Friedrich-Alexander-Universität Erlangen-Nürnberg
Technische Fakultät

Lehrstuhl für Informatik 6 – Datenmanagement

Masterarbeit

Wikipedia in times of BigData – Einsatz von NoSQL-Systemen zur Verwaltung und Analyse von Wikipediainhalten

von

Patrick Kaltenmaier

15.07.2016

Erstprüfer
Prof. Dr. Klaus Meyer-Wegener

Zweitprüfer
Dipl.-Inf. Peter Schwab

Prüfungsrechtliche Erklärung Ich versichere, dass ich die Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Erlangen, 15.07.2016

Abstract

This thesis examines the power of modern NoSQL systems in conjunction with WOM to analyse and manage the article's data of wikipedia. The Wiki Object Model (WOM) is a maschine readable semi-structured representation of markup produced by different wiki engines. The systems studied are native XML databases (Sedna, BaseX, eXist-db, X-Hive, Oracle Berkley DB XML and Apache Xindice), XML enabled databases (PostgrSQL, MySQL, MariaDB), JSON based document stores (Couchbase Server, MongoDB), Wide Column Stores (Apache HBase) and multi model databases (ArangoDB, OrientDB). As a foundation for the further analysis of suitability the analytical database's expected data volumn, query profile and workload is investigated. Subsequently, the listed systems are evaluated regarding to their functional and non-functional suitability for the given use case. The functional part is derived by the system's capability of executing the given query profile – query driven schema design. The non-functional part is dictated by the amount of data which needs to be processed, and the expected workload. The list of potential system candidates is narrowed down to two whose ressource consumtion and funcitional suitability for the given usecase is described in further detail. The proposed research is the technical foundation for choosing and deploying the first prototype of an analytical database unlocking the structured data of wikipedia.

Zusammenfassung

Diese Arbeit untersucht den Leistungsumfang moderner NoSQL-Systeme zur Verwaltung und Analyse des Datenbestands der Wikipedia im WOM-Format. WOM steht für Wiki Object Model und ist eine maschinenlesbare, semi-strukturierte Repräsentation für Markup unterschiedlichster Wiki-Engines. Zu den untersuchten Systemen gehören native XML-Datenbanken (Sedna, BaseX, eXist-db, X-Hive, Oracle Berkley DB XML und Apache Xindice), XML-enabled-Databases (PostgrSQL, MySQL, MariaDB), JSON-basierte Document Stores (Couchbase Server, MongoDB), Wide Column Stores (Apache HBase), sowie zwei Multi-Model-Datenbanken (ArangoDB, OrientDB). Als Ausgangspunkt für die Systemanalyse werden die Anforderungen an die geforderte Analysedatenbank hinsichtlich erwartetem Datenvolumen, Queryprofil und Workload konkretisiert. Anschließend erfolgt die funktionale und nicht-funktionale Bewertung der gelisteten Systeme. Die funktionale Eignung wird direkt aus der Fähigkeit zur Verarbeitung der Queries abgeleitet – Query Driven Schema Design. Die nicht-funktionale Eignung ist durch das erwartete Datenvolumen und den Workload bestimmt. Die Liste der möglichen Kandidatensysteme wird dabei auf zwei eingegrenzt, deren Funktionsumfang und Ressourcenbedarf detailliert analysiert wird. Dies dient als Ausgangspunkt für die initiale Systemauswahl und Inbetriebnahme eines ersten Systemprototyps.

Inhaltsverzeichnis

1 Einleitung **1**

2 Verwandte Arbeiten **3**

3 Anforderungsanalyse **4**
3.1 Datenanalyse . . . 4
3.1.1 Format . . . 5
3.1.2 Volumen . . . 19
3.2 Queryanalyse . . . 23
3.3 Workload . . . 24
3.3.1 Abfragecharakteristik . . . 24
3.3.2 Änderungscharakteristik . . . 25

4 Datenbanksystemanalyse **26**
4.1 Native XML-Datenbanken . . . 26
4.1.1 Grundlegende Eigenschaften . . . 27
4.1.2 Funktionale Eignung . . . 27
4.1.3 Nicht-funktionale Eignung . . . 28
4.1.4 Fazit . . . 31
4.2 Relationale Datenbanksysteme mit XML Datentyp . . . 31
4.2.1 Grundlegende Eigenschaften . . . 32
4.2.2 Funktionale Eignung . . . 33
4.2.3 Nicht-funktionale Eignung . . . 34
4.2.4 Fazit . . . 34
4.3 JSON-basierte Document Stores . . . 35
4.3.1 Grundlegende Eigenschaften . . . 35
4.3.2 Funktionale Eignung . . . 36
4.3.3 Nicht-funktionale Eignung . . . 56
4.3.4 Fazit . . . 65
4.4 Multi-Model Datenbanken . . . 66
4.4.1 Grundlegende Eigenschaften . . . 67
4.4.2 Fazit . . . 69
4.5 Wide Column Stores in Kombination mit Map-Reduce . . . 70
4.5.1 Grundlegende Eigenschaften . . . 70
4.5.2 HBase – Schemaentwurf . . . 74
4.5.3 Funktionale Eignung . . . 77
4.5.4 Nicht-funktionale Eignung . . . 79
4.5.5 Fazit . . . 81

5 Zusammenfassung **82**

6 Ausblick **84**

Abbildungsverzeichnis 86

Auflistungsverzeichnis 86

Tabellenverzeichnis 87

Literatur 88

1 Einleitung

Motivation Die Wikipedia ist mit 40,1 Millionen Artikeln, verfasst in 254 Sprachen, die größte Online-Enzyklopädie der Welt[1]. Neben einer großen Leser- und Autorengemeinde hat sich ein weites Feld von Wissenschaftlern formiert, die Wikipedia-Inhalte als Datenbasis für ihre Forschungen einsetzen. Die Open Source Research Group der Friedrich-Alexander-Universität Erlangen-Nürnberg hat 2011, zur Unterstützung dieser Forschergemeinde den Sweble Wikitext Parser [DR11b], zusammen mit einem semi-strukturierten Datenmodell zur Repräsentation von Wiki-Artikeln, veröffentlicht. Das in diesem Kontext entwickelte Wiki Object Model (WOM) [DR11a] ermöglicht die einfache, maschinelle Verarbeitung von Wikipedia-Artikeln und dient als Basis für Inhaltsumformungen und Refactorings [DR13], sowie für das Auffinden feingranularer Änderungen zwischen verschiedenen Artikelversionen [DR14]. Das in dieser Technologie steckende Potential, nutzen jedoch nur wenige. Der Grund dafür liegt in der rechenaufwändigen Umwandlung des Wiki-Markups in WOM.

Forschungsfrage Es stellt sich daher die Frage: „Ist es möglich den Leistungsumfang moderner NoSQL-Systeme und WOM zu kombinieren und eine öffentlich oder im Rahmen des deutschen Forschungsnetz zugängliche Analysedatenbank zu betreiben, die inhaltsbezogene Suchen auf der WOM-Repräsentation des Datenbestands der Wikipedia ermöglicht?“

Ausgangssituation Die für wissenschaftliche Arbeiten relevanten Daten stecken dabei in den Artikeln, den Verweisen zwischen den Artikeln, sowie in ihrer Historie. Als Ausgangspunkt für die Arbeit wird eine Momentaufnahme des Datenbestands der Wikipedia[2] einmalig auf dem High-Performace-Computing-Cluster des Regionalen Rechenzentrums Erlangen in WOM umgewandelt. Das Einspielen neuer Artikelversionen erfolgt später inkrementell und kommt ohne den Einsatz von Spezialhardware aus. Das eingesetzte Datenbanksystem selbst muss ebenfalls auf Commodity Hardware betrieben werden können und ohne den Erwerb kommerzieller Lizenzen, von Personen innerhalb und außerhalb der FAU benutzt werden dürfen. Der zu verwaltende Datenbestand umfasst etwa 8 Milliarden Artikelversionen[3,4], das entspricht, ausgehend von einem durchschnittlichen Speicherbedarf von 42,69 KB[5] pro Artikelversion, einem Datenvolumen von rund 342 TB. Der erwartete Wert für die WOM-Repräsentation liegt noch einmal deutlich höher.

[1] https://stats.wikimedia.org/EN/TablesArticlesTotal.htm (besucht am 07.01.2016)
[2] https://dumps.wikimedia.org/backup-index.html (besucht am 08.07.2016)
[3] http://www.businessinsider.de/most-edited-wikipedia-2016-1?r=US&IR=T (besucht am 08.04.2016)
[4] https://en.wikipedia.org/wiki/Wikipedia:Size_of_Wikipedia (besucht am 08.04.2016)
[5] https://stats.wikimedia.org/EN/TablesArticlesBytesPerArticle.htm (besucht am 08.04.2016)

Zielsetzung Ziel dieser Arbeit ist es, den Grundstein für dieses ambitionierte Projekt zu legen und zu ermitteln, welche/s Datenbanksystem/e für den Aufbau einer solchen Analysedatenbank geeignet sind. Im Zentrum der Überlegungen steht die direkte Verwaltung und Analyse der semi-strukturierten WOM-Repräsentation der Artikel, sowie die Erschließung der enthaltenen Strukturinformation. Auf diese Weise grenzt sich die vorliegende Arbeit klar von einer direkten, textuellen Speicherung, sowie von den Forschungsgebieten des Text Minings und Natural Language Processings ab. Weiterhin beschränken sich die Hochrechnungen und Eignungsanalysen dieser Arbeit auf die Auswahl eines Datenbanksystems zur Verwaltung des Datenbestands der englischen Wikipedia. Die englische Wikipedia ist die größte und am häufigsten für wissenschaftliche Arbeiten verwendete Wikipedia-Installation. Ihr Datenvolumen umfasst etwa 13 Prozent[1] des Gesamtdatenvolumens aller Wikipedia-Installationen und ist, mit etwas über 800 Millionen Artikelversionen[3], ein wichtiger Meilenstein hin zur Verarbeitung der kompletten Wikipedia.

Methodik Die Menge potentieller Systemkandidaten für diese Aufgabe ist groß und beginnt bei einer Vielzahl nativer XML-Datenbanken (Sedna, BaseX, eXist-db, X-Hive, Oracle Berkley DB XML und Apache Xindice), gefolgt von XML-enabled-Databases (PostgrSQL, MySQL, MariaDB), über JSON-basierten Document Stores (Couchbase Server, MongoDB), Wide Column Stores (Apache HBase), bis hin zum Einsatz von Multi-Model-Datenbanken (ArangoDB, OrientDB). Um ein besseres Verständnis der zu verarbeitenden Daten zu erhalten wird zunächst eine umfassende Datenanalyse durchgeführt. Der erste Teil beschreibt die wichtigsten konzeptionellen Eigenschaften des WOM-Formats und hält bisher undokumentierte Änderungen zwischen der initialen WOM-Spezifikation (WOM 1.0) und der aktuellen WOM-Version (WOM 3.0) fest. Der zweite Teil der Datenanalyse beinhaltet eine Hochrechnung des zu erwartenden Datenvolumens. Dieser Aspekt hat einen besonders hohen Stellenwert, da viele Systeme, allein durch die schiere Masse der zu verarbeitenden Daten, ausscheiden. Zudem wird der Ressourcenverbrauch möglicher Kandidatsysteme auf Basis dieser Daten ermittelt. Im Anschluss an die Datenanalyse folgt die Queryanalyse. In diesem Abschnitt werden die Zugriffsmuster auf die Daten diskutiert, um daraus Datenbankschemata und funktionale Anforderungen an die Systeme abzuleiten – Query Driven Schema Design. Die letzte wichtige Kenngröße und abschließender Teil der Anforderungsanalyse ist der Workload. Er diktiert, zusammen mit dem Datenvolumen, die nicht-funktionale Eignung der Systeme. Im nächsten Schritt werden die grundlegenden Eigenschaften, sowie die funktionale- und nicht-funktionale Eignung aller zuvor aufgelisteten Datenbanksysteme diskutiert. Jeder Unterabschnitt endet mit einem Fazit, in dem die insgesamte Eignung der betrachteten Systeme bewertet wird. Vereinzelt kommt es vor, dass Systeme bereits nach der Diskussion einzelner Teilaspekte als ungeeignet eingestuft werden können. In solchen Fällen wird die Vierteilung in grundlegende Eigenschaften, funktionale Eignung, nicht-funktionale Eignung und Fazit aufgebrochen und direkt zum Fazit übergegangen. Bei Systemen, die sich während der Analyse als potentiell geeignet herausgestellt haben, fällt die Diskussion der funktionalen und nicht-funktionalen Eignung detaillierter aus. Zusätzlich wird in diesen Fällen im nicht-funktionalen Teil der Ressourcenbedarf für die Verwaltung der englischen Wikipedia berechnet und damit der

Ausgangspunkt für die Inbetriebnahme des ersten Systemprototypen geschaffen.

Aufbau Die Arbeit beginnt mit dem einleitenden Abschnitt *Verwandte Arbeiten* (2), gefolgt von dem ersten Research-Kapitel, *Anforderungsanalyse* (3), mit den Unterkapiteln *Datenanalyse* (3.1), *Queryanalyse* (3.2) und *Workload* (3.3). Die *Datenbanksystemanalyse* (4) untersucht die funktionale und nicht-funktionale Eignung von *Nativen XML-Datenbanken* (4.1), *Relationalen Datenbanksystemen mit XML-Datentyp* (4.2), *JSON-basierten Document Stores* (4.3), *Multi-Model Datenbanken* (4.4) sowie *Wide Column Stores* (4.5) und endet jeweils mit einem abschließenden Fazit zur Eignung der diskutierten Systeme. Der Schluss fasst alle gewonnen Erkenntnisse zusammen (5) und gibt einen *Ausblick* (6) auf bisher undokumentierte Erkenntnisse und Ideen.

2 Verwandte Arbeiten

Mit WikiBrain [Sen+14] und WikiParq [KN16] existieren im Wesentlichen zwei verwandte Arbeiten. WikiBrain ist ein Analyse-Framework für Wikipedia, das unter anderem Möglichkeiten zur Extraktion des reinen textuellen Inhalts von Artikeln, sowie der Ausführung von Volltextsuchen mit Apache Lucene[6] bereitstellt und den Zugang zu dem Link-, Redirect- und Kategorie-Graph der Wikipedia ermöglicht [Sen+14, S. 3]. Es ist möglich sprachenneutral auf Artikel zuzugreifen und beispielsweise ausgehend von dem Artikelidentifier des Artikels „Germany“ der englischen Wikipedia, Artikel mit dem Thema „Deutschland“ in anderssprachigen Wikipedia-Installationen zu finden [Sen+14, S. 4]. Weiterhin ist die Implementierung von sechs Semantic-Relatedness-Algorithmen enthalten und es wird die Abfrage von Daten aus Wikidata[7], sowie die Extraktion von geographischen Informationen, die mit den Artikel verknüpft sind, ermöglicht [Sen+14, S. 3].

Abgesehen von den *core data structures* [Sen+14, S. 3] stellt WikiBrain damit eher eine Sammlung von High-Level-Funktionen zur Analyse der Wikipedia, mit klarem Fokus auf Semantic-Relatedness und Volltextsuche, dar und ist nur schwer mit den Anwendungsmöglichkeiten einer zentralen Analysedatenbank zur Extraktion beliebiger Artikelinhalte vergleichbar. Zudem muss jeder WikiBrain-Anwender aufs Neue die von der Wikipedia bereitgestellten Artikel-Dumps[8] herunterladen und auf seiner verfügbaren Hardware einlesen. Die Verarbeitung der Daten ist dabei auf Multi-Core-Parallelität beschränkt [Sen+14, S. 4], wobei der hier vorgestellte Ansatz die Ausführung der Abfragen über die CPUs aller Maschinen im Analysecluster verteilt.

WikiPraq positioniert sich mehr im Bereich Natural-Language-Processing [KN16, S. 4147] und Concept-Mapping [KN16, S. 4147], ermöglicht dabei jedoch auch Low-Level-Funktionen, wie die Ausgabe des textuellen Inhalts bestimmter Artikel [KN16, S. 4145].

[6] https://lucene.apache.org/core/ (besucht am 13.07.2016)
[7] https://www.wikidata.org/wiki/Wikidata:Main_Page (besucht am 13.07.2016)
[8] https://dumps.wikimedia.org/backup-index.html (besucht am 13.07.2016)

WikiPraq wandelt die von der Wikipdia bereitgestellten Artikel-Dumps[8] dazu in ein spezielles, spaltenorientiertes Datenformat namens „Parquet“ [KN16, S. 4143f.] um. Dieses kann mit Hilfe von Apache Spark[9] [Zah+10] verteilt innerhalb von Amazon EC2[10], auf Basis von Hadoop YARN[11] oder innerhalb eines Apache Mesos Clusters[12] verarbeitet werden [KN16, S. 4145]. Die *Parquet*-Dateien verschiedener Wikiinstallationen sind dabei öffentlich unter http://semantica.cs.lth.se/wikiparq/ verfügbar [KN16, S. 4141].

WikiPraq ähnelt dem vorliegenden Ansatz durch die zentrale Bereitstellung der Datenbasis (*Parquet*-Dateien), sowie der stark verteilten Datenverarbeitung durch Apache Spark. Im Gegensatz zu WikiPraq konzentriert sich die vorliegende Arbeit nicht auf High-Level-Analysen, wie Natural-Language-Processing oder Concept-Mapping, sondern hat vielmehr den Aufbau einer Low-Level-Analysedatenbank zur Erschließung der Strukturinformation von Wikipediainhalten zum Ziel, die als Datenbasis für die Erstellung solcher Anwendungen dienen kann. Für das fertige System wird weiterhin die Bereitstellung eines möglichst aktuellen, im Idealfall sogar eines Live-Datenbestands, angestrebt (vgl. Kapitel 3.3) und nicht nur die Extraktion eines Schnappschusses der Daten.

3 Anforderungsanalyse

Die Anforderungsanalyse ist untergliedert in die Kapitel Datenanalyse (3.1), Queryanalyse (3.2) und Workload (3.3). Die Datenanalyse beschreibt die wichtigsten konzeptionellen Eigenschaften des WOM-Formats und hält bisher undokumentierte Änderungen zwischen der initialen WOM-Spezifikation (WOM 1.0) und der aktuellen WOM-Version (WOM 3.0) fest. Weiterhin ist eine Hochrechnung des erwarteten Datenvolumens für die Verwaltung der Artikelhistorie der englischen Wikipedia im WOM-Format enthalten. Die Queryanalyse fasst eine Reihe exemplarischer Abfragen zusammen und bildet die Basis der funktionalen Eignungsanalyse des nächsten Kapitels. Im Kapitel Workload wird die Abfrage- und Änderungscharakteristik der geplanten Anaylsedatenbank beschrieben. Der Workload bildet, zusammen mit der Hochrechnung des Datenvolumens, den Ausgangspunkt für die nicht-funktionale Bewertung der im weiteren Verlauf dieser Arbeit untersuchten Systeme.

3.1 Datenanalyse

Die Datenanalyse vermittelt einen Eindruck über das Format (Abschnitt 3.1.1) und Volumen (Abschnitt 3.1.2) der zu speichernden Daten.

[9] http://spark.apache.org/ (besucht am 13.06.2016)
[10] https://aws.amazon.com/ec2/ (besucht am 13.06.2016)
[11] http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/index.html (besucht am 13.06.2016)
[12] http://mesos.apache.org/ (besucht am 13.06.2016)

3.1.1 Format

WOM steht für Wiki Object Model und ist ein Datenformat, welches speziell für die Verwaltung von Informationen entwickelt wurde, die in Form von Markup unterschiedlicher Wiki-Engines kodiert sind. WOM 1.0 wurde erstmals im Juli 2011 von der Open Research Group der Friedrich Alexander Universität Erlangen-Nürnberg spezifiziert [DR11a]. Die aktuelle Version, WOM 3.0, ist im September 2014[13] erschienen.

Bei der Konvertierung eines Wikiartikels nach WOM werden systematisch alle, im Fließtext enthaltenen Markupelemente, in ein strukturiertes Datenformat umgewandelt. Der Fließtext selbst wird während des Umwandlungsvorgangs als Nutzdaten eingebettet. WOM stellt somit eine alternative Repräsentation des ursprünglichen Markups dar und verfolgt im Wesentlichen zwei Designziele [DR13, S. 1]:

1. Wiki-Entwickler sollen bei der Implementierung komplexer Funktionen, wie dem Bereitstellen visueller Editoren, der gezielten Abfrage von Inhalten, der Ausführung konsistenter Formatierungsänderungen, der Datenspeicherung und dem Austausch von Daten zwischen unterschiedlichen Wiki-Engines, unterstützt werden.

2. Die Pflege und Wartung großer Wiki-Installationen soll erleichtert werden. Dohrn et al. [DR13, S. 5ff.] beschreiben einen Katalog von Transformationen, die unter Zuhilfenahme von WOM, für die Anwendung konsistenter Markup-Umformungen innerhalb eines Artikels oder über Artikelgrenzen hinweg, verwendet werden können.

Um den Einstieg in WOM 3.0 zu erleichtern, werden kurz die wichtigsten Aspekte der WOM-1.0-Spezifikation zitiert. Die Beschreibung wird an einigen Stellen um die, in WOM 3.0, neu eingeführten Funktionen ergänzt. Diese sind speziell gekennzeichnet und waren notwendig, da es zum aktuellen Zeitpunkt (April 2016) keine offizielle Spezifikation der WOM 3.0 gibt. Die Angaben stammen aus [DR11a], [DR13] oder wurden im persönlichen Gespräch mit dem WOM-Entwickler, Hannes Dohrn, erarbeitet.

WOM 3.0 In WOM gespeicherte Dokumente bilden eine Baumstruktur, vergleichbar mit dem Document Object Model der HTML. Die vorhandenen Knoten orientieren sich direkt an den Syntaxelementen moderner Wiki-Engines [DR13, S. 3]. Auf diese Weise erklärt sich auch die starke Anlehnung an XHTML 1.0. Ein Teil der Knotenmenge wurde aus XHTML 1.0 übernommen[14], die restlichen Knoten sind WOM-spezifisch. Eine genaue Beschreibung aller neu eingeführten Knoten, sowie eine Auflistung aller aus XHTML 1.0 übernommenen Knoten, einschließlich ihres angepassten Datenmodells, liefert der Abschnitt Knotentypen. Das Datenmodell von WOM ist über Verschachtelungsregeln festgelegt. Sie definieren, welche Knotentypen zulässige Kindknoten eines bestimmten Elternk-

[13] http://sweble.org/2014/09/sweble-2-0-released/ (besucht am 26.04.2016).

[14] Übernommen wurden Namen, Semantik und Knotenattribute. Das Datenmodell und damit die Liste der zulässigen Kindkonten, wurde an die erweiterte Funktionalität von WOM angepasst (für weitere Details siehe Abschnitt 3.1.1).

notens sind. WOM bildet somit ein **semistrukturiertes, hierarchisches Datenmodell**, dessen Struktur sich direkt aus der Struktur der verarbeiteten Quelldokumente ableitet.

Syntax der Datenmodellbeschreibung Die Syntax für die Beschreibung des Datenmodells orientiert sich strikt an der Originalspezifikation (vgl. [DR11a, S. 13]).

$e*$	Kleene Operator: Null oder mehrmaliges Vorkommen von e.
$e+$	Ein oder mehrmaliges Vorkommen von e.
e_{opt}	Null oder einmaliges Vorkommen von e.
$e_1, e_2, \ldots e_n$	Sequenz: Zuerst e_1, gefolgt von e_2, bis hin zu e_n.
(e)	Gruppierung
$\{e_1, e_2, \ldots e_n\}$	Menge: Jeweils genau ein Vorkommen von $e_1, e_2, \ldots e_n$.
$[s]$	Menge: Genau ein Element aus der Menge s.
$s_1 \backslash s_2$	Mengendifferenz: Alle Elemente aus s_1, die nicht in s_2 enthalten sind.
<any>	Jedes beliebige Element.

Mixed Flag: Taucht das Mixed Flag in der Liste der zulässigen Kindelemente auf, können die aufgeführten Knoten zusätzlich mit Fließtext gemischt werden[15].

Knotentypen Die Menge aller WOM-Knoten setzt sich aus einem **Subset** der in XHTML 1.0 Transitional[16] definierten Knoten, vereinigt mit einer Menge in WOM neu eingeführter Knoten, zusammen. Abbildung 1[17] entstammt der WOM 1.0 Spezifikation. Die blaue Box listet alle durch WOM 1.0 neu eingeführten und damit in XHTML 1.0 nicht enthaltenen Knoten. Die rote Box enthält alle XHTML 1.0 Knoten, die **nicht** Teil des Wiki Object Models sind. Die Menge aller WOM 1.0 Knoten besteht somit aus den Elementen der blauen Box und allen XHTML 1.0 Knoten, die **nicht** in der roten Box aufgeführt sind. WOM 3.0 führt zusätzlich die Knoten *article*, *articles*, *for*, *repl*, *rtd*, *subst* und *text* ein.

Bei den aus **XHTML 1.0 übernommen Knoten** wurden sowohl die Namen, als auch die zulässigen Attribute, sowie die Semantik, übernommen, nicht jedoch ihre Verschachtelungsregeln. Hier definiert WOM eigene Regelsätze. Diese Regeln legen fest, welche Knotentypen gültige Kindknoten eines bestimmten Elternknotens sind und werden in der Originalspezifikation auch als *„Content Model“* oder gültiger Kontext bezeichnet. Eine Auflistung aller aus XHTML 1.0 übernommenen Knoten mit ihrem angepassten Datenmodell folgt direkt im Anschluss an diesen Absatz. Die Originalbeschreibung ist auf Seite 19 bis 23 der WOM 1.0 Spezifikation zu finden [DR11a, S. 19-23].

Die Knoten und ihre zulässigen Kindelemente werden mit folgender Kurzschreibweise beschrieben:

[15] Hinweis: Dieses Konstrukt wird ab WOM 3.0 vollständig durch den *text* Knoten ersetzt.
[16] https://www.w3.org/2002/08/xhtml/xhtml1-transitional.xsd (besucht am 16.03.2016).
[17] *XWML* steht für „XML Wiki Markup Language“ oder einfacher-, WOM gespeichert als XML.

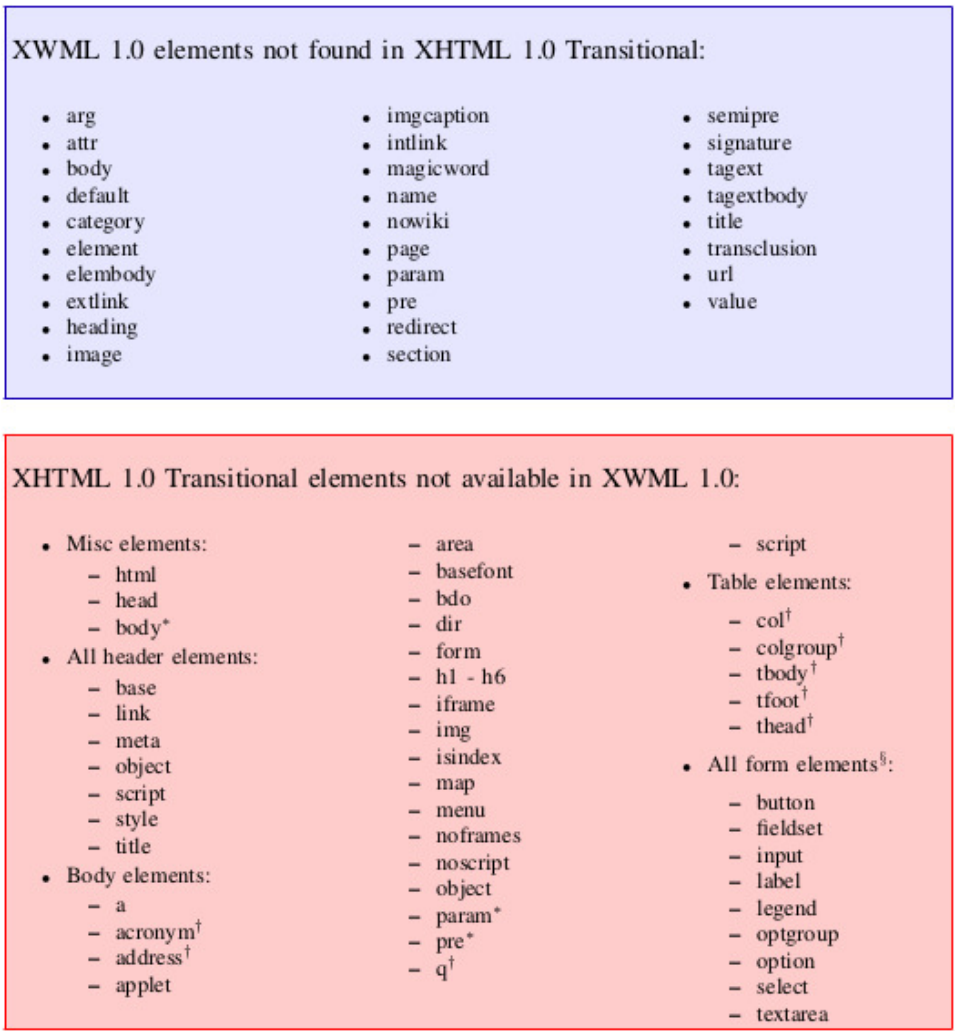


Abbildung 1: Blaue Box: Alle in WOM neu eingeführten Knoten. Rote Box: Alle XHTML 1.0 Knoten, die **nicht** teil von WOM sind [DR11a, S. 40].

.

<**Knotenname**> → <Kindelemente>

Die in eckigen Klammern stehenden Elemente referenzieren „Element Groups“. Diese fassen einzelne Knoten zu Knotenmengen zusammen und werden detailliert im Abschnitt Verschachtelungsregeln bzw. in Abbildung 2 beschrieben.

abbr → [inline elements]*

b → [inline elements]*

big → [inline elements]*

blockquote → [block elements]*

br → -keine-

capiton → [inline elements]*

center → [block elements]*

cite → [inline elements]*

code → [inline elements]*

dd → [block elements]*

del → [flow elements]*

dfn → [inline elements]*

div → [flow elements]*

dl → ([preprocessor elements] | dt | dd)*

dt → [inline elements]*

em → [inline elements]*

font → [inline elements]*

hr → -keine-

i → [inline elements]*

ins → [flow elements]*

kbd → [inline elements]*

li → [block elements]*

ol → ([preprocessor elements] | li)*

p → [inline elements]*

s → [inline elements]*

samp → [inline elements]*

small → [inline elements]*

span → [inline elements]*

strike → [inline elements]*

strong → [inline elements]*

sub → [inline elements]*

sup → [inline elements]*

table → ([preprocessor elements] | caption)$_{opt}$ ([preprocessor elements] | tbody)$_{opt}$

tbody → ([preprocessor elements] | tr)*

td → [block elements]*

th → [block elements]*

tr → ([preprocessor elements] | th | td)*

tt → [inline elements]*

u → [inline elements]*

ul → ([preprocessor elements] | li)*

var → [inline elements]*

Eine detaillierte Erklärung der Knotensemantik, sowie eine vollständige Attributliste aller Knoten, inklusive der Angabe des Wertebereichs aller Attribute, liefert die XHTML 1.0 Transitional[18].

