen werden. Dies führt jedoch zu längeren Ausführungszeiten der HiveQL-Abfragen bei der Ermittlung des multidimensionalen Datenmodells (MDM) als auch bei der Generierung des Sternschemas in Hive. Erklärt wird dieses Verhalten durch die Anzahl der benötigten Mappers und Reducers. Hierbei muss bei jeder HiveQL-Abfrage die gesamte, unkomprimierte N-Triples-Datenmenge ausgelesen werden. Bei der Datenmenge S1 liegt die Anzahl der Mappers konstant bei 67, welche jeweils von den DataNodes abgearbeitet werden müssen.

Wird dagegen PARQUET als Datenformat gewählt, muss die Hive-Tabelle *QB_ Triples_ opt* anhand der External Hive-Table generiert werden. Einmalig werden dafür 67 Mappers benötigt. Diese Optimierung reduziert die Ausführungszeiten beim Auslesen des MDMs und die Generierung der Hive-Tabellen im Sternschema deutlich, da durch die Komprimierung der Daten durchschnittlich nur 34 Mappers benötigt werden. Im Ergebnis wird der ETL-Prozess um 357 Sekunden schneller ausgeführt.

Die Ausfühungsdauer wird noch deutlicher verringert, wenn bei der Generierung der Tabelle *QB_ Triples_ opt* nach der Spalte *predicate* partitioniert wird. Zwar dauert die Generierung der optimierten Hive-Tabelle deutlich länger (470 Sekunden anstatt 146 Sekunden), jedoch wird die Ausführungszeit beim Auslesen des MDMs von insgesamt 406 Sekunden auf 29 Sekunden verringert. Grund hierfür ist das Auslesen der MDM-Informationen nach bestimmten Prädikaten, z. B. *qb:measure* und *qb:dimension*. Diese Informationen werden durch die Partitionierung in seperaten Ordner gespeichert, was beim Auslesen dazu führt, dass lediglich ein Mapper benötigt wird. Zusätzlich wird die Generierung des Sternschemas in Hive durch die Partitionierung von 1524 Sekunden auf 450 Sekunden verringert. Die Reduzierung der Ausführungsdauer wird dadurch erreicht, dass nur die Daten aus den Partitionsordnern ausgelesen werden müssen, die die benötigten Informationen enthalten (z. B. *skos:member* und *skos:narrower*). Insgesamt wird der ETL-Prozess durchschnittlich um 1128 Sekunden schneller ausgeführt.

Werden die beiden vorgestellten Optimierungen kombiniert führt das Datenformat PAR-

QUET bei der Generierung der Faktentabelle, bei der die größte Datenmenge verarbeitet werden muss, zu einer Reduzierung der Ausführungsdauer. Grund hierfür ist die Komprimierung der Datenmenge. Das Datenformat PARQUET reduziert auch in diesem Fall die Anzahl der benötigten Mappers bei der Generierung der Faktentabelle, was eine kürzere Ausführungsdauer von durchschnittlich 20 Sekunden zur Folge hat.

Zusammenfassend hat die Untersuchung folgende Auswirkungen auf den ETL-Prozess ergeben:

- Ohne Optimierungen benötigt der ETL-Prozess am Längsten. Grund hierfür ist die Anzahl der Mappers, die benötigt wird, um eine HiveQL-Abfrage zu beantworten.
- Mit PARQUET als Datenformat wird die Ausführungsdauer des ETL-Prozesses um insgesamt 357 Sekunden reduziert.
- Durch die Partitionierung nach der *predicate*-Spalte wird der ETL-Prozess um 1485 Sekunden schneller als ohne Optimierungen und 1128 Sekunden schneller als mit dem Datenformat PARQUET ausgeführt.
- Die Kombination der beiden Optimierungen weist bei der Generierung der Faktentabelle bereits bei S1 einen Unterschied von durchschnittlich 20 Sekunden auf.

Aus dieser Evaluation geht hervor, dass die kombinierte Optimierung zur kürzesten Ausführungsdauer des ETL-Prozesses führt. Aus diesem Grund wird bei allen weiteren Experimenten die kombinierte Optimierung angewendet.

5.4.2. Horizontale Speicherung der Datenmengen ins HDFS

Die Verteilung der komprimierten N-Triples-Dateien ins HDFS wurde bei der Datenmenge S1 durchschnittlich in 6 Sekunden durchgeführt. Bei S10 und S20 führten die Messungen zum Teil zu sehr unterschiedlichen Ergebnissen. Die horizontale Speicherung der Datenmenge S10 benötigte unabhängig der Anzahl an DataNodes zwischen 60 und 216 Sekunden. Die für die Datenmenge S20 benötigte Zeit für den Upload ins HDFS lag zwischen 116 und 181 Sekunden. Dies ist auf die Netzwerk-Auslastung des AWS bei der Durchführung der Experimente zurückzuführen. Ein eindeutiges Ergebnis konnte daher im Rahmen der Evaluation nicht ermittelt werden.

5.4.3. ETL-Prozess bei horizontaler Skalierung mit S1 und S10

In diesem Abschnitt wird die horizontale Skalierung mit 3, 6 und 9 DataNodes sowie mit der Datenmenge S1 und S10 untersucht. Gemessen wurden die Ausführungszeiten der Komponenten 1-4, da Komponente 5 (MDM-2-Mondrian) bei allen Experimenten durchschnittlich in 0,06 Sekunden ausgeführt wurde. Abbildung 5.3 und 5.4 zeigen bei S1 und S10 die gesamten Ausführungszeiten der einzelnen Komponenten bei unterschiedlichen Clustergrößen.

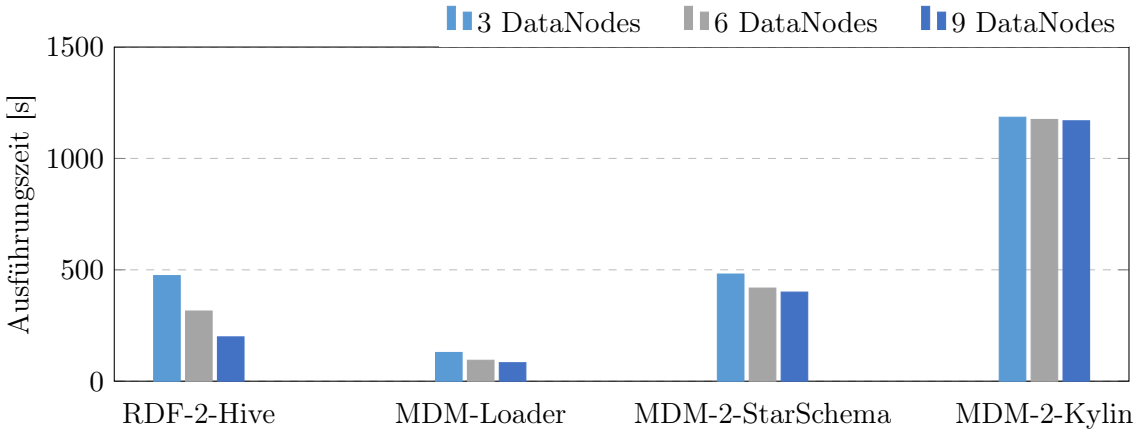

Abbildung 5.3.: Ausführungsdauer der einzelnen Komponenten des ETL-Prozesses mit der Datenmenge S1 sowie 3, 6 und 9 DataNodes.

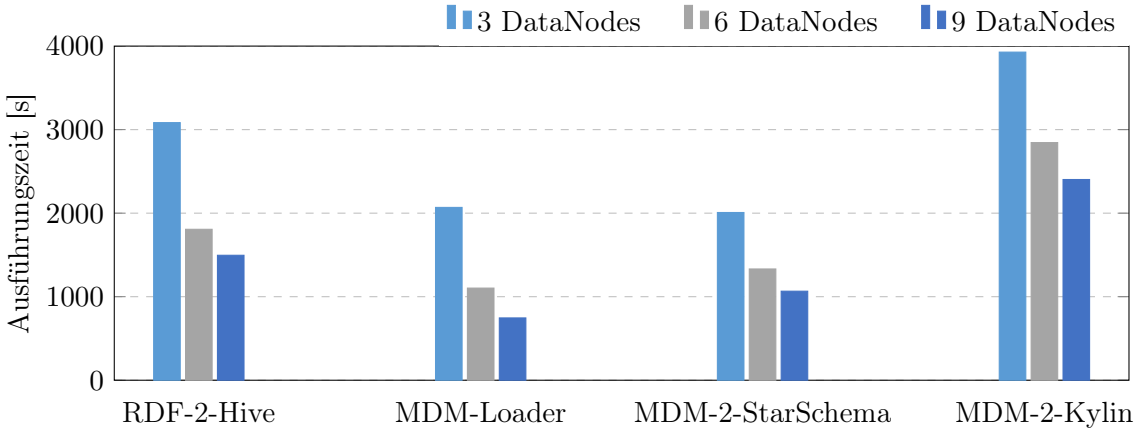

Abbildung 5.4.: Ausführungsdauer der einzelnen Komponenten des ETL-Prozesses mit der Datenmenge S10 sowie 3, 6 und 9 DataNodes.

Evaluation der 1. Komponente RDF-2-Hive

Die horizontale Skalierung ist bei der Generierung der Hive-Tabelle *QB_ Triples_ opt* deutlich erkennbar. Wie im vorangegangen Abschnitt erläutert, werden bei der Generierung der optimierten Hive-Tabelle mit Partition und PARQUET alle Triples aus den N-Triples-Dateien geladen. Infolgedessen reduziert das Hinzufügen von DataNodes die Ausführungsdauer der *Create-Table-As-Select*-Abfrage. Eine größere Anzahl an DataNodes führt dazu, dass mehr Mappers und mehr Reducers parallel ausgeführt werden können.

Evaluation der 2. Komponente MDM-Loader

Wird bei der MDM-Loader-Komponente die horizontale Skalierung betrachtet, lässt sich eine Reduzierung der Laufzeit bei den Datenmengen S1 und S10 erkennen. Abbildungen 5.5 und 5.6 zeigen eine detailliertere Auflistung der Ausführungszeiten der MDM-Loader-Komponente zum Auslesen der Measures, Dimensionen, Hierachien, Levels und Attribute.

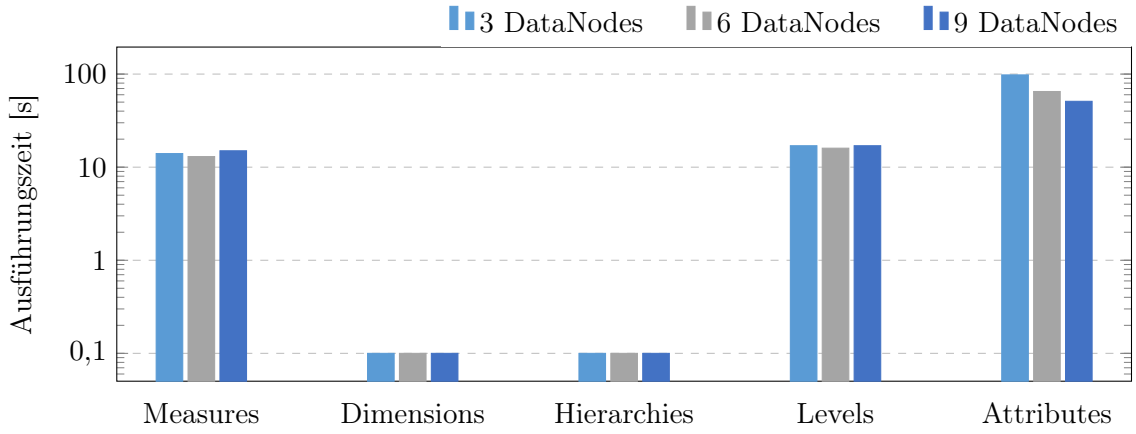

Abbildung 5.5.: Ausführungsdauer der zweiten Komponente MDM-Loader bei der Datenmenge S1 sowie 3, 6 und 9 DataNodes.

Die horizontale Skalierung ist beim Auslesen der Measures und Levels nicht erkennbar. Aufgrund der Partitionierung werden die HiveQL-Abfragen sowohl bei der Datenmenge S1 als auch bei S10 in einem MapReduce-Job mit einem Mapper umgewandelt. Daher kann bei der verwendeten Datenmenge durch das Hinzufügen von DataNodes die Ausführungsdauer nicht reduziert werden. Ferner wird beim Auslesen der Dimensionen und Hierarchien kein MapReduce-Job generiert, da die benötigten Daten durch die Partitionierung direkt

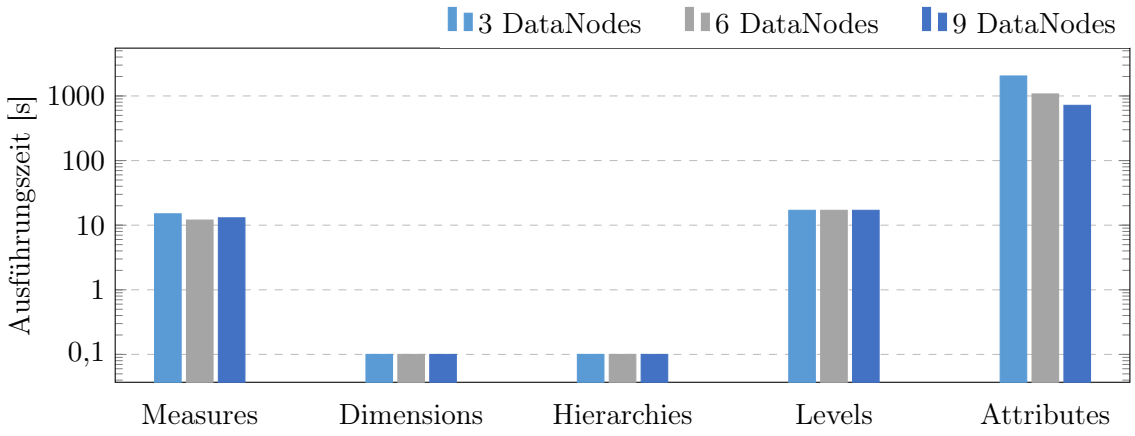

Abbildung 5.6.: Ausführungsdauer der zweiten Komponente MDM-Loader bei der Datenmenge S10 sowie 3, 6 und 9 DataNodes.

auslesen werden. Dagegen wird die HiveQL-Abfrage zum Auslesen der Attribute in einem MapReduce-Job mit vielen Mappern und Reducern umgewandelt. Infolgedessen hat die horizontale Skalierung einen Einfluss auf die Ausführungsdauer. Tabelle 5.5 listet die durchschnittlichen Ausführungszeiten zum Auslesen der Attribute auf.

	3 DataNodes	**6 DataNodes**	**9 DataNodes**
Skalierung 1	98s	65s	51s
Skalierung 10	2037s	1074s	716s

Tabelle 5.5.: Ausführungsdauer in Sekunden der MDM-Loader-Komponente zum Auslesen der Attribute.

Der MapReduce-Job bei S1 mit 3 DataNodes benötigt durchschnittlich 98 Sekunden, bei 6 DataNodes 65 Sekunden (33,7% schneller) und bei 9 DataNodes 51 Sekunden (48,0% schneller als mit 3 DataNodes und 21,5% schneller als mit 6 DataNodes).

Das Auslesen der Attribute bei der Datenmenge S10 benötigt mit 3 DataNodes durchschnittlich 2037 Sekunden, mit 6 DataNodes 1074 Sekunden (47,3% schneller) und mit 9 DataNodes 716 Sekunden (64,9% schneller als mit 3 DataNodes und 33,3% schneller als mit 6 DataNodes). Die Verdopplung der DataNodes führt daher zu einer Halbierung der Ausführungsdauer. Ferner beträgt der Unterschied zwischen dem Cluster mit 6 und 9 DataNodes 33%. Bei diesen Messungen ist die horizontale Skalierung sehr deutlich erkennbar.

Evaluation der 3. Komponente: MDM-2-StarSchema

Die Ausführungszeiten der dritten Komponenten MDM-2-StarSchema - die Generierung der Fakten- und Dimensionstabellen in Hive - sind für die Datenmenge S1 in Abbildung 5.7 und für die Datenmenge S10 in Abbildung 5.8 aufgelistet.

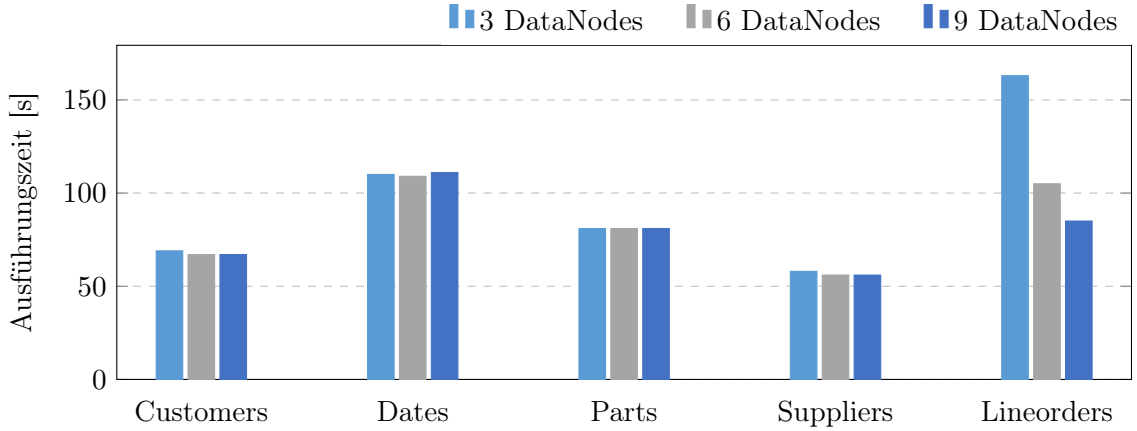

Abbildung 5.7.: Ausführungsdauer der dritten Komponente MDM-2-StarSchema bei der Datenmenge S1 sowie 3, 6 und 9 DataNodes.

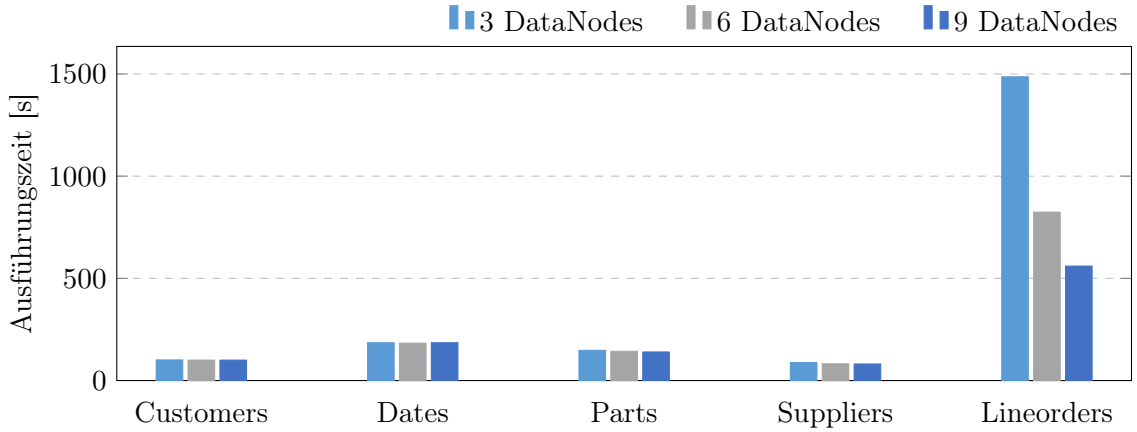

Abbildung 5.8.: Ausführungsdauer der dritten Komponente MDM-2-StarSchema bei der Datenmenge S10 sowie 3, 6 und 9 DataNodes.

Analog zur Komponente MDM-Loader sind die Datenmengen S1 und S10 aufgrund der Partitionierung zu klein, um bei der Generierung der Dimensionstabellen einen Unterschied in den Ausführungszeiten mit 3, 6 und 9 DataNodes zu erkennen. Jedoch wird

deutlich, dass bei der Generierung der Date-Tabelle mehr Zeit benötigt wird als bei den anderen Dimensionstabellen. Dies ist mittels der zwei Hierarchien innerhalb der Dimension zu erklären. Die Date-Tabelle wird in zwei Durchläufe generiert. In der ersten Phase wird die Tabelle mit den Attributen und den Levels der ersten Hierarchie erstellt. In der zweiten Phase werden die Spalten für die Levels der zweiten Hierarchie zur bestehenden Hive-Tabelle hinzugefügt.

Des Weiteren ist erkennbar, dass die horizontale Skalierung bei der Generierung der Faktentabelle eine deutliche Reduzierung der Aufführungszeit zur Folge hat. In diesem Fall können durch mehr DataNodes mehr Mappers und mehr Reducers parallel ausgeführt werden. Tabelle 5.6 listet die durchschnittlichen Ausführungszeiten zum Auslesen der Attribute auf.

	3 DataNodes	**6 DataNodes**	**9 DataNodes**
Skalierung 1	163s	105s	85s
Skalierung 10	1486s	824s	560s

Tabelle 5.6.: Ausführungsdauer in Sekunden der MDM-2-Kylin-Komponente zum Generieren der Faktentabelle.

Bei der Datenmenge S1 und 3 DataNodes werden 163 Sekunden, bei 6 DataNodes 105 Sekunden (35,6% schneller) und bei 9 DataNodes 85 Sekunden (47,9% schneller als bei 3 DataNodes und 19,0% schneller als bei 6 DataNodes) benötigt. Im Hinblick auf die Datenmenge S10 werden bei 3 DataNodes 1486 Sekunden, bei 6 DataNodes 824 Sekunden (44,5% schneller) und bei 9 DataNodes 560 Sekunden (62,3% schneller als mit 3 DataNodes und 32,0% als mit 6 DataNodes) benötigt. In beiden Fällen ist die horizontale Skalierung und die damit verbundene Reduzierung der Ausführungszeiten erkennbar.