Als Nächstes folgt eine Auflistung der von **WOM neu eingeführten Knoten**, inklusive Angabe der jeweiligen Knotenfunktionalität und der zulässigen Kindknoten. Alle in WOM 3.0 neu eingeführten Knoten erhalten die Suffix$_{v3}$. Bisher existiert ausschließlich eine Spezifikation zu WOM 1.0 [DR11a]. Die genaue Funktionsweise der nachträglich eingeführten Elemente wurde in einem persönlichen Gespräch mit dem WOM-Entwickler Hannes Dohrn erarbeitet. Die vollständige Auflistung aller Knotenattribute mit Datentyp und Wertebereich ist unter [DR11a, S. 23-30] zu finden. Alle neu eingeführten Datentypen werden unter [DR11a, S. 10-13] beschrieben. Die aus der XHTML 1.0 Transitional[18] übernommenen Datentypen sind unter [DR11a, S. 9] gelistet.

arg

- Beschreibung: Beinhaltet das Argument eines Seiten-Templates und wird als Teil der *transclusion* Anweisung angegeben.
- Kindelemente: name$_{opt}$ value

[18] https://www.w3.org/2002/08/xhtml/xhtml1-transitional.xsd (besucht am 16.03.2016).

article$_{v3}$

- Beschreibung: Das Wurzelelement eines Wikiartikels.
- Kindelemente: body

articles$_{v3}$

- Beschreibung: Erlaubt die Speicherung mehrerer Artikel innerhalb eines WOM-Dokuments.
- Kindelemente: article$_{v3}$+

attr

- Beschreibung: Beschreibt das Attribut einer tag extension (*tagext*) oder eines beliebigen anderen XML-Knotens. Name und Wert werden über die knoteneigenen Attribute *name* und *value* festgelegt.
- Kindelemente: -keine-

body

- Beschreibung: Allgemeiner Knoten zur Speicherung von Blockelementen innerhalb einer *page* oder *section*.
- Kindelemente: [block elements]*

category

- Beschreibung: Fügt eine *page* einer bestimmten Kategorie hinzu.
- Kindelemente: page

comment

- Beschreibung: Beinhaltet einen Kommentar.
- Kindelemente: *xhtml:Text*

default

- Beschreibung: Beinhaltet den Standardparameter einer template page. Wird bei Aufruf einer *transclusion* Anweisung kein Parameter angegeben, wird der mittels *default* spezifizierte Parameter übergeben.
- Kindelemente: (text$_{v3}$ | [preprocessor elements])*

element

- Beschreibung: Repräsentiert ein beliebiges XML-Element.
- Kindelemente: attr* elemboy$_{opt}$

elembody

- Beschreibung: Inhalt eines beliebigen, mittels *element* spezifiziertem XML-Element.
- Kindelemente: <any>*

extlink

- Beschreibung: Verweis auf eine externe URL.
- Kindelemente: title$_{opt}$

for$_{v3}$

- Beschreibung: Beinhaltet spezifische Syntaxelemente der Quellmarkups. Die zugehörigen WOM-Knoten werden in Form von WOM-Erweiterungen des jeweiligen Wikitext-Parsers spezifiziert und sind nicht Teil des WOM-Standards.
- Kindelemente: <any>

heading

- Beschreibung: Repräsentiert die Überschrift einer *section*. Bei Angabe der Überschriften mittels der entsprechenden HTML-Syntax (<h1> bis <h6>), werden alle HTML-Attribute als Attribute von *heading* übernommen. Die Ebene x der Überschrift (<hx> mit

$1 <= x <= 6$) wird in Form des *level* Attributs der umschließenden *section* gespeichert.

- Kindelemente: (text$_{v3}$ | [inline elements])*

image

- Beschreibung: Eine Abbildung. *Imgcaption* ist nur als Kindelement eingerahmter Abbildungen erlaubt (siehe [DR11a, 10 f.] für weitere Details.)
- Kindelemente: imgcaption$_{opt}$

imgcaption

- Beschreibung: Titel einer Abbildung.
- Kindelemente: (text$_{v3}$ | [inline elements])*

intlink

- Beschreibung: Verweis auf eine interne Wikiseite.
- Kindelemente: title$_{opt}$

magicword

- Beschreibung: Magicword zur Speicherung spezieller Steueranweisungen der exportierenden Wiki-Engine.
- Kindelemente: -keine-

name

- Beschreibung: *Name*-Elemente spezifizieren den Namen eines *arg*, *param* oder *transclusion* Knotens. Sie müssen nach Expansion einer Seitenvorlage, Zeichenketten mit validen Namensangaben enthalten. Welche Namen valide sind, ist bestimmt durch den umschließenden Kontext.
- Kindelemente: (text$_{v3}$ | [preprocessor elements])*

nowiki

- Beschreibung: Beinhaltet Text, der ohne weitere Interpretation, von der verarbeitenden Wiki-Engine angezeigt werden soll.
- Kindelemente: *xhtml:Text*

page

- Beschreibung: Das Wurzelelement einer Wikiseite.
- Kindelemente: redirect$_{opt}$ category* body

param

- Beschreibung: Parameter einer *transclusion* Anweisung.
- Kindelemente: name default$_{opt}$

pre

- Beschreibung: Bereits vorformatierter Text. Wird von der verarbeitenden Wiki-Engine nicht weiter interpretiert und mit einer fest vorgegebenen Schriftart angezeigt.
- Kindelemente: *xhtml:Text*

redirect

- Beschreibung: Zeigt an, dass es sich bei der angezeigten Seite um eine Weiterleitung auf eine andere Wikiseite handelt und gibt die Zielseite an.
- Kindelemente: -keine-

repl$_{v3}$

- Beschreibung: Beinhaltet die standardisierte WOM-Repräsentation enginespezifischer Markup-Elemente.
- Kindelemente: <any>

rtd$_{v3}$

- Beschreibung: Konserviert die genaue syntaktische Repräsentation eines Markup-Elements und ermöglicht somit die syntaktische Rekonstruktion des Quellmarkups.
- Kindelemente: *xhtml:Text*

section

- Beschreibung: Abschnitt einer Wikiseite mit Überschrift und Inhalt.
- Kindelemente: heading body

semipre

- Beschreibung: Bereits vorformatierter, geparster Inhalt, der mit einer fest vorgegebenen Schriftart angezeigt wird.
- Kindelemente: (text$_{v3}$ | [inline elements] \{image, big, small, sub, sup, front})*

signature

- Beschreibung: Die digitale Unterschrift eines Autors.
- Kindelemente: -keine-

subst$_{v3}$

- Beschreibung: Kombiniert ein enginespezifisches Syntax-Element mit seiner standardisierten WOM-Repräsentation, falls ein entsprechendes Konstrukt vorhanden ist.
- Kindelemente: repl$_{opt}$ for

tagext

- Beschreibung: Abbildung einer benutzerdefinierten Markup-Erweiterung (sogenannte tag extension).
- Kindelemente: attr* tagextbody$_{opt}$

tagextbody

- Beschreibung: Der textuelle Inhalt einer benutzerdefinierten Markup-Erweiterung (tag extension).
- Kindelemente: *xhtml:Text*

text$_{v3}$

- Beschreibung: Speichert den textuellen Inhalt eines Markup-Elements. Sämtlicher, semantisch bedeutsamer Fließtext eines Artikels wird in Form von Textknoten gespeichert. Auf diese Weise können nachträglich eingefügte Textformatierungen des WOM-Outputs von echten, inhaltstragenden Textbestandteilen des Quellartikels unterschieden werden.
- Kindelemente: *xhtml:Text*

title

- Beschreibung: Repräsentiert den Titel eines internen oder externen Links. Dieser Knoten darf nicht leer sein oder ausschließlich aus Leerzeichen bestehen.
- Kindelemente: (text$_{v3}$ | [inline elements] \ [inline link elements])*

transclusion

- Beschreibung: Fügt eine Seitenvorlage an der Stelle der *transclusion* Anweisung ein. Die Seitenvorlage wird vorverarbeitet und expandiert, bevor die *transclusion* Anweisung mit dem textuellen Inhalt der Seitenvorlage ersetzt wird.
- Kindelemente: name arg*

url

- Beschreibung: Abbildung einer URL. Die Zielressource wird mit Hilfe des *target* Attributs von *url* festgelegt.

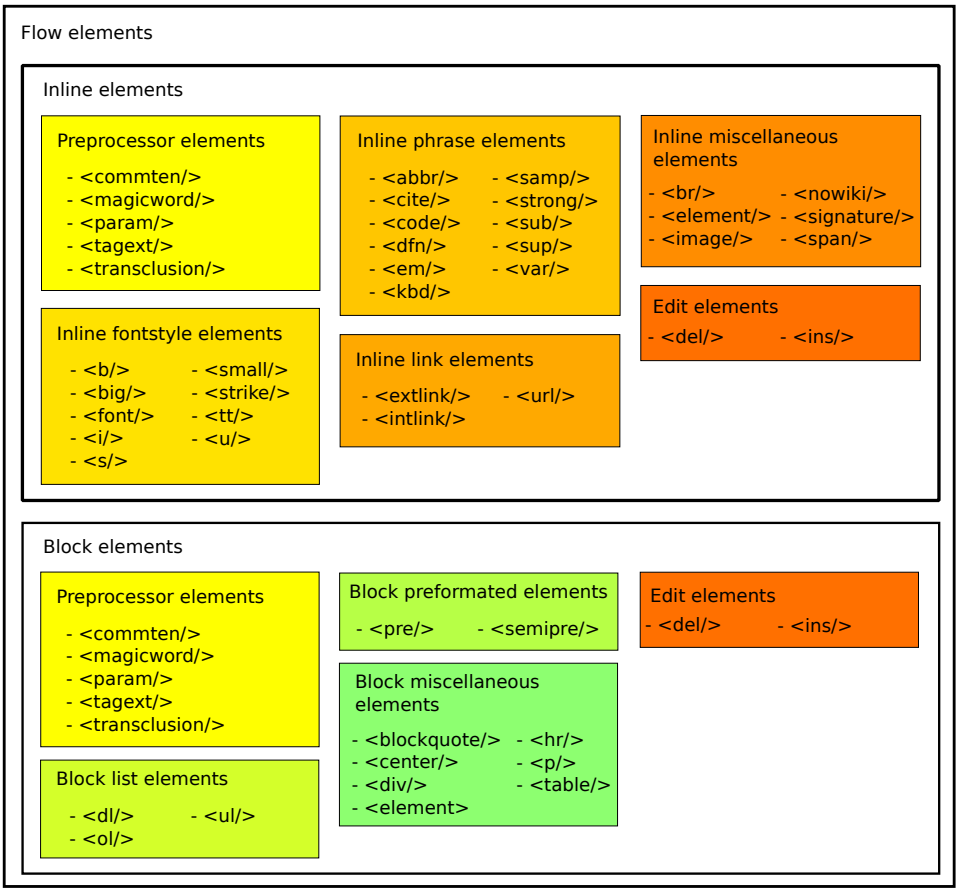


Abbildung 2: WOM-Element-Groups

- Kindelemente: -keine-

value

- Beschreibung: Beinhaltet den Wert eines *arg* Knotens.
- Kindelemente: ($text_{v3}$ | [preprocessor elements])*

Verschachtelungsregeln Die Verschachtelungsregeln legen fest, welche Knotentypen zulässige Kindknoten eines bestimmten Elternknotens sind und definieren auf diese weise das Datenmodell von WOM. Die grundsätzliche Organisationseinheit bilden dabei sogenannte ***Element Groups*** [DR11a, 15 ff.] und ***Element Content Groups*** [DR11a, 17 f.]. Sie fassen einzelne Knoten zu logischen Knotenmengen zusammen und erleichtern die Definition der zulässigen Eltern- Kindbeziehungen der unter Knotentypen gelisteten Knoten.

Die zulässigen Kindknoten des Subsets der aus XHTML 1.0 übernommenen Knoten sind über die ***Element Groups*** definiert (siehe Abbildung 2).

Eine spezielle Eigenheit von Wikitext und WOM sind die [preprozessor elements]. Sie sind

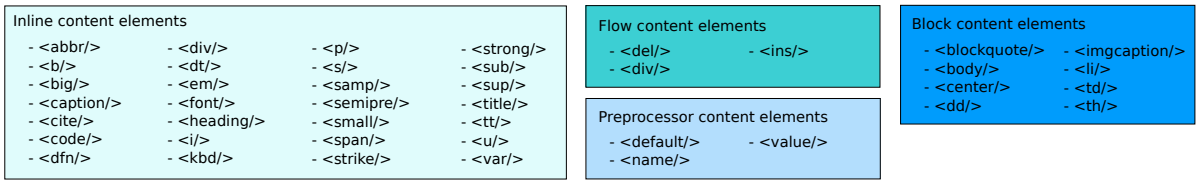


Abbildung 3: WOM-Element-Content-Groups

nicht Bestandteil von XHTML 1.0 und wurden explizit zu den [inline], [block] und damit zu der umschließenden Gruppe der [flow elements] hinzugefügt. Weiterhin wurden alle Knoten, die Kindknoten aufnehmen und keine der drei zuvor genannten Elementgruppen als direkte Kinder akzeptieren, um [preprocessor elements] erweitert (vgl. Abbildung 2 bzw. [DR11a, 15 ff.] und XHTML 1.0 Transitional[19]).

Die zulässigen Kindknoten der von WOM neu eingeführten Knoten werden mittels der ***Element Content Groups*** beschrieben (siehe Abbildung 3).

Neben der Redefinition der [inline], [block] und [flow elements] und dem expliziten Hinzufügen der [preprocessor elements] zu bestimmten Knoten verbietet WOM die Verwendung des *tr* Knotens als direktes Kind von Tabellen. Auf diese Weise wird die Mischung des table body und table row Knotens verhindert[DR11a, 18 f.]. Die Einschränkung, Text grundsätzlich in *p* Knoten einbetten zu müssen bzw. [block elements] an den Stellen zu verwenden, an den XHTML 1.0 Transitional [flow elements] erlaubt, wurde in WOM 3.0 aufgehoben.

Spezielles Zu den speziellen Eigenschaften von WOM gehört der Umgang mit Präprozessor-Elementen und die Rekonstruktion des ursprünglichen, vom User erzeugten, Wiki-Markups aus der WOM-Repräsentation.

Präprozessor-Elemente, namentlich *transclusions*, *tag extensions* und *parser functions*, können an nahezu jeder beliebigen Stelle in Wiki-Markup vorkommen. Durch Expansion dieser Konstrukte wäre es daher möglich, Wikitext zu erzeugen, der nach der Umwandlung in WOM zu einer syntaktisch inkorrekten WOM-Repräsentation führt. Präprozessor-Elemente (siehe Abbildung 2) dürfen daher ausschließlich zu gültigen Konten expandiert werden. Welche Knoten im Einzelnen gültig sind, ergibt sich aus dem jeweiligen Kontext des Präprozessor-Elements [DR11a, 13 f.].

Dazu folgendes Beispiel aus der WOM 1.0 Spezifikation:

```
1  <ul>
2    <li>Item 1</li>
3    <li>Item 2</li>
4    {{SomeTemplate}}
5  </ul>
```

[19] https://www.w3.org/2002/08/xhtml/xhtml1-transitional.xsd (besucht am 16.03.2016)

Das Wikitext-Fragment zeigt eine Aufzählung (unorderd list) mit zwei Aufzählungselementen (list items) und eine, innerhalb der Aufzählung eingebundene, Seitenvorlage mit Namen „SomeTemplate“.

WOM erlaubt innerhalb eines *ul* Knotens ausschließlich *li* Knoten oder [preprocessor elements] als direkte Kinder (siehe Abschnitt Knotentypen). Die Verfasser der Seitenvorlage „SomeTemplate“ müssen daher sicherstellen, dass ausschließlich *li* Knoten oder andere [preprocessor elements], für die die gleichen Einschränkungen gelten, auf oberster Ebene von „SomeTemplate“ auftauchen. Anderenfalls gäbe es keine gültige WOM-Repräsentation des erzeugten Wikitexts und die zugehörige Wikiseite könnte nicht verarbeitet werden.

WOM ist als generisches „Wiki Object Model“ konzipiert. Aus diesem Grund ist es nicht möglich, für jedes syntaktische Feature jeder Wiki-Engine ein entsprechendes WOM-Äquivalent anzubieten. Trotzdem darf bei der Umwandlung in WOM und einem anschließenden Export zurück in das Ursprungsformat keine Information verloren gehen. WOM löst dieses Problem durch die Definition des Substitution Tags, $subst_{v3}$, und der Einführung enginespezifischer Parsererweiterungen. Jeder Wikidialekt verfügt somit über einen eigenen WOM-Parser, mit für ihn spezifischen Erweiterungen. Trifft ein solcher WOM-Parser auf ein innerhalb einer Erweiterung spezifiziertes Konstrukt, erzeugt er einen *subst* und einen for_{v3} Knoten. Der for_{v3} Knoten ist ein direktes Kind von $subst_{v3}$ und enthält die XML-Repräsentation der eingelesenen Markup-Erweiterung. Auf diese Weise ist sichergestellt, dass die komplette Semantik des Quellmarkups bei einer Konvertierung nach WOM und anschließendem Export zurück in das Quellformat erhalten bleibt.

Sämtliche, in Form einer Parsererweiterung gespeicherte Information, kann ausschließlich von der erzeugenden Wiki-Engine interpretiert werden. Um für andere Engines, die kein direktes Syntax-Äquivalent dieser Konstrukte besitzen, so viel Information wie möglich zu konservieren, wurde das $repl_{v3}$ Tag eingeführt. Es beinhaltet eine alternative, zum WOM-Standard konforme, Darstellung der Inhalte einer Parsererweiterung. Der $repl_{v3}$ Knoten ist ein optionaler Bestandteil von $subst_{v3}$ und wird angelegt, wenn sich die Semantik einer Parsererweiterung, ganz oder in Teilen, mit Hilfe von Elementen aus dem WOM-Standard beschreiben lässt.

Neben der Definition spezifischer Markup-Elemente erlauben Wiki-Engines oftmals unterschiedliche Schreibweisen für ein semantisch gleiches Konstrukt. Um neben der semantischen Repräsentation der Daten auch die exakte Syntax rekonstruieren zu können, wurde das rtd_{v3} Tag eingeführt. Definiert die erzeugende Wiki-Engine verschiedene Schreibweisen eines Konstrukts, erhält der zugehörige WOM-Knoten ein oder mehrere rtd_{v3} Knoten als Kinder. Diese rtd_{v3} Knoten speichern jeweils exakt das an dieser Stelle verwendete Markup-Element. Auf diese Weise kann das Quellmarkup einer Wikiseite, auch nach der Umwandlung in WOM, vollständig wiederhergestellt werden.

Erweiterungen Für die direkte Verwaltung von WOM-Dokumenten mittels eines Document Stores muss sowohl die Artikelversion, als auch das Erstellungsdatum direkt in den

XML- und JSON-Repräsentationen der Artikel verankert werden. In WOM 3.0 werden diese Daten als Metadaten behandelt. Das spiegelt die Anwendungssicht wider, da das eingelesene Wiki-Markup diese Informationen nicht direkt enthält.

Aus Datenbanksicht sind die Artikelversion und das Erstellungsdatum Attribute eines Artikels. Genau genommen ist ein Markup-String erst durch seinen Titel (diese Information steckt direkt im Markup) und seine Version eindeutig identifizierbar. Aus konzeptioneller Sicht ist es daher notwendig, WOM 3.0 die Artikelversion (*@articleVersion*) und das Erstellungsdatum (*@timestamp*) als Attribute des *article* Knotens hinzuzufügen.

Um nicht für jede Wikipediainstanz (en, de, ru etc.) eine eigene *Collection* (einen eigenen *Bucket*) anlegen zu müssen, wird dem *article* Knoten weiterhin das *@lang* Attribut hinzugefügt. In *@lang* wird ein Kürzel für die Sprache der jeweiligen Artikelversion hinterlegt. Dies ist notwendig, da die Collection-Anzahl bei den meisten Document-Stores begrenzt ist und einige Systeme zur Verwendung einer relativ kleinen Anzahl an *Collections* pro Cluster (Zehn und weniger) raten. Dies trifft insbesondere für Couchbase zu und wird in Abschnitt 4.3.3 näher beschrieben.

Serialisierung WOM kann als XML oder JSON serialisiert werden. Ein WOM-Dokument, genauer eine XML- oder JSON-Datei, enthält dabei einen einzelnen Artikel oder eine Menge von Artikeln. WOM bildet diese Funktion über das *article* und *articles* Tag ab. Der *articles* Knoten ist optional und fasst mehrere *article* zu einer Artikelmenge zusammen. Je nachdem, welches der beiden Elemente in der Dokumentwurzel steht, beinhaltet die Datei einen einzelnen Artikel oder eine Sammlung von Artikeln.

WOM-JSON führt zusätzlich zwei spezielle Schlüssel-Präfix-Elemente ein. Ein „!“ am Anfang eines Schlüssels steht für eine Property, ein „@“ für ein Attribut. Die *type* Property gibt den Typ des gespeicherten WOM-Knotens an, z.B. „text“, für einen Text Knoten, *children* beinhaltet die Menge der zulässigen Kindknoten und *value* repräsentiert den direkten Wert eines Knotens, falls dieser keine weiteren Kindknoten akzeptiert. In JSON-Sprechweise besteht ein WOM-Dokument damit aus einer Hierarchie von JSON-Objekten. Jedes dieser JSON-Objekte verfügt über ein spezielles Schlüssel-Wert-Paar *„!type“:„Knotentyp“*, welches den Typ des jeweiligen JSON-Objekts angibt. Akzeptiert das JSON-Objekt gemäß dem WOM-Datenmodell Kindobjekte, verfügt es über den Schlüssel *„!children “*, welcher wiederum eine Menge weiterer JSON-Objekte als Wert aufnimmt. Erlaubt das WOM-Datenmodell keine weiteren Kindelemente, erhält es den Schlüssel *„!value“*, welcher den eigentlichen Inhalt des JSON-Objekts als Wert speichert. Besitzt ein WOM-Knoten Attribute, erhält das zugehörige JSON-Objekt pro Attribut ein Schlüssel-Wert-Paar der Form *„@AttributName“:„AttributWert“*. Eine solche XML- oder JSON-Datei wird als „WOM document“ bezeichnet, ein einzelner Artikel als „document fragment“.

Konsolidierendes Beispiel Abgerundet wird die Analyse des Datenformats mit der Betrachtung eines kurzen Beispieldatensatzes. Dazu wird zunächst das Rendering einer

kleinen Wikiseite, gefolgt von ihrem Markup und der zugehörigen WOM-XML und WOM-JSON-Repräsentation, abgedruckt.

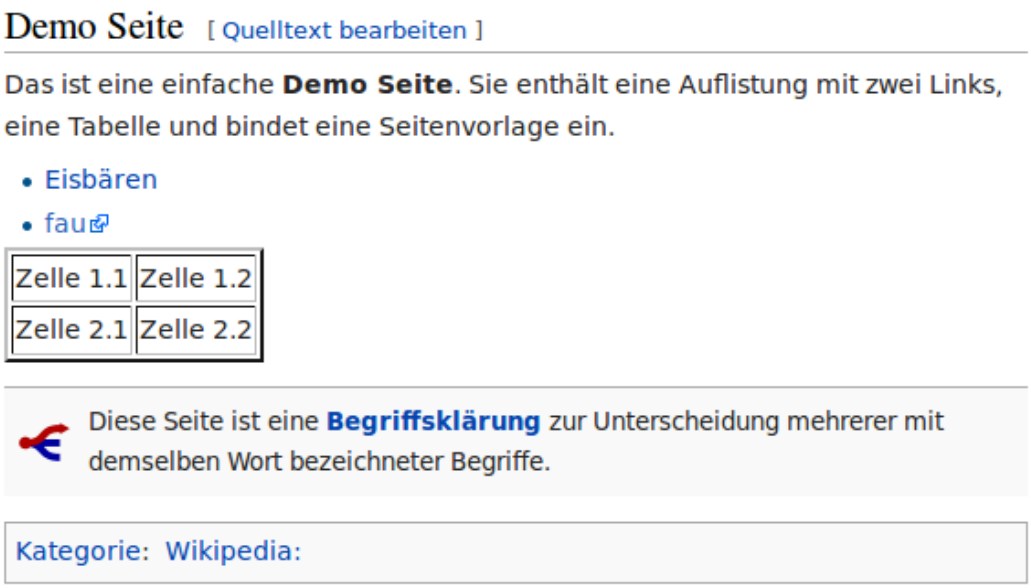


Abbildung 4: Demo Seite – HTML Rendering.

```
1  == Demo Seite ==
2
3  Das ist eine einfache '''Demo Seite'''. Sie enthält eine Auflistung mit zwei Links, eine
       ↪ Tabelle und bindet eine Seitenvorlage ein.
4
5  * [[Eisbär]]en
6  * [http://www.fau.de fau]
7
8  {| border="2"
9   |-
10  | Zelle 1.1 || Zelle 1.2
11  |-
12  | Zelle 2.1 || Zelle 2.2
13  |}
14
15 {{Begriffsklärung}}
```

Listing 1: Demo Seite – Wiki Markup.

```
1  <?xml version="1.1" encoding="UTF-8" standalone="no"?>
2  <article xmlns="http://sweble.org/schema/wom30" xmlns:mww="
       ↪ http://sweble.org/schema/mww30" version="3.0" title
       ↪ ="SimplePage" articleVersion="13" timestamp
       ↪ ="2016-06-17T14:19:47.688+02:00" lang="en">
3    <body>
4      <section level="2">
5        <heading>
6          <rtd>==</rtd>
7          <text> Demo Seite </text>
8          <rtd>==</rtd>
9        </heading>
10       <body>
11         <text>
12
13 </text>
14         <p topgap="0" bottomgap="0">
15           <text>Das ist eine einfache </text>
16           <b>
17             <rtd>'''</rtd>
18             <text>Demo Seite</text>
19             <rtd>'''</rtd>
20           </b>
21           <text>. Sie enthält eine Auflistung mit zwei Links,
                 ↪ eine Tabelle und bindet eine Seitenvorlage
                 ↪ ein.</text>
22         </p>
23         <text>
24
25 </text>
26         <ul>
27           <li>
28             <rtd>*</rtd>
29             <text> </text>
30             <subst>
31               <repl>
32                 <intlink target="Eisbär">
33                   <title>
```

```
                  <text>Eisbären</text>
                </title>
              </intlink>
            </repl>
            <for>
              <mww:intlink target="Eisbär" postfix="en">
                <rtd>[[Eisbär]]en</rtd>
              </mww:intlink>
            </for>
          </subst>
          <rtd>
</rtd>
        </li>
        <li>
          <rtd>*</rtd>
          <text> </text>
          <extlink target="http://www.fau.de">
            <rtd>[http://www.fau.de </rtd>
            <title>
              <text>fau</text>
            </title>
            <rtd>]</rtd>
          </extlink>
        </li>
      </ul>
      <text>

</text>
      <table border="2">
        <rtd>{| border="2"
</rtd>
        <tbody>
          <tr>
            <rtd> |-
</rtd>
            <td>
              <rtd> |</rtd>
              <text> Zelle 1.1 </text>
            </td>
            <td>
              <rtd>||</rtd>
              <p topgap="0" bottomgap="0">
                <text> Zelle 1.2 </text>
              </p>
              <text>
</text>
            </td>
          </tr>
          <tr>
            <rtd> |-
</rtd>
            <td>
              <rtd> |</rtd>
              <text> Zelle 2.1 </text>
            </td>
            <td>
              <rtd>||</rtd>
              <p topgap="0" bottomgap="0">
                <text> Zelle 2.2</text>
              </p>
              <text>
</text>
            </td>
          </tr>
        </tbody>
        <rtd> |}</rtd>
      </table>
      <text>

</text>
      <p topgap="0" bottomgap="0">
        <mww:transclusion>
          <rtd>{{</rtd>
          <mww:name>
            <text>Begriffsklärung</text>
          </mww:name>
          <rtd>}}</rtd>
        </mww:transclusion>
      </p>
      <text>
</text>
    </body>
  </section>
 </body>
</article>
```

Listing 2: Demo Seite – WOM-XML-Repräsentation.

```
{
  "!type": "article",
  "@xmlns": "http://sweble.org/schema/wom30",
  "@version": "3.0",
  "@title": "SimplePage",
  "@articleVersion": "13",
  "@timestamp": "2016-06-17T14:19:47.688+02:00",
  "@lang": "en",
  "@xmlns:mww": "http://sweble.org/schema/mww30",
  "!children": [
    {
      "!type": "body",
      "!children": [
        {
          "!type": "section",
          "@level": "2",
          "!children": [
            {
              "!type": "heading",
              "!children": [
                {
                  "!type": "rtd",
                  "!children": [
                    {
                      "!type": "#text",
                      "!value": "\u003d\u003d"
                    }
                  ]
                },
                {
                  "!type": "text",
                  "!children": [
                    {
                      "!type": "#text",
                      "!value": " Demo Seite "
                    }
                  ]
                },
                {
                  "!type": "rtd",
                  "!children": [
                    {
                      "!type": "#text",
                      "!value": "\u003d\u003d"
                    }
                  ]
                }
              ]
            },
            {
              "!type": "body",
              "!children": [
                {
                  "!type": "text",
                  "!children": [
                    {
                      "!type": "#text",
                      "!value": "\n\n"
                    }
                  ]
                },
                {
                  "!type": "p",
                  "@topgap": "0",
                  "@bottomgap": "0",
                  "!children": [
                    {
                      "!type": "text",
                      "!children": [
                        {
                          "!type": "#text",
                          "!value": "Das ist eine einfache "
                        }
                      ]
                    },
                    {
                      "!type": "b",
                      "!children": [
                        {
                          "!type": "rtd",
                          "!children": [
                            {
                              "!type": "#text",
                              "!value": "\u0027\u0027\u0027"
                            }
                          ]
                        },
                        {
                          "!type": "text",
                          "!children": [
```

```
                    {
                      "!type": "#text",
                      "!value": "Demo Seite"
                    }
                  ]
                },
                {
                  "!type": "rtd",
                  "!children": [
                    {
                      "!type": "#text",
                      "!value": "\u0027\u0027\u0027"
                    }
                  ]
                }
              ]
            },
            {
              "!type": "text",
              "!children": [
                {
                  "!type": "#text",
                  "!value": ". Sie enthält eine
                        ↪ Auflistung mit zwei Links,
                        ↪ eine Tabelle und bindet eine
                        ↪  Seitenvorlage ein."
                }
              ]
            }
          ]
        },
        {
          "!type": "text",
          "!children": [
            {
              "!type": "#text",
              "!value": "\n\n"
            }
          ]
        },
        {
          "!type": "ul",
          "!children": [
            {
              "!type": "li",
              "!children": [
                {
                  "!type": "rtd",
                  "!children": [
                    {
                      "!type": "#text",
                      "!value": "*"
                    }
                  ]
                },
                {
                  "!type": "text",
                  "!children": [
                    {
                      "!type": "#text",
                      "!value": " "
                    }
                  ]
                },
                {
                  "!type": "subst",
                  "!children": [
                    {
                      "!type": "repl",
                      "!children": [
                        {
                          "!type": "intlink",
                          "@target": "Eisbär",
                          "!children": [
                            {
                              "!type": "title",
                              "!children": [
                                {
                                  "!type": "text",
                                  "!children": [
                                    {
                                      "!type": "#text",
                                      "!value": "Eisbären"
                                    }
                                  ]
                                }
                              ]
                            }
                          ]
                        }
                      ]
                    }
                  ]
                },
                {
                  "!type": "for",
                  "!children": [
                    {
                      "!type": "mww:intlink",
                      "@target": "Eisbär",
                      "@postfix": "en",
                      "!children": [
                        {
                          "!type": "rtd",
                          "!children": [
                            {
                              "!type": "#text",
                              "!value": "[[Eisbär]]en"
                            }
                          ]
                        }
                      ]
                    }
                  ]
                }
              ]
            }
          ]
        },
        {
          "!type": "rtd",
          "!children": [
            {
              "!type": "#text",
              "!value": "\n"
            }
          ]
        }
      ]
    },
    {
      "!type": "li",
      "!children": [
        {
          "!type": "rtd",
          "!children": [
            {
              "!type": "#text",
              "!value": "*"
            }
          ]
        },
        {
          "!type": "text",
          "!children": [
            {
              "!type": "#text",
              "!value": " "
            }
          ]
        },
        {
          "!type": "extlink",
          "@target": "http://www.fau.de",
          "!children": [
            {
              "!type": "rtd",
              "!children": [
                {
                  "!type": "#text",
                  "!value": "[http://www.fau.de "
                }
              ]
            },
            {
              "!type": "title",
              "!children": [
                {
                  "!type": "text",
                  "!children": [
                    {
                      "!type": "#text",
```

Listing 3: Demo Seite – WOM-JSON-Repräsentation (gekürzt auf 255 Zeilen).