Evaluation der 4. Komponente: MDM-2-Kylin

Der Cube-Build-Prozess in Apache Kylin (Komponente 4) wird durch die horizontale Skalierung der Datenmenge S1 nicht beeinflusst (s. Abbildung 5.3). Bei 3 DataNodes benötigt der Prozess 1185 Sekunden, bei 6 DataNodes 1175 Sekunden und bei 9 DataNodes 1169 Sekunden. Dieses Verhalten lässt sich dadurch erklären, dass die Datenmenge S1 nicht ausreicht, um eine kürzere Ausführungsdauer durch das Hinzufügen von DataNodes zu erreichen. Der Cube-Build-Prozess berechnet in allen Experimenten 14 verschiedene Cuboids. Bei S1 werden für diese Vorberechnungen im Durchschnitt 1 Mapper und 1 Reducer

benötigt. Die horizontale Skalierung hat daher bei dieser Datenmenge keinen Einfluss auf die Ausführungsdauer des Cube-Build-Prozesses in Kylin.

Dagegen ist die horizontale Skalierung der Datenmenge S10 in der Komponente MDM-2-Kylin erkennbar (s. Abbildung 5.4). Der Cube-Build-Prozess benötigt bei 3 DataNodes durchschnittlich 3928 Sekunden, bei 6 DataNodes 2845 Sekunden (27,6% schneller) und bei 9 DataNodes 2403 Sekunden (38,8% schneller als mit 3 DataNodes und 15,5% schneller als mit 6 DataNodes). Eine Verdopplung der DataNodes von 3 auf 6 führt zwar zu einer Reduzierung der Ausführungsdauer, jedoch nicht um die erwarteten 50%. Dieses Verhalten wird folgendermaßen erklärt:

- Das Hinzufügen von DataNodes hat zunächst eine Auswirkung auf den Cube-Build-Prozess. Zu Beginn werden viele Mappers und Reducers benötigt, um die großen Cuboids des OLAP Cubes zu berechnen. Je weiter der Prozess voranschreitet, desto kleiner werden die Cuboids. Dies führt zu einer geringeren Anzahl an Mappern und Reducern. Ab einem gewissen Zeitpunkt hat die Anzahl der DataNodes auf die Berechnungszeit keine messbare Auswirkung mehr, da weniger Mapper für die Berechnungen benötigt werden, als DataNodes im Cluster vorhanden sind.
- Der letzte Schritt des Cube-Build-Prozesses - die Generierung der HFile für den Bulk-Import in die HBase-Datenkbank - wird bei der Datenmenge S10 mithilfe von drei Mappern generiert. In der Evaluation beanspruchte dieser Schritt im Durchschnitt 30% des Cube-Build-Prozesses. Durch die geringe Anzahl an Mappern hat die Erhöhung der DataNodes jedoch keinen Einfluss auf die Ausführungsdauer der HFile-Generierung.

5.4.4. ETL-Prozess bei horizontaler Skalierung mit S20

In diesem Abschnitt wird der ETL-Prozess bei der Datenmenge S20 und 9 DataNodes untersucht. Zum Vergleich werden die Datenmengen S1 und S10 bei 9 DataNodes herangezogen.

Analog zur den Datenmengen S1 und S10 wurden die Ausführungszeiten der Komponenten 1-4 gemessen. Abbildung 5.9 zeigt die Ausführungsdauer der einzelnen Komponenten im Vergleich zu den Datenmengen S1 und S10 bei einem Cluster mit 9 DataNodes.

Die Durchführung der RDF-2-Hive- und MDM-Loader-Komponente benötigt bei doppelter

Datenmenge und gleicher Anzahl an DataNodes doppelt so viel Zeit.

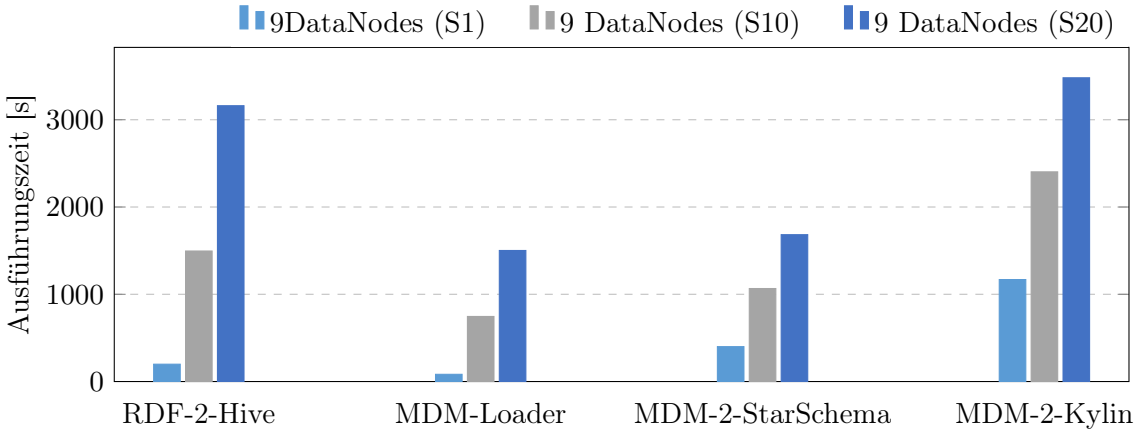

Abbildung 5.9.: Ausführungsdauer der einzelnen Komponente des ETL-Prozesses bei S1, S10 und S20 mit 9 DataNodes.

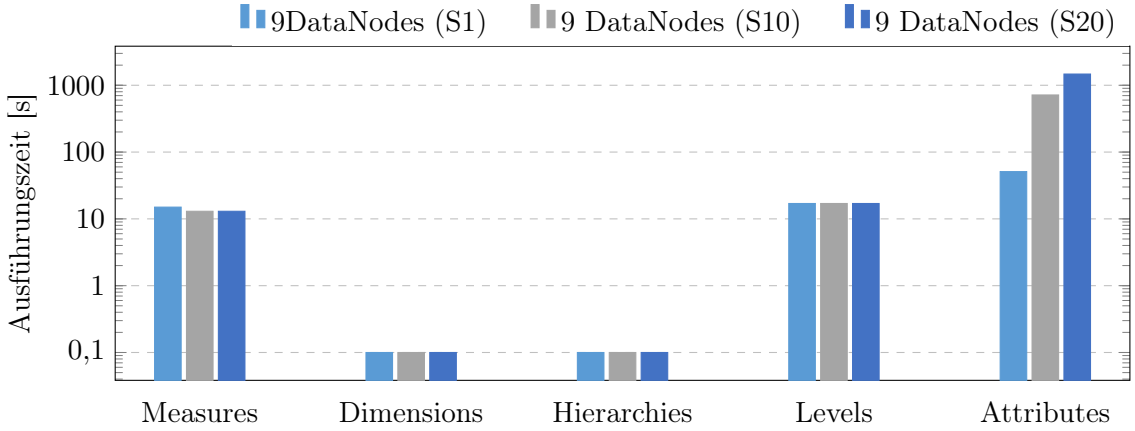

Abbildung 5.10.: Ausführungsdauer der zweiten Komponente MDM-Loader bei S1, S10 und S20 sowie 9 DataNodes.

Abbildung 5.10 zeigt die Ausführungsdauer der MDM-Loader-Komponente. Erneut ist erkennbar, dass die Ausführungsdauer der MDM-Loader-Komponente beim Auslesen der Measures, Dimensionen, Hierarchien und Levels bei allen Datenmengen konstant bleibt. Grund hierfür ist die Partitionierung nach der *predicate*-Spalte.

Das Auslesen der Attribute bei gleichbleibender Anzahl an DataNodes führt bei doppelter Datenmenge auch zur doppelten Ausführungsdauer. Bei S1 werden 51 Sekunden, bei S10 716 Sekunden und bei S20 1472 Sekunden benötigt. Der Unterschied zwischen S1 und S10

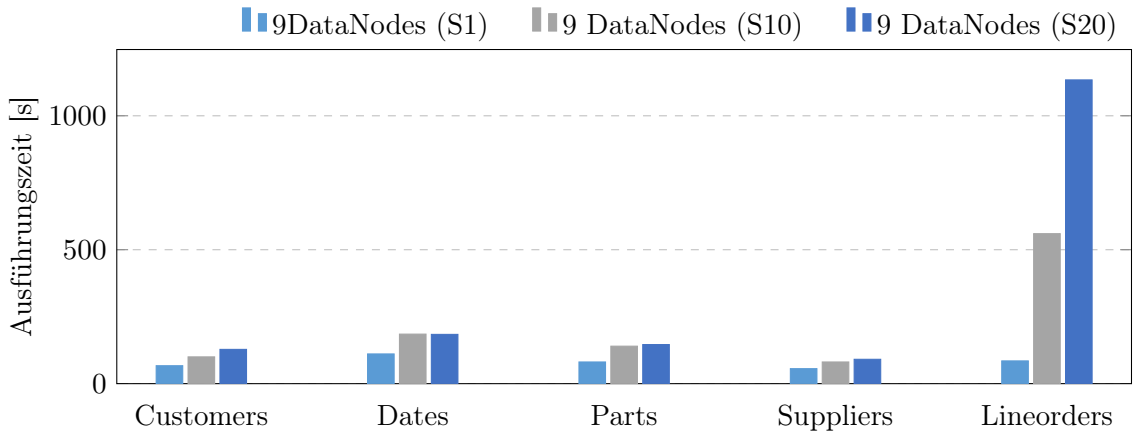

Abbildung 5.11.: Ausführungsdauer der dritten Komponente MDM-2-StarSchema bei S1, S10 und S20 sowie 9 DataNodes.

ist jedoch weit größer als das 10-fache. Begründet wird dieses Verhalten durch die Anzahl der Mappers und Reducers. Zudem werden die Mappers bei S1 schneller ausgeführt, da sie weniger Daten in den jeweiligen Partitionsordnern verarbeiten müssen.

Abbildung 5.11 zeigt die Ausführungsdauer der MDM-2-StarSchema-Komponete bei S1, S10 und S20 mit 9 DataNodes. Die Generierung des Sternschemas mit 9 DataNodes wurde bei der Datenmenge S1 in 400 Sekunden, bei S10 in 1066 Sekunden und bei S20 in 1683 Sekunden abgeschlossen. Erkennbar ist eine leichte Erhöhung der Ausführungsdauer bei der Generierung der Dimensionstabellen. Bei 9 DataNodes benötigt die Generierung der Faktentabelle bei doppelter Datenmenge wiederum doppelt so viel Zeit. Bei der Datenmenge S10 wurden 560 Sekunden und bei S20 1134 Sekunden gemessen.

5.4.5. Vergleich des ETL-Prozesses mit MySQL und Open Virtuoso

Zum Vergleich des hier vorgestellten ETL-Prozesses wird die relationale Datenbank MySQL (Version 5.6.28) sowie der RDF Store Open Virtuoso (Version 7.2.1) herangezogen. Die Experimente wurden auf einer AWS-Instanz vom Typ *m4.xlarge* durchgeführt.