3.1.2 Volumen

Die englische Wikipedia besteht zum aktuellen Zeitpunkt (April 2016) aus 5.123.064 Artikeln[20], mit einer Versionshistorie von 808.187.367 Revisionen[21]. Im Durchschnitt verfügt dabei jeder Artikel über 21,07 Revisionen[22]. Dieser Durchschnittswert ist allerdings nur bedingt aussagekräftig. Zu dem Artikel „Friedrich-Alexander-Universität Erlangen-Nürnberg“ (de.wikipedia.org) gibt es beispielsweise 653 Versionen[23], der Artikel „Deutschland“ (de.wikipedia.org) wurde 14.088[24] mal editiert und der Artikel „United States“ (en.wikipedia.org) wird mit 36.966 Revisionen[25] gelistet. Vergleicht man die Anzahl der Artikel innerhalb der englischen Wikipedia mit der Summe aller Artikel sämtlicher Wikipedia Installationen, beinhaltet die englische Wikipedia ca. 13% aller Wikipedia Artikel[26]. Extrapoliert man die Artikelzahlen der englischen Wikipedia auf die gesamte Wikipedia, müssen die Werte in etwa verzehnfacht werden.

Laut der letzten Messung von `stats.wikimedia.org` (Stand 02.2014) belegt ein Artikel, gespeichert als Wiki-Markup, im Durchschnitt 42,69 KB[27]. Für eine Hochrechnung dieses Messwerts auf die durchschnittliche Artikelgröße in den unterschiedlichen WOM-Repräsentationen wurde ein Stichprobe von 18.307 Artikeln der englischen Wikipedia[28] entnommen. Dabei wurden die jeweiligen Artikelgrößen bei Serialisierung als Wiki-Markup, LZMA2 komprimiertes WOM-JSON[29], LZMA2 komprimiertes flaches WOM-JSON[30], WOM-XML, WOM-JSON und flaches WOM-JSON gemessen. Die Stichprobe enthält ausschließlich die aktuelle Version der Artikel und ist frei von Doppelgängern. Die genauen Größenverteilungen können den Diagrammen in Abbildung 5 entnommen werden. Für die Y-Achsen der Plots wurde eine logarithmische Skala gewählt. Zur besseren Lesbarkeit enthält der Vergleich der Medianwerte im unteren Teil der Y-Achse (bis einschließlich 12) Achsenbeschriftungen in Zweierschritten, sowie zusätzliche Beschriftungslinien auf der Hälfte jedes Teilschritts entlang der kompletten Achse. Eine deskriptive

[20] https://stats.wikimedia.org/EN/TablesArticlesTotal.htm (besucht am 11.07.2016)
[21] http://www.businessinsider.de/most-edited-wikipedia-2016-1?r=US&IR=T (besucht am 08.04.2016)
[22] https://de.wikipedia.org/wiki/Wikipedia:Statistik (besucht am 08.04.2016)
[23] https://tools.wmflabs.org/xtools-articleinfo/?article=Friedrich-Alexander-Universit%C3%A4t+Erlangen-N%C3%BCrnberg&project=de.wikipedia.org (besucht am 08.04.2016)
[24] https://tools.wmflabs.org/xtools-articleinfo/?article=Deutschland&project=de.wikipedia.org (besucht am 08.04.2016)
[25] https://tools.wmflabs.org/xtools-articleinfo/?article=United_States&project=en.wikipedia.org (besucht am 08.04.2016)
[26] https://en.wikipedia.org/wiki/Wikipedia:Size_of_Wikipedia (besucht am 08.04.2016)
[27] https://stats.wikimedia.org/EN/TablesArticlesBytesPerArticle.htm (besucht am 08.04.2016)
[28] Die Daten stammen aus den ersten 13 enwiki-20160113-pages-meta-history* Dumps der englischen Wikipedia vom 13.01.2016; siehe https://dumps.wikimedia.org/enwiki/20160113/ (besucht am 22.06.2016).
[29] Die WOM-JSON-Repräsentation wurde in Kapitel 3.1.1 eingeführt (vgl. Listing 3).
[30] Bezeichnet eine spezielle WOM-JSON-Repräsentation, in der die ursprüngliche Baumstruktur der WOM-Knoten in eine lineare Knotenliste umgewandelt wurde (vgl. Abschnitt 4.3.2; Listing 28).

Datenformat	Median	Avg	Std. Abw.	Min	Max
Wiki-Markup	13,55 KB	27,16 KB	37,93 KB	0,001 KB	985,52 KB
LZMA2-kompr. hierarchisches WOM-JSON	7,50 KB	12,74 KB	15,35 KB	0,2 KB	292,58 KB
LZMA2-kompr. flaches WOM-JSON	9,74 KB	16,19 KB	19,25 KB	0,24 KB	364,29 KB
WOM-XML	34,12 KB	66,36 KB	90,34 KB	0,19 KB	1906,07 KB
Hierarchisches WOM-JSON	97,78 KB	189,08 KB	263,09 KB	0,24 KB	6783,98 KB
Flaches WOM-JSON	260,71 KB	546,61 KB	823,10 KB	0,32 KB	20843,46 KB

Tabelle 1: Stichprobe über 18.307 Artikeln der englischen Wikipedia – Tabellarische Auswertung der Artikelgrößen in Abhängigkeit vom Datenformat

Beschreibung der wichtigsten statistischen Größen liefert Tabelle 1.

Die erhobenen Messwerte können nicht direkt für die Hochrechnung des Datenvolumens der englischen Wikipedia verwendet werden. Dazu ist der Umfang der Stichprobe, verglichen mit den absoluten Artikelzahlen der englischen Wikipedia, zu gering. Es lassen sich jedoch Rückschlüsse auf den Faktor zwischen der durchschnittlichen Artikelgröße in Wiki-Markup und den durchschnittlichen Speicherverbrauch der Artikel in den unterschiedlichen WOM-Repräsentationen ziehen. Multipliziert mit den exakten Daten von `stats.wikimedia.org` kann so das zu erwartende Datenvolumen grob extrapoliert werden. Für die Berechnung des Faktors zwischen Wiki-Markup und den unterschiedlichen WOM-Repräsentationen wurde der Median der erhobenen Artikelgrößen verwendet.

Sei d_{rep} die durchschnittliche Artikelgröße aller Artikel der englischen Wikipedia im Februar 2014[27], m_{wm} der Median der gemessenen Artikelgrößen in Wiki-Markup, m_r der Median der gemessenen Artikelgrößen in den unterschiedlichen WOM-Repräsentationen r und f_r der Faktor zwischen m_{wm} und m_r, dann ergibt sich folgender Zusammenhang:

$$x = d_{rep} \cdot f_r \tag{1}$$

bzw.

$$x = d_{rep} \cdot \frac{m_r}{m_{wm}} \tag{2}$$

wobei x für den hochgerechneten Wert der durchschnittlichen Artikelgröße je WOM-Repräsentation steht. Die Werte für m_{wm} und m_r können der ersten Spalte von Tabelle 1 entnommen werden. Die Ergebnisse der Hochrechnung sind in Tabelle 2 abgebildet.

Als Vorverarbeitung für die vorliegende Messung wurden die ersten 13 enwiki-20160113-

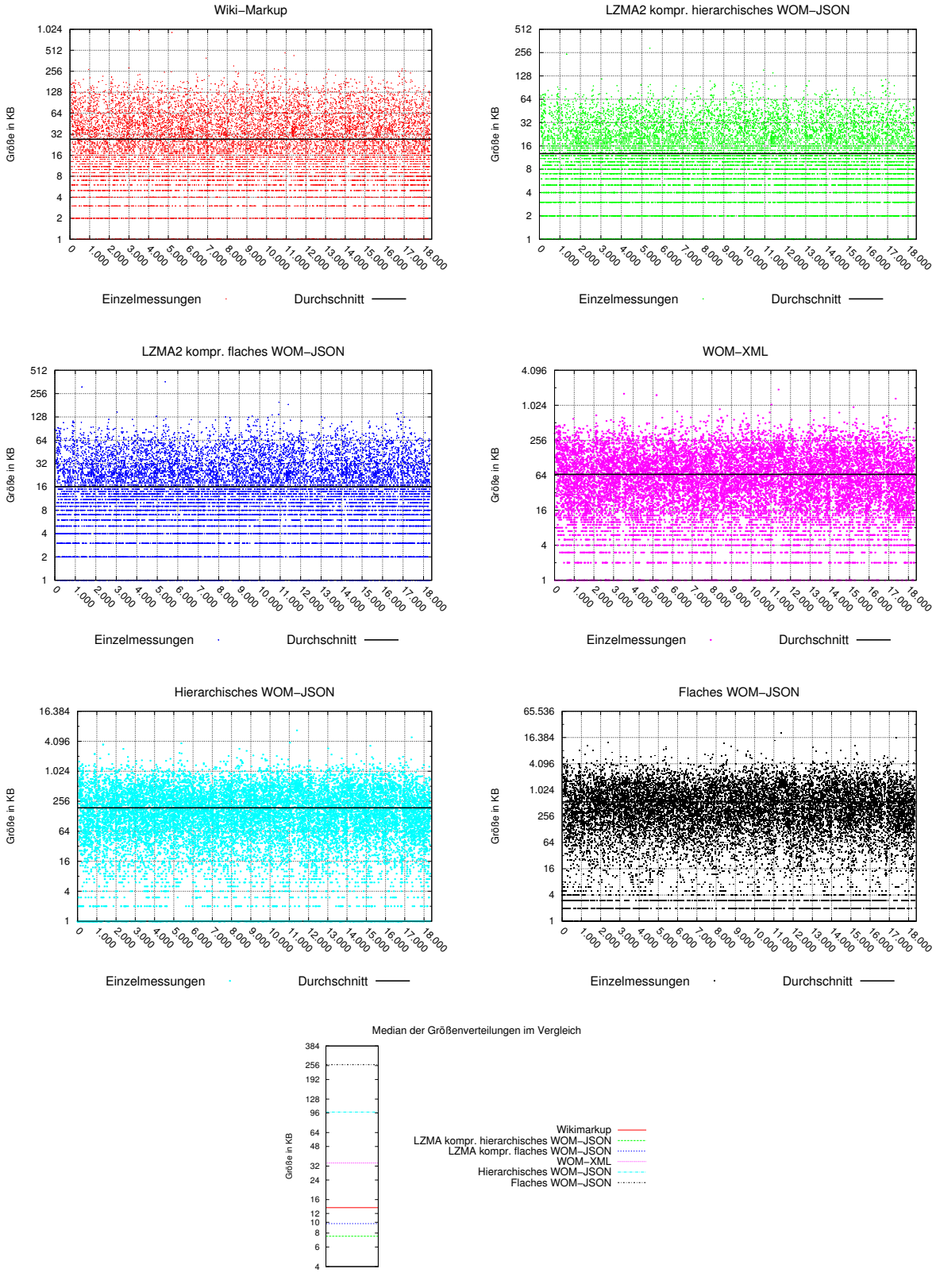


Abbildung 5: Stichprobe über 18.307 Artikeln der englischen Wikipedia – Grafische Auswertung der Artikelgrößen in Abhängigkeit vom Datenformat

Datenformat	Faktor zu Wiki-Markup f_r	Erwartete durchschn. Artikelgröße x	Geschätztes Datenvolumen EN Wikipedia
LZMA2 kompr. hierarchisches WOM-JSON	0,5535	23,63 KB	18,90 TB
LZMA2 kompr. flaches WOM-JSON	0,7188	30,69 KB	24,55 TB
WOM-XML	2,5181	107,50 KB	86,00 TB
Hierarchisches WOM-JSON	7,2162	308,06 KB	246,45 TB
Flaches WOM-JSON	19,2405	821,38 KB	671,10 TB

Tabelle 2: Hochrechnung des erwarteten Datenvolumens der englischen Wikipedia in Abhängigkeit vom Datenformat

pages-meta-history* Dumps der englischen Wikipedia (Stand 13.01.2016)[31] auf dem High-Performace-Computing-Cluster des Regionalen Rechenzentrums in Erlangen in WOM umgewandelt. Jeder der 13 XML-Dumps wurde für die Umwandlung in 20 bis 30 Einzeldateien, sogenannte „*Splits*“, zerlegt. Jede der Einzeldateien enthält die Versionshistorie eines oder mehrerer Artikel und steht nach der Vorverarbeitung als *.avro* Datei[32] zur Verfügung. Jede *.avro* Datei beinhaltet die LZMA komprimierten WOM-XML-Repräsentationen der Artikel ihres zugehörigen *Splits*.

Die Vorverarbeitung und Bereitstellung der *.avro* Dateien wurde von WOM-Entwickler Hannes Dohrn übernommen. Die aktuell verfügbare Datenbasis geparster Wikipedia-Artikel befindet sich auf dem File-Server der Open Source Research Group der Friedrich-Alexander-Universität Erlangen-Nürnberg und kann unter[33] abgerufen werden. Der aktuell vorliegende Datenbestand umfasst 7,4% aller am 13.01.2016 exportierten Artikel der englischen Wikipedia. Eine Vergrößerung der untersuchten Stichprobe oder die Verarbeitung aller bisher in WOM umgewandelter Artikelversionen ist, auf Grund des hohen Rechenaufwands, mit der für diese Arbeit zur Verfügung stehenden Hardware[34] nicht möglich. Die Erhebung der vorgestellten Daten hat insgesamt knapp 13 Stunden gedauert. In Summe wurden 19.098.755 Einzelversionen dekomprimiert, deserialisiert und unterdessen geprüft, ob es sich um die aktuellste Version handelt. Die jeweils letzte Version der Artikel wurde in die beschriebenen Ausgabeformate konvertiert und dabei die Artikelgröße gemessen. Die Verarbeitung aller bisher zur Verfügung stehenden Einzelversionen würde rund 62 Wochen dauern und ist daher im Kontext dieser Arbeit nicht möglich.

[31] https://dumps.wikimedia.org/enwiki/20160113/ (besucht am 22.06.2016)
[32] https://avro.apache.org/docs/current/ (besucht am 22.06.2016)
[33] http://faui2o2h.cs.fau.de:9080/wpra/
[34] Intel Core 2 Quad Q9550, 8 GB RAM

3.2 Queryanalyse

Abfragen werden anhand ihrer Suchparametern, der Granularität des Abfrageergebnis oder ihrer zeitlichen Dimension unterschieden.

Art der Suchparameter Knoten können grundsätzlich anhand ihres **Typs**, **Inhalts**, gesetzter **Attribute** oder ihres **Kontexts** unterschieden werden. Die Suchparameter können dabei beliebig kombiniert werden.

Queries in natürlicher Sprache:

1. Liste alle Einzelnachweise der aktuellen Version des Artikels X [Typ].
2. Liste alle Links, die in der aktuellen Version des Artikels X vorkommen und auf die Webseite Y verweisen [Typ + Inhalt].
3. Liste den Text aller Überschriften der Ebene Z der aktuellen Version des Artikels X auf [Typ + Attribut].
4. Liste alle Links der aktuellen Version des Artikels X, die innerhalb einer Tabelle stehen [Typ + Kontext].

Granularität des Abfrageergebnis Auch die Granularität der Abfrageergebnisse spielt eine wichtige Rolle. In manchen Fällen möchte man lediglich wissen, in welchem Artikel etwas zu finden ist und es reicht, wenn die Datenbank mit den Namen der Artikel bzw. einer **Liste aus Artikelidentifiern** antwortet. Dient die Abfrage zur Eingrenzung des Suchraums einer weiterführenden Analyse, müssen **ganze Artikel** oder **einzelne Abschnitte** eines Artikels als Antwort zurückgegeben werden.

Queries in natürlicher Sprache:

1. Liste die Identifier aller aktuellen Artikel, die auf Webseite X verweisen [Identifier].
2. Durchsuche die aktuelle Version aller Artikel nach Absätzen, deren Überschrift die Zeichenfolge X enthält und liste die Inhalte dieser Absätze als Teilbäume [Teilbäume].
3. Liste den kompletten WOM-Tree aller aktuellen Artikel, die auf Webseite X verweisen [komplette Artikel].

Zeitliche Dimension Es wird zwischen „**historischen Queries**“ und „**aktuellen Queries**“ unterschieden. Letztere beziehen Informationen aus den **aktuellen Versionen der Artikel** (sprich aus der letzten Revision eines jeden Artikels). „Historische Queries“ extrahieren Informationen aus einem **Quer-** oder **Längsschnitt der Vergangenheit**. Bei einem Querschnitt werden die Revisionen der Artikel betrachtet, die zum damaligen Zeitpunkt die

neueste Revision darstellten. Bei einem Längsschnitt wird eine Serie von Revisionen desselben Artikels untersucht, um zum Beispiel etwas über dessen Entwicklung zu erfahren.

Queries in natürlicher Sprache:

1. Liste alle aktuellen Artikel, die einen Verweis auf Artikel Y enthalten [aktuell].
2. Liste alle aktuellen Artikel, die Bold-Knoten mit dem Text „foo“ enthalten [aktuell].
3. Liste die Revision des Artikels X, in der der Verweis auf Artikel Y hinzukam [Längsschnitt].
4. Liste die Änderungen, die sich zwischen den Revisionen des Artikels X im Zeitraum t_a . . . t_b ergeben haben [Längsschnitt].
5. Liste alle Artikel, die zum Zeitpunkt t auf den Artikel Y verwiesen [Querschnitt].

3.3 Workload

Der Workload der Datenbank ist von komplexen Analysen auf den Inhalten der Wikipedia geprägt. Dabei wird die Datenbank einmal initial in einem Batch-Prozess befüllt und anschließend inkrementell, mittels vieler kleiner Einfügeoperationen, aktuell gehalten.

3.3.1 Abfragecharakteristik

Zu erwarten sind hauptsächlich lange laufende Queries, die viele Millionen Artikel verarbeiten und punktuell oder im Batch-Betrieb ausgeführt werden. Wie viele Artikelrevisionen dabei untersucht werden, hängt im Wesentlichen von der zeitlichen Dimension der Abfrage ab.

Aktuelle Queries untersuchen ausschließlich die letzte Version der Artikel. Basierend auf den Zahlen von `stats.wikimedia.org`[35] für Februar 2016 müsste eine Analyse der aktuellen Versionen aller Artikel der englischen Wikipedia ca. $5,1 \cdot 10^6$ Einträge verarbeiten bzw. $38,8 \cdot 10^6$, wenn Aussagen über die Gesamtheit der letzten Versionen aller Wikipedia Artikel weltweit getroffen werden sollen.

Historische Queries werden in Quer- und Längsschnittanalysen aufgeteilt und untersuchen die Versionsgeschichte der Artikel.

[35] `https://stats.wikimedia.org/EN/TablesArticlesTotal.htm` (besucht am 09.04.2016)

Längsschnitte analysieren die Historie einzelner Artikel. Die Anzahl der Versionen pro Artikel schwankt zwischen eins, bei einem neu angelegten Artikel und mehreren 10.000. Laut bussinessinsider.de[36] handelt der meisteditierte Artikel der englischen Wikipedia von dem früheren US Präsidenten Georg W. Bush (Stand 01.2016) und zählt aktuell 45.924 Versionen[37]. Im schlechtesten Fall verarbeitet eine Längsschnittanalyse damit $4.6 \cdot 10^4$ Artikelversionen. Auch ist der Artikel über Georg W. Bush nicht der Einzige mit einer derart umfangreichen Versionshistorie. 11 der 15 meist editierten Artikel der englischen Wikipedia zählen mehr als $2 \cdot 10^4$ Änderungen.

Querschnittanalysen ermitteln Zusammenhänge über eine Population von Wikiartikelen in einem Zeitintervall $t_x \ldots t_y$, wobei t_x und t_y zwei beliebige Zeitpunkte in der Vergangenheit sind und der Zusammenhang $t_x < t_y$ gilt. Prinzipiell muss für jeden Artikel der Population zunächst eine Längsschnittanalyse durchgeführt werden, mit deren Hilfe die Versionshistorie auf die zum Zeitraum t_x bis t_y aktuelle(en) Version(en) eingegrenzt wird. Danach folgt die eigentliche Querschnittanalyse. Diese ähnelt grundsätzlich einer aktuellen Query. Der Unterschied liegt im Intervall $t_x \ldots t_y$. Bei einer aktuellen Query wird ausschließlich die letzte Version eines Artikel untersucht. Der, einer Querschnittanalyse vorausgehende Längsschnitt, kann hingegen mehrere Versionen eines Artikels als Ergebnis zurückliefern. Die Anzahl in einem Querschnitt zu untersuchender Artikel ist dabei abhängig von der betrachteten Artikelpopulation (z.B. en.wikipedia.org), der Länge des zu betrachteten Zeitintervalls $t_x \ldots t_y$ und der individuellen Änderungshäufigkeit im Zeitraum $t_x \ldots t_y$.

3.3.2 Änderungscharakteristik

Die Änderungscharakteristik der Analysedatenbank und das Versionsmanagement der Wikipedia sind eng miteinander verzahnt. Die Wikipedia verwaltet keine inkrementelle Artikelhistorie, stattdessen wird bei jeder Änderung eine neue Artikelversion angelegt. Einmal angelegte Artikelversionen werden vom System ausschließlich lesend abgefragt. Soll ein Artikel editiert werden, muss zunächst seine letzte Version von der Anwendung gelesen, modifiziert und anschließend der gesamte geänderte Artikel als neue Version in die Datenbank zurückgeschrieben werden. Die Konsistenzsicherung ist dabei Aufgabe der Anwendung.

Aus Sicht der Analysedatenbank ergeben sich daraus zwei Konsequenzen:

1. Schreiboperationen sind auf Insert-Operationen beschränkt.

[36] http://www.businessinsider.de/most-edited-wikipedia-2016-1?r=US&IR=T (besucht am 08.04.2016)

[37] https://tools.wmflabs.org/xtools-articleinfo/?article=George+W+Bush&project=en.wikipedia.org (besucht am 08.04.2016)

2. Selbst kleine Änderungen führen zu hohen Schreiblasten, da stets ganze Artikel geschrieben werden, anstatt einzelne Artikelabschnitte zu modifizieren.

Basierend auf den Zahlen von stats.wikimedia.org[38] für Februar 2016 verarbeitet die englische Wikipedia rund 3,4 Millionen Editiervorgänge pro Monat, das entspricht ca. 1,4 Einfügeoperationen pro Sekunde. Die Gesamtheit aller Wikipedia-Installationen verzeichnete im Februar 2016 rund 13 Millionen Editiervorgänge pro Monat bzw. 5,2 Einfügeoperationen pro Sekunde. Eine Analysedatenbank, die einen Live Datenbestand der englischen Wikipedia enthält, müsste somit in der Lage sein, neben der Ausführung der eigentlichen Abfragen, einen Strom von 1,4 Einfügeoperationen pro Sekunde zu verarbeiten.

4 Datenbanksystemanalyse

In diesem Kapitel werden die charakteristischen Eingenschaften aller eingangs als geeignet eingestufter Datenbanksysteme besprochen und anschließend eine Vorauswahl, entlang der wichtigsten funktionalen und nicht-funktionalen Eigenschaften, getroffen.

Funktional stellt die Vielfalt der Suchparameter die größte Herausforderung dar. Die Queryanalyse hat gezeigt, WOM-Dokumente sollen gezielt nach bestimmten Knoteninhalten, Knotentypen oder nach Knoten mit speziellen Attributen durchsucht werden. Des Weiteren soll der direkte oder indirekte Knotenkontext als Argument für Suchanfragen dienen.

Aus nicht-funktionaler Sicht ist das Datenvolumen die wichtigste Kenngröße. Gemäß der Datenanalyse umfasst die Versionshistorie der englischen Wikipedia rund $800 \cdot 10^6$ Einträge. Ausgehend von 308 KB pro WOM-Dokument (vgl. Tabelle 2) ergibt sich unkomprimiert ein Datenvolumen von rund 246 Terabyte bzw. 2,5 Petabyte, wenn man auf den Datenbestand der kompletten Wikipedia extrapoliert. Auf Grund der enormen Datenmengen und der Forderung, das Endsystem auf Commodity Hardware betreiben zu können, haben im nicht-funktionalen Bereich gute horizontale Skalierbarkeit und transparente Datenkompression den höchsten Stellenwert.

Nachfolgend wird die Eignung von nativen XML-Datenbanken (Abschnitt 4.1), relationalen Datenbanken mit XML-Datentyp (Abschnitt 4.2), JSON basierten Document Stores (Abschnitt 4.3), Multi-Model-Datenbanken (Abschnitt 4.4), sowie Wide Column Stores in Kombination mit Map-Reduce (Abschnitt 4.5) untersucht.

4.1 Native XML-Datenbanken

Auf Grund der großen konzeptionellen Ähnlichkeit zwischen WOM und XML und der direkten Abbildung von WOM auf XML (XWML), wird zunächst der Einsatz nativer XML-Datenbanken geprüft.

[38] https://stats.wikimedia.org/EN/TablesDatabaseEdits.htm (besucht am 09.04.2016)

4.1.1 Grundlegende Eigenschaften

Im Bereich nativer XML-Datenbanken hat sich XQuery[39] als Anfragesprache etabliert. XQuery wurde von der W3C[40] definiert und soll den Umgang mit XML-Daten vereinheitlichen. XPath[41] ist ein Teil von XQuery und ermöglicht die Formulierung von Pfadausdrücken, genauer die gezielte Auswahl einzelner XML-Knoten innerhalb eines Dokuments. Die XQuery-FLWOR-Ausdrücke[42] ermöglichen Abfragen und Aggregationen über mehrere XML-Dokumente einer *Collection*[43]. Da XQuery ursprünglich als reine Abfragesprache konzipiert wurde, implementieren native XML-Datenbanken zusätzlich die XQuery Update Facility (XQUF)[44]. Auf diese Weise wird XQuery um die Möglichkeit erweitert, schreibend auf XML-Dokumente zuzugreifen und somit zur vollwertigen Data-Manipulation-Language (DML) aufgewertet [NM, S. 2]. Zur Beschleunigung der Abfragen können einzelne XML-Knoten gezielt indiziert werden. Darüber hinaus verfügen die meisten Systeme über integrierte Queryoptimizer bzw. Query-Rewrite-Engines, die die automatisierte Nutzung der Indexstrukturen ermöglichen. Einige wenige Systeme[45,46] erlauben zusätzlich die Ausführung von XSLT-Transformationen[47] innerhalb der Datenbank.

4.1.2 Funktionale Eignung

Rein funktional lassen sich die verschiedenen Anfragetypen mit ihren unterschiedlichen Suchparametern problemlos auf den Funktionsumfang von XQuery abbilden[48,49]. Dies gilt insbesondere für kontextbasierte Abfragen, welche direkt mit Hilfe der XPath-Achsen[50] für den Zugriff auf Eltern-, Kind- und Nachbarknoten umgesetzt werden können. Die Granularität des Ergebnisses ist dabei frei wählbar. XQuery ermöglicht den Zugriff auf einzelne Knoten, Teilbäume oder ganze XML-Dokumente[48]. Die Suche entlang der verschiedenen zeitlichen Achsen (Analyse der aktuellen Versionen, Quer- und Längsschnitte) wird mit Hilfe der FLWOR-Ausdrücke umgesetzt.