Bei der Evaluation der MySQL-Datenbank wurden die Konfigurationswerte analog zu Listing 5.2 angepasst. Des Weiteren wurde der MySQL-Cache deaktiviert, um unabhängige Ergebnisse bei der Evaluation der Abfragen zu erreichen (s. Abschnitt 5.5.5). Bei der Konfiguration von Open Virtuoso wurde eine hohe Speicherplatzuweisung festgelegt. Die

verwendete Konfiguration ist in Listing 5.3 dargestellt.

```
1 key_buffer = 32M
2 max_allowed_packet = 16M
```

Listing 5.2: Die verwendete MySQL-Konfiguration bei der Evaluation.

```
1 MaxQueryMem = 8G
2 NumberOfBuffers = 1105000
3 MaxDirtyBuffers = 812500
```

Listing 5.3: Die bei der Evaluation verwendete Konfiguration von Open Virtuoso.

Abbildung 5.12 zeigt die gemessene Ausführungsdauer für den Import der unterschiedlichen Datenmengen S1, S10 und S20 in die MySQL-Datenbank sowie in den RDF Store Open Virtuoso. Zudem ist hierin die gesamte Ausführungsdauer der durchgeführten ETL-Durchgänge abgebildet.

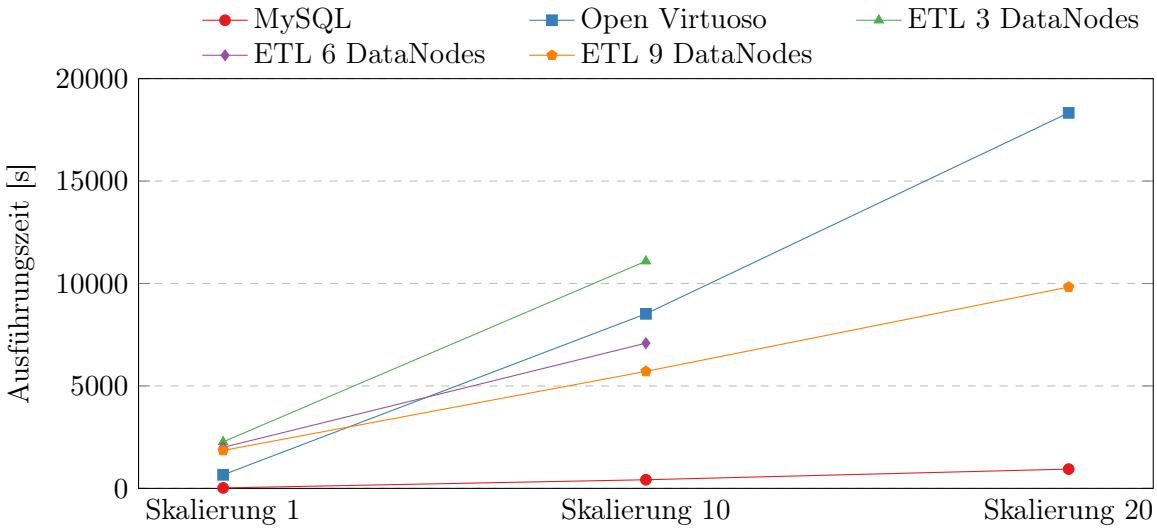

Abbildung 5.12.: Ausführungsdauer der dritten Komponente MDM-2-StarSchema bei S1, S10 und S20 sowie 9 DataNodes.

Es zeigt sich, dass der Import der TBL-Daten in die MySQL-Datenbank bei allen Skalierungen deutlich schneller durchgeführt wird. Außerdem benötigt der ETL-Prozess mit 3 DataNodes und der Datenmenge S1 und S10 länger als der Bulk-Import der gleichen

Datenmenge in den RDF Store Open Virtuoso.

Zwar benötigt der ETL-Prozess bei der Datenmenge S1 mehr Zeit, jedoch ändert sich dies bereits bei der Datenmenge S10. Hier wird eine schnellere Ausführung durch die horizontale Skalierung erzielt.

Zusammenfassend bringt der hier vorgestellte ETL-Prozess bei einer kleinen Datenmenge keinen Vorteil gegenüber nicht horizontal skalierenden Systemen. Jedoch hat die horizontale Skalierung bei größeren Datenmengen einen Vorteil gegenüber den in dieser Arbeit verwendeten RDF Store Open Virtuoso.

5.5. Ausführung analytischer Abfragen

In diesem Abschnitt werden die Antwortzeiten der analyitschen Abfragen des Star Schema Benchmarks (SSB) evaluiert. Der Benchmark stellt 13 verschiedene SQL-Abfragen Q1 - Q13 zur Evaluation bereit. In der vorangegangenen Arbeit von Kämpgen und Harth wurden äquivalente SPARQL- und MDX-Abfragen definiert (vgl. [KH13] und die dazugehörige Projektseite[6]). Diese Abfragen werden in den Folgenden Abschnitten auf ihre Antwortzeiten bei unterschiedlichen Clustergrößen hin untersucht.

Bei der Evaluation der analytischen Abfragen wurde zunächst der Mondrian-Cache deaktiviert und zwei Warm-up-Durchgänge durchgeführt. Anschließend wurden die Abfragen Q1 - Q13 mit unterschiedlicher Anzahl an DataNodes jeweils drei Mal ausgeführt.

5.5.1. Evaluation der analytischen Abfragen mit S1

Abbildung 5.13 zeigt die Antwortzeiten der analytischen MDX-Abfragen in Millisekunden bei der Datenmenge S1 und einer Clustergröße von 3, 6 und 9 DataNodes. Analog führt Abbildung 5.14 die Ausführungszeiten der äquivalenten SQL-Abfragen auf.

Zu erkennen ist, dass die horizontale Skalierung bei der Datenmenge S1 die Antwortzeiten nur weniger MDX-Abfragen beeinflusst. Bei den äquivalenten SQL-Abfragen ist die horizontale Skalierung nicht messbar. Dies wird durch die Größe des OLAP Cubes in Kylin begründet. Bei der Datenmenge S1 wird ein OLAP Cube mit der Größe von 4,34 GB erstellt. Eine HBase-Region in Kylin hat jedoch die Größe von 10 GB. Infolgedessen wird

[6] s. Projektseite unter `http://people.aifb.kit.edu/bka/ssb-benchmark/`.

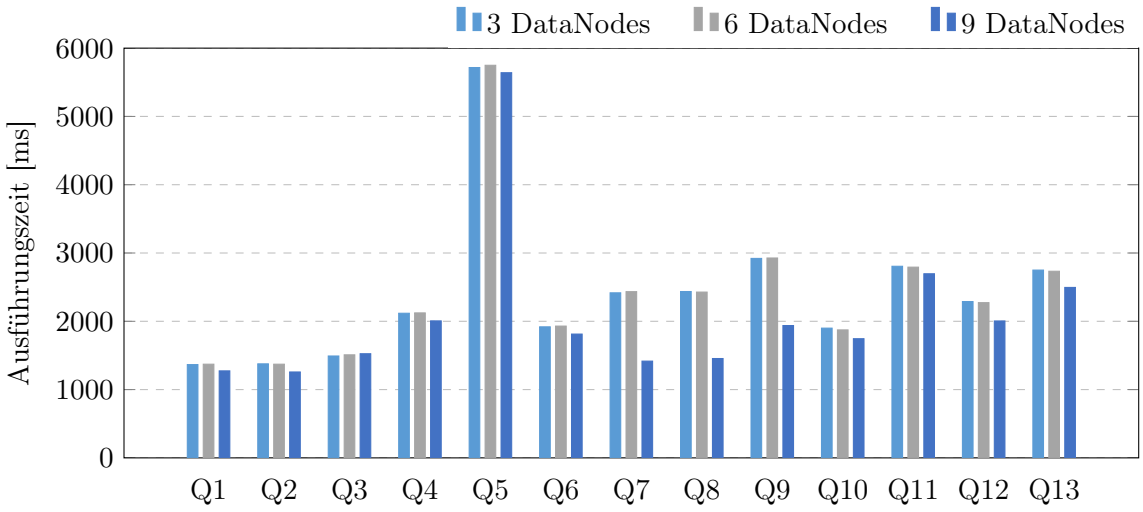

Abbildung 5.13.: Ausführungsdauer der analytischen MDX-Abfragen bei der Datenmenge S1.

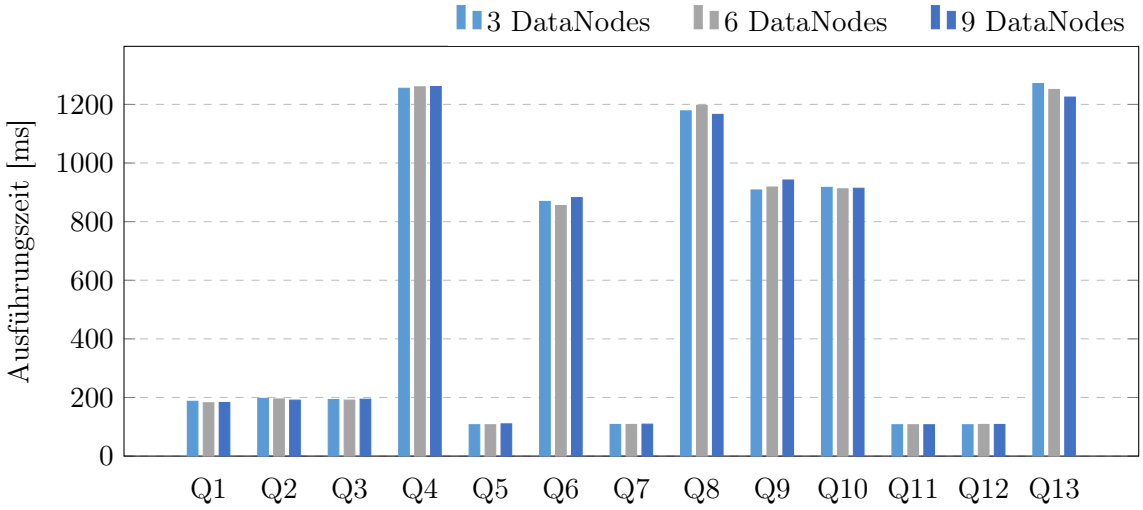

Abbildung 5.14.: Ausführungsdauer der analytischen SQL-Abfragen bei der Datenmenge S1.

bei dieser Datenmenge der OLAP Cube nur auf einem RegionServer gespeichert. Bei einer Abfrage muss folglich die gesamte Datenmenge aus dem RegionServer gelesen werden. Eine horizontale Skalierung hat aus diesem Grund keine Auswirkung.

Auffällig sind die Unterschiede in der Ausführungsdauer der MDX- und SQL-Abfragen. Bei Kylin liefern 10 der 13 SQL-Abfragen in weniger als einer Sekunde ein Ergebnis zurück.

Im Gegensatz dazu wird bereits bei der Datenmenge S1 keine MDX-Abfrage unter einer Sekunde beantwortet. Dieses Verhalten wird wiefolgt erklärt:

- Generell ist die Transformation einer MDX-Abfrage in ein SQL-Statement in Mondrian mit einer zeitlichen Latenz verbunden.
- In der Regel wird zur Beantwortung einer MDX-Abfrage das Statement in mehrere SQL-Abfragen umgewandelt. Die generierten SQL-Abfragen werden an Kylin gesendet und nacheinander ausgeführt. Dies führt letztendlich zu einer deutlich längeren Antwortzeit einer einzelnen MDX-Abfrage im Vergleich zu den äquivalenten SQL-Statements.

5.5.2. Evaluation der analytischen Abfragen mit S10

Abbildung 5.15 stellt die Antwortzeiten der analytischen MDX-Abfragen in Millisekunden bei der Datenmenge S10 und einer Clustergröße mit 3, 6 und 9 DataNodes dar. Analog führt Abbildung 5.16 die Ausführungszeiten der äquivalenten SQL-Abfragen auf.

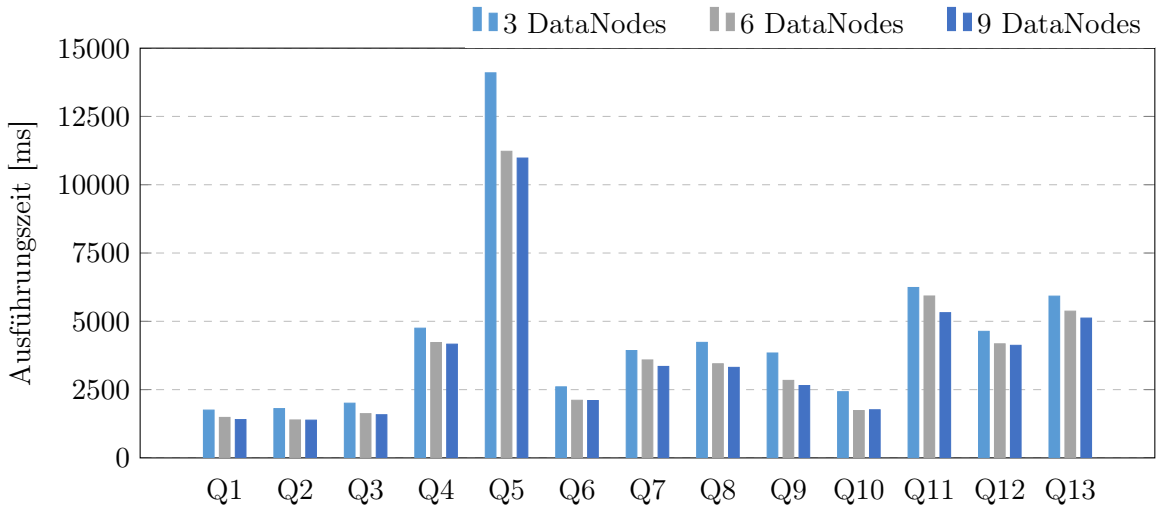

Abbildung 5.15.: Ausführungsdauer der analytischen MDX-Abfragen bei der Datenmenge S10.

Ähnlich zur Datenmenge S1 benötigt die MDX-Abfrage Q5 am Längsten. Zudem hat die horizontale Skalierung bei fast allen Abfragen einen messbaren Effekt auf die Ausführungszeiten zur Folge, z. B. liefert die Abfrage Q5 mit 3 DataNodes in 14,1 Sekunden, bei 6

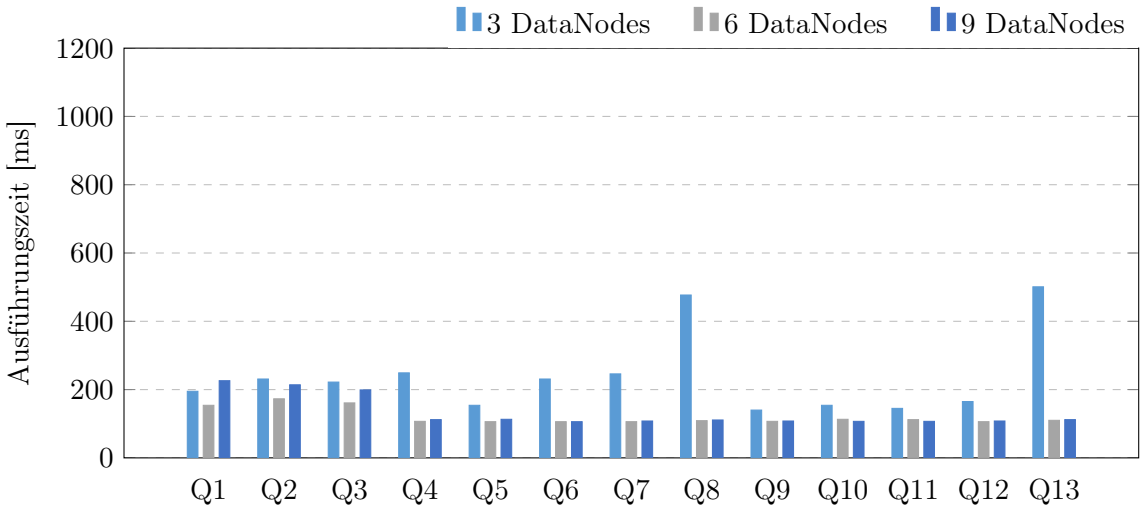

Abbildung 5.16.: Ausführungsdauer der analytischen SQL-Abfragen bei der Datenmenge S10.

DataNodes in 11,2 Sekunden und bei 9 DataNodes in 10,9 Sekunden ein Ergebnis zurück.

Die horizontale Skalierung führt auch bei den äquivalenten SQL-Abfragen zu einer Reduzierung der Ausführungszeiten. Bei der Evaluation wurden die Abfragen Q1, Q2 und Q3 mit 9 DataNodes zwar etwas langsamer ausgeführt, jedoch wird dies durch die generell sehr kurzen Antwortzeiten dieser Abfragen begründet. Sie unterscheiden sich um 30 Millisekunden.

Bei den SQL-Abfragen Q4, Q6, Q7, Q8 und Q13 liefert das Cluster mit 6 DataNodes in einer kürzeren Zeitspanne das Ergebnis zurück. Der OLAP Cube in Kylin hat bei der Datenmenge S10 eine Größe von 39,9 GB. Dadurch werden vier Regionen in HBase benötigt und über das Cluster horizontal verteilt. Bei 6 DataNodes können daher alle Regionen unabhängig voneinander auf verschiedenen DataNodes geladen werden.

Weiter fällt auf, dass alle Antwortzeiten der SQL-Abfragen bei der Datenmenge S10 unter 0,6 Sekunden liegen. Dieses Verhalten wird durch die Anzahl der Regionen in HBase begründet. Bei der Datenmenge S1 wurde für den OLAP Cube nur eine Region erstellt. Dies führt dazu, dass die gesamte Datenmenge, die sortiert in der Region gespeichert wurde, geladen werden muss. Eine parallele Ausführung findet in diesem Fall nicht statt. Bei der Datenmenge S10 werden vier Regionen für die Speicherung des OLAP Cubes benötigt. Während der Abfragen werden hierbei die Daten parallel verarbeitet.

5.5.3. Evaluation der analytischen Abfragen mit S20

In diesem Abschnitt werden die 13 MDX- und SQL-Abfragen bei der Datenmenge S20 und 9 DataNodes evaluiert. Abbildung 5.17 führt die gemessenen Antwortzeiten der MDX-Abfragen im Vergleich zu den Datenmengen S1 und S10 auf. Analog werden in Abbildung 5.18 die Ausführungszeiten der SQL-Abfragen aufgelistet.

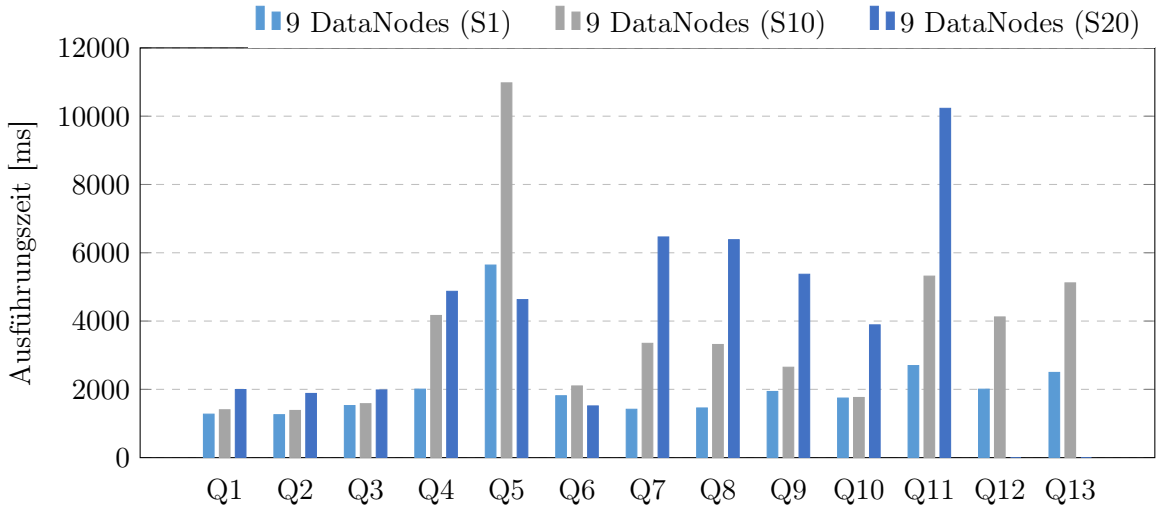

Abbildung 5.17.: Ausführungsdauer der analytischen MDX-Abfragen bei 9 DataNodes sowie Datenmenge S1, S10 und S20.

Für die Evalaution wurde im Vorfeld erwartet, dass die Ausführungszeiten der Abfragen bei gleichbleibender Anzahl der DataNodes, jedoch größer werdender Datenmenge zunehmen werden. Dieses Verhalten trat bei allen MDX-Abfragen bis auf Q5 und Q6 ein.

Des Weiteren wurden die Abfragen Q12 und Q13 nicht erfolgreich ausgeführt. Apache Kylin bricht nach 600 Sekunden die Ausführung der Abfrage ab. Dieser Schwellwert konnte während der Evaluation nicht erhöht werden. Zudem zeigte die Analyse der Fehlermeldung, dass Mondrian die Abfragen Q12 und Q13 in SQL-Abfragen umwandelt, die in der WHERE-Bedingung eine IN-Anweisung enthalten. Eigene Recherchen hierzu haben ergeben, dass Kylin zum Zeitpunkt der Evaluation solche SQL-Abfragen nicht effizient ausführen kann.