Für eine Analyse der aktuellen Version aller Artikel werden die Einträge anhand ihres

[39] https://www.w3.org/TR/xquery-30/ (besucht am 16.05.2016)
[40] https://www.w3.org/Consortium/ (besucht am 15.06.2016)
[41] https://www.w3.org/TR/xpath-31/ (besucht am 16.05.2016)
[42] http://www.w3schools.com/xsl/xquery_flwor.asp (besucht am 20.05.2016)
[43] Eine *Collection* bezeichnet eine Sammlung von Dokumenten und ist das NoSQL-Äquivalent zur Datenbank von relationalen Datenbankmanagementsystemen.
[44] https://www.w3.org/TR/xquery-update-10/ (besucht am 16.05.2016)
[45] http://docs.basex.org/wiki/XSLT_Module (besucht am 16.05.2016)
[46] http://exist-db.org/exist/apps/doc/xsl-transform.xml (besucht am 16.05.2016)
[47] XSLT steht für „Extensible Stylesheet Language Transformation“ und ist eine, von der W3C definierte, Sprache zur Umwandlung von XML-Dokumenten in andere XML-Dokumente https://www.w3.org/TR/xslt (besucht am 16.05.2016)
[48] http://www.w3schools.com/xsl/xpath_syntax.asp (besucht am 17.05.2016)
[49] http://exist-db.org/exist/apps/demo/examples/basic/mondial.html (besucht am 20.05.2016)
[50] http://www.w3schools.com/xsl/xpath_axes.asp (besucht am 12.05.2016)

Titels gruppiert (XQuery 3.0 group by[51]) und pro Gruppe der Datensatz mit der jeweils größten Versionsnummer ausgegeben (XQuery fn:max[52]). Titel und Version befinden sich in den *title* und *articleVersion* Attributen des *article* Knotens. Für eine Querschnittanalyse werden ausschließlich Einträge zurückgegeben, deren Erstellungsdatum (*timestamp* Attribut, *article* Knoten) innerhalb eines vorgegebenen Zeitintervalls liegt. Bei einem Längsschnitt werden die Artikel anhand ihres Titels (*title* Attribut, *article* Knoten) ausgewählt und die Versionsnummer nicht weiter eingegrenzt. Die *timestamp* und *articleVersion* Attribute des *article* Knotens wurden nachträglich in Abschnitt 3.1.1 eingeführt.

4.1.3 Nicht-funktionale Eignung

Ob diese Klasse von Datenbanksystemen in der Lage ist die geforderten Datenmengen zu bewältigen, kann ausschließlich durch Betrachtung konkreter Systeme beurteilt werden. DB-Engines[53] listet im nicht-kommerziellen Bereich Sedna[54], BaseX[55] und eXitst-db[56]. Weitere Recherchen haben X-Hive[57], Oracle Berkeley DB XML[58] und Apache Xindice[59] hervorgebracht.

Sedna, X-Hive und Apache Xindice werden nicht mehr aktiv weiterentwickelt. Der letzte Eintrag im News Ticker auf www.sedna.org stammt vom 28.11.2011[60]. Der letzte, offiziell auf der Sedna-Projektseite verlinkte Build ist auf den 27.06.2014 datiert[61] und der letzte commit auf sourceforge stammt vom 28.01.2012[62]. Die letzte X-Hive Version wurde am 13.06.2002 veröffentlicht[57]. Apache Xindice ist seit dem 15.08.2011 unter Verwaltung von Apache Attic[59].

Exist-db steht unter der L-GPL[63], BaseX wird unter den Bedingungen der BSD Lizenz veröffentlicht[64]. Beide Systeme können somit kostenlos genutzt werden und Anwendungen, die intern auf die Systeme zurückgreifen, dürfen, unter Einhaltung der jeweiligen Lizenzbedingungen, an Dritte weitergegeben werden. Oracle Berkeley DB XML wird ebenfalls unter einer, zur GPL kompatiblen, Open-Source-Lizenz vertrieben. Die Lizenzvereinbarungen verbieten jedoch die Weitergabe („redistribution“) aller auf ihr basierenden Anwendungen. Anwendungen, die Oracle Berkeley DB XML intern zur Datenspeicherung

[51] https://www.w3.org/TR/xquery-30/#id-group-by (besucht am 15.06.2016)
[52] http://www.xqueryfunctions.com/xq/fn_max.html (besucht am 15.06.2016)
[53] http://db-engines.com/de/ranking/native+xml+dbms (besucht am 12.05.2016)
[54] http://www.sedna.org/ (besucht am 12.05.2016)
[55] http://basex.org/ (besucht am 12.05.2016)
[56] http://exist-db.org/exist/apps/homepage/index.html (besucht am 12.05.2016)
[57] http://xml.coverpages.org/X-HiveNative-XML-Database.html (besucht am 07.04.2016)
[58] http://www.oracle.com/us/products/database/berkeley-db/xml/overview/index.html (besucht am 12.05.2016)
[59] http://xml.apache.org/xindice/ (besucht am 12.05.2016)
[60] Stand: 07.04.2016
[61] http://www.modis.ispras.ru/FTPContent/sedna/all-builds/ (besucht am 07.04.2016)
[62] https://sourceforge.net/projects/sedna/files/?source=navbar (besucht am 07.04.2016)
[63] http://exist-db.org/exist/apps/doc/legal.xml (besucht am 16.05.2016)
[64] http://basex.org/open-source/ (besucht am 16.05.2016)

verwenden und von Nutzern außerhalb der eigenen Organisation genutzt werden sollen, müssen eine kostenpflichtige Berkeley DB Lizenz erwerben[65]. Aus diesem Grund scheidet auch Oracle Berkeley DB XML als möglicher Systemkandidat aus.

Nach Ausschluss aller eingestellten Projekte und Prüfung der Lizenzbedingungen verbleiben eXist-db und baseX als mögliche Systemkandidaten. Keines der Systeme bietet integrierte Möglichkeiten für den verteilungstransparenten Zugriff auf einzelne *Collections*, den Clusterbetrieb oder das Collection-Sharding an [NM, S. 5][66,67,68]. BaseX wendet ein eigenes, auf speziellen Eigenschaften der BaseX Storage-Engine basierendes, Verfahren zur transparenten Datenkompression an [GHS07, S. 631] [Grü+06]. Exist-db speichert XML-Dokumente unkomprimiert. Freire et al. beziffern die Kompressionsraten des von BaseX eingesetzten Algorithmus auf 1,40 bis 1,47[Fre+12, S. 54]. Die Messung untersucht unter anderem den Speicherverbrauch von BaseX und eXist bei der Verwaltung von vier unterschiedlich großen XML-Datenbeständen aus dem medizinischen Umfeld. BaseX und eXist befinden sich dabei jeweils in ihrer Default-Konfiguration und es werden keine zusätzlichen Indexstrukturen angelegt. Exist-db benötigt bei der Messung im Schnitt 1,43 mal mehr Speicher als die Rohdaten. Der Faktor steigt im Verlauf der Messung von 1,29 bei 0,56 GB Rohdaten, auf 1,67 bei 60 GB Rohdatenvolumen an.

Als Letztes wird überprüft, wie viele XML-Dokumente mit einer einzelnen baseX bzw. eXist Instanz verwaltet werden können. Da es in diesem Punkt für keines der Produkte aussagekräftige Referenzanwendungsfälle gibt, die klarstellen, welche Daten- und Dokumentmengen die Systeme verwalten können, wird bei der weiteren Analyse auf zwei Studien von Freire et al. und einen eXist-spezifischen Nutzerbericht von Robert Ellwell[69], Master-Absolvent der University of Texas Austin und früherer Softwarearchitekt bei AetherQuest Solutions, zurückgegriffen.

Die BaseX-Dokumentation beinhaltet eine Liste mit Volumenmetriken unterschiedlicher BaseX-Installationen[70]. Der Verwendungszweck, die Zugriffsmuster oder konkrete abfragebezogene Metriken werden nicht angegeben. Das Gleiche gilt im Wesentlichen für die aufgeführte Case Study zur Verwaltung von Twitter Nachrichten[71]. Die eXist-Projektseite listet das „Office of the Historian's“[72] und das Content-Management-System easyDITA[73] als größte eXist-Nutzer. Das „Office of the Historian's“ verwaltet einen Datenbestand von knapp 100.000 staatlichen Dokumenten[74] der US-Regierung. Die Produktbeschreibung

[65] http://www.oracle.com/technetwork/database/berkeleydb/downloads/licensing-098979.html (besucht am 12.05.2016)
[66] http://exist-db.org/exist/apps/doc/documentation.xml (besucht am 18.05.2016)
[67] http://files.basex.org/publications/xmlprague2013/2013/Dirk-Kirsten-Distributed-XQuery.pdf (besucht am 18.05.2016)
[68] http://docs.basex.org/wiki/Table_of_Contents (besucht am 18.05.2016)
[69] http://robertelwell.info/about.php (besucht am 19.05.2016)
[70] http://docs.basex.org/wiki/Statistics (besucht am 17.06.2016)
[71] http://docs.basex.org/wiki/Twitter (besucht am 17.06.2016)
[72] https://history.state.gov/ (besucht am 17.06.2016)
[73] http://easydita.com/features/ (besucht am 17.06.2016)
[74] http://exist-db.org/exist/apps/homepage/references.html (besucht am 17.06.2016)

von easyDITA spricht von „millions of topics“[73] die mit easyDITA verwaltet werden können. Es ist jedoch unklar, wie exakt diese Angaben sind und in wie weit der Funktionsumfang von eXist für den Einsatz innerhalb von easyDITA erweitert wurde.

Aus den Studien von Freire et al. lassen sich folgende Erkenntnisse ziehen:

- Die Ausführungszeit populationsbasierter Queries ist bei baseX und eXist, sowohl indiziert[Fre+16, S.10 unten] als auch nicht indiziert[Fre+12, S.55 Conclusions], abhängig von der Collectiongröße und der Komplexität der zu verarbeitenden Queries.
- BaseX und eXist bewältigen auf Consumer-Hardware im unteren Leistungssegment[75], bei Dokumentgrößen von 30 bis 606,2 KB und ohne weitere Optimierung durch Indizes, populationsbasierte Queries aus dem medizinischen Umfeld bis zu Collectiongrößen von ca. 60.000 Einträgen. Danach steigt die mittlere Querylaufzeit auf mehrere Minuten[Fre+12, S. 54].
- BaseX und eXist sind, auf Consumer-Hardware im mittleren Leistungssegment[76] und bei gezielter Indizierung des Datenbestands, in der Lage ein Beispielset von 8 populationsbasierte Queries aus dem medizinischen Umfeld, bis zu Collectiongrößen von ca. 100.000 Einträgen (3,2 GB), zu verarbeiten. Die mittlere Querylaufzeit von eXist übersteigt mit 99 Sekunden die Eine-Minuten-Marke. Die mittlere Querylaufzeit von BaseX ist mit 9,2 Sekunden deutlich geringer. Die am längsten laufende Query von eXist ist nach 170 Sekunden verarbeitet. BaseX produziert im schlechtesten Fall nach maximal 31 Sekunden ein Ergebnis[Fre+16, S.8,11 f.].
- BaseX benötigt bei Collectiongrößen von 600.000 Dokumenten mit einer Größe von 30 bis 606,2 KB und Verwendung von Consumer-Hardware im unteren Leistungssegment[75], ohne vorherige Indizierung des Datenbestands, ca. 16 Minuten für die Ausführung populationsbasierter Queries aus dem medizinischen Umfeld[Fre+12, S. 54]. Die gleiche Messung wurde bei eXist auf Grund zu langer Querylaufzeiten abgebrochen.

Robert Ellwell[69] hat die Eignung von eXist-db für die Verwaltung, Abfrage und Indizierung von Output der Stanford CoreNLP Suite[77] geprüft. Dabei wurde ein Testdatenbestand von 50.000 Dokumenten mit einer mittleren Dateigröße von 150 KB in eine auf High-End-Hardware[78] ausgeführte eXist-Instanz eingelesen. Den Messungen zufolge benötigt eine einzelne Einfügeoperation im Schnitt 200 ms. Jegliche Bemühungen den Datenimport zu beschleunigen blieben laut Ellwell erfolglos[79].

[75] AMD Athlon 64X2 5600+x2, 3,9 GB RAM, Ubuntu 12.04 LTS; Festplattensetup und Dateisystemkonfiguration unbekannt [Fre+12, S.54 mitte]

[76] Intel Core i7-3770S CPU 8 x 3.10GHz, 8 GB RAM, Ubuntu 12.04 LS 64 Bit; Festplattensetup und Dateisystemkonfiguration unbekannt [Fre+16, S.9 unten]

[77] http://stanfordnlp.github.io/CoreNLP/ (besucht am 19.05.2016)

[78] Amazon m2.4xlarge Instanz mit 8 virtuellen CPUs, 68,4 GB RAM und 2 x 840 GB Festplattenspeicher; Dateisystemkonfiguration unbekannt.

[79] http://robertelwell.info/blog/exist-db-not-ready-for-high-scale/ (besucht am 19.05.2016)

Strnad et al. bestätigen die relativ lange Laufzeit von Einfügeoperationen bei eXist-db im Vergleich zu anderen nativen XML-Datenbanken [SMJ13, S. 125]. Rückschlüsse auf die absolute Einfüge-Performance von BaseX unter den Bedingungen von Ellwell lassen sich auf Basis der Ergebnisse jedoch nicht ziehen. Dazu sind die verwendeten Hardware-Setups zu unterschiedlich[80] und die rein grafische Beschreibung der Messergebnisse von Strnad et al. zu ungenau.

4.1.4 Fazit

Auf Basis der vorliegenden Daten wird die maximale, von BaseX und eXist verarbeitbare Dokumentenmenge, bei Einsatz von Commodity Hardware, auf ca. 400.000 bis 600.000 Dokumente pro Datenbankinstanz geschätzt. Basierend auf einer Collectiongröße von 600.000 Dokumenten pro Instanz wären für die Verarbeitung der kompletten Versionshistorie der englischen Wikipedia ca. 1334 Datenbankinstanzen notwendig. Auf Grund dieser Hochrechnung und der Tatsache, dass weder BaseX noch eXist-db für den Clusterbetrieb geeignet sind, werden beide Systeme als ungeeignet für den vorliegenden Anwendungsfall eingestuft.

4.2 Relationale Datenbanksysteme mit XML Datentyp

Relationale Datenbanken mit XML-Funktionalität, so genannte „XML Enabled Databases“, stellen die zweite Möglichkeit zur Verwaltung von XML dar. Die Bezeichnung „XML enabled“ fasst dabei verschiedene Techniken zusammen und reicht von einem starren tabellenbasierten Mapping, über den Einsatz objektrelationaler Mappingtechniken, bis hin zur Speicherung der XML-Dokumente in einem speziellen XML-Datentyp oder als XML interpretierbaren BLOB oder CLOB [Bou99, S. 7f., 19f.]. Welche Techniken geeignet sind, hängt von der Art der vorliegenden XML-Daten ab.

Grundsätzlich wird zwischen daten- und dokumentenzentriertem XML unterschieden [Bou99, S. 3-6]. Datenzentriertes XML ist durch eine weitgehend reguläre Struktur mit feingranularen Datensätzen, deren kleinste unabhängige Dateneinheit PCDATA-only- oder reine Attribut-Elemente sind, gekennzeichnet. Weiterhin wird nahezu vollständig auf die Mischung von PCDATA und Plaintext verzichtet. Dokumentenzentriertes XML verfügt über eine weniger reguläre oder gänzlich irreguläre Struktur und beeinhaltet hauptsächlich grobgranulare Datensätze, deren kleinste gemeinsame Dateneinheit ganze, oftmals mit Plaintext gemischte, Elemente sind. Dokumentenzentriertes XML ist meist handgeschrieben oder wird aus händisch generierten Daten erzeugt [Bou99, S. 5].

[80] Strnad et al. verwenden für ihre Messungen einen Intel(R) Core(TM) 2 Duo T5500 1.66 GHz, mit 1,5 GB RAM und Debian 6.0.4 (Kernel 2.6.35-5-amd64); Festplattensetup und Dateisystemkonfiguration unbekannt.

Für die Verwaltung von datenzentriertem XML kommen prinzipiell alle drei der eingangs beschriebenen Speichermöglichkeiten in Betracht. Bei der Anwendung des tabellenbasierten Mappings ist in vielen Fällen ein zusätzlicher Transformationsschritt (vor der Ablage in und nach dem Lesen aus der Datenbank) notwendig, da die zu speichernden XML-Dokumente in einem speziellen Format vorliegen müssen [Bou99, S.7 f.]. Dokumentenzentriertes XML lässt sich am Besten in Form eines eigenen XML-Datentyps bzw. als XML interpretierbarer BLOB oder CLOB speichern [Bou99, S. 19].

Gemäß der Definition von Bourret [Bou99, S. 5] zählt WOM-XML zu den dokumentenzentrierten XML-Formaten. WOM-Dokumente werden aus handgeschriebenen Wikiartikeln erzeugt und haben keine reguläre Struktur. Die Struktur setzt sich vielmehr aus den Formatierungselementen der zugehörigen Wikiartikel zusammen. Für die Verwaltung der entstehenden XML-Dokumente werden daher relationale Datenbanksysteme benötigt, welche die XML-Daten direkt, mittels eines speziellen XML-Datentyps, speichern und die Mischung von SQL- und XPath-Ausdrücken für die anschließende Datenanalyse erlauben.

4.2.1 Grundlegende Eigenschaften

Auf Grund der weiten Verbreitung und des vielfach vorhandenen Wissens über relationale Datenbanksysteme, wird in diesem Abschnitt ausschließlich eine Systemauswahl getroffen und die Lizenzen der entsprechenden Produkte überprüft.

Als mögliche Systemkandidaten werden PostgrSQL[81] und MariaDB[82] untersucht. PostgreSQL wird unter der PostgreSQL-Lizenz vertrieben, einer liberalen Open-Source-Lizenz, ähnlich der BSD- oder MIT-Lizenz[83]. MariaDB steht unter der GPL-Version 2, Client-Libraries werden unter der LGPL-Version 2.1 bereitgestellt[84]. MariaDB wird MySQL[85] auf Grund des Lizenzmodells vorgezogen. Die MySQL Community-Edition wird inklusive Client-Libraries unter der GPL-Version 2 vertrieben[86] und ist damit ebenfalls frei verfügbar. Oracle hat jedoch das Recht, das Lizenzmodell nachträglich zu ändern, was, bei einer Änderung zu einer kommerziellen Lizenz, die Nutzung von MySQL auf lange Sicht unmöglich machen würde[87]. Aus diesem Grund wird MariaDB gegenüber MySQL der Vorzug gegeben.

[81] https://www.postgresql.org/ (besucht am 27.05.2016)
[82] https://mariadb.org/ (besucht am 27.05.2016)
[83] https://www.postgresql.org/about/licence/ (besucht am 27.05.2016)
[84] https://mariadb.com/kb/en/mariadb/licensing-faq/ (besucht am 27.05.2016)
[85] https://www.mysql.de/products/community/ (besucht am 27.05.2016)
[86] https://www.mysql.de/products/community/ (besucht am 27.05.2016)
[87] http://www.informatik-aktuell.de/betrieb/datenbanken/mariadb-als-strategische-entscheidung.html (besucht am 27.05.2016)

4.2.2 Funktionale Eignung

Beide Systeme bieten Möglichkeiten zum direkten Speichern von XML und erlauben die Mischung von SQL- und XPath-1.0-Ausdrücken.

PostgreSQL implementiert einen eigenen XML-Datentyp[88], auf den mittels verschiedener XML-Funktionen zugegriffen werden kann[89]. Der Funktionsumfang lässt sich im Wesentlichen durch die Funktionen *XMLPARSE*, *XMLSERIALIZE*, *xpath* und *xpath_exists* beschreiben.

XMLPARSE wandelt einen String bzw. „character data“ mit XML-Markup in den PostgreSQL-eigenen XML-Datentyp um und kann optional das Format der eingelesen Daten gegen eine W3C XML-Schema-Definition validieren[88]. *XMLSERIALIZE* stellt die inverse Operation zu *XMLPARSE* dar und exportiert die PostgreSQL-XML-Repräsentation zurück in einen String bzw. *character*, *character varying* oder *text*[88].

Die Analyse der gespeicherten XML-Dokumente erfolgt mit *xpath* und *xpath_exists*. Beide Funktionen werten die Instanz eines XML-Datentyps mit Hilfe eines XPath 1.0-Ausdrucks aus. *Xpath* gibt die gefundenen Knoten als Array zurück, *xpath_exists* liefert einen Wahrheitswert der positiv wird, falls es Knoten innerhalb des XML-Dokuments gibt, die der Struktur des XPath-Ausdrucks entsprechen. Ein direktes Setzen von Sekundärindizes auf einzelne Knoten eines XML-Dokuments ist nicht möglich[88]. Die einzige Möglichkeit zur Beschleunigung von XPath-Abfragen ist die Indizierung der konkreten XPath-Ausdrücke selbst[90]. Dabei existieren jedoch Fälle, in denen selbst das Setzen eines Indexes, kein Performance-Plus bringt[91].

MariaDB stellt für die Verwaltung von XML die Funktion *ExtractValue* zur Verfügung. *ExtractValue* interpretiert einen String als XML und erwartet als zweites Argument einen XPath-1.0-Ausdruck für dessen Analyse. Der Rückgabewert besteht ausschließlich aus PCDATA, was das Extrahieren einzelner XML-Fragmente unmöglich macht. Beinhaltet der gegebene String kein gültiges XML, wird eine Fehlermeldung ausgegeben. Möglichkeiten zur Validierung gegenüber einer DTD oder eines XML-Schemas stehen nicht zur Verfügung[92]. Weiterhin gibt es keinerlei Möglichkeiten zur Indizierung spezieller XML-Konten oder einzelner XPath-Ausdrücke. Auch das künstliche Hinzufügen virtueller Spalten und

[88] https://www.postgresql.org/docs/current/static/datatype-xml.html (besucht am 28.05.2016)
[89] https://www.postgresql.org/docs/current/static/functions-xml.html (besucht am 28.05.2016)
[90] https://wiki.postgresql.org/wiki/XML_Indexing (besucht am 28.05.2016)
[91] http://stackoverflow.com/questions/6504278/postgresql-index-on-xpath-expression-gives-no-speed-up (besucht am 28.05.2016)
[92] https://mariadb.com/kb/en/mariadb/extractvalue/ (besucht am 28.05.2016)

deren Indizierung wird durch einen Bug in der Virtual-Column-Implementierung (mit Status „Won't fix") verhindert[93,94].

4.2.3 Nicht-funktionale Eignung

Beide Systeme sind entweder direkt oder mit Hilfe von Erweiterungen in der Lage, Tabellen über mehrere Serverinstanzen aufzuteilen – Tabellen Sharding. Bei PostgrSQL bedarf es dazu einer Erweiterung namens Citus[95]. MariaDB realisiert diese Funktion mittels der Spider-Storage-Engine[96].

4.2.4 Fazit

Sowohl PostgreSQL als auch MariaDB bieten die Möglichkeit XML-Dokumente als Ganzes schemalos zu verwalten und durch die Kombination von SQL- und XQuery-Ausdrücken zu analysieren. Der Funktionsumfang ist jedoch sehr limitiert. Keines der Systeme stellt Funktionen für eine gezielte Indizierung einzelner Knoten innerhalb eines XML-Dokuments bereit. PostgreSQL erlaubt, im Gegensatz zu MariaDB, die Indizierung konkreter XPath-Ausdrücke. Auf diese Weise lassen sich bekannte Abfragen eingeschränkt[91] beschleunigen, Ad-Hoc-Queries profitieren davon jedoch nicht.

Ronald Bourret beschreibt in seiner Arbeit eine Art Workaround für die Indizierung von XML-Dokumenten[Bou99, S. 19]. Dabei wird pro benötigtem XML-Index eine weitere Tabelle, eine sogenannte „side table", angelegt. Diese enthält eine Spalte mit dem indizierten Wert und einen Fremdschlüssel auf die eigentliche Datentabelle. Die Inhalte der Zusatztabellen werden mit Triggern aktuell gehalten und mit konventionellen Indexmechanismen indiziert. Bei der Abfrage von Daten wird, statt direkt mit XQuery auf die XML-Daten zuzugreifen, auf die *side table* zugegriffen. Solche oder ähnliche Vorgehensweisen[91] sind grundsätzlich möglich, erhöhen jedoch die Systemlast beim Einfügen neuer Datensätze und setzen voraus, dass die Datenbanknutzer alle auf diese Weise angelegten Indexstrukturen kennen, da kein automatischer Queryrewrite stattfinden kann.

Auf Grund der hohen funktionalen Einschränkungen, allen voran der fehlenden Indizierung einzelner XML-Knoten und dem unkalkulierbaren Mehraufwand, der bei dem manuellen Nachrüsten der fehlenden Funktionalität entsteht, werden sowohl PostgreSQL als auch MariaDB als mögliche Systemkandidaten verworfen.

[93] http://openlife.cc/blogs/2010/october/mariadb-52-using-mariadb-column-store-and-virtual-columns-indexing (besucht am 28.05.2016)
[94] https://bugs.launchpad.net/maria/+bug/608641 (besucht am 28.05.2016)
[95] https://www.citusdata.com/product/citus#pricing (besucht am 29.05.2016)
[96] https://mariadb.com/kb/en/mariadb/spider-storage-engine-overview/ (besucht am 29.05.2016)

4.3 JSON-basierte Document Stores

Nach Ausschluss aller XML-gestützten Ansätze wird als Nächstes die Eignung JSON-basierter Document Stores untersucht. Laut db-engines.com sind MongoDB[97], Couchbase Server[98] und CouchDB[99] die drei am häufigsten genutzten Systeme im nicht-kommerziellen Bereich oder mit dualem Lizenzmodell[100]. Gemäß der aktuellen Gartner-Empfehlung[101] und auf Grund der fehlenden, deklarativen Query-Language von CouchDB, werden nachfolgend ausschließlich MongoDB und Couchbase Server näher untersucht.

4.3.1 Grundlegende Eigenschaften

Beide Systeme verwalten JSON-Dokumente in Form von schemalosen Dokumentensammlungen, sogenannten *Collections* (MongoDB-Sprechweise) oder *Buckets* (Couchbase-Bezeichnung). Für den späteren Zugriff erhält jedes Dokument eine eindeutige Dokumenten-ID, über die es direkt von der Anwendung angesprochen werden kann. Die Reaktionszeit bei direkten Key-Value-Zugriffen, bzw. bei der Ausführung von CRUD[102]-Operationen, ist sowohl bei MongoDB als auch bei Couchbase äußerst gering (einstelliger Millisekunden-Bereich und kleiner) und skaliert, in Abhängigkeit von der verwendeten Hardware, bis zu mehreren 100.000 Operationen pro Sekunde[103]. Mit dem Aggregation-Framework[104] von MongoDB und Couchbase N1QL[105] stehen neben CRUD-Zugriffen auch deklarative Abfragesprachen zur Verfügung, deren Ausführung gezielt durch das Setzen von Indizes auf einzelnen JSON-Keys optimiert werden kann. Beide Produkte stellen dafür eine Vielzahl verschiedener Indexstrukturen zur Verfügung[106,107]. Zur transparenten Datenkompression der verwalteten BSON- bzw. JSON-Dokumente setzt sowohl MongoDB als auch Couchbase den Snappy-Algorithmus[108] von Google ein[109,110]. Weiterhin ist die Systemarchitektur beider Systeme inhärent auf horizonta-

[97] https://www.mongodb.com/ (besucht am 23.06.2016)
[98] http://www.couchbase.com/nosql-databases/couchbase-server (besucht am 23.06.2016)
[99] http://couchdb.apache.org/ (besucht am 23.06.2016)
[100] http://db-engines.com/de/ranking/document+store (besucht am 23.06.2016)
[101] https://www.gartner.com/doc/reprints?id=1-2PMFPEN&ct=151013 (besucht am 23.06.2016)
[102] CRUD ist ein Akronym und steht für die Datenbankoperationen *create*, *read*, *update* und *delete*.
[103] http://info.couchbase.com/rs/northscale/images/couchbase_benchmark.pdf (besucht am 23.06.2016)
[104] https://docs.mongodb.com/manual/meta/aggregation-quick-reference/ (besucht am 23.06.2016)
[105] http://www.couchbase.com/n1ql (besucht am 23.06.2016)
[106] https://docs.mongodb.com/manual/indexes/#index-types (besucht am 24.06.2016)
[107] http://developer.couchbase.com/documentation/server/4.5/concepts/indexing.html (besucht am 24.06.2016)
[108] http://google.github.io/snappy/ (besucht am 02.07.2016)
[109] https://www.mongodb.com/blog/post/new-compression-options-mongodb-30 (besucht am 02.07.2016)
[110] http://blog.couchbase.com/couchbase-103-q-and-a (besucht am 02.07.2016)

le Skalierung ausgelegt, wodurch die Verteilung von *Collections* (*Buckets*) über mehrere Serverinstanzen[111,112], sowie die Verwaltung großer Datenbestände im Terabytebereich[113,114] ermöglicht wird.

4.3.2 Funktionale Eignung

Trotz vieler konzeptioneller Gemeinsamkeiten unterscheiden sich Couchbase und MongoDB grundlegend in ihren Möglichkeiten zur Datenabfrage.

Couchbase Server bietet, neben dem Key-Value-basierten Zugriff, zwei High-Level-Konzepte zur Abfrage von Daten an. Zum Einen N1QL[115], eine deklarative, an SQL angelehnte Query-Language, zum Anderen eine Art materialisierter, mit JavaScript-Funktionen definierter Sichten, genannt MapReduce-Views[116].

N1QL Die grundlegenden Sprachelemente von N1QL (ausgesprochen „Nickel“) werden auf der Herstellerseite[117] in neuen kurzen Beispielen erläutert und mit ihren zugehörigen SQL-Gegenstücken verglichen. Eine ausführliche Beschreibung aller Sprachkonstrukte liefert die N1QL-Language-Reference[118]. Wie in der Gegenüberstellung auf der Projektseite[117] zu sehen, ist die Syntax und Semantik von SQL und N1QL sehr ähnlich. An einigen Stellen wurde N1QL jedoch um wichtige Konzepte für den Umgang mit JSON-spezifischen Konstrukten erweitert. Eines dieser Konzepte sind die Colleciton-Operatoren *IN* und *WITHIN*, die in Kombination mit den Range-Prädikaten *ANY* und *EVERY* die Suche in tief verschachtelten JSON-Arrays ermöglichen[119]. Die fiktive Query

```
1  SELECT name
2  FROM contacts
3  WHERE ANY child IN children
4  SATISFIES child.age > 14 END
```

[111] https://docs.mongodb.com/manual/core/sharding-introduction/ (besucht am 24.06.2016)
[112] http://developer.couchbase.com/documentation/server/current/concepts/distributed-data-management.html (besucht am 24.06.2016)
[113] https://docs.mongodb.com/manual/reference/limits/ (besucht am 24.06.2016)
[114] http://www.couchbase.com/customers/paypal.html (besucht am 24.06.2016)
[115] http://developer.couchbase.com/documentation/server/4.5/getting-started/first-n1ql-query.html (besucht am 24.06.2016)
[116] http://developer.couchbase.com/documentation/server/4.5/architecture/querying-data-with-views.html (besucht am 24.06.2016)
[117] www.couchbase.com/n1ql (besucht am 24.06.2016)
[118] http://developer.couchbase.com/documentation/server/4.5/n1ql/n1ql-language-reference/index.html#n1ql-lang-ref (besucht am 24.06.2016)
[119] http://developer.couchbase.com/documentation/server/4.5/n1ql/n1ql-language-reference/collectionops.html (besucht am 24.06.2016)

untersucht den Inhalt des *Buckets contacts* auf JSON-Dokumente, die auf oberster Ebene einen JSON-Array *children* besitzen. Mit *ANY* und *IN* wird der direkte Inhalt des JSON-Arrays *children* nach einem JSON-Objekt mit dem Schlüssel *age* durchsucht und geprüft, ob dessen Wert größer als 14 ist. Existiert mindestens ein solches Objekt gibt *ANY* „true“ zurück und es wird der Wert des *name* Schlüssels aller JSON-Dokumente im *Bucket contacts* ausgegeben, die das formulierte Suchkriterium erfüllen. Als Ergebnis kommt ein neues JSON-Objekt mit einem Array *results* zurück, welches alle gefundenen Ergebnisse als JSON-Objekte enthält.

```
{
  ...
  "results": [ { "name": "dave" }, ... ],
  ...
}
```

Durch das Ersetzen von *IN* mit *WITHIN* werden nicht nur die direkten Inhalte von *children*, sondern auch der Inhalt aller beinhalteten Objekte gegen das in *SATISFIES* formulierte Kriterium, geprüft. Dass es sich dabei um JSON-Objekte und nicht etwa einfache Werte handelt, wird implizit durch den Ausdruck *child.age* innerhalb von *SATISFIES* festgelegt.