Werden die äquivalenten SQL-Abfragen in Abbildung 5.18 betrachtet, wird deutlich, dass die SQL-Abfragen schneller als die MDX-Abfragen ausgeführt werden. Einzig die Abfragen Q4, Q8 und Q13 der Datenmenge S1 benötigen knapp über eine Sekunde. Durch

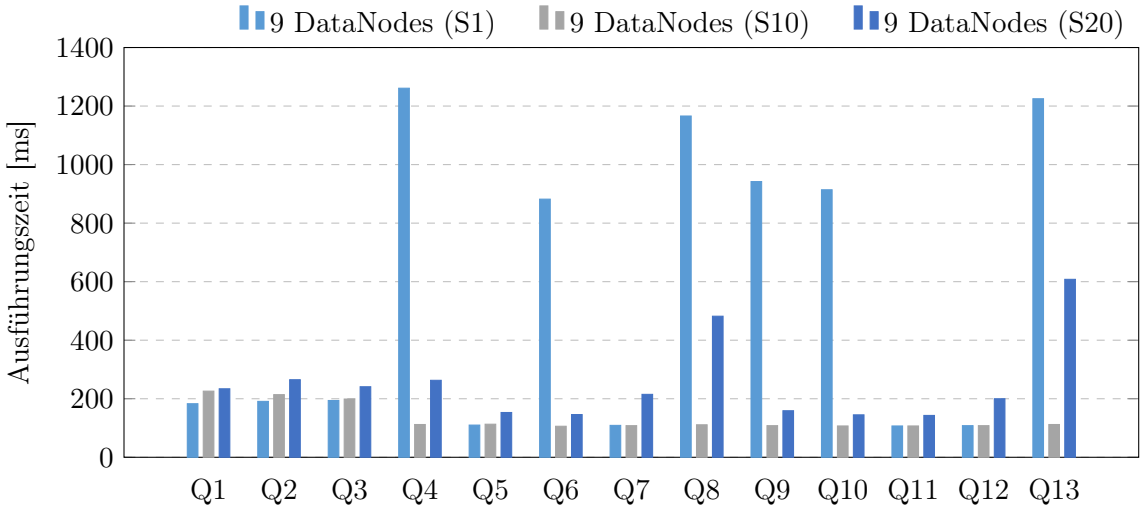

Abbildung 5.18.: Ausführungsdauer der analytischen SQL-Abfragen bei 9 DataNodes sowie Datenmenge S1, S10 und S20.

die zu geringe Datenmenge S1 wird der gesamte OLAP Cube (Größe: 4,34 GB) in einer HBase-Region gespeichert. Die horizontale Skalierung hat hierbei keinen Einfluss auf die Antwortzeiten. Bei der Datenmenge S10 beträgt die Größe des OLAP Cubes 39,9 GB (4 Regionen) und bei S20 bereits 79,4 GB (8 Regionen). In der Folge können durch das Hinzufügen von DataNodes die Antwortzeiten reduziert werden. Aus diesem Grund benötigten die SQL-Abfragen bei doppelter Datenmenge S20 nicht doppelt so lang in der Ausführung. Ausnahmen bilden hier die Abfragen Q8 und Q13, bei denen die Ausführungsdauer länger war. Die Ursache für dieses Verhalten konnte im Rahmen der Evaluation jedoch nicht ermittelt werden.

5.5.4. Evaluation mit aktiviertem Mondrian-Cache

In den bisherigen Messungen wurde der Mondrian-Cache bewusst deaktiviert, um die horizontale Skalierung von Apache Kylin untersuchen zu können. Vor der Evaluation mit aktiviertem Mondrian-Cache bestanden folgende Annahmen:

- Die Antwortzeiten der MDX-Abfragen sollten messbar kürzer sein als ohne Mondrian-Cache.
- Ab einer gewissen Datenmenge sollte der Mondrian-Cache - aufgrund der begrenzten Cache-Größe - die benötigten Daten in Kylin anfragen. Dessen ungeachtet sollten die

MDX-Abfragen bei größerer Datenmenge weiterhin schneller ausgeführt werden als ohne Mondrian-Cache.

Die Evaluation mit aktiviertem Mondrian-Cache führte zu folgendem Ergebnis:

- Nach einer ersten Warm-Up-Phase wurden die Daten zur Beantwortung der 13 MDX-Abfragen direkt aus dem Mondrian-Cache gelesen. Dabei wurden alle MDX-Abfragen in durchschnittlich 56ms beantwortet.
- Auch bei der Datenmenge S20 wurden die Daten aus dem Mondrian-Cache geladen. Im Rahmen der Evaluation konnte bei keiner Datenmenge die Begrenzung des Mondrian-Caches erreicht werden.

Infolgedessen führt der erste Aufruf einer MDX-Abfrage zu den in den vorherigen Abschnitten aufgelisteten Antwortzeiten. Eine erneute Ausführung der MDX-Abfrage führt dazu, dass das Ergebnis direkt aus dem Cache in wenigen Millisekunden geladen werden.

In der Praxis ist die Anzahl der Abfragen jedoch nicht auf 13 Stück begrenzt. Aus diesem Grund kann zwar der Mondrian-Cache einen Vorteil bieten, jedoch gilt es zu überprüfen, bei welcher Datenmenge und bei welcher Anzahl an Anfragen der Mondrian-Cache die benötigten Informationen aus Kylin nachladen muss.

5.5.5. Vergleich mit MySQL und Open Virtuoso

Analog zu Abschnitt 5.4.5 sollen die Antwortzeiten der analytischen Abfragen im Folgenden mit denen der relationalen Datenbank MySQL und dem RDF Store Open Virtuoso verglichen werden. Tabelle 5.7 listet die gemessenen Ausführungszeiten der 13 Abfragen bei den Datenmengen S1, S10 und S20 auf.

Wie erwartet, steigen die Antwortzeiten der SQL-Abfragen bei MySQL mit zunehmender Datenmenge an. Die SQL-Abfragen wurden erfolgreich bei allen Datenmengen ausgeführt. Bei S1 werden 81 Sekunden, bei S10 1516 Sekunden und bei S20 3273 Sekunden benötigt.

Unter Verwendung von Open Virtuoso konnte die SPARQL-Abfrage Q12 bereits bei der Datenmenge S1 nicht erfolgreich ausgeführt werden. Aus diesem Grund wurde diese Abfrage bei den weiteren Datenmengen S10 und S20 nicht mehr berücksichtigt. Des Weiteren fallen die unterschiedlichen Antwortzeiten auf. Obwohl die Evaluation mehrere Male durchgeführt wurde, unterschieden sich die Ausführungszeiten bei unterschiedlicher Datenmenge zum Teil sehr stark voneinander, z. B. wird die SPARQL-Abfrage Q5 bei S1 in 979 Sekun-

den ausgeführt, während sie bei der Datenmenge S10 nur 18 Sekunden und bei S20 nur 25 Sekunden benötigte. Ein ähnliches Verhalten wurde bei der Abfrage Q7 festgestellt. Bei der Datenmenge S1 wird diese Abfrage in 160 Sekunden, bei S10 in 7488 Sekunden und bei S20 in 354 Sekunden ausgeführt. Eine Begründung hierfür ist nicht ersichtlich.

Alles in allem können mit dem vorgestellten System sowohl bei MDX- als auch bei SQl-Abfragen deutlich kürzere Antwortzeiten erzielt werden als mit MySQL oder Open Virtuoso. Dieser Unterschied ist bei größeren Datenmengen am Signifikantesten.

5.6. Fazit der Evaluation

Die Evaluation hat gezeigt, dass die Ausführungszeit des ETL-Prozesses für kleine Datenmengen im Vergleich zum Bulk-Import in MySQL oder Open Virtuoso deutlich größer ist. Bei der größeren Datenmengen S10 und bereits mit 6 DataNodes hat der ETL-Prozess einen deutlichen Vorteil gegenüber dem RDF Store Open Virtuoso.

Des Weiteren werden die analytischen MDX- und SQL-Abfragen bei allen Datenmengen deutlich schneller beantwortet als in der relationalen Datenbank MySQL und dem RDF Store Open Virtuoso. Dies wird durch die horizontale Speicherung der Daten in HBase erziel, welche eine effizientere Ausführung der analytischen Abfragen ermöglichen.

Name	**Q1**	**Q2**	**Q3**	**Q4**	**Q5**	**Q6**	**Q7**	**Q8**	**Q9**	**Q10**	**Q11**	**Q12**	**Q13**	**Sum**
MySQL (S1)	1.2s	0.9s	0.9s	12.8s	12.5s	12.2s	8.4s	6.4s	6.2s	2.4s	9.2s	4.3s	4.1s	81.5s
Open Virtuoso (S1)	3.4s	0.2s	0.1s	22.5s	979.1s	1.8s	159.7s	3.4s	1.9s	1.8s	273.5s	N/A	49.6s	1497.0s
ETL MDX (S1)	1.3s	1.3s	1.5s	2.0s	5.6s	1.8s	1.4s	1.5s	1.9s	1.7s	2.7s	2.0s	2.5s	27,2s
ETL SQL (S1)	0.2s	0.2s	0.2s	1.3s	0.1s	0.9s	0.1s	1.1s	0.9s	0.9s	0.1s	0.1s	1.2	7,3s
MySQL (S10)	12.0s	8.6s	8.6s	158.0s	156.5s	155.1s	158.4s	134.8s	131.8s	131.6s	164.2s	161.1s	135.6s	1516.3s
Open Virtuoso (S10)	574.1s	1.9s	0.5s	159.2s	18.3s	4.1s	7488.1s	53.1s	63.7s	96.9s	1334.4s	N/A	18.4s	9812.7s
ETL MDX (S10)	1.4s	1.4s	1.6s	4.2s	11.0s	2.1s	3.3s	3.3s	2.6s	1.8s	5.3s	4.1s	5.1s	47,2s
ETL SQL (S10)	0.2s	0.2s	0.2s	0.1s	0.1s	0.1s	0.1s	0.1s	0.1s	0.1s	0.1s	0.1s	0.1s	1,6s
MySQL (S20)	25.5s	17.8s	17.8s	346.1s	340.6s	338.0s	346.4s	288.5s	278.8s	279.1s	352.5s	350.7s	291.2s	3273.0s
Open Virtuoso (S20)	1156.0s	3.8s	1.0s	128.2s	25.0s	9.0s	353.9s	32.5s	520.9s	373.8s	10015.0s	N/A	82.1s	12701,2s
ETL MDX (S20)	2.0s	1.9s	2.0s	4.9s	4.6s	1.5s	6.5s	6.4s	5.4s	3.9s	10.3s	N/A	N/A	49,4s
ETL SQL (S20)	0.2s	0.3s	0.2s	0.3s	0.2s	0.2s	0.2s	0.5s	0.2s	0.1s	0.1s	0.2s	0.6s	3,3s

Tabelle 5.7.: Vergleich der Antwortzeiten zwischen MySQL, Open Virtuoso und dem vorgestellten System mit 9 DataNodes.

6. Fazit und Ausblick

Im folgenden Kapitel werden die Ergebnisse dieser Arbeit zusammengefasst. Des Weiteren wird ein Ausblick auf weitere Entwicklungen gegeben und Herausforderungen an zukünftige Arbeiten dargestellt.

6.1. Zusammenfassung

Im Rahmen dieser Arbeit wurde ein umfangreicher ETL-Prozess implementiert, der automatisiert Statistical Linked Data im RDF Data Cube Vocabulary in eine horizontal skalierende OLAP Engine transformiert und für Analysen bereitstellt. Dabei wurden Open-Source-Technologien aus dem Big-Data-Umfeld eingesetzt. Neben der Ausführungsdauer des ETL-Prozesses wurden auch die Antwortzeiten analytischer MDX- und SQL-Abfragen evaluiert.

Die Evaluation ergab, dass die analytischen Abfragen bereits bei einer relativ kleinen Datenmenge in einer deutlich kürzen Zeit beantwortet werden als äquivalente Abfragen in MySQL und Open Virtuoso. In diesen Fällen ist der Vorteil durch die horizontale Speicherung der statistischen Daten in Kylin bemerkbar. Des Weiteren wurde im Rahmen der Evaluation festgestellt, dass der Einsatz von Mondrian zwar die Möglichkeit bietet, MDX-Abfragen in einer horizontal skalierbaren Umgebung auszuführen, jedoch die Ausführung aufgrund der zeitlichen Latenz während der Generierung der SQL-Abfragen nicht so effizient ist, wie äquivalente SQL-Abfragen.

Ferner wurde ermittelt, dass der hier vorgestellte ETL-Prozess erst ab einer größeren Datenmenge einen Vorteil gegenüber Import-Vorgängen bei nicht-horizontal skalierenden Systemen bietet. Der Einsatz von Apache Kylin ist mit dem Bereistellen der Fakten- und den Dimensionstabellen in Hive im Sternschema verbunden. Ein weiterer Grund ist der Overhead, der bei der Generierung der MapReduce-Jobs entsteht. Bei einer kleinen Da-

tenmenge hat die horizontale Skalierung keinen messbaren Effekt auf die Ausführung der MapReduce-Jobs.

Die im Abschnitt 1.2 vorgestellten Zielsetzungen haben nach der Evaluation Folgendes ergeben:

(V1) Die Dauer des ETL-Prozesses bei großen Datensätzen mit vielen Zusatzinformationen ist zufriedenstellend, da innerhalb der RDF-Daten die nötigen Informationen für das multidimensionale Datenmodell (Metadaten und Daten) effizient ausgelesen werden können.

(V2) Bei einer Aktualisierung des Datenbestands muss der ETL-Prozess auch in diesem System neu durchgeführt werden.

(V3) Bei der Hinzunahme neuer Daten muss der ETL-Prozess und die Generierung des OLAP Cubes lediglich für die neuen Daten durchgeführt werden. Kylin bietet durch den Einsatz einer Datum-Spalte die Möglichkeit, einen OLAP Cube mit neuen Daten aus einem definierbaren Zeitintervall zu generieren und mit einem bestehenden OLAP Cube zusammenzuführen.

(V4) Zusatzinformationen in den Datensätzen werden bei der Erstellung des multidimensionalen Datenmodells in diesem System zum Teil gefiltert. Zwar werden alle Attribute von konkreten Dimensionsinstanzen bei der Generierung des OLAP Cubes berücksichtigt, doch aufgrund des Sternschemas werden weiterführende Informationen nicht mit einbezogen.

Im nächsten Abschnitt wird ein Ausblick und weitere Ideen für zukünftige Arbeiten beschrieben.

6.2. Ausblick und weitere Ideen

Im Allgemeinen liegen die RDF-Daten nicht im benötigten N-Triples-Format vor. Jedoch konnte die Transformation der RDF-Daten im Rahmen der Abschlussarbeit nicht parallelisiert werden. Diese Herausforderung gilt es in einer zukünftigen Arbeit zu untersuchen.

Des Weiteren werden Verknüpfungen zu neuen Informationen aus verschiedenen Datenquellen in der vorgestellten Lösung nicht berücksichtigt. Weiterführende Analysen sollten die Möglichkeit untersuchen, die verlinkten Daten in einem Sternschema zusammenzuführen. Dies hätte einen enorm großen OLAP Cube zur Folge. Durch die horizontale Skalierung

kann Apache Kylin jedoch analytische Abfragen auf Grundlage einer beliebig großen Datenmenge interaktiv ausführen.

Eine weiterführende Ausarbeitung kann einen Vergleich zu horizontal skalierenden RDF Stores ziehen. Hierfür stellt Virtuoso ab der Enterprise Version 6 eine kommerzielle Clusterlösung bereit. Alternativ kann *4store*[1], ein horizontal skalierender RDF Store aus dem Open-Source-Bereich, zur Evaluation verwendet werden.

Die Optimierung der Ausführungsdauer sollte außerdem Gegenstand einer weiteren Arbeit sein. Hinsichtlich der MDM-Loader- und der MDM-2-Starschema-Komponente können neue Open-Source-Projekte, z. B. Apache Spark[2] oder Clouderas *Impala*[3], eine kürzere Ausführungsdauer des ETL-Prozesses erzielen. Zudem plant die Open Souce Community von Kylin den Einsatz von Apache Spark beim Cube-Build-Prozess[4]. Erste Ergebnisse haben gezeigt, dass die Generierung des OLAP Cubes mit Apache Spark den Cube-Build-Prozess erheblich verkürzen kann.

Die Antwortzeiten der analytischen MDX-Abfragen sind im Vergleich zu äquivalenten SQL-Abfragen deutlich höher. Wie bereits in der Evaluation festgestellt wurde, konnten ab einer gewissen Datenmenge einige wenige MDX-Abfragen nicht effizient durch Kylin beantwortet werden. Ausschlaggebend hierfür ist die MDX-zu-SQL-Transformation, denn Mondrian generiert SQL-Abfragen mit einer *IN*-Anweisung in der *WHERE*-Bedingung. Eigene Recherchen haben ergeben, dass Kylin zum Zeitpunkt der Evaluation solche SQL-Abfragen nicht effizient ausführen kann. Zukünftig sollte die Optimierung der MDX-zu-SQL-Transformation Gegenstand weiterer Untersuchungen sein. Hierzu würde eine neue Methode im *KylinDialect* genügen, die bei der Generierung der SQL-Abfrage anstelle einer *IN*-Anweisung eine *OR*-verknüpfte Bedingung erstellt.

Apache Kylin wird in der Version 2.0 die Möglichkeit bereitstellen, einen sogenannten *Hybrid OLAP Cube* zu definieren. Dabei können projektübergreifend analytische Abfragen auf Grundlage verschiedener OLAP Cubes definiert werden. Dies führt zur folgenden Überlegung: Für jeden statistischen Datensatz im QB-Vokabular, der nach dem Linked-Data-Prinzip veröffentlichte Daten enthält, kann jeweils ein OLAP Cube in Kylin erstellt werden. Der ETL-Prozess kann in diesem Fall pro Datensatz einzeln ausgeführt werden. Änderungen oder neue Verlinkungen im Datensatz führen dazu, dass der ETL-Prozess

[1] s. 4store-Webseite unter `http://4store.org/`.
[2] s. Apache Spark Webseite unter `http://spark.apache.org/`.
[3] s. Cloudera Impala Webseite unter `http://impala.io/`.
[4] s. Spark-Test unter `http://kylin.apache.org/blog/2015/09/09/fast-cubing-on-spark/`.

lediglich für diesen einzelnen Datensatz durchgeführt werden muss.

Der vorgestellte ETL-Prozess zeigt das Potential von Big-Data-Technologien. Durch die horizontal skalierende Architektur ist es möglich, eine enorm große RDF-Datenmenge in kurzer Zeit generisch in Kylin zu integrieren und interaktiv mit SQL- oder MDX-Abfragen zu analysieren. Einerseits stellt dieses System das Fundament bereit, MDX-Abfragen auf Grundlage von Apache Hadoop und Apache Kylin auszuführen. Andererseits können mit dem ETL-Prozess eine Vielzahl von unterschiedlichen Statistical Linked Data untersucht werden.

A. Anhang

Im Rahmen der Abschlussarbeit wurde bei Github eine Projektseite angelegt. Diese ist unter der URL `https://github.com/sjelsch/etl-evaluation` aufrufbar. Auf dieser Seite ist der Quellcode des ETL-Prozesses sowie eine ausführbare JAR-Datei mit Anleitung zu finden. Des Weiteren werden die Schritte für die Datengenerierung mit dem Star Schema Benchmark sowie die eingesetzte DataStructureDefinition-Datei auf der Seite beschrieben.

Neben einer Auflistung der verwendeten analytischen Abfragen beinhaltet diese Projektseite auch eine kurze Beschreibung der OLAP-Abfragen. Ferner sind die Ergebnisse der Evaluation für jeden einzelnen Durchgang aufgelistet.

A.1. KylinDialect in Mondrian

An dieser Stelle wird der entwickelte *KylinDialect* für die Interaktion von Mondrian mit Apache Kylin vorgestellt. Dieser Quellcode wurde im Rahmen der Abschlussarbeit im Mondrian-Projekt in einem Pull Request[1] bei Github festgehalten.

```
1  package mondrian.spi.impl;
2
3  import java.sql.Connection;
4  import java.sql.SQLException;
5
6  /**
7   * Implementation of {@link mondrian.spi.Dialect} for Kylin.
8   *
9   * @author Sébastien Jelsch
10  * @since Dez 28, 2015
11  */
```

[1] s. Pull Request unter `https://github.com/pentaho/mondrian/pull/480`.

```
12 public class KylinDialect extends JdbcDialectImpl {
13     public static final JdbcDialectFactory FACTORY =
14             new JdbcDialectFactory(KylinDialect.class, DatabaseProduct.KYLIN) {
15                 protected boolean acceptsConnection(Connection connection) {
16                     return super.acceptsConnection(connection);
17                 }
18             };
19
20     /**
21      * Creates a KylinDialect.
22      */
23     public KylinDialect(Connection connection) throws SQLException {
24         super(connection);
25     }
26
27     @Override
28     public boolean allowsCountDistinct() {
29         return false;
30     }
31
32     @Override
33     public boolean allowsJoinOn() {
34         return true;
35     }
36 }
```

Listing A.1: Java-Klasse *KylinDialect* in Mondrian.

A.2. RDF-Prefixe

In Tabelle A.1 werden alle in dieser Arbeit verwendeten Prefixe, die dazugehörigen URIs und die Namen der Vokabulare aufgelistet.

Prefix	**Namespace URI**	**Vocabulary**
qb4o	`http://purl.org/qb4olap/cubes#`	Vocabulary for Business Intelligence
qb	`http://purl.org/linked-data/cube#`	RDF Data Cube Vocabulary
rdf	`http://www.w3.org/1999/02/22-rdf-syntax-ns#`	RDF Core
rdfh-inst	`http://lod2.eu/schemas/rdfh-inst#`	BIBM RDF Vocabulary (Instances)
rdfh	`http://lod2.eu/schemas/rdfh#`	BIBM RDF Vocabulary (Schema)
rdfs	`http://www.w3.org/2000/01/rdf-schema#`	RDF Schema
skos	`http://www.w3.org/2004/02/skos/core#`	Simple Knowledge Organization System
skosclass	`http://ddialliance.org/ontologies/skosclass#`	SKOS extension for classifications
xkos	`http://purl.org/linked-data/xkos#`	SKOS Extension for Statistics
xsd	`http://www.w3.org/2001/XMLSchema#`	Schema Datatypes in RDF and OWL

Tabelle A.1.: Auflistung der verwendete Prefixe mit zugehörigen URIs und Vokabulare.

Literaturverzeichnis

[AFR11] Abelló, Alberto, Jaume Ferrarons und Oscar Romero: *Building Cubes with MapReduce*. In: *Proceedings of the ACM 14th international workshop on Data Warehousing and OLAP*, Seiten 17–24. ACM, 2011.