Ausgehend von dem unter 3.1.1 beschriebenen Datenformat und der Annahme, dass sich alle Artikel sämtlicher Wikipediainstanzen in einem *Bucket articles* befinden, lässt sich die Analyse der Artikel, bei Einsatz der unter 3.2 festgelegten Suchparameter, wie folgt umsetzen; Die Unterscheidung der einzelnen Knotentypen erfolgt über den *!type* Schlüssel, Knotenattribute sind durch ein *@* gekennzeichnet und der Knoteninhalt befindet sich im *!value* eines der in *!children* enthaltenen *#text* Knoten. Der jeweilige Kontenkontext ist implizit durch den Inhalt der *!children* Arrays gegeben und wird mit dem N1QL *IN* oder *WITHIN* Statement analysiert. Die Query

> „Liste die Dokumenten-ID der letzten Version aller Artikel der englischen Wikipedia, die Absätze der Ebene zwei, mit fett gedrucktem Fließtext und einer Überschrift mit der Zeichenfolge *X*, beinhalten“

verwendet alle im Kontext der Queryanalyse geforderten Suchparameter und kann wie folgt formuliert werden:

```
SELECT meta(articles).id
FROM articles
WHERE @lang = "en"
      AND ANY section WITHIN !children
          SATISFIES section.!type = "section"
          AND section.@level = 2
          AND ANY heading IN section.!children
              SATISFIES heading.!type = "heading"
              AND ANY text WITHIN heading.!children
                  SATISFIES text.!type = "#text"
                  AND CONTAINS (text.!value, X)
                  END
              END
          AND ANY sectionBody IN section.!children
              SATISFIES sectionBody.!type = "body"
              AND ANY bold IN sectionBody.!children
                  SATISFIES bold.!type = "b"
                  END
              END
```

```
20            END
21    GROUP BY @title
22    HAVING @articleVersion = MAX (@articleVersion)
```

Die Granularität des Abfrageergebnisses wird dabei durch die SELECT-Anweisung festgelegt. Soll statt der Dokumenten-ID der Inhalt der gefundenen Absätze als Teildokument zurückgegeben werden, muss die SELECT-Anweisung wie folgt abgeändert werden:

```
1    SELECT !children
2    ...
```

Die Rückgabe der kompletten Artikel geschieht mit:

```
1    SELECT *
2    ...
```

Für die Suche entlang der zeitlichen Achsen werden entweder das *WHERE* Statement, in Kombination mit dem *title* (Längsschnitt), oder *GROUP BY* und *HAVING*, zusammen mit dem *@timestamp* oder der *@articleVersion* (Querschnitt), verwendet. Die Formulierung aktueller Queries bzw. eines Querschnitts über die aktuelle Version aller Artikel der englischen Wikipedia wurde bereits im vorherigen Absatz demonstriert. Die Ausgabe aller vom 01.01.2016 bis 07.01.2016 erstellten Artikelversionen erhält man mit

```
1    SELECT *
2    FROM articles
3    WHERE @lang = "en"
4          AND STR_TO_MILLIS (@timestamp)
5          BETWEEN STR_TO_MILLIS ("2016-01-01") AND STR_TO_MILLIS ("2016-01-07")
```

wobei angenommen wird, dass *@timestamp* in einem Couchbase-kompatiblen Datumsformat[120] vorliegt. Historische Queries, bzw. den Längsschnitt der kompletten Versionshistorie eines Artikels (in diesem Beispiel des Artikels „Germany“), liefert die folgende Abfrage:

```
1    SELECT *
2    FROM articles
3    WHERE @title = "Germany"
4          AND @lang = "en"
```

MapReduce-Views sind benutzerdefinierte, inkrementell angelte Indexstrukturen, die den einfachen und effizienten Zugang zu häufig benötigten Daten ermöglichen. Jede *View* ist exakt einem *Bucket* zugeordnet. Es ist nicht möglich mit einer *View* auf Daten verschiedener *Buckets* zuzugreifen[121].

MapReduce-Views, kurz *Views*, werden mit zwei benutzerdefinierten, serverseitigen JavaScript-Funktionen, Map und Reduce, erzeugt. Die Map-Funktion stellt ein „Mapping“

[120] http://developer.couchbase.com/documentation/server/current/n1ql/n1ql-language-reference/datefun.html (besucht am 25.06.2016)

[121] http://developer.couchbase.com/documentation/server/4.1/developer-guide/views-operation.html (besucht am 25.06.2016)

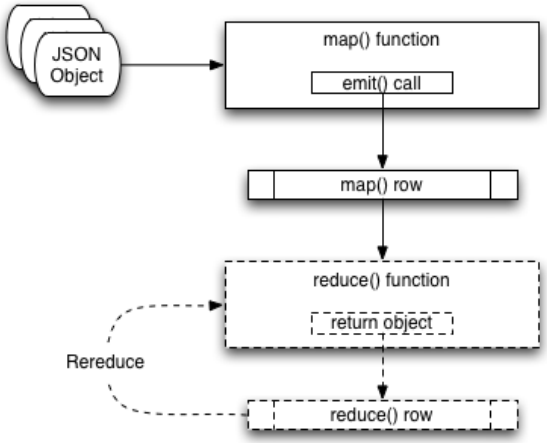


Abbildung 6: Couchbase – Teilschritte beim Erstellen einer *View* (Bildquelle siehe[122])

zwischen den Eingabedaten (den Daten innerhalb eines *Buckets*) und den Ausgabedaten (dem Inhalt der View) dar und legt somit deren Inhalt fest. Die Map-Funktion wird beim Erstellen der *View* für jedes Dokument des zugehörigen Couchbase-Buckets einmal aufgerufen. Werden neue Dokumente in den *Bucket* eingefügt oder bestehende Einträge modifiziert, wird der Vorgang inkrementell, ausschließlich für die geänderten Dokumente, wiederholt[122].

Über die Reduce-Funktion können Aggregationen auf den, in der Map-Phase extrahierten Daten, ausgeführt werden. Die Angabe einer Reduce-Funktion ist optional. Wurde eine Reduce-Funktion definiert, wird sie beim Zugriff auf die *View* automatisch aufgerufen. Dieses Verhalten kann über das Setzen eines Parameters („reduce=false“) geändert werden[122]. Die einzelnen Teilschritte beim Erstellen einer View werden in Abbildung 6 veranschaulicht.

Der wichtigste Teil der View-Definition ist die Map-Funktion. Die Map-Funktion hat grundsätzlich folgende Signatur:

```
1    function(doc, meta) { emit(key, [value1, value2, ... , valueN]); }
```

Der Übergabeparameter *doc* enthält ein JSON- oder Binär-Dokument des zugehörigen Couchbase-Buckets. Über *meta* können Meta-Informationen zu dem übergebenen Dokument abgerufen werden. Das Meta-Objekt enthält folgende Schlüssel[123]:

- *id* := Die eindeutige Dokumenten-ID des übergebenen Dokuments.
- *rev* := Eine interne Versions-ID, die von Couchbase zur Verwaltung unterschiedlicher Dokumentenversionen verwendet wird. Die Werte von *rev*

[122] http://developer.couchbase.com/documentation/server/4.1/developer-guide/views-writing.html (besucht am 26.06.2016)
[123] http://developer.couchbase.com/documentation/server/4.1/developer-guide/views-stored-data.html (besucht am 26.06.2016)

sind nicht konsistent und sollten daher nicht direkt in der Anwendung verwendet werden.

- *type* := Der Typ des übergebenen Dokuments. Der Wert „json“ weist auf ein JSON-Dokument hin, „base64“ steht für ein Binärdatum.
- *flags* := Ein Flag, das beim Speichern des übergebenen Dokuments gesetzt wurde. Der Wert ist abhängig von der verwendeten Client-Library.
- *expiration* := Gültigkeitsdauer des übergebenen Dokuments als Unix-Zeitstempel.

Der Inhalt der *View* wird über die *emit*-Funktion festgelegt. Jeder Aufruf von *emit* erzeugt einen neuen Eintrag in der *View*, wobei *emit* beliebig of innerhalb der Map-Funktion aufgerufen werden kann.

Die *emit*-Funktion erwartet zwei Parameter, einen *key*, der für das Sortieren und die Auswahl bestimmter Inhalte der *View* verwendet wird, sowie einen *value*, der aggregierbare Nutzdaten enthält. Als *key* und *value* können beliebige JSON-Datenstrukturen, angefangen bei einfachen Strings, über Arrays, bis hin zu verschachtelten JSON-Objekten, verwendet werden. Auch die Angabe eines leeren *values* ist möglich. Egal wie *key* und *value* formatiert sind, das View-System fügt jedem Eintrag der *View* die eindeutige Dokumenten-ID des Quelldokuments hinzu. Auf diese Weise muss die Dokumenten-ID nicht manuell angegeben werden und die Datensätze sind unabhängig von den verwendeten Werten für *key* und *value* eindeutig identifizierbar. Der *key* selbst darf nicht leer sein[122]. Abbildung 7(a) veranschaulicht den Aufruf einer fiktiven Map-Funktion, die Dokumente mit Mitarbeiterstammdaten verarbeitet.

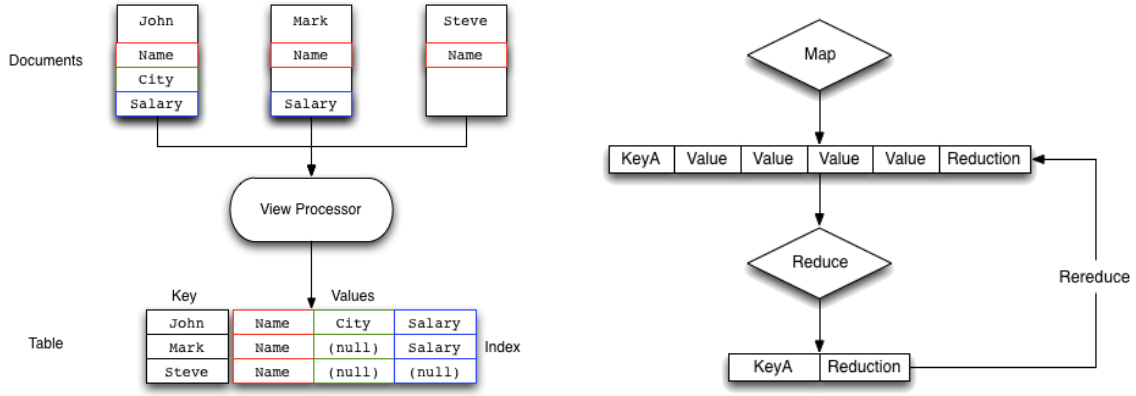
```
1    function(doc, meta) { emit(doc.name, [doc.city, doc.salary]); }
```

Die erzeugte *View* enthält den Nachnamen der Mitarbeiter als Schlüssel und ordnet jedem/er Angestellten die Stadt in der er/sie wohnt und sein/ihr Gehalt zu. Mit Hilfe dieser *View* kann das Gehalt aller MitarbeiterInnen, deren Nachname mit „A“ beginnt, ausgegeben werden.

Die Signatur der Reduce-Funktion ist wie folgt Aufgebaut:

```
1    function(key, values, rereduce)
2    {
3      ...
4      return result;
5    }
```

Der *key* der Reduce-Funktion leitet sich aus dem, im Map-Schritt verwendeten, Schlüssel der *emit*-Funktion ab. Wurden mehrwertige Schlüssel, wie z.B. ein JSON-Array, verwendet, wird der Aufbau des *keys* zusätzlich durch die Angabe des *group_level*-Parameters, beim Zugriff auf die *View*, beeinflusst. Über das Setzen des *group_levels* kann festgelegt werden welche Teile, des im vorangegangenen Map-Schritt angegebenen Schlüssels, für die Gruppierung der Daten und damit als *key* der Reduce-Funktion verwendet werden sollen. Bei Angabe eines JSON-Arrays mit Datumswerten der Form „[Jahr,Monat,Tag]“



(a) Map-Schritt (b) (Re)reduce-Schritt

Abbildung 7: Couchbase – Die Map- und Reduce-Operationen (Bildquelle siehe[122])

ergeben sich folgende Gruppierungsmöglichkeiten[122,124]:

- *group_level* = 0 $\longmapsto$ Gruppierung auf Basis des kompletten Arrays.
- *group_level* = 1 $\longmapsto$ Gruppierung nach Jahren.
- *group_level* = 2 $\longmapsto$ Gruppierung nach Jahren und Monaten.
- *group_level* = 3 $\longmapsto$ Gruppierung nach Jahren, Monaten und Tagen (hat in diesem Fall den selben Effekt wie das Setzen von *group_level* = 0).

In den *values* werden, unter Berücksichtigung des *group_level*-Parameters, alle im Map-Schritt ausgewählten Datensätze mit dem selben *emit key* zusammengefasst. Über den *rereduce*-Parameter kann ermittelt werden, ob es sich bei dem Reduce-Aufruf um den initialen Funktionsaufruf nach Erstellen der *View* oder um eine inkrementelle Aktualisierung, einen sogenannten „rereduce“, handelt. Im Fall von „*rereduce*=false“ enthält der *values*-Parameter ein Array mit allen, dem aktuellen *key* im Map-Schritt zugeordneten, Einzelwerten. Bei „*rereduce*=true“ beinhaltet der *values*-Parameter ebenfalls ein Array, diesmal jedoch mit den Ergebnissen der vorangegangenen Reduce-Schritte[122].

Für die Definition einer fiktiven *View* mit den Nachnamen der MitarbeiterInnen einer Firma als *key* und den Vornamen als *value*, ergibt sich folgende Map-Funktion:

```
1    function(doc, meta) { emit(doc.lastName, doc.firstName); }
```

Die Reduktion der vorangegangenen *View* auf eine Liste der häufigsten Mitarbeiternachnamen, zusammen mit deren Häufigkeit, kann mit folgender Reduce-Funktion erreicht werden:

```
1    function(key, values, rereduce)
2    {
```

[124] http://developer.couchbase.com/documentation/server/4.1/developer-guide/views-querying.html (besucht am 27.06.2016)

```
3       if (rereduce)
4       {
5         var result = 0;
6         for (var i = 0; i < values.length; i++)
7         {
8           result += values[i];
9         }
10        return result;
11      }
12      else
13      {
14        return values.length;
15      }
16    }
```

Eine grafische Visualisierung der Funktionsweise von *Reduce* und *Rereduce* liefert Abbildung 7(b).

Die View-Definitionen werden in Couchbase in sogenannten Design-Dokumenten („design documents“) hinterlegt. Jedes dieser Design-Dokumente ist einem *Bucket* zugeordnet. Die *Views* innerhalb eines Design-Dokuments bilden eine Einheit. Wird eine *View* auf Grund von Änderungen des zugehörigen *Buckets* aktualisiert, werden alle *Views* im selben Desgin-Dokuments mit aktualisiert, unabhängig davon, ob sich die Werte der enthaltenen Daten tatsächlich geändert haben[121].

Der Inhalt einer *View* ist im Vergleich zu den Daten des zugehörigen *Buckets eventually consistent*[125]. Das Aktualisieren einer *View* kann auf zwei verschiedene Arten angestoßen werden. Einmal über die Index-Engine, welche die *Views* eines Design-Dokuments asynchron, in einem vorgegebenen Zeitintervall oder nach Modifikation einer bestimmten Anzahl von Dokumenten aktualisiert, oder client-seitig über das Setzen eines *stale*-Parameters beim Zugriff auf die *View*. Die Werte des Intervalls und die minimale Anzahl modifizierter Bucket-Einträge, ab der der Inhalt einer View aktualisiert wird, kann individuell pro Design-Dokument festgelegt werden. Bei Angabe des *stale*-Parameters kann zwischen den Werten „*stale*=ok“, „*stale*=false“ und „*stale*=update_after“ gewählt werden[121].

Das Setzen von „*stale*=ok“ führt im Allgemeinen zu den niedrigsten Reaktionszeiten, liefert die *View* jedoch in ihrem aktuell vorliegenden Zustand aus. Es besteht daher das Risiko einen veralteten Datenbestand zu lesen. Im Fall von „*stale*=false“ wird die *View* vor dem Zugriff aktualisiert. Auf diese Weise sind die gelesenen Daten garantiert aktuell, es muss jedoch mit potentiellen Verzögerungen gerechnet werden bis die Index-Engine das inkrementelle Indexupdate durchgeführt hat. Bei „*stale*=update_after“ wird zunächst der aktuelle Stand der *View* ausgeliefert und im Anschluss, nach Abarbeitung der Anfrage, der Index aktualisiert. *Update_after* ist die Standardeinstellung und wird verwendet, wenn kein *stale*-Parameter gesetzt ist[121].

Für die Abfrage von Inhalten einer *View* kann ein konkreter *key*, eine Liste von *keys* oder eine *key-range* angegeben werden. Die Ausgabe wird dabei, auf Basis der Unicode-Repräsentation des im Map-Schritt angegebenen *keys*, aufsteigend sortiert (*„123456790*

[125] http://developer.couchbase.com/documentation/server/4.1/developer-guide/views-intro.html (besucht am 25.06.2016)

< aAbBcCdDeEfFgGhH. . . " bzw. *„a < á < A < Á < b"*). Für die Invertierung der Sortierreihenfolge kann der Parameter *„descending=true"* gesetzt werden. Für die range-basierte Abfrage von Inhalten stehen die Parameter *„start_key"* bzw. *„startkey_docid"* und *„endkey"* bzw. *„endkey_docid"* zur Verfügung. Dabei ist zu beachten; die Auswertung der angegebenen Suchbereiche erfolgt nachdem die Ausgabe sortiert wurde. Das Setzen von *„descending=true"* entspricht im Allgemeinen einem Vertauschen des Start- und End-Keys[124].

Der Zugriff auf die *Views* erfolgt über eine REST-API. Die Zugriffs-URL des Endpunkts hat dabei folgenden Aufbau

```
GET http://[localhost]:8092/[bucket-name]/_design/[ddoc-name]/_view/[view-name]
```

wobei *[bucket-name]* für den Namen des *Buckets* steht, *[ddoc-name]* den Namen des Design-Dokuments mit der entsprechenden View-Definition enthält und *[view-name]* den Namen der gewünschten *View* beinhaltet. Für eine vollständige Auflistung aller Zugriffsparameter, inklusive weiterer Abfragebeispiele siehe[124].

Die MapReduce-Views eignen sich hervorragend für die Beschleunigung von Abfragen entlang der unterschiedlichen zeitlichen Achsen. Mit Hilfe der *View*

```
function(doc, meta)
{
  // key format: [year, month, day, hour, minute]
  emit(dateToArray(doc.@timestamp)[0:4], null);
}
```

und der Range-Query *?startkey=[2016,1,1,0,0]&endkey=[2016,1,7,24,0]* können beispielsweise die Dokumenten-IDs aller vom 01.01.2016 bis 07.01.2016 erstellten Artikelversionen abgerufen werden (Querschnitt). Soll neben der Dokumenten-ID auch der Titel und die Versionsnummer angezeigt werden, kann die *View* wie folgt erweitert werden:

```
function(doc, meta)
{
  // key format: [year, month, day, hour, minute]
  emit(dateToArray(doc.@timestamp)[0:4], [doc.@title, doc.@articleVersion]);
}
```

Für die Beschleunigung von aktuellen Queries und Längschnitten eignet sich die *View*

```
function(doc, meta) { emit([doc.@title, doc.@articleVersion], doc.@articleVersion); }
```

wobei für die Analyse der aktuellsten Versionen ein zusätzlicher Reduce-Schritt erforderlich ist. Die Einträge der *View* müssen dabei, durch Setzen des Parameters *group_level=1*, anhand ihres Artikeltitels gruppiert werden.

```
function (key, values, rereduce)
{
  var maxValue = values[0][1];
  var indexOfMaxValue = 0;
  for(var i = 1; i < values.length; i++)
  {
    if(values[i][1] > maxValue)
    {
      maxValue = values[i][1];
      indexOfMaxValue = i;
```

```
11        }
12      }
13      return values[indexOfMaxValue];
14    }
```

Die Abfrage aller Versionen des Artikels „Germany“ erfolgt z.B. über *?startkey=["Germany",-1]&endkey=["Germany",null]&reduce=false*. Die letzte Version aller Artikel der englischen Wikipedia erhält man mit *?startkey=[0,0]&endkey=[null,null]&reduce=true&group_level=1*. Den letzten drei View-Definitionen liegt jeweils das unter 3.1 beschriebenen Datenformat zugrunde.

MongoDB bietet mit der *Aggregation Pipeline*, den *Single Purpose Aggregation Operations* und der in JavaScript geschriebenen MapReduce-Funktionen drei verschiedene Möglichkeiten zur Abfrage von Daten an[126].

Aggregation Pipeline Die *Aggregation Pipeline* ist die von den MongoDB-Entwicklern empfohlene Möglichkeit zur Abfrage von Daten, da alle auf diesem Weg formulierten Queries direkt in native C++ Funktionen übersetzt werden können und im Allgemeinen über die besten Performance-Charakteristiken verfügen. Das *aggregate* Kommando ist wie folgt aufgebaut

```
1  db.runCommand (
2    {
3      aggregate: "<collection>",
4      pipeline: [ <stage>, <...> ],
5      explain: <boolean>,
6      allowDiskUse: <boolean>,
7      cursor: <document>,
8      bypassDocumentValidation: <boolean>,
9      readConcern: <document>
10   }
11 )
```

wobei die Angabe von *explain*, *allowDiskUse*, *cursor*, *bypassDocumentValidation* und *readConcern* optional ist.

Mit *aggregate* wird der Name der Quell-Collection, in der sich die zu analysierenden Daten befinden, festgelegt. Über *explain* wird die Ausgabe von Meta-Informationen zur Abarbeitung der Pipeline-Stufen (wie z.B. die Nutzung von Indizes etc.) aktiviert. Durch das Setzen von *allowDiskUse* können die Pipeline-Operatoren Zwischenergebnisse in ein temporäres Verzeichnis („_tmp“) unterhalb des Datenbankverzeichnisses schreiben. An *cursor* kann ein Dokument mit Parametern zur Definition des Ergebnis-Cursors übergeben werden. *BypassDocumentValidation* deaktiviert die Validierung der Ausgabedokumente, falls diese im letzten Pipeline-Schritt, durch Angabe von *$out*, zurück in eine MongoDB *Collection* geschrieben werden. Der *readConcern* legt das Konsistenz-Level der Abfrageergebnisse fest[127].

[126] https://docs.mongodb.com/manual/aggregation/ (besucht am 28.06.2016)
[127] https://docs.mongodb.com/manual/reference/command/aggregate/ (besucht am 28.06.2016)

Der *readConcern* ist standardmäßig auf „local“ gesetzt, wodurch das Abfrageergebnis die jeweils aktuellste Kopie der Daten des gerade angesprochenen MongoDB-Knotens (Replica-Set-Mitglieds) erhält. Es ist somit nicht sichergestellt, ob die ausgelieferten Daten bereits von einer Mehrheit der Replica-Set-Mitglieder als „geschrieben“ bestätigt wurden und somit bei Ausfall des aktuellen Knotens weiter zur Verfügung stehen. Die zweite Option für das Setzten des *readConcerns* ist „majority“. In diesem Fall wird ebenfalls die aktuellste Kopie der Daten des momentan angesprochenen Knotens ausgeliefert. Vor der Auslieferung wird jedoch sichergestellt, dass eine Mehrheit der Mitglieder des zugehörigen Replica-Sets die Daten als „geschrieben“ bestätigt hat. Das Setzen des *readConcerns* „majority“ ist ausschließlich bei Verwendung der WiredTiger-Storage-Engine möglich und erfordert zusätzlich die Aktivierung des *enableMajorityReadConcern* Konfigurationsparameters bei allen *mongod* Instanzen, sowie den Einsatz des *replica set election protocols*[128] Version 2[129].

Über das *pipeline* Array werden die eigentlichen Verarbeitungsstufen der *Aggregation Pipeline* angegeben und das Abfrageergebnis festgelegt. Zur Auswahl stehen:

- $project
- $match
- $redact
- $limit
- $skip
- $unwind
- $group
- $sample
- $sort
- $geoNear
- $lookup
- $out
- $indexStats

Mit dem in Version 3.2 hinzugefügten *$lookup* Operator ist die *Aggregation Pipeline* die einzige Zugriffsmethode, die gleichzeitig Daten aus mehreren *Collections* verarbeiten kann. $lookup führt einen *left outer join* mit den Dokumenten der in *aggregate* angegebenen *Collection* und den Dokumenten der im *from*-Parameter von $lookup definierten *Collection* durch.

```
1 {
2   $lookup:
3   {
4     from: <collection to join>,
5     localField: <field from the input documents>,
6     foreignField: <field from the documents of the "from" collection>,
7     as: <output array field>
8   }
9 }
```

Das *localField* gibt das Feld der Eingabedokumente an, anhand dessen die Dokumente verbunden werden sollen. In *foreignField* steht das für die Verbundoperation verwendete Feld der in *from* angegebenen Collection-Dokumente. Die Inhatle der Parameter *localField* und *foreignField* werden während der Verbundoperation auf Gleichheit geprüft.

[128] https://docs.mongodb.com/manual/release-notes/3.2/#rel-notes-rs-enhancements (besucht am 29.06.2016)

[129] https://docs.mongodb.com/manual/reference/read-concern/#readconcern.%22local%22 (besucht am 29.06.2016)

Über den *as*-Parameter wird den Eingabedokumenten ein Feld zur Aufnahme der jeweiligen Ergebnisse der Join-Operation hinzugefügt[130]. Die vollständige Beschreibung aller Pipeline-Stufen würde den Rahmen dieses Abschnitts sprengen. Hier wird deshalb auf das Kapitel „Pipeline Aggregation Stages“[131] der MongoDB-Dokumentation verwiesen.

Eine wichtige konzeptionelle Einschränkung der *Aggregation Pipeline* ist die fehlende Möglichkeit zur Verarbeitung von Baumstrukturen. Der JSON-Output von WOM (siehe Listing 3, Kapitel 3.1.1) bildet jedoch eine Baumstruktur, mit einem *article* Knoten als Wurzel und einem Array von Kindkonten (*!children*), die wiederum *!children* Arrays mit weiteren Kindknoten enthalten. Für die Durchführung der unter 3.2 beschriebenen Queries wäre somit ein syntaktisches Konstrukt notwendig, mit dem transitiv, ausgehend vom *article* Knoten, die Kinder und Kindeskinder des *!children* Arrays untersucht werden können, ohne dabei vorher ihre genaue Verschachtellungstiefe zu kennen. Die *Aggregation Pipeline* bietet jedoch keine solche Funktion. Zudem ist die maximale Verschachtellungstiefe der BSON-Dokumente auf 100 begrenzt[132].

Das Benutzerhandbuch von MongoDB nennt eine Anpassung des Datenformats als möglichen Lösungsweg. Laut der dortigen Empfehlungen sollen tief verschachtelte Dokumenten-Bäume in flache Knotenlisten verwandelt werden, wobei jeder Einzelknoten genug Informationen trägt, um seine frühere Position im Baum zu rekonstruieren. Für die Formatkonvertierung werden folgende fünf Verfahren vorgestellt[133]:

1. ***Child References*** Jeder Knoten speichert einen Identifier seiner direkten Kinder. Mit diesem Verfahren kann auf die Eltern und direkten Kinder eines Knotens zurückgegriffen werden. Aussagen über die genaue Position im Baum sind, bei Betrachtung eines einzelnen Knotens, nicht möglich.

2. ***Parent References*** Jeder Knoten speichert einen Identifier seines Elternknotens. So können der jeweilige Elternknoten und die direkten Kinder eines Knotens ermittelt werden. Für die Analyse der Baumstruktur sind jedoch viele Einzelabfragen notwendig.

3. ***Array of Ancestors*** Jeder Knoten speichert ein Array mit den IDs aller seiner Vorfahren, bis hin zum Wurzelknoten. Mit dieser Methode können durch Betrachtung eines einzelnen Knotens alle seine Vorfahren und damit seine Position im Baum ermittelt werden. Weiterhin kann, bei gegebener Vorfahrenliste, der komplette Teilbaum mit allen Kinder eines Knotes abgefragt werden. Der Einsatz dieser Methode erfordert eine relativ statische Baumstruktur, da bei Replatzierung eines Kontens im Baum die Vorfahrenliste aller alten und neuen Kinder angepasst werden muss. Das

[130] https://docs.mongodb.com/manual/reference/operator/aggregation/lookup/#pipe._S_lookup (besucht am 28.06.2016)
[131] https://docs.mongodb.com/manual/reference/operator/aggregation-pipeline/ (besucht am 28.06.2016)
[132] https://docs.mongodb.com/manual/reference/limits/#bson-documents (besucht am 29.06.2016)
[133] https://docs.mongodb.com/manual/tutorial/model-tree-structures/ (besucht am 29.06.2016)

Beispiel im MongoDB Handbuch speichert zusätzlich die ID des direkten Elternknotens. Diese Information ist redundant und steckt im letzten Element des Vorfahren-Arrays.

4. ***Materialized Paths*** Jeder Knoten speichert einen, mit einem beliebigen *Separator* getrennten, String der IDs seiner Vorfahren. Für dieses Verfahren gelten die selben Rahmenbedingungen wie für den Einsatz eines *Array of Ancestors*. Zusätzlich wird das Durchsuchen der Vorfahrenpfade mit regulären Ausdrücken ermöglicht. Auf diese Weise können neben absoluten auch relative Vorfahrenpfade für die Knotenabfrage verwendet werden.

5. ***Nested Sets*** Jeder Knoten speichert den Identifier seines Elternknotens und zwei Stop-Zähler, die während einer *Depth-First-Traversel* des Baums berechnet werden. Der Stop-Zähler wird nach dem Abstieg in einen Kindknoten und vor dem Aufstieg in den jeweiligen Elternknoten inkrementiert. Beim Abstieg wird der Wert des Stop-Zählers in die Variable *left* des neu erreichten Kindknotens geschrieben. Die *right* Variable wird vor dem Aufstieg in den Elternknoten mit dem Wert des Stop-Zählers initialisiert. Mit diesem Verfahren können in einer Abfrage sowohl der Elternknoten, als auch die Kinder und Kindeskinder eines bestimmten Knotens ermittelt werden. Voraussetzung ist auch hier eine statische Baumstruktur.

Die beispielhafte Anwendung aller Verfahren zeigen die Listings 23 bis 27.

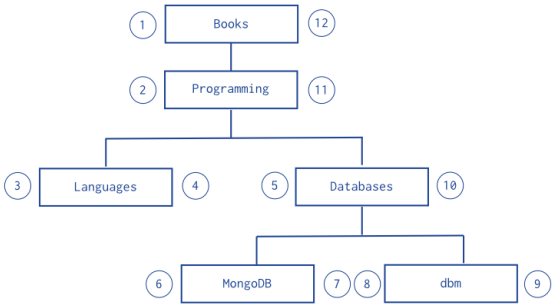


Abbildung 8: Baumstruktur mit verschiedenen Bücherkategorien

```
1  { _id: "Books", children: ["Programming"] },
2  { _id: "Programming", children: ["Languages", "Databases"] },
3  { _id: "Languages", children: [] },
4  { _id: "Databases", children: ["MongoDB", "dbm"] },
5  { _id: "MongoDB", children: [] },
6  { _id: "dbm", children: [] }
```

Listing 23: Child References

```
1  { _id: "Books", parent: null },
2  { _id: "Programming", parent: "Books" },
3  { _id: "Languages", parent: "Programming" },
4  { _id: "Databases", parent: "Programming" },
5  { _id: "MongoDB", parent: "Databases" },
6  { _id: "dbm", path: "Databases" }
```

Listing 24: Parent References

```
1  { _id: "Books", ancestors: [] },
2  { _id: "Programming", ancestors: ["Books"] },
3  { _id: "Languages", ancestors: ["Books","Programming"] },
4  { _id: "Databases", ancestors: ["Books","Programming"] },
5  { _id: "MongoDB", ancestors: ["Books","Programming", "
       ↪ Databases"] },
6  { _id: "dbm", ancestors: ["Books","Programming", "Databases"]
       ↪ }
```

Listing 25: Array of Ancestors

```
1  { _id: "Books", path: null },
2  { _id: "Programming", path: ",Books," },
3  { _id: "Languages", path: ",Books,Programming," },
4  { _id: "Databases", path: ",Books,Programming," },
5  { _id: "MongoDB", path: ",Books,Programming,Databases," },
6  { _id: "dbm", path: ",Books,Programming,Databases," }
```

Listing 26: Materialized Paths

```
1  { _id: "Books", parent: null, left: 1, right: 12 },
2  { _id: "Programming", parent: "Books", left: 2, right: 11 },
3  { _id: "Languages", parent: "Programming", left: 3, right: 4
       ↪ },
4  { _id: "Databases", parent: "Programming", left: 5, right: 10
       ↪ },
5  { _id: "MongoDB", parent: "Databases", left: 6, right: 7 },
6  { _id: "dbm", parent: "Databases", left: 8, right: 9 }
```

Listing 27: Nested Sets

Für den vorliegenden Anwendungsfall wurde eine Kombination aus *Materialized Paths* und *Array of Ancestors* verwendet. Jeder Knoten erhält zunächst eine dokumentenlokale Knoten-ID („*!nodeId*), über die er im Kontext des jeweiligen WOM-Dokuments eindeutig identifizierbar ist. Für die Ausführung kontexbasierter Suchen wird der *Materialized Path* der Knotentypen aller Vorfahren in einem String *!ancestorTypes* gespeichert. So kann durch die Anwendung eines einzelnen regulären Ausdrucks ermittelt werden, ob es beispielsweise Tabellen in dem aktuellen Dokument gibt, die als direktes oder indirektes Kind einen Link enthalten oder, ob Absätze mit fett gedrucktem Text existieren. Die genaue Position der Knoten im Baum wird über ein *Array of Ancestors*, mit den IDs aller Vorfahrenknoten, ermittelt. Die *Ancestor-IDs* dienen so als Rückreferenz zu allen Vorfahrenknoten und ersetzen die fehlenden Traversierungs-Operationen. Auch können über die Vorfahren-IDs eindeutige Eltern-Kind-Beziehungen abgelesen werden.

Listing 28 zeigt die Anwendung der beschriebenen Modifikationen auf den JSON-Output der in Listing 3 abgebildeten „Demo Seite“.

```
{
  "!type": "article",
  "!nodeId": "0",
  "!ancestorIds": [],
  "!ancestorTypes": null,
  "@xmlns": "http://sweble.org/schema/wom30",
  "@version": "3.0",
  "@title": "SimplePage",
  "@articleVersion": "13",
  "@timestamp": "2016-06-17T14:19:47.688+02:00",
  "@lang": "en",
  "@xmlns:mww": "http://sweble.org/schema/mww30",
  "nodes": [
    {
      "!type": "body",
      "!nodeId": 1,
      "!ancestorIds": [0],
      "!ancestorTypes": ",article,",
    },
    {
      "!type": "section",
      "!nodeId": 2,
      "!ancestorIds": [0,1],
      "!ancestorTypes": ",article,body,",
      "@level": "2",
    },
    {
      "!type": "heading",
      "!nodeId": 3,
      "!ancestorIds": [0,1,2],
      "!ancestorTypes": ",article,body,section,",
    },
    {
      "!type": "rtd",
      "!nodeId": 4,
      "!ancestorIds": [0,1,2,3],
      "!ancestorTypes": ",article,body,section,heading,",
    },
    {
      "!type": "#text",
      "!nodeId": 5,
      "!ancestorIds": [0,1,2,3,4],
      "!ancestorTypes": ",article,body,section,heading,rtd,",
      "!value": "\u003d\u003d"
    },
    {
      "!type": "text",
      "!nodeId": 6,
      "!ancestorIds": [0,1,2,3],
      "!ancestorTypes": ",article,body,section,heading,",
    },
    {
      "!type": "#text",
      "!nodeId": 7,
      "!ancestorIds": [0,1,2,3,6],
      "!ancestorTypes": ",article,body,section,heading,text,",
      "!value": " Demo Seite ",
    },
    {
      "!type": "rtd",
      "!nodeId":8,
      "!ancestorIds": [0,1,2,3],
      "!ancestorTypes": ",article,body,section,heading,",
    },
    {
      "!type": "#text",
      "!nodeId": 9,
      "!ancestorIds": [0,1,2,3,8],
      "!ancestorTypes": ",article,body,section,heading,rtd,",
      "!value": "\u003d\u003d",
    },
    {
      "!type": "body",
      "!nodeId": 10,
      "!ancestorIds": [0,1,2],
      "!ancestorTypes": ",article,body,section,",
    },
    {
      "!type": "text",
      "!nodeId": 11,
      "!ancestorIds": [0,1,2,3,10],
      "!ancestorTypes": ",article,body,section,body,",
    },
    {
      "!type": "#text",
      "!nodeId": 12,
      "!ancestorIds": [0,1,2,3,10,11],
      "!ancestorTypes": ",article,body,section,body,text,",
      "!value": "\n\n"
    },
    {
      "!type": "p",
      "!nodeId": 13,
      "!ancestorIds": [0,1,2,3,10,11],
      "!ancestorTypes": ",article,body,section,body,",
      "@topgap": "0",
      "@bottomgap": "0",
    },
    {
      "!type": "text",
```

Listing 28: Demo Seite – flache WOM-JSON-Repräsentation (gekürzt auf 100 Zeilen).

Nach Anpassung des Datenformats kann auch mit den Möglichkeiten der *Aggregation Pipeline*, die, während der Diskussion der funktionalen Eignung von Couchbase, eingeführte Referenz-Query

> „Liste die Dokumenten-ID der letzten Version aller Artikel der englischen Wikipedia, die Absätze der Ebene zwei, mit fett gedrucktem Fließtext und einer Überschrift mit der Zeichenfolge *X* beinhalten."

ausgeführt werden. Die Abfrage zerfällt dabei in mehrere Einzelschritte und wird auch nicht allein mit den Mitteln der *Aggregation Pipeline* umgesetzt, sondern benötigt stückweise prozedurale Programmlogik für die Verarbeitung von Zwischenergebnissen. Zum besseren Vergleich der beiden Systeme wird ebenfalls von einer einzelnen *Collection* „*articles*" ausgegangen, in der die Einzelversionen aller Artikel sämtlicher Wikipedia-Instanzen verwaltet werden[134].

Listing 29 zeigt den ersten Teil der Abfrage und ermittelt die letzte Version aller Artikel der englischen Wikipedia, die eine Absatzüberschrift mit der Zeichenfolge „X" enthalten. Pro gefundenem Dokument wird die Dokumeten-ID, sowie die Knoten-ID jeder Überschrift die das Suchkriterium erfüllt, als auch die Liste der Knoten-IDs aller ihrer Vorfahren zurückgegeben. Das JSON-Format des Ergebnisses ist zusätzlich als Kommentar in Zeile 1 beschrieben.

```
1  // [ { _id: <documentId>, nodes: { !nodeId: <nodeId>, !ancestorIds: [<id>, <id> ...] } } ]
2  var headings = db.articles.aggregate ( [
3    { $match :  { @lang : 'en' } }, // Suche auf Artikel der englischen Wikipedia beschränken
4    {
5      $group :  { _id: "!title",
6             latestVersion: { $max: "@articleVersion" }
7           }
8    }, // aktuellste Version der jeweiligen Artikel ermitteln
9    { $match :  { @articleVersion : latestVersion } }, // Dokumentenmenge auf die jeweils
         ↪ aktuellste Version der Artikel beschränken
10   { $unwind: {  path: "$nodes" } }, // Self-Join des article Knotes mit dem beinhalteten nodes
         ↪  array
11   {
12     $match :  { $and: [
13               { nodes.!type : '#text' },
14               { nodes.!ancestorTypes : /.*,section,heading,text,$/ },
15               { nodes.!value : /.*X.*/ }
16             ]
17           }
18   }, // Dokumentenmenge auf Dokumente beschränken, die eine Absatzüberschriften mit der
         ↪ Zeichenfolge 'X' enthalten
19   {
20     $project : { _id : 1, "nodes.!nodeId": 1, "nodes.!ancestorIds": 1 }
21   } // Dokumenten-ID, Knoten-ID und Liste der Knoten-IDs aller Vorfahren jeder gefundenen Ü
         ↪ berschrift auswählen
22 ]);
```

Listing 29: Pseudocode der Verarbeitung einer komplexen Abfrage mit der *Aggregation Pipeline* – Teil 1

[134] Dieses Vorgehen dient ausschließlich dem besseren Vergleich von MongoDB und Couchbase und ist nicht als Empfehlung zu verstehen. Für weitere Einzelheiten siehe Absatz 4.3.3

Im nächsten Schritt wird für jede gefundene Überschrift aller Einzeldokumente überprüft, ob es sich um eine Absatzüberschrift der Ebene zwei handelt und ob der zugehörige Absatz fett gedruckten Text enthält. Wurde ein solcher Absatz gefunden, wird die Dokumenten-ID der umschließenden Artikelversion zur Ergebnismenge hinzugefügt und in nachfolgenden Iterationen nicht weiter betrachtet.

```
1  var documentIdSet = new Set();
2
3  for (var i = 0; i < headings.length; i++)
4  {
5    var documentId = headings[i]._id;
6    if (documentIdSet.has(documentId))
7    {
8      // Wir wissen bereits, dass dieses Dokument ein valides Ergebnis ist.
9      continue;
10   }
11
12   var arrayOfAncestorIds = headings[i].nodes.!ancestorIds;
13   /*
14    * in der ersten Query sind wir bis zum textuellen Inhalt der Überschrift abgestiegen. Auf
        ↪ Grund des Datenmodells von WOM wissen wir, dass die zugehörige Section drei Ebenen h
        ↪ öher im Baum zu finden ist (vgl. Zeile 56 und Zeile 30 Listing 9)
15    */
16   var sectionNodeId = arrayOfAncestorIds[arrayOfAncestorIds.length - 3];
17   // Alle Vorfahren-IDs bis einschließlich des ermittelten Absatzes
18   var acestorIdsUpToSection = arrayOfAncestorIds.slice(0,arrayOfAncestorIds.length - 3);
19
20   var queryResult = db.articles.findOne (
21     {
22       $and: [
23         { _id: documentId },
24         { nodes: $elemMatch: { !nodeId: sectionNodeId, @level: 2 } }, // Test ob die Absatzü
             ↪ berschrift wirklich eine Überschrift der Ebene 2 ist.
25         { nodes: $elemMatch:
26           {
27             !nodeType: 'b',
28             !ancestorIds: { $all: acestorIdsUpToSection }
29           }
30         } // Testet ob fett geschriebener Text in dem gefundenen Absatz existiert.
31       ]
32     },
33     { _id: 1 }
34   );
35
36   // Testet ob fett geschriebener Text in dem gefundenen Absatz existiert.
37   if (queryResult != null)
38   {
39     documentIdSet.add(documentId);
40   }
41 }
```

Listing 30: Pseudocode der Verarbeitung einer komplexen Abfrage mit der *Aggregation Pipeline* – Teil 2

Für die zweite Teilabfrage wurde die CRUD-Operation *findeOne*[135], in Kombination mit den Array-Operatoren *$elemMatch*[136] und *$all*[137], verwendet. Falls ausschließlich auf die Mittel der *Aggregation Pipeline* zurückgegriffen werden soll, kann der Aufruf von *findeOne* durch die Pipeline-Stufen *$match*[138], *$limit*[139] und *$project*[140] ersetzt werden.

Listing 29 und 30 zeigen; die Formulierung von Abfragen mit dem kompletten unter 3.2 geforderten Parameterraum ist auch mit den Mitteln der *Aggregation Pipeline* möglich. Aus der Notwendigkeit zur Aufteilung in mehrere Einzelabfragen ergibt sich, je nach Selektivität der ersten Teilabfrage, eine mehr oder weniger große Anzahl an Round-Trips-Operationen zwischen Client und Server. Diese wirken sich potentiell negativ auf die Abfrage-Performance aus. Ein möglicher Weg zur Vermeidung der Round-Trips ist der Einsatz der Eval-Funktion[141], mit deren Hilfe die komplette JavaScript-Logik, inklusive der Collction-Zugriffe, serverseitig ausgeführt werden kann. Die Verwendung von *eval* ist allerdings seit MongoDB Version 3.0 *deprecated* und es muss mit dem Wegfallen der Funktionalität in zukünftigen MongoDB-Versionen gerechnet werden.

Die Granularität des Abfrageergebnisses ergibt sich aus der Projektionsanweisung der zweiten Teilabfrage (Zeile 33 Listing 30). In der bisherigen Version wird die Dokumenten-ID zurückgegeben. Diese ist bereits aus der ersten Teilabfrage bekannt und wird daher nicht weiter verarbeitet. Wird Zeile 33 weggelassen und damit eine leere Projektionsanweisung übergeben, gibt *findOne* den kompletten Artikel zurück. Für die Rückgabe aller ermittelten Absätze als Teilbäume muss zunächst der Code in Zeile 6 bis 10 entfernt und anschließend die Projektion in Zeile 33 wie folgt modifiziert werden:

```
1    { _id: 1,
2      nodes: $elemMatch: { !ancestorIds: { $all: acestorIdsUpToSection } }
3    }
```

Für die Suche entlang der zeitlichen Achsen wird entweder der einleitende *$match* Schritt oder eine Kombination der folgenden *$group* und *$match* Schritte der ersten Teilabfrage verwendet. Die Formulierung einer aktuellen Query bzw. eines Querschnitts über die aktuellen Versionen aller Artikel der englischen Wikipedia wurde bereits im vorherigen Absatz demonstriert. Die Ausgabe aller vom 01.01.2016 bis 07.01.2016 erstellten Artikelversionen erhält man mit

[135] https://docs.mongodb.com/manual/reference/method/db.collection.findOne/#db.collection.findOne (besucht am 01.07.2016)
[136] https://docs.mongodb.com/manual/reference/operator/query/elemMatch/#op._S_elemMatch (besucht am 01.07.2016)
[137] https://docs.mongodb.com/manual/reference/operator/query/all/#op._S_all (besucht am 01.07.2016)
[138] https://docs.mongodb.com/manual/reference/operator/aggregation/match/#pipe._S_match (besucht am 01.07.2016)
[139] https://docs.mongodb.com/manual/reference/operator/aggregation/limit/#pipe._S_limit (besucht am 01.07.2016)
[140] https://docs.mongodb.com/manual/reference/operator/aggregation/project/#pipe._S_project (besucht am 02.07.2016)
[141] https://docs.mongodb.com/manual/reference/command/eval/#dbcmd.eval (besucht am 01.07.2016)

```
// [ { _id: <documentId>, nodes: { !nodeId: <nodeId>, !ancestorIds: [<id>, <id> ...] } } ]
var headings = db.articles.aggregate ( [
  { $match : { $and: [
               { @lang : 'en' },
               { $gte: [ new Date(@timestamp), new Date("2016-01-01T00:00:00") ] },
               { $lte: [ new Date(@timestamp), new Date("2016-01-07T23:59:59") ] }
           ]
         }
  },
  { $unwind: {   path: "$nodes" } }, // Self-Join des article Knotes mit dem
  ...
]);
```

wobei angenommen wird, dass *@timestamp* in einem MongoDB-kompatiblen Datumsformat[142] vorliegt. Historische Queries bzw. den Längsschnitt der kompletten Versionshistorie eines Artikels (in diesem Beispiel des Artikels „Germany“) liefert die folgende Abfrage:

```
// [ { _id: <documentId>, nodes: { !nodeId: <nodeId>, !ancestorIds: [<id>, <id> ...] } } ]
var headings = db.articles.aggregate ( [
  { $match : { $and: [
               { @lang : 'en' },
               { @title : 'Germany' }
           ]
         }
  },
  { $unwind: {   path: "$nodes" } }, // Self-Join des article Knotes mit dem
  ...
]);
```

MapReduce Das MapReduce-Kommando ist die zweite Möglichkeit zur Analyse verteilter *Collections* und hat folgende Syntax[143]:

```
db.runCommand(
  {
    mapReduce: <collection>,
    map: <function>,
    reduce: <function>,
    finalize: <function>,
    out: <output>,
    query: <document>,
    sort: <document>,
    limit: <number>,
    scope: <document>,
    jsMode: <boolean>,
    verbose: <boolean>,
    bypassDocumentValidation: <boolean>
  }
)
```

Die wichtigsten Parameter sind *mapReduce*, *map*, *reduce*, *finalize*, *out*, *query* und *jsMode*.

[142] https://docs.mongodb.com/manual/reference/bson-types/#document-bson-type-date (besucht am 02.07.2016)

[143] https://docs.mongodb.com/manual/reference/command/mapReduce/#dbcmd.mapReduce (besucht am 03.07.2016)

- ***mapReduce*** legt den Namen der Eingabe-Collection fest.
- ***map*** enthält eine benutzerdefinierte Map-Funktion.
- ***reduce*** empfängt eine benutzerdefinierte Reduce-Funktion.
- ***finalize*** beinhaltet eine weitere JavaScript-Funktion zur abschließenden Verarbeitung des Reduce-Outputs.
- ***out*** legt den Namen der Ausgabe-Collection fest oder aktiviert die inline-Ausgabe des Ergebnisses in Form eines JSON-Dokuments. Für eine nähere Beschreibung des *out*-Parameters siehe folgenden Abschnitt des MongoDB-Handbuchs[144].
- ***query*** ermöglicht das Filtern der Eingabe-Collection durch Angabe von Query-Operatoren. Die Query-Operatoren erreichen die selbe Selektivität wie *find*[145] oder *findOne*[146], erlauben jedoch keine Aggregationen.
- ***jsMode*** steuert die Verwaltung der Zwischenergebnisse zwischen Map- und Reduce-Schritt. Diese werden entweder flüchtig, als JavaScript-Objekte im Hauptspeicher („*jsMode=true*“), oder persistent, als BSON-Dokumente in ein temporäres Verzeichnis („*_tmp*“) unterhalb des Datenbankverzeichnisses („*jsMode=false*“) gespeichert. Der *jsMode* ist standardmäßig deaktiviert[147].

Die Parameter *finalize*, *query*, und *jsMode* sind optional. Eine detaillierte Beschreibung aller Parameter kann dem Abschnitt „mapReduce“ des MongoDB-Handbuchs[143] entnommen werden. Die Map-Funktion hat grundsätzlich folgenden Aufbau

```
1 function()
2 {
3   ...
4   emit(this.<someKey>, <someValue>);
5   ...
6 }
```

und sollte weder direkt auf die Datenbank zugreifen, noch sonstige Seiteneffekte haben. Der Zugriff auf die einzelnen Collection-Elemente erfolgt über die *this*-Funktion. Das Ergebnis des Map-Aufrufs wird durch ein- oder mehrmaliges Aufrufen der *emit*-Funktion an den Reduce-Schritt weitergegeben. Pro *emit*-Aufruf wird ein weiteres Zwischenergebnis erzeugt, wobei der Wert eines einzelnen *emit*-Aufrufs auf die Größe von 8 MB (die Hälfte der maximalen BSON-Dokumentgröße)[148] beschränkt ist.

Die Signatur der Reduce-Funktion ist wie folgt aufgebaut:

[144] https://docs.mongodb.com/manual/reference/command/mapReduce/#mapreduce-out-cmd (besucht am 03.07.2016)
[145] https://docs.mongodb.com/manual/reference/method/db.collection.find/ (besucht am 03.07.2016)
[146] https://docs.mongodb.com/manual/reference/method/db.collection.findOne/ (besucht am 03.07.2016)
[147] Bei Aktivierung des *jsModes* können maximal 500.000 Dokumente in der Map-Funktion emittet werden[143].
[148] https://docs.mongodb.com/manual/reference/command/mapReduce/#mapreduce-map-cmd (besucht am 03.07.2016)

```
function(key, values)
{
  ...
  return result;
}
```

Der *values*-Parameter fasst alle, im Map-Durchgang extrahierten Werte mit dem selben Schlüssel, zusammen. Der Wert des zugehörigen Schlüssels ist über den gleichnamigen Parameter *key* abrufbar. Genau wie für die Map-Funktion gilt auch für Reduce; sowohl der *key*- als auch der *values*-Parameter dürfen eine Größe von maximal 8 MB (die Hälfte der maximalen BSON-Dokumentgröße)[148] nicht überschreiten. Die Reduce-Funktion sollte weiterhin weder direkt auf die Datenbank zugreifen noch sonstig Seiteneffekte haben und der Typ des *return* Objekts muss mit dem Typ der im Map-Schritt extrahierten *values* übereinstimmen. Weiterhin ist nicht garantiert, dass alle in der Map-Phase extrahierten Werte mit dem selben Schlüssel auf einmal reduziert werden. Die Logik von Reduce muss daher kommutativ, assoziativ und idempotent sein bzw. folgende Bedingungen erfüllen[149]:

```
  reduce( key, [ A, B ] ) == reduce( key, [ B, A ] ) &&
  reduce(key, [ C, reduce(key, [ A, B ]) ] ) == reduce( key, [ C, A, B ] ) &&
  reduce( key, [ reduce(key, valuesArray) ] ) == reduce( key, valuesArray )
```

Die Finalize-Funktion hat folgende Signatur

```
function(key, reducedValue)
{
  ...
  return modifiedObject;
}
```

und empfängt den Schlüssel (*key*) und das fertige Ergebnis (*reducedValue*) des Reduce-Druchgangs einer Schlüsselpartition. Die Finalize-Funktion sollte dabei nicht direkt auf die Datenbank zugreifen und seiteneffektfrei sein[150].

Auf Grund der Größenbeschränkung des *value*-Parameters von *emit* und die Limitierung der Größe des *values*-Parameter der Reduce-Funktion auf 8 MB[148], ist das MapReduce-Kommando nicht für Aggregationen mit Zwischenergebnissen in der Größe ganzer Dokumente geeignet. Dies trifft insbesondere zu, wenn die Einzeldokumentgröße 8 MB übersteigt oder wenn eine große Anzahl von Dokumenten auf wenige Einzelpartitionen reduziert wird. Letzteres reizt die Größenlimitierung des *values*-Parameter der Redcue-Funktion aus, Ersteres scheitert direkt im Map-Schritt bei Aufruf von *emit*. Ein Beispiel für eine Aggregation, die beide Limits ausreizt, ist die Suche nach der aktuellen Version aller Artikel der englischen Wikipedia.

Für die Ausführung aktueller Queries muss daher entweder vollständig auf den Einsatz von *MapReduce* verzichtet, oder eine *Collection* mit den letzten Versionen aller Artikel sämtlicher Wikipedia-Instanzen (bzw. aller Versionen einer bestimmten Instanz) angelegt

[149] https://docs.mongodb.com/manual/reference/command/mapReduce/#mapreduce-reduce-cmd (besucht am 03.07.2016)

[150] https://docs.mongodb.com/manual/reference/command/mapReduce/#mapreduce-finalize-cmd (besucht am 03.07.2016)

werden. Eine solche *Collection* kann über die *$out* Pipeline-Stufe der *Aggregation Pipeline* erstellt und nachfolgend mit dem MapReduce-Kommando analysiert werden. Längsschnitte entlang der Historie eines Artikels (in diesem Fall des Artikels „Germany“) können wie folgt formuliert werden:

```
1  var mapFunction = fuction { ... };
2  var reduceFunction = function(key, values) { ... };
3
4  db.runCommand (
5    {
6      mapReduce: articles,
7      map: mapFunction,
8      reduce: reduceFunction,
9      out: { inline: 1 },
10     query: { $and: [
11           { @lang : 'en' },
12           { @title : 'Germany' }
13       ]
14     }
15   }
16 )
```

Querschnittanalysen funktionieren analog. Für die Verarbeitung aller vom 01.01.2016 bis 07.01.2016 erstellten Artikelversionen der englischen Wikipedia ergibt sich folgender *query*-Parameter:

```
1  ...
2      query: { $and: [
3             { @lang : 'en' },
4             { $gte: [ new Date(@timestamp), new Date("2016-01-01T00:00:00") ] },
5             { $lte: [ new Date(@timestamp), new Date("2016-01-07T23:59:59") ] }
6         ]
7      }
8  ...
```

Auch was die Granularität des Abfrageergebnisses angeht, entstehen Einschränkungen. Ausgehend von der im Abschnitt Aggregation Pipeline eingeführten flachen WOM-Repräsentation und der nicht unerheblichen Anzahl von Artikeln, die größer als 8 MB sind (vgl. Abbildung 5), ist bei der Rückgabe ganzer Dokumente mit Fehlern zu rechnen. Die Verarbeitung von Dokumententeilen ist grundsätzlich möglich, es muss jedoch auf die Anzahl und Höhe der ausgewählten Teilbäume geachtet werden. Ist sie zu groß oder die Bäume zu hoch, kann bei großen Artikeln das 8-MB-Limit[148] bei Übergabe der gruppierten Zwischenergebnisse an die Reduce-Funktion greifen. Der selbe Effekt tritt ein, wenn zu viele kleine Teilergebnisse unterhalb des selben *keys* zusammengefasst werden. Die Selektivität der Map-Anweisung sollte daher möglichst hoch sein. Im Idealfall werden numerische Metriken (das gesuchte Element kommt im Artikel x vor) oder die Knoten-ID (wo befindet sich das gesuchte Element im Artikel x) an die Reduce-Funktion weitergegeben. Diese kann die Ergebnisse weiter aggregieren oder direkt als Liste zurückgeben. In diesem Fall werden die Ergebnisse über den *key* der Reduce-Funktion identifiziert.

Was die Suchparameter angeht, sind den Möglichkeiten prinzipiell keine Grenzen gesetzt. Jede Abfrage, die sich in JavaScript formulieren lässt, kann grundsätzlich ausgeführt werden. Einschränkungen entstehen hier ausschließlich durch den Aufwand bei der Programmierung und die Ausführungsdauer der Abfragen. Aus Sicht des MapReduce-Kommandos

wäre es auch möglich den unmodifizierten WOM-Output (vgl. Absatz 3.1.1) zu verarbeiten. Durch die geringeren Dokumentgrößen (vgl. Tabelle 5) könnten so auch Teilbäume, sofern die beinhalteten Knotenmengen disjunkt[151] sind, weitgehend uneingeschränkt verarbeitet werden. Die Möglichkeiten aller anderen Zugriffsmethoden reduzieren sich damit jedoch auf eine Vorauswahl der Artikel entlang der zeitlichen Achsen, da lediglich der Titel, die Artikelversion und das Erstellungsdatum als Attribute des Wurzelknotens erreichbar sind. Ein solches Vorgehen ist grundsätzlich möglich, auf Grund der vergleichsweise geringen Performance des MapReduce-Kommandos, im Vergleich zur *Aggregation Pipeline*, aber nicht empfehlenswert[152].

Single Purpose Aggregation Operations, namentlich *count*[153], *distinct*[154] und *group*[155], fassen häufig verwendete Aggregationsoperationen in einer Funktion zusammen. Ihre eingeschränkte Funktionalität und fehlende Konsistenz bei Anwendung auf Sharded-Collections machen sie jedoch für den vorliegenden Anwendungsfall nur bedingt einsetzbar. Die *group* Funktion ist von Grund auf nicht auf Sharded-Collecitons anwendbar[155]. Ihr Funktionsumfang wird in Anbetracht der Datenmengen der Wikipedia nicht näher untersucht. *Count* zählt die Elemente einer *Collection*, produziert allerdings im verteilten Fall, während der Neuverteilung von Daten („chunk migration“[156]) oder bei Existenz verweister Dokumente („orphaned documents“[157]) inkorrekte Werte[153]. *Distinct* liefert eine doppelgängerfreie Liste bestimmter Attribute einer *Collection* und ist an keine weiteren Einschränkungen gebunden[154]. Alle *Single Purpose Aggregation Operations* können im Verlauf eines Aufrufs ausschließlich auf Dokumente einer *Collection* zugreifen[153,154,155].

4.3.3 Nicht-funktionale Eignung

Sowohl Couchbase als auch MongoDB haben durch ihren Einsatz bei Industriegrößen wie Nielsen[158], PayPal[159] oder ebay[160] ihre Fähigkeit zur Verwaltung großer Datenmengen

[151] Sind die Teilbäume nicht disjunkt, werden Dokumentabschnitte mehrfach verarbeitet. Dies steigert das Datenvolumen der Zwischenergebnisse und erhöht das Risiko, die Maximalgröße des *values*-Parameter der Reduce-Funktion zu überschreiten.

[152] https://sysdig.com/blog/mongodb-showdown-aggregate-vs-map-reduce/ (besucht am 04.07.2016)

[153] https://docs.mongodb.com/manual/reference/command/count/ (besucht am 28.06.2016)

[154] https://docs.mongodb.com/manual/reference/command/distinct/ (besucht am 28.06.2016)

[155] https://docs.mongodb.com/manual/reference/command/group/#dbcmd.group (besucht am 28.06.2016)

[156] https://docs.mongodb.com/manual/core/sharding-chunk-migration/ (besucht am 28.06.2016)

[157] https://docs.mongodb.com/manual/reference/glossary/#term-orphaned-document (besucht am 28.06.2016)

[158] http://www.couchbase.com/customers/nielsen.html (besucht am 04.07.2016)

[159] http://www.couchbase.com/customers/paypal.html (besucht am 04.07.2016)

[160] https://www.mongodb.com/presentations/storing-ebays-media-metadata-mongodb-0 (besucht am 04.07.2016)

unter Beweis gestellt. Trotzdem unterliegen beide Systeme einer Reihe individueller Einschränkungen, die für den erfolgreichen Einsatz beachtet werden müssen. Daneben spielt der Ressourcenbedarf, sowie Funktionsumfang der kostenlosen Editionen eine wichtige Rolle für die Beurteilung der nicht-funktionalen Eignung.

Couchbase Server hat ein Soft-Limit von 10 *Buckets* pro Cluster. Der Wert ist einstellbar, sollte jedoch nicht überschritten werden[161]. Die Couchbase-Entwickler empfehlen daher, initial alle Dokumente in einem einzelnen *Bucket* zu speichern und die unterschiedlichen Dokumenttypen mit einem Typ-Attribut zu unterscheiden[162]. Die maximale Schlüssellänge der Dokument-ID beträgt 250 Bytes. Ein einzelnes JSON-Dokument darf nicht größer als 20 MB sein.

Für die MapReduce-Views gelten weiterhin folgende Einschränkungen[161]:

- ***Key-Size*** Die maximale Key-Größe innerhalb eines *emit*-Aufrufs beträgt 4096 bytes.
- ***Key-Value-Size*** Der Payload aller *emit*-Aufrufe (Schlüssel + Wert) pro Dokument pro View darf eine Größe von 1 MB nicht überschreiten.
- ***Function-Timeout*** Die Ausführungsdauer aller Map- und Reduce-Funktionen eines Design-Dokuments ist pro Bucket-Eintrag auf 1000 ms begrenzt.
- ***Document-Size-For-Indexing*** Die maximale Größe eines indexierbaren Dokuments ist standardmäßig auf 20 MB begrenzt (maximale Dokumentgröße).

Couchbase Server ist grundsätzlich für den Betrieb auf Commodity Hardware ausgelegt[163]. Zur Konkretisierung dieses Begriffs nennt die Couchbase-Dokumentation folgende Zahlen:

Ressource	Minimal	Empfohlen
CPU Kerne	4	8+
RAM	4 GB	16 GB+
HDD	8 GB	16 GB+
Netzwerk	1 Gbit/s	1 Gbit/s+

Tabelle 3: Couchbase – Commodity Hardware[164]

[161] http://developer.couchbase.com/documentation/server/current/clustersetup/server-setup.html (besucht am 04.07.2016)
[162] http://developer.couchbase.com/documentation/server/current/clustersetup/server-setup.html (besucht am 04.07.2016)
[163] http://blog.couchbase.com/how-many-nodes-part-3-hardware-considerations (besucht am 05.07.2016)
[164] http://developer.couchbase.com/documentation/server/4.5/install/sizing-general.html (besucht am 04.07.2016)

Ein Produktivsetup sollte dabei immer mindestens aus drei Knoten bestehen[165].

Um eine möglichst feingranulare Zuweisung der verfügbaren Hardwareressourcen, sowie eine optimale Anpassung an den bestehenden Workload zu gewährleisten, führt Couchbase das Konzept des *Multidimensional Scaling* (MDS)[166] ein. Die Datenbankfunktionaliät ist dabei in die Services *Core Data*-, *Index*-, *Query*- und *Search* unterteilt, wobei jeder Service einzeln, pro Knoten im Cluster, aktiviert werden kann.

- ***Core Data*** stellt die CRUD-Funktionalität und *Views* bereit. Der *Core Data Service* muss auf mindestens einem Knoten des Clusters aktiviert werden.
- ***Index*** erstellt und verwaltet die globalen Sekundärindizes (GSI) (optional).
- ***Query*** ermöglicht die Ausführung von N1QL-Queries (optional).
- ***Search*** dient der Bereitstellung und Abfrage von Volltextindizes (optional).

Hardwareressourcen Als Ausgangspunkt für eine ungefähre Abschätzung der benötigten Hardwareressourcen dient der *Core Data Service*. Sein Ressourcenbedarf kann relativ statisch, mit Hilfe der im Abschnitt „Deployment Considerations“[167] enthaltenen Formeln der Couchbase-Dokumentation, berechnet werden. Die dafür notwendigen Variablen und Konstanten fassen Tabelle 4 und 5 zusammen.

Variablen	Kürzel	Wert
Anzahl der Dokumente	d_{anz}	800 Millionen^{k}
Größer der Dokumenten_ID	d_{id}	100 Bytes^{s}
Dokumentengröße	d_{gr}	308.000 Bytes^{m}
Arbeitsspeicher pro Serverknoten	ram_{node}	128 GB^{s}
Memory-Quota[168,169]	ram_{quota}	-^{s}
Prozentsatz der im Cache vorgehaltenen Daten	wsp	-^{s}
Anzahl der Replikate pro Dokument	r	-

k) Konstante; m) Messwert (siehe 3.1.2); r) Anhand von Richtwert festgelegt; s) Schätzwert

Tabelle 4: Couchbase – Variablen für die Berechnung der Hardwareressourcen des *Core Data Service*

[165] http://blog.couchbase.com/how-many-nodes-part-1-introduction-sizing-couchbase-server-20-cluster (besucht am 05.07.2016)
[166] http://developer.couchbase.com/documentation/server/current/architecture/services-archi-multi-dimensional-scaling.html#concept_v4w_zmj_vs (besucht am 04.07.2016)
[167] http://developer.couchbase.com/documentation/server/4.5/install/sizing-data.html (besucht am 05.07.2016)
[168] Die Memory Quota gibt an wie viel Arbeitsspeicher maximal pro Knoten für den *Core Data Service* genutzt werden darf.
[169] http://developer.couchbase.com/documentation/server/current/architecture/db-engine-architecture.html (besucht am 06.07.2016)

Konstanten	Kürzel	Wert
Headroom[170]	hr	30%^{r}
Metadaten pro Dokument	d_{meta}	56 Byte^k
High-Water-Mark[171,169]	wm_{high}	85%^{r}
Low-Water-Mark[172,169]	wm_{low}	75%^{r}

k) Konstante; m) Messwert (siehe 3.1.2); r) Anhand von Richtwert festgelegt; s) Schätzwert

Tabelle 5: Couchbase – Konstanten für die Berechnung der Hardwareressourcen des *Core Data Service*

Das gesamte Datenvolumen aller Dokumente ($data_{nutz}$) ist gegeben als

$$data_{nutz} = d_{anz} \cdot d_{gr} \cdot (1 + r) \tag{3}$$

Der Arbeitsspeicherverbrauch aller *Data Nodes* (ram_{ges}) ergibt sich aus

$$ram_{ges} = \frac{(data_{meta} + ws) \cdot (1 + hr)}{wm_{high}} \tag{4}$$

wobei das Metadatenvolumen ($data_{meta}$) geben ist als

$$data_{meta} = d_{anz} \cdot (d_{meta} + d_{id}) \cdot (1 + r) \tag{5}$$

Das Datenvolumen aller im Arbeitsspeicher vorgehaltenen Dokumente, genannt *Working Set* (ws), ist gegeben als der Faktor

$$ws = data_{nutz} \cdot wsp \tag{6}$$

Die Anzahl der benötigten Server-Knoten ($nodes$) ergibt sich aus

$$nodes = \frac{ram_{ges}}{ram_{quota}} \tag{7}$$

Das maximale Datenvolumen ($data_{max}$) ist die Summe aus Nutz- und Metadaten,

$$data_{max} = data_{nutz} + data_{meta} \tag{8}$$

[170] Gibt den Arbeitsspeicherbedarf zur Verwaltung von Metadaten an. Die Couchbase-Dokumentation empfiehlt einen Wert von 25% für SSDs und 30% für Festplatten http://docs.couchbase.com/admin/admin/Concepts/bp-sizingGuidelines.html (besucht am 06.07.2016).

[171] Grenzwert ab dem Elemente aus dem Cache entfernt werden. Das Entfernen dauert so lange an bis die Low-Water-Mark erreicht ist. Die Higth-Water-Mark ist ein Prozentsatz der Memory-Quota.

[172] Grenzwert bis zu dem Elemente aus dem Cache entfernt werden. Die Low-Water-Mark ist ein Prozentsatz der Memory-Quota.

wobei die Dokumente vor ihrer Persistierung, mit Hilfe des Snappy-Algorithmus von Google, komprimiert werden. Die Couchbase-Dokumentation macht hier keine offiziellen Angaben zu den erreichten Kompressionsraten. Die Messungen von Freire et al. [Fre+16, S. 11] lassen jedoch auf eine mittlere Kompressionsrate von $3,8$ schließen. Die minimal erreichte Kompressionsrate lag bei 2.6, der Maximalwert bei 4.44. Als grobe Näherung für den Effektivwert ($data_{eff}$) wird daher folgende Formale angenommen:

$$data_{eff} = data_{nutz} \cdot 0,3 \tag{9}$$

Tabelle 6 zeigt den Ressourcenbedarf des *Core Data Service* für die Verwaltung aller Artikel der englischen Wikipedia bei Vorhaltung von 5% der Daten im Cache und einer *Memory-Quota* von 124 GB pro Server-Knoten.

Ressource	**Anzahl der Replikate pro Dokument**		
	$r = 0$	$r = 1$	$r = 2$
Nutzdatenvolumen ($data_{nutz}$)	246,4 TB	492,8 TB	739,2 TB
Working-Set (ws)	12,32 TB	24,64 TB	36,96 TB
Metadatenvolumen ($data_{meta}$)	128,8 GB	249,6 GB	374,4 GB
Maximales Datenvolumen ($data_{max}$)	246,52 TB	493,05 TB	739,57 TB
Effektives Datenvolumen ($data_{eff}$)	73,96 TB	147,92 TB	221,87 TB
Arbeitsspeicherverbrauch (ram_{ges})	19,04 TB	38,07 TB	57,10 TB
Anzahl der Serverknoten ($nodes$)	154	307	460
Maximales Datenvolumen pro Knoten	1,61 TB		
Effektives Datenvolumen pro Knoten	482 GB		

Tabelle 6: Couchbase – Ressourcenbedarf des *Core Data Service* für die Verwaltung der englischen Wikipedia in Abhängigkeit vom Replikationsfaktor der Dokumente

Neben dem Ressourcenbedarf für den *Core Data Service*, müssen weitere Kapazitäten für den *Indexing* und *Query Service* eingeplant werden. Die genauen Hardwareanforderungen lassen sich jedoch nur schwer statisch im Vorfeld abschätzen und müssen vielmehr im laufenden Betrieb ermittelt werden. Grundsätzlich ist festzuhalten:

- Index-Service-Nodes profitieren von viel Arbeitsspeicher und hoher I/O-Performance[173].
- Data- und Index-Service-Instanzen sind beide I/O-intensiv und sollten nach Möglichkeit auf verschiedenen Server-Knoten ausgeführt werden[180]. Auf diese Weise werden konkurrierende Festplattenzugriffe vermieden und die Leistungsfähigkeit beider Dienste maximiert.
- Die Ausführung von N1QL-Queries benötigt hauptsächlich CPU-Leistung, wobei die

[173] http://developer.couchbase.com/documentation/server/4.5/architecture/storage-architecture.html (besucht am 06.07.2016)

Verarbeitung unabhängiger Queries stark parallelisierbar ist[174]. Je mehr leistungsfähige CPUs einer Query-Node zur Verfügung stehen, desto besser ist ihre Performance.

Die Hochrechnungsergebnisse in Tabelle 5 zeigen; bereits ohne redundante Speicherung der Dokumente ($r = 0$) wären, bei Vorhaltung von 5% der Daten im Cache, 154 Serverknoten mit einer *Memory-Quota* von 124 GB pro Knoten für die Verwaltung der englischen Wikipedia notwendig. Das effektiv verwaltete Datenvolumen pro Knoten liegt allerdings nur bei 482 GB, das entspricht einem Verhältnis von etwa 3,9 GB Nutzdaten pro einem Gigabyte RAM. Berücksichtigt man den zusätzlichen Ressourcenbedarf pro Knoten, der aus dem Einsatz der Community-Edition bzw. der gleichzeitigen Ausführung des *Index Service* resultiert, ist der Arbeitsspeicherbedarf des *Core Data Service* in dieser Konfiguration zu hoch.

Ausgehend von einer symmetrischen Skalierung aller Services wird der Prozentsatz der im Cache vorgehaltenen Daten auf 1% reduziert. Zusätzlich wird die *Memory-Quota* des *Core Data Service* von 124 GB auf 80 GB heruntergesetzt, wodurch pro Serverknoten Arbeitsspeicher für den *Index Service* frei wird. Zusätzlich wird pro Knoten von 25%[175] mehr persistentem Speicher für die Verwaltung der Indizes ausgegangen. Tabelle 7 zeigt den angepassten Ressourcenbedarf für die symmetrische Skalierung aller Services bei 128 GB RAM pro Serverknoten.

Ressource	**Anzahl der Replikate pro Dokument**		
	$r = 0$	$r = 1$	$r = 2$
Nutzdatenvolumen ($data_{nutz}$)	246,4 TB	492,8 TB	739,2 TB
Working-Set (ws)	2,64 TB	4,93 TB	7,39 TB
Metadatenvolumen ($data_{meta}$)	128,8 GB	249,6 GB	374,4 GB
Maximales Datenvolumen ($data_{max}$)	246,52 TB	493,05 TB	739,57 TB
Effektives Datenvolumen ($data_{eff}$)	73,96 TB	147,92 TB	221,87 TB
Arbeitsspeicherverbrauch (ram_{ges})	4,23 TB	7,92 TB	11,87 TB
Anzahl der Serverknoten ($nodes$)	53	99	149
Maximales Datenvolumen pro Knoten	4,90 TB + 0.25 · 4,9 TB = 6,13 TB		
Effektives Datenvolumen pro Knoten	1,45 TB + 0.25 · 4,9 TB = 2,68 TB		

Tabelle 7: Couchbase – Ressourcenbedarf für die Verwaltung der englischen Wikipedia in Abhängigkeit vom Replikationsfaktor der Dokumente bei symmetrischer Skalierung aller *Services*

[174] http://developer.couchbase.com/documentation/server/4.5/install/sizing-query.html (besucht am 06.07.2016)

[175] Hierbei handelt es sich um einen Schätzwert, der exakt, erst im laufenden Betrieb bestimmt werden kann.

Community-Edition Die Community-Edition von Couchbase wird unter einer eigenen Open-Source-Lizenz[176] vertrieben und ist für den Produktiveinsatz freigegeben[177]–, ihr Funktionsumfang unterliegt jedoch einigen Einschränkungen[178]; Es fehlen unter anderem Software-Updates (Bugfix- und Security-Updates), der Zugang zu professionellem Support, sowie eine Reihe erweiterter Funktionen, wie Rack-Awareness, LDAP-Integration und erweiterte Backup- und Recovery-Tools. Für den vorliegenden Anwendungsfall relevant ist die fehlende In-Memory-Optimierung der globalen Sekundärindizes[179], der Ausschluss des *Multidimensional Scalings*[180], sowie die fehlende *High Performance Concurrent Query Execution*.

Die In-Memory-Optimierung der globalen Sekundärindizes (MOI) verspricht eine schnellere Indizierung großer Datenbestände und eine bessere Performance bei der Ausführung von N1QL-Queries[181]. Weitere Details zu den *Memory-Optimized Global Secondary Indexes* liefern die folgenden Couchbase-Blog-Einträge[182,183].

Multidimensional Scaling wurde im vorherigen Absatz kurz erläutert und ermöglicht das Deployment des *Core Data*-, *Index*-, *Query*- und *Search Service* auf unterschiedlichen Server-Knoten. In der Community-Edition steht ausschließlich *Symetric Scaling* zur Verfügung, wobei die Services entweder auf allen Server-Knoten aktiviert oder deaktiviert werden[184]. Sowohl der *Core Data Service* als auch der *Index Service* sind sehr I/O-lastig[173]. Bei Einsatz der Community-Edition ist daher darauf zu achten, die I/O Operationen beider Dienste auf unterschiedliche, physikalische Speichermedien zu verlagern. Anderenfalls kommt es zu Performance-Einbußen.

Die maximal nutzbare Anzahl von CPUs des *Query Service* ist in der Community-Edition auf vier beschränkt[185]. Auf Servern mit vielen CPUs wird so die Zahl gleichzeitig ausführbarer Queries limitiert. Unter den gegebenen Hardwareanforderungen relativiert sich dieser Effekt jedoch über die symmetrische Skalierung. Ausgehend von ca. acht virtuellen CPUs pro Server-Knoten und den CPU-Anforderungen der *Buckets* (eine CPU pro *Bucket*), der View-Engine und des *Index Service*, stehen für die Verarbeitung der Queries

[176] http://www.couchbase.com/community (besucht am 09.07.2016)
[177] http://www.couchbase.com/nosql-databases/downloads# (besucht am 09.07.2016)
[178] http://developer.couchbase.com/documentation/server/current/introduction/editions.html (besucht am 06.07.2016)
[179] http://developer.couchbase.com/documentation/server/current/introduction/editions.html#couchbase-editions__query-index-moi (besucht am 09.07.2016)
[180] http://developer.couchbase.com/documentation/server/current/architecture/services-archi-multi-dimensional-scaling.html#concept_v4w_zmj_vs (besucht am 06.07.2016)
[181] http://developer.couchbase.com/documentation/server/current/introduction/editions.html#couchbase-editions__query-index-moi (besucht am 12.07.2016)
[182] http://blog.couchbase.com/2016/may/faster-indexing-and-query-with-memory-optimized-global-secondary-indexes-gsi---part-ii (besucht am 12.07.2016)
[183] http://blog.couchbase.com/2016/may/efficient-io-with-circular-writes-on-global-indexes-in-couchbase-server-4.5 (besucht am 12.07.2016)
[184] Der Core Data Service muss aktiviert werden, da er die Kernfunktionalität von Couchbase bereitstellt[180].
[185] http://developer.couchbase.com/documentation/server/current/introduction/editions.html#couchbase-editions__concurrent-query-exec (besucht am 06.07.2016)

von Grund auf nicht mehr als vier virtuelle CPUs zur Verfügung.

MongoDB Beim Einsatz von MongoDB müssen folgende Einschränkungen beachtet werden:

- Das MapReduce-Kommando kann während eines *emit*-Aufrufs, sowie bei Übergabe des *values*-Parameter an die Reduce-Funktion, maximal 8 MB (die Hälfte der maximalen Dokumentengröße) verarbeiten[148].
- Bei Verwendung des MapReduce-Kommandos ist standardmäßig die BSON-Serialisierung der JavaScript-Objekte zwischen Map- und Reduce-Schritt aktiviert. Bei Deaktivierung dieser Funktion ist die Anzahl der *emit*-Aufrufe im Map-Schritt auf 500.000 begrenzt[186].
- Das globale Limit für die Größe eines BSON-Dokuments liegt bei 16 MB. Dies gilt auch für alle Funktionen die eine Ergebnismenge *inline*, als BSON-Dokument, zurückgeben[187]. Dieses Limit kann grundsätzlich durch die Verwaltung der Dokumente als Binärdaten überwunden werden[188]. Die Dokumente werden dabei in eine Reihe 255 kB großer Binär-Chunks unterteilt, wobei jeder Chunk als Binäres-Feld eines BSON-Dokuments in einer speziellen *chunks Collection* abgelegt wird. Die zugehörigen Metadaten liegen in einer separaten *Collection files*[188]. Inhaltsbezogene Suchen mit der *Aggregation Pipeline* oder *MapReduce* sind dabei nicht möglich.
- MongoDB lässt schreibende Operationen nur auf dem *Primary* eines *Replica-Sets* zu[189]. Um I/O intensive Workloads durch Hinzufügen eines weiteren *Shards* zu unterstützen, müssen statt einem, n Serverknoten hinzugefügt werden, wobei n der Anzahl der Mitglieder eines Replika-Sets entspricht bzw. den Replikationsfaktor der Daten widerspiegelt. Da I/O für den vorliegenden Anwendungsfall eine eher untergeordnete Rolle spielt (siehe Kapitel 3.3 Workload – 1,4 Einfügeoperationen pro Sekunde), kann diese Einschränkung prinzipiell vernachlässigt werden.

Hardwareressourcen Der für den vorliegenden Anwendungsfall notwendige Ressourcenbedarf wird anhand zweier Berechnungsbeispiele der MongoDB-Webseite ermittelt[190]. Den Ausgangspunkt der Berechnung bilden die Kenngrößen in Tabelle 8.

[186] https://docs.mongodb.com/manual/reference/command/mapReduce/ (besucht am 29.06.2016)
[187] https://docs.mongodb.com/manual/core/map-reduce/ (besucht am 29.06.2016)
[188] https://docs.mongodb.com/manual/core/gridfs/ (besucht am 09.07.2016)
[189] https://docs.mongodb.com/manual/core/replica-set-secondary/ (besucht am 09.07.2016)
[190] https://www.mongodb.com/blog/post/capacity-planning-and-hardware-provisioning-mongodb-ten-minutes (besucht am 09.07.2016)
[191] Ein viertel Prozent des Nutzdatenvolumens.
[192] Die Memory Quota gibt an wie viel Arbeitsspeicher maximal pro Serverknoten für MongoDB genutzt werden darf.

Kenngröße	Kürzel	Wert
Anzahl der Dokumente	d_{anz}	800 Millionen[k]
Dokumentengröße	d_{gr}	821,38 KB[m]
Nutzdatenvolumen	$data_{nutz}$	671,10 TB[e]
Datenvolumen der Indizes	$data_{index}$	1,68 TB[s,191]
Arbeitsspeicher pro Serverknoten	ram_{node}	128 GB[s]
Memory-Quota[192]	ram_{quota}	-[s]
Prozentsatz der im Cache vorgehaltenen Daten	wsp	-[s]
Anzahl der Replikate pro Dokument	r	-

k) Konstante; m) Messwert (siehe 3.1.2); r) Anhand von Richtwert festgelegt; e) Errechneter Wert; s) Schätzwert

Tabelle 8: MongoDB – Kenngrößen für die Berechnung der Hardwareressourcen

Das Working-Set (ws) aller im Arbeitsspeicher vorgehaltener Dokumente ist gegeben als:

$$ws = data_{nutz} \cdot wsp \tag{10}$$

Der gesamte Arbeitsspeicherverbrauch (ram_{ges}) ist die Summe aus dem Working-Set und dem Speicherverbrauch der Indizes.

$$ram_{ges} = ws + data_{index} \tag{11}$$

Die Anzahl der *Shards* ($shards$) ergibt sich aus dem Working-Set und dem verfügbaren Arbeitsspeicher pro Serverknoten.

$$shards = \frac{ram_{ges}}{ram_{quota}} \tag{12}$$

Das benötigte Datenvolumen pro Serverkonten($data_{node}$) ist der Quotient aus dem Nutzdatenvolumen und der Anzahl der *Shards*.

$$data_{node} = \frac{data_{nutz}}{shards} \tag{13}$$

Die Anzahl der physikalischen Serverknoten ergibt sich aus dem Produkt von Replikationsfaktor und der Menge der benötigten *Shards*.

$$nodes = shards \cdot r \tag{14}$$

Tabelle 9 zeigt den Ressourcenbedarf von MongoDB für die Verwaltung aller Artikel der englischen Wikipedia, bei Vorhaltung von 1% der Daten im Cache und einer *Memory-*

Quota von 124 GB pro Server-Knoten.

Ressource	**Anzahl der Replikate pro Dokument**		
	$r = 0$	$r = 1$	$r = 2$
Gesamtdatenvolumen	671,10 TB	1,34 PB	2,01 PB
Working-Set (ws)		6,71 TB	
Arbeitsspeicherverbrauch (ram_{ges})		8,39 TB	
Anzahl der Shards ($shards$)		55	
Anzahl der Serverknoten ($nodes$)	68	136	204
Datenvolumen pro Knoten ($data_{node}$)		9,87 TB	

Tabelle 9: MongoDB – Ressourcenbedarf für die Verwaltung der englischen Wikipedia in Abhängigkeit vom Replikationsfaktor der Dokumente

Open-Source-Edition Die Open-Source-Edition von MongoDB steht unter der AGPL v3.0[193], die zugehörigen Treiber werden unter der Apache-Lizenz v2.0[193] veröffentlicht. Vergleicht man die Kernfunktionalität von Open-Source- und Enterprise-Edition, ergeben sich kaum Unterschiede. Die einzige funktionale Einschränkung ist die fehlende In-Memory-Storage-Engine. Diese spielt, auf Grund der Größe des zu verwaltenden Datenbestands (sieh Tabelle 9), jedoch nur eine untergeordnete Rolle und befindet sich zudem im Beta-Stadium. Ansonsten zeichnet sich die Enterprise-Edition durch erweiterte Angebote für Unternehmen, wie professionellen Support und Consulting, sowie spezielle Monitoring-, Management und BI-Lösungen aus[194].

4.3.4 Fazit

Funktional sowie nicht-funktional ergeben sich wesentliche Unterschiede zwischen Couchbase Server und MongoDB. Funktional bietet Couchbase mit N1QL eine SQL-artige Querylanguage, die bereits mit Sprachmitteln die Analyse tief verschachtelter JSON-Dokumente erlaubt. Weiterhin steht eine View-Engine zur direkten Referenzierung relevanter Document-IDs zur Verfügung. Die Kombination beider Funktionen erlaubt ein frühzeitiges Eingrenzen der Eingabemenge von N1QL Queries im WHERE-Statement, sodass beispielsweise die wiederholte Ausführung der selben Gruppierungsoperation in unterschiedlichen Queries verhindert werden kann.

Die deklarativen Zugriffsmöglichkeiten von MongoDB erlauben keine direkte Analyse von Baumstrukturen und erzwingen eine aufwändige, sowie speicherplatzintensive Anpassung des Datenformats (vgl. Tabelle 1). Zudem müssen Abfragen, die Aussagen über

[193] https://www.mongodb.com/community/licensing (besucht am 09.07.2016)
[194] https://www.mongodb.com/products/mongodb-enterprise-advanced (besucht am 09.07.2016)

Dokumentknoten in unterschiedlichen Verschachtellungstiefen machen, in mehrere Einzelabfragen zerlegt werden. Für die Verarbeitung der entstehenden Zwischenergebnisse ist zusätzlicher Programmcode notwendig. Dies verkompliziert die Abfragen und erhöht die Anzahl der Round-Trips zwischen Client und Server. Die gesteigerte Round-Trip-Zahl wirkt sich wiederum negative auf die Abfrageperformance aus. Das MapReduce-Kommando ist, durch Einsatz von JavaScript, bei der Verarbeitung von Baumstrukturen an keine Limitierungen gebunden, unterliegt dafür jedoch strikten Einschränkungen, was die Größe der zu verarbeitenden Zwischenergebnisse angeht. Zudem ist die Performance des MapReduce-Kommandos deutlich geringer als die der deklarativen Abfragen.

Im nicht-funktionalen Bereich ist die Anzahl der *Buckets* bei Couchbase auf zehn limitiert. Dadurch ergibt sich die Notwendigkeit eines zusätzlichen Sprachen-Attributs im *article*-Knoten, womit die Artikel aller Wikiinstanzen gemeinsam in einem *Bucket* verwaltet werden können. Weiterhin wurden in der Community-Edition Funktionen zur Ressourcenverteilung und Queryoptimierung gestrichen. Aus diesem Grund muss bei Einsatz der Community-Edition pro Knoten insgesamt leistungsfähigere Hardware eingesetzt werden.

Die Kernfunktionalität von Open-Source- und Enterprise-Edition ist bei MongoDB nahezu identisch. Für Einschränkungen bei der Datenverarbeitung sorgt jedoch die gesteigerte Dokumentengröße des angepassten JSON-Formats in Kombination mit der Maximalgröße eines BSON-Dokuments von 16 MB. Ausgehend von der Formatanpassung in Kapitel 4.3.2 und den Messergebnissen in Tabelle 1, übersteigt das größte gemessene Dokument den zulässigen Maximalwert um etwa vier Megabyte. Weiterhin handelt es sich bei dem Messwert um ein lokales Maximum. Es muss davon ausgegangen werden, dass im Datenbestand der gesamten Wikipedia noch größere Artikel existieren.

Auf Grund des vergleichsweise hohen Datenvolumens (vgl. Tabelle 7 und 9) und der Überschreitung der maximalen Dokumentengröße von 16 MB, wird MongoDB für den vorliegenden Anwendungsfall als technisch ungeeignet und finanziell unwirtschaftlich eingestuft. Couchbase Server ist damit das einzige System im Bereich der JSON-basierten Document Stores, welches als geeignet für den vorliegenden Anwendungsfall eingestuft werden kann.

4.4 Multi-Model Datenbanken

Im Bereich der Multi-Model Datenbanken wurden ArangoDB[195] und OrientDB[196] untersucht.

[195] https://www.arangodb.com/ (besucht am 20.05.2016)
[196] http://orientdb.com/ (besucht am 20.05.2016)

4.4.1 Grundlegende Eigenschaften

Beide Systeme kombinieren die Eigenschaften eines JSON basierten Document Stores und einer Graph-Datenbank in einer Storage-Engine. Sowohl ArangoDB als auch OrientDB bieten eine API für den dokumentenorientierten Zugriff[197,198], sowie eine API für den graphorientierten Zugriff[199,200] und implementieren zusätzlich den TinkerPop-Blueprints-Standard[201] für den Zugriff auf Graphdatenbanken[202,203].

ArangoDB ist ein, auf In-Memory-Verarbeitung optimierter, Document Store mit zusätzlichen Funktionen zur Modellierung von Knoten und Kanten eines Graphen[204,205]. Die Datenhaltung ist grundsätzlich persistent, basiert jedoch auf der Verwaltung von *Collections* in Form von „memory mapped data files“, welche gemäß der Speicherverwaltung des Hostbetriebssystems in den physikalischen Hauptspeicher ein- und ausgelagert werden[206]. Für kurze Zugriffszeiten müssen sich alle häufig verwendeten Datenfiles, inklusive Indexstrukturen, im Hauptspeicher befinden[207]. Zu den wichtigsten Funktionen gehören das Collection-Sharding über mehrere Datenbankinstanzen, Joins zwischen Dokumenten unterschiedlicher *Collections*, ein einstellbarer Durability-Level[208], das Setzen von Sekundärindizes[209] auf Dokumentattribute, und Transaktionen[205] mit ACID Eigenschaften über ein oder mehrere Dokumente. Für die Auswertung der gespeicherten Daten wurde AQL[210], eine ArangoDB spezifische Query-Language, entwickelt, deren Funktionsumfang durch benutzerdefinierte, serverseitig ausgeführte JavaScript-Funktionen[211] ergänzt werden kann.

OrientDB geht den umgekehrten Weg und präsentiert sich als verteilte Graphdatenbank, deren interne Datenrepräsentation auf logischer Ebene aus JSON-Dokumenten be-

[197] https://docs.arangodb.com/Documents/index.html (besucht am 26.05.2016)
[198] http://orientdb.com/docs/2.2/Document-Database.html (besucht am 26.05.2016)
[199] https://docs.arangodb.com/Graphs/index.html (besucht am 26.05.2016)
[200] http://orientdb.com/docs/2.2/Graph-Database-Tinkerpop.html (besucht am 26.05.2016)
[201] https://github.com/tinkerpop/blueprints/wiki (besucht am 26.05.2016)
[202] https://github.com/arangodb/blueprints-arangodb-graph/wiki (besucht am 26.05.2016)
[203] http://orientdb.com/docs/2.2/Graph-Database-Tinkerpop.html (besucht am 26.05.2016)
[204] https://www.arangodb.com/faq/#what-is-arangodb-and-for-what-kind-of-applications-is-it-designed-for (besucht am 23.05.2016)
[205] https://www.arangodb.com/key-features/ (besucht am 20.05.2016)
[206] https://www.arangodb.com/faq/#how-much-space-does-arangodb-need-in-the-file-system (besucht am 23.05.2016)
[207] http://stackoverflow.com/questions/24380071/memory-usage-of-arangodb (besucht am 20.05.2016)
[208] https://www.arangodb.com/faq/#what-durability-guarantees-does-arangodb-offer (besucht am 20.05.2016)
[209] https://docs.arangodb.com/IndexHandling/IndexBasics.html (besucht am 26.05.2016)
[210] https://docs.arangodb.com/Aql/Basics.html (besucht am 20.05.2016)
[211] https://docs.arangodb.com/Foxx/index.html (besucht am 20.05.2016)

steht[212]. OrientDB kann als reine In-Memory-Datenbank oder als persistenter Datenspeicher betrieben werden[213]. Im Fall der In-Memory-Konfiguration befinden sich alle Daten flüchtig im Hauptspeicher und gehen beim Beenden des Serverdienstes verloren. Bei der Verwendung als persistenter Datenspeicher werden die Daten dauerhaft auf Festplatte geschrieben. Der Datenzugriff kann durch das Setzen entsprechender Disk-Cache-Größen beschleunigt werden[214].

Die kleinste Speichereinheit ist ein *Record*. *Records* werden weiter in *Documents, RecordBytes, Vertexes* und *Edges* untergliedert[215]. Eine besondere, aus dem Multi-Model Ansatz resultierende Funktion ist die Erweiterung des Dokumentenmodells durch sogenannte *relationships*. *Relationships* bilden eine Art persistenten *Document* Pointer, mit deren Hilfe 1:1, 1:n oder n:m Verweise bzw. *Links* zwischen Dokumenten, erzeugt werden können[216]. Beim Zugriff auf ein Dokument werden alle enthaltenen Verweise transparent und mit konstanter Laufzeit durch die jeweils referenzierten Dokumente ersetzt[212].

Die Datenhaltung kann schemalos, schemagestützt oder teilweise schemabehaftet erfolgen. Die Definition von Schemata erfolgt mit Hilfe von *Classes*[217]. Eine solche Klasse kann die Präsenz bestimmter Properties in einem *Record* erzwingen und den Datentyp des zugehörigen Values, sowie optional dessen Wertebereich, festlegen. Jeder gespeicherte *Record* wird einer Klasse zugeordnet, deren Daten sich auf einen oder mehrere *Cluster*[218] verteilen. *Cluster* sind vergleichbar mit den *Collections* eines Document Stores und können über mehrere Serverinstanzen verteilt werden – Cluster Sharding[219]. Zum weiteren Funktionsumfang gehört das Setzen von Sekundärindizes[220] und Transaktionen[221] mit ACID-Eigenschaften über mehrere Dokumente[222].

Als Abfragesprache kommt ein eigener SQL-Dialekt[223] zum Einsatz, dessen Sprachumfang um Funktionen zur Traversierung von Graphen erweitert wurde[224]. Clients können einzelne Queries oder ganze Queryfolgen, sogenannte SQL-Batches, an den Server schicken[225]. Weiterhin werden alle innerhalb einer JVM lauffähigen Programmiersprachen für

[212] http://orientdb.com/why-orientdb/ (besucht am 23.05.2016)
[213] http://orientdb.com/docs/2.1/Memory-storage.html (besucht am 23.05.2016)
[214] http://orientdb.com/docs/2.2/Performance-Tuning.html (besucht am 23.05.2016)
[215] http://orientdb.com/docs/last/Concepts.html (besucht am 25.05.2016)
[216] http://orientdb.com/docs/2.0/orientdb.wiki/Tutorial-Relationships.html (besucht am 25.05.2016)
[217] http://orientdb.com/docs/last/Tutorial-Classes.html (besucht am 25.05.2016)
[218] http://orientdb.com/docs/last/Tutorial-Clusters.html (besucht am 25.05.2016)
[219] http://orientdb.com/docs/last/Distributed-Sharding.html (besucht am 25.05.2016)
[220] http://orientdb.com/docs/last/Indexes.html (besucht am 25.05.2016)
[221] http://orientdb.com/docs/last/Transactions.html (besucht am 25.05.2016)
[222] http://orient-database.narkive.com/MGTTmUe6/orientdb-multi-key-transactions (besucht am 25.05.2016)
[223] http://orientdb.com/docs/last/SQL.html#orientdb-sql-dialect (besucht am 25.05.2016)
[224] http://orientdb.com/docs/last/Commands.html (besucht am 25.05.2016)
[225] http://orientdb.com/docs/2.0/orientdb.wiki/SQL-batch.html (besucht am 25.05.2016)

die Definition serverseitiger Scripte unterstützt[226].

4.4.2 Fazit

Für keines der beiden Systeme lohnt sich eine genauere Überprüfung der funktionalen und nicht-funktionalen Eignung.

ArangoDB scheidet auf Grund seiner Konzeption als In-Memory-Datenbank als möglicher Systemkandidat aus. Die ArangoDB-Entwickler raten ausdrücklich von der Verwendung von ArangoDB für die Verwaltung großer, verteilter Datenbestände im Terabytebereich ab[227]. Die Hochrechnungen im Abschnitt Datenvolumen haben gezeigt, die Analyse der Wikipedia ist genau solch einen Anwendungsfall und ArangoDB ist daher nicht geeignet.

OrientDB wurde auf Grund der nachweislichen Stabilitätsprobleme verworfen. Trotz wiederholter Aufnahme in Gartner's *Magic Quadrant for Operational Database Management Systems*[228,229] und diverser, auf der OrientDB-Homepage gelisteter, Produktivinstallationen[230], berichten Communitymitglieder über schwerwiegende Mängel in der Software von OrientDB. Dazu zählen unter anderem Stabilitätsprobleme bei Einsatz von OrientDB in einem verteilten Setup, zeitweise fehlerhaften Transaktionsimplementierungen, Bugreports, die von Entwicklern bestätigt und später, unbehoben, von CEO geschlossen wurden, sowie gravierende Sicherheitslücken im User-Management[231]. Die Gartner-Studie bestätigt dies und spricht von einem hohen Prozentsatz an Referenzkunden, die Fehler in der Software als Problem beschrieben[229]. Ein Blick in den Issue-Tracker von OrientDB bestärkt den Eindruck. Stand 25.04.2016 wurden 307 offene Bugreports gelistet[232]. OrientDB wird unter Berücksichtigung der gesammelten Fakten als instabil und für den Produktiveinsatz ungeeignet eingestuft und somit als möglicher Systemkandidat ausgeschlossen.

[226] http://orientdb.com/docs/2.0/orientdb.wiki/Javascript-Command.html (besucht am 25.05.2016)
[227] https://www.arangodb.com/faq/#for-which-use-cases-is-arangodb-not-the-perfect-choice (besucht am 20.05.2016)
[228] https://www.gartner.com/doc/2610218 (besucht am 20.05.2016)
[229] https://www.gartner.com/doc/reprints?id=1-2PMFPEN&ct=151013 (besucht am 23.06.2016)
[230] http://orientdb.com/customers/ (besucht am 26.05.2016)
[231] http://orientdbleaks.blogspot.de/2015/06/the-orientdb-issues-that-made-us-give-up.html (besucht am 26.05.2016)
[232] https://github.com/orientechnologies/orientdb/issues?q=is%3Aissue+is%3Aopen+label%3Abug (besucht am 20.05.2016)

4.5 Wide Column Stores in Kombination mit Map-Reduce

Ein grundlegend anderer Ansatz zur Verwaltung und Analyse von WOM ist der kombinierte Einsatz von Wide Column Stores und Map-Reduce[DG08]. Die beiden dominierenden Systeme im Open-Source-Bereich sind Apache Cassandra[233] und Apache HBase[234,235]. Cassandra ist primär auf hohe Schreiblasten optimiert, während HBase eher für leselastige Zugriffe konzipiert wurde[236]. Da der Fokus der Wikipedia Research Appliance auf dem Einsatz als Analysedatenbank liegt und der Workload somit typischer weise aus lesenden Zugriffen besteht, wird nachfolgend ausschließlich die Eignung von HBase untersucht.

4.5.1 Grundlegende Eigenschaften

Auf einer abstrakten Ebene ist das Datenmodell von HBase vergleichbar mit einem hierarchischen Key-Value Store, auf dessen Datenwerte, über eine Hierarchie von drei bzw. vier Schlüsseln, wahlfrei zugegriffen werden kann. Jede Schlüsselebene ist sortiert und die enthaltenen Schlüssel können vom Nutzer frei definiert werden. Die Elemente der obersten Schlüsselmenge können durch Angabe eines *Filters* gezielt nach bestimmten Schlüsselwertpaaren durchsucht werden. Die Suche profitiert von der Sortierung und ermöglicht so ein effizientes Eingrenzen der Datenmenge. Im Gegensatz zu allen bisher diskutierten Systemen, werden die Nutzdaten bei HBase binär abgespeichert, was grundsätzlich den Einsatz clientseitiger Kompressionsalgorithmen ermöglicht. Weiterhin weist HBase jedem Datensatz eine Versionsnummer zu bzw. ermöglicht die range-basierte Abfrage von Datensätzen auf Basis der Versionsinformation. Das Anlegen des Versionszählers kann sowohl automatisch von HBase als auch benutzerdefiniert beim Einfügen eines neuen Datensatzes erfolgen.

HBase ist die freie Implementierung von Googles Big Table[Cha+08] und wurde für die verteilte Speicherung großer Mengen strukturierter Daten entwickelt. Der Einsatz von HBase setzt ein solides Verständnis der zu erwartenden Zugriffsmuster voraus, da auf Grund der hierarchischen Natur nur in eine Richtung effizient zugegriffen werden kann bzw. diese Einschränkung, je nach Anwendungsfall, gezielt durch Denormalisierung aufgehoben werden muss. Der hohe Stellenwert des Schemas unterscheidet Wide Column Stores grundsätzlich von anderen NoSQL-Systemen, die oft als schemalos gelten. Das Schema eines Wide Column Stores ist bestimmt durch dessen Key-Value-Hierarchie, über die sowohl die Verteilung der Datensätze im Cluster, als auch physikalische Speichereigenschaften, wie Cache-Größen oder das Schlüsselencoding, bestimmt werden.

Das Ziel bei der Schemamodellierung liegt darin, Datensätze, die logisch zusammen gehören und die mit hoher Wahrscheinlichkeit gemeinsam verarbeitet werden, auch physikalisch eng zusammen zu speichern. Dem gegenüber steht die Vermeidung sogenannter

[233] http://cassandra.apache.org/ (besucht am 29.05.2016)
[234] https://hbase.apache.org/ (besucht am 29.05.2016)
[235] http://db-engines.com/de/ranking/wide+column+store (besucht am 29.05.2016)
[236] http://bigdatanoob.blogspot.de/2012/11/hbase-vs-cassandra.html (besucht am 29.05.2016)

„*write hotspots*“ bzw. der Konzentration des gesamten Schreibzugriffs auf einzelne Clusterknoten. Ein ideales Schema erreicht eine hohe Lokalität zusammengehörender Datensätze und verteilt Schreiblasten gleichmäßig über den gesamten Cluster[237].

Datenmodell Das Datenmodell von HBase besteht im Wesentlichen aus acht Bestandteilen, die im Folgenden kurz vorgestellt werden. Die genaue Funktionsweise wurde dem HBase Reference Guide[238], Kapitel 22 bis 29, entnommen. Eine Visualisierung der Konzepte liefert Abbildung 9.

- **Namespace** Ein Namensraum ist eine logische Gruppierung von Tabellen und vergleichbar mit der Datenbank eines relationalen Datenbankmanagement Systems.
- **Table** Eine Tabelle besteht aus mehreren Zeilen (*Rows*) und wird einmalig, während der Schemadefiniton, einem Namensraum zugeordnet.
- **Row** Eine Zeile besitzt einen Zeilenschlüssel (*Row Key*) und beinhaltet eine oder mehrere Spalten (*Columns*). Zeilen werden aufsteigend, in lexikografischer Ordnung, gemäß ihrem Zeilenschlüssel gespeichert. Der *Row Key* ist das wichtigste Design-Element und sollte stets so gewählt werden, dass zusammengehörige Zeilen, basierend auf der Sortierreihenfolge, eng zusammenliegen.
- **Column** Eine Spalte besteht aus einer Spaltenfamilie (*Column Family*) und einem *Column Qualifier*. Der *Column Qualifier* wird durch einen Doppelpunkt (:) von der *Column Family* getrennt.
- **Column Family** Eine Spaltenfamilie stellt eine physikalische Gruppierung von Spalten (*Columns*) und ihren zugehörigen Spaltenwerten dar. Über die *Column Families* können physikalische Speichereigenschaften der gruppierten Spalten, wie z.B. die Größe des Schreibcaches, die Kompression der Spaltenwerte oder das Encoding der zugehörigen *Row Keys*, festgelegt werden. Der Bezeichner einer *Column Family* besteht ausschließlich aus druckbaren Zeichen und wird, im Unterschied zu den *Column Qualifiern*, statisch, zum Zeitpunkt der Schemadefinition, angelegt. Die Wahl der *Column Families* sollte daher mit Bedacht erfolgen, da eine nachträgliche Änderung mit einigem Aufwand verbunden ist.
- **Column Qualifier** Der *Column Qualifier* wird an die *Column Family* angefügt und referenziert, zusammen mit dem Zeitstempel (*Timestamp*), einen konkreten Datensatz. Ein *Column Qualifier* besteht aus einer beliebigen Folge von Bytes und kann zur Laufzeit modifiziert werden. Die Anzahl der *Column Qualifier* pro *Row* ist nicht festgelegt und kann sich von *Row* zu *Row* stark unterscheiden.
- **Cell** Eine Zelle in HBase ist exakt durch das Tupel {row, column, timestamp} bzw.

[237] http://www.doag.org/formes/servlet/DocNavi?action=getFile&did=6873888&file=2015-NN-Andreas_Buckenhofer-Wide-column_Stores_fuer_Architekten-Praesentation.pdf (besucht am 31.05.2016)

[238] http://hbase.apache.org/book.html#datamodel (besucht am 31.05.2016)

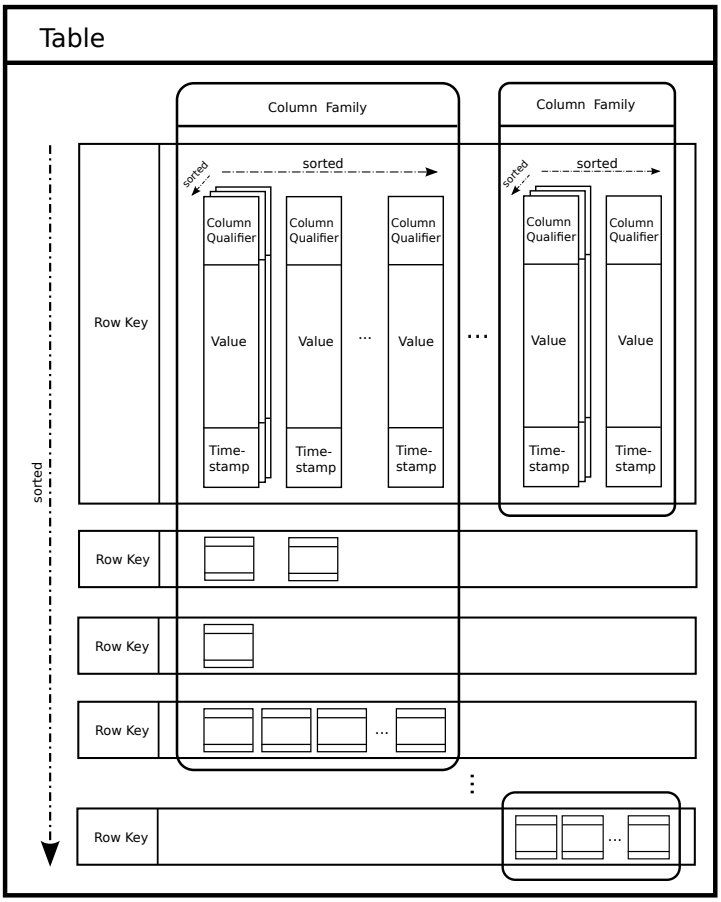
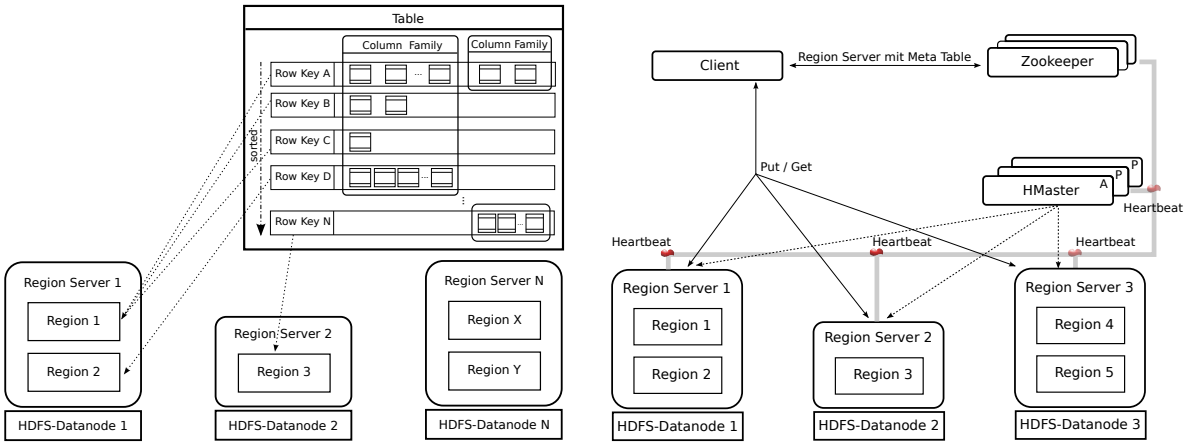


Abbildung 9: HBase – Datenmodell

{row, column family : column qualifier, timestamp} festgelegt und speichert die eigentlichen Nutzdaten als uninterpretierte Bytefolge. Die Zellen, oftmals auch nur als Key-Value Paare bezeichnet, bilden die eigentlichen Datensätze in HBase.

- **Timestamp** Jeder gespeicherte Datensatze (*Cell*) enthält neben den Nutzdaten eine Zeitstempel (*Timestamp*). Dieser repräsentiert die Version des gespeicherten Datensatzes und kann beim Einfügen vom Nutzer frei definiert werden. Wird kein Wert angegeben, setzt der verarbeitende Region Server den Zeitstempel standardmäßig auf den Zeitpunkt, an dem die Daten physikalisch ins Dateisystem geschrieben wurden. Im Gegensatz zu den *Row Keys* und *Column Qualifiers* (*Column Keys*) besteht der Zeitstempel aus einem long integer und enthält typischerweise einen Zeitwert in Form eines Unix-Zeitstempels oder eine vergleichbare, scalare Zeiteinheit. Die Datensätze (*Cells*) werden in absteigender Reihenfolge gespeichert, sodass beim Zellenzugriff der aktuellste Eintrag zuerst erscheint. Wird beim Zugriff auf eine Speicherzelle keine bestimmte Version angegeben, gibt HBase standardmäßig die aktuellste Version des Datensatzes zurück.



(a) Zusammenhang zwischen Rows und Regions (b) Systemkomponenten

Abbildung 10: HBase – Cluster-Architektur

HBase - Abbildung des Datenmodells auf die Cluster-Architektur Bei der Schemamodellierung mit HBase ist es wichtig ein grundlegendes Verständnis dafür zu entwickeln, wie die einzelnen Elemente des Datenmodells die Verteilung der Daten im Cluster und deren Zugriffscharakteristik beeinflussen. Aus diesem Grund werden im Folgenden die wichtigsten HBase-Komponeten vorgestellt und ein Zusammenhang zwischen ihnen und dem Datenmodell hergestellt. Dabei werden ausschließlich die für die Datenmodellierung relevanten Bestandteile diskutiert. Eine vollständige Beschreibung der HBase-Architektur liefern Carol McDonalds Blog Post „An In-Depth Look at the HBase Architecture“[239] oder Kapitel 63 bis 73 des HBase Reference Guides[240].

HBase Tabellen werden horizontal in sogenannte *Regions* aufgeteilt. Jede *Region* enthält eine Teilmenge der Tabellendaten und besteht im Wesentlichen aus einer sortierten, zusammenhängenden Menge aus *Rows*. Die *Rows* werden auf Basis ihres *Row Keys* sortiert und auf die einzelnen *Regions* verteilt (Abbildung 10(a))[241]. Die Verwaltung der *Regions* ist Aufgabe der *Region Server*. *Region Server* persistieren ihre Daten innerhalb von *Hadoop HDFS* und setzen deshalb direkt auf *HDFS Data Nodes* auf. Neben *Region Servern* und *Data Nodes* benötigt HBase noch *Zookeepr*[242] als verteilten Koordinierungsdienst und einen oder mehrere *HMaster*[239] für die Koordinierung der *Region Server* (Abbildung 10(b)). Eine detaillierte Beschreibung der Interaktion zwischen *Client*, *HMaster*, *Zookeeper* und *Region Servern* würde den Rahmen dieses Kapitels sprengen. Hier sei erneut auf den Blog-Post „An In-Depth Look at the HBase Architecture“ von Carol McDonald verwiesen[239].

Jeder Region Server besitzt einen *Write Ahead Log* (WAL) und einen globalen Lesecache,

[239] https://www.mapr.com/blog/in-depth-look-hbase-architecture (besucht am 02.06.2016)
[240] http://hbase.apache.org/book.html#_architecture (besucht am 02.06.2016)
[241] https://blog.cloudera.com/blog/2013/04/how-scaling-really-works-in-apache-hbase/ (besucht am 10.06.2016)
[242] https://zookeeper.apache.org/ (besucht am 13.07.2016)

den sogenannten *Block Cache*. Der *Write Ahead Log* liegt im HDFS und speichert noch nicht dauerhaft gespeicherte Datensätze persistent zwischen. Der *Block Cache* hält die am häufigsten gelesenen Datensätze im Hauptspeicher. Beim Erreichen der maximalen Cachegröße werden immer die jeweils ältesten Einträge aus dem Cache entfernt, sodass sich stets die zuletzt abgefragten Einträge bzw. die „Least Recently Used Cells“ im *Block Cache* befinden. Schreibende Zugriffe (*Put Requests*) werden ebenfalls im Hauptspeicher zwischengespeichert. Zu diesem Zweck existiert pro *Column Family* pro *Region* genau ein Schreibcache, der sogenannte *Memstore*. Jede *Memstore*-Instanz beinhaltet eine sortierte Menge *Cells* einer *Column Family*. Die persistente Datenspeicherung erfolgt in Form sogenannter *HFiles*. Die *HFiles* liegen redundant gespeichert im Hadoop Distributed File System und verwalten eine indizierte Menge HBase *Cells*. Für eine detaillierte Beschreibung der HFiles siehe[239,243].

Übersteigt der Speicherverbrauch eines einzelnen *Memstores*, oder der Speicherverbrauch der Gesamtheit aller *Memstores* innerhalb einer Region, einen bestimmten Grenzwert, werden die Inhalte aller *Memstores* der betroffenen *Region* in einen neuen *HFile* geschrieben[243]. Dieser Vorgang wird als *Region Flush*[239] bezeichnet.

Durch die Ausführung von *Region Flushs* nimmt die Anzahl der HFiles immer weiter zu. Nicht selten verteilt sich der Inhalt einer *Row* über mehrere *HFiles* und im Extremfall müssen beim Ausführen eines Scans oder eines *Get Requests* alle *HFiles* einer *Region* durchsucht werden. Wird die Anzahl der *HFiles* zu groß, ergeben sich negative Einflüsse auf die Systemperformance[243]. HBase wendet daher zwei Operationen zur Reduktion der *HFiles* an. Die Erste heißt *Minor Compaction* und fügt wenige kleine, aufeinanderfolgende *HFiles* zu einem größeren *HFile* zusammen. Die Zweite heißt *Major Compaction* und bündelt alle *HFiles* einer *Region* zu einem großen *HFile*. Wann welche Compaction-Strategie angewendet wird und wie viele *HFiles* in einer *Minor Compaction* zusammengeführt werden, legt die *Compaction Policy* fest. Die Auswahl der richtigen *Compaction Policy* hat bei hohen Schreiblasten großen Einfluss auf die Reaktionszeiten von HBase, da sowohl zu hohe als auch zu niedrige *Compaction* Raten die Cluster-Performance negativ beeinträchtigen[244]. Die Notwendigkeit der *Compactions* ist einer der Gründe, warum HBase eingangs als leseoptimiert beschrieben wurde.

4.5.2 HBase – Schemaentwurf

Beim Schemaentwurf sollten folgende Punkte beachtet werden:

- **Row Key** Der *Row Key* ist das wichtigste Design-Element. Er steuert die Verteilung der Daten auf die einzelnen *Regions* bzw. *Region Server* im Cluster. *Row Keys* sollten so gewählt werden, dass zusammenhängende Daten möglichst eng zusammen, in der selben *Region* oder auf dem selben *Region Server*, gespeichert

[243] http://blog.cloudera.com/blog/2012/06/hbase-io-hfile-input-output/ (besucht am 02.06.2016)
[244] http://hbase.apache.org/book.html#regions.arch (besucht am 02.06.2016)

werden. Schreibende Lasten sollten dabei möglichst gleichmäßig über alle *Regions* des Clusters verteilt werden[245].

- **Column Family** Die Dimensionierung des Write-Caches erfolgt auf Basis der *Column Families*. Es existiert genau ein Write-Cache pro *Region* pro *Column Family*, die Anzahl der *Column Families* sollte daher gering gehalten werden. Die HBase-Dokumentation spricht von ein bis drei *Column Families* pro Tabelle als typischen Wert[246]. Die Namen von *Column Families* müssen weder selbstdokumentierend noch sprechend sein. Kurze Namen sorgen für eine bessere Systemperformance[247].
- **Cell** HBase ist standardmäßig auf Zellgrößen von 100 KB und kleiner ausgelegt[248]. Für die Verwaltung sogenannter *Medium-sized Objects* (MOBs) muss die HBase-Konfiguration speziell angepasst werden. Die Optimierung erfolgt dabei auf Spaltenbasis und ermöglicht eine effiziente Verwaltung von Zellen mit einem Datenvolumen zwischen 100 KB und 10 MB[247].

Basierend auf den bisherigen Erkenntnissen wird folgendes Schema vorgeschlagen (siehe Tabelle 10):

	Wert	**Beispiel**
Namespace	wikipedia	„wikipdia“
Tabellenname	Wikiinstanz	„de“
Row Key	Artikelname ohne Leerzeichen	„Weinbau in Deutschland“ → „WeinbauinDeutschland“
Column Family	markup	„markup“
Column Qualifier	wom	„wom“
Cell value	Komprimiertes WOM-JSON	-
Timestamp	Long.max() - Erstellungszeitpunkt als Unixtime	1464976059

Tabelle 10: HBase – Schemaentwurf

Als *Row Key* wurde der Artikelname ohne Leerzeichen gewählt. Auf diese Weise verteilen sich Schreiblasten gemäß der lexikografischen Ordnung der Artikelnamen im Cluster. Insbesondere entstehen so beim Anlegen neuer Artikel keine Write-Hotspots, wie es beispielsweise bei der Verwendung des Erstellungsdatums, ohne zusätzliche Metrik, der Fall wäre. Eine *Row* enthält die komplette Versionshistorie eines Artikels. Die einzelnen Artikelversionen werden direkt auf den *Timestamp* der HBase *Cells* abgebildet.

Diese Design-Entscheidung hat Vor- und Nachteile. Der entscheidende Vorteil dabei ist die range-basierte Abfrage der Zeitstempel. Durch diese Funktion kann bereits beim *Scan*

[245] http://hbase.apache.org/book.html#rowkey.design (besucht am 03.06.2016)
[246] http://hbase.apache.org/book.html#regionserver_sizing_rules_of_thumb (besucht am 03.06.2016)
[247] http://hbase.apache.org/book.html#table_schema_rules_of_thumb (besucht am 03.06.2016)
[248] http://hbase.apache.org/book.html#hbase_mob (besucht am 03.06.2016)

einer Tabelle (siehe funktionale Eignung) gezielt