[AKB13] Arres, Billel, Nadia Kabbachi und Omar Boussaid: *Building OLAP cubes on a Cloud Computing environment with MapReduce*. In: *Computer Systems and Applications (AICCSA), 2013 ACS International Conference on*, Seiten 1–5. IEEE, 2013.

[BG13] Bauer, Andreas und Holger Günzel: *Data-Warehouse-Systeme: Architektur, Entwicklung, Anwendung*. dpunkt. verlag, 2013.

[BGH13] Back, D William, Nicholas Goodman und Julian Hyde: *Mondrian in Action: Open source business analytics*. Manning Publications Co., 2013.

[BHBL09] Bizer, Christian, Tom Heath und Tim Berners-Lee: *Linked Data - The Story So Far*. Semantic Services, Interoperability and Web Applications: Emerging Concepts, Seiten 205–227, 2009.

[BL06] Berners-Lee, Tim: *Linked Data: Design Issues*, 2006. aufgerufen am 25. Oktober 2015.

[BLFM04] Berners-Lee, Tim, Roy Fielding und Larry Masinter: *Uniform resource identifier (URI): Generic syntax*. Technischer Bericht, World Wide Web Consortium, 2004.

[BLHL+01] Berners-Lee, Tim, James Hendler, Ora Lassila et al.: *The semantic web*. Scientific american, 284(5):28–37, 2001.

[BPSM+98] Bray, Tim, Jean Paoli, C Michael Sperberg-McQueen, Eve Maler und François Yergeau: *Extensible markup language (XML)*. World Wide Web Consortium Recommendation REC-xml-19980210. `http://www.w3.`

org/TR/1998/REC-xml-19980210, 16, 1998.

[BWBN14] Bendler, Johannes, Sebastian Wagner, Dipl-Vw Tobias Brandt und Dirk Neumann: *Taming Uncertainty in Big Data.* Business & Information Systems Engineering, 6(5):279–288, 2014.

[CCS93] Codd, Edgar F, Sharon B Codd und Clynch T Salley: *Providing OLAP (on-line analytical processing) to user-analysts: An IT mandate.* Codd and Date, 32, 1993.

[CDG+06] Chang, Fay, Jeffrey Dean, Sanjay Ghemawat, Wilson C Hsieh, Deborah A Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes und Robert E Gruber: *Bigtable: A Distributed Storage System for Structured Data.* To appear in OSDI, Seite 1, 2006.

[CEMK+15] Chevalier, Max, Mohammed El Malki, Arlind Kopliku, Olivier Teste und Ronan Tournier: *Implementing Multidimensional Data Warehouses into NoSQL.* In: *17th International Conference on Entreprise Information Systems (ICEIS), Barcelona, Spain*, 2015.

[CMEF+13] Cudré-Mauroux, Philippe, Iliya Enchev, Sever Fundatureanu, Paul Groth, Albert Haque, Andreas Harth, Felix Leif Keppmann, Daniel Miranker, Juan F Sequeda und Marcin Wylot: *NoSQL databases for RDF: an empirical evaluation.* In: *The Semantic Web–ISWC 2013*, Seiten 310–325. Springer, 2013.

[DG04] Dean, Jeffrey und Sanjay Ghemawat: *MapReduce: Simplified Data Processing on Large Clusters.* OSDI '04: Sixth Symp. on Operating System Design and Implementation, Seiten 137–150, 2004.

[Dor15] Dorschel, Joachim: *Praxishandbuch Big Data: Wirtschaft–Recht–Technik.* Springer-Verlag, 2015.

[EV12] Etcheverry, Lorena und Alejandro A. Vaisman: *QB4OLAP: A Vocabulary for OLAP Cubes on the Semantic Web.* In: *Proceedings of the Third International Workshop on Consuming Linked Data, COLD 2012, Boston, MA, USA*, 2012.

[Fas14] Fasel, Daniel: *Big Data–Eine Einführung.* HMD Praxis der Wirtschaftsinformatik, 51(4):386–400, 2014.

[Geo11] George, Lars: *HBase: The Definitive Guide*. O'Reilly Media, Inc., 2011.

[GGD08] Gluchowski, Peter, Roland Gabriel und Carsten Dittmar: *Management Support Systeme und Business Intelligence*. Auflage, Berlin, 2008.

[GGL03] Ghemawat, Sanjay, Howard Gobioff und Shun-Tak Leung: *The Google file system*. In: *ACM SIGOPS operating systems review*, Band 37, Seiten 29–43. ACM, 2003.

[GK06] Gluchowski, Peter und Hans-Georg Kemper: *Quo vadis business intelligence*. BI-Spektrum, 1(1):12–19, 2006.

[HH02] Hannig, Uwe und Andreas Hahn: *Der Deutsche Markt für Data Warehousing und Business Intelligence*. In: *Knowledge Management und Business Intelligence*, Seiten 219–228. Springer, 2002.

[HHR+09] Hausenblas, Michael, Wolfgang Halb, Yves Raimond, Lee Feigenbaum und Danny Ayers: *Scovo: Using Statistics on the Web of Data*. In: *The Semantic Web: Research and Applications*, Seiten 708–722. Springer, 2009.

[HKJR10] Hunt, Patrick, Mahadev Konar, Flavio Paiva Junqueira und Benjamin Reed: *ZooKeeper: Wait-free Coordination for Internet-scale Systems*. In: *USENIX Annual Technical Conference*, Band 8, Seite 9, 2010.

[HKRS07] Hitzler, Pascal, Markus Krötzsch, Sebastian Rudolph und York Sure: *Semantic Web: Grundlagen*. Springer-Verlag, 2007.

[Inm05] Inmon, William H: *Building the Data Warehouse*. John Wiley & Sons, 2005.

[Int99] Intelligence, Business: *The IBM Solution, Datawarehousing and OLAP, Mark Whitehorn and Mary Whitehorn*, 1999.

[Kem11] Kempa, Martin: *Multidimensional Expressions (MDX)*. Datenbank-Spektrum, 11(2):123–126, 2011.

[KH11] Kämpgen, Benedikt und Andreas Harth: *Transforming Statistical Linked Data for Use in OLAP Systems*. In: *Proceedings of the 7th international conference on Semantic systems*, Seiten 33–40. ACM, 2011.

[KH13] Kämpgen, Benedikt und Andreas Harth: *No Size Fits All – Running*

the Star Schema Benchmark with SPARQL and RDF Aggregate Views. In: *The Semantic Web: Semantics and Big Data*, Seiten 290–304. Springer, 2013.

[KH14] King, Stefanie und Ivo Hajnal: *Big Data: Potential und Barrieren der Nutzung im Unternehmenskontext.* Springer-Verlag, 2014.

[KR13] Kimball, Ralph und Margy Ross: *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling.* John Wiley & Sons, 2013.

[KWZ98] Krahl, Daniela, Ulrich Windheuser und Friedrich-Karl Zick: *Data Mining: Einsatz in der Praxis.* Addison-Wesley, 1998.

[Lan01] Laney, Doug: *3D Data Management: Controlling Data Volume, Velocity and Variety.* META Group Research Note, 6:70, 2001.

[LS99] Lassila, Ora und Ralph R Swick: *Resource Description Framework (RDF): Model and Syntax Specification.* Recommendation, World Wide Web Consortium, 1999. Siehe `http://www.w3.org/TR/REC-rdf-syntax/`.

[MB00] Mucksch, Harry und Wolfgang Behme: *Das Data Warehouse-Konzept.* Architektur-Datenmodelle-Anwendungen, Wiesbaden, 2000.

[OOC09] O'Neil, Pat, E O'Neil und Xuedong Chen: *Star Schema Benchmark-Revision 3*, 2009.

[PAG06] Pérez, Jorge, Marcelo Arenas und Claudio Gutierrez: *Semantics and Complexity of SPARQL.* In: *International semantic web conference*, Band 4273, Seiten 30–43. Springer, 2006.

[PC95] Pendse, Nigel und Richard Creeth: *The OLAP report.* Business Intelligence, 1995.

[RSS15] Rahm, Erhard, Gunter Saake und KAIUWE SATTLER: *Verteiltes und Paralleles Datenmanagement.* Springer-Verlag, 2015.

[RW12] Redmond, Eric und Jim R Wilson: *Sieben Wochen, sieben Datenbanken: Moderne Datenbanken und die NoSQL-Bewegung.* O'Reilly Germany, 2012.

[SHWC05] Spofford, George, Sivakumar Harinath, Christopher Webb und Francesco Civardi: *MDX Solutions: With Microsoft SQL Server Analysis Services 2005 and Hyperion Essbase.* John Wiley & Sons, Inc., 2005.

[SRC10] Shafer, Jeffrey, Scott Rixner und Alan L Cox: *The Hadoop Distributed Filesystem: Balancing Portability and Performance.* In: *Performance Analysis of Systems & Software (ISPASS), 2010 IEEE International Symposium on*, Seiten 122–133. IEEE, 2010.

[SW02] Strauch, Bernhard und Robert Winter: *Vorgehensmodell für die Informationsbedarfsanalyse im Data Warehousing.* In: *Vom Data Warehouse zum Corporate Knowledge Center*, Seiten 359–378. Springer, 2002.

[Tot00] Totok, Andreas: *Modellierung von OLAP- und Data-Warehouse-Systemen.* Deutscher Universitäts-Verlag, 2000.

[TSJ+09] Thusoo, Ashish, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Suresh Anthony, Hao Liu, Pete Wyckoff und Raghotham Murthy: *Hive - A Warehousing Solution Over a Map-Reduce Framework.* Proceedings of the VLDB Endowment, 2(2):1626–1629, 2009.

[TSJ+10] Thusoo, Ashish, Joydeep Sen Sarma, Namit Jain, Zheng Shao, Prasad Chakka, Ning Zhang, Suresh Antony, Hao Liu und Raghotham Murthy: *Hive - A Petabyte Scale Data Warehouse Using Hadoop.* In: *Data Engineering (ICDE), 2010 IEEE 26th International Conference on*, Seiten 996–1005. IEEE, 2010.

[VLH+10] Vrandecic, Denny, Christoph Lange, Michael Hausenblas, Jie Bao und Li Ding: *Semantics of Governmental Statistics Data.* Proceedings of the WebSci10, 2010.

[WDS13] Weidner, Martin, Jonathan Dees und Peter Sanders: *Fast OLAP Query Execution in Main Memory on Large Data in a Cluster.* In: *Big Data, 2013 IEEE International Conference on*, Seiten 518–524. IEEE, 2013.

[WZP05] Whitehorn, Mark, Robert Zare und Mosha Pasumansky: *Fast track to MDX.* Springer Science & Business Media, 2005.

[ZPdMS+13] Zancanaro, Airton, LD Pizzol, Rafael de Moura Speroni, José Leomar Todesco und Fernando O Gauthier: *Publishing Multidimensional Statistical Linked Data.* In: *Proceedings of the Fifth International Conference on Information, Process, and Knowledge Management*, Seiten 290–304, 2013.

[ZW13] Zhang, Y.S. und S. Wang: *OLAP Query Processing Method Oriented to Database and HADOOP Hybrid Platform*, Oktober 24 2013. US Patent App. 13/514,296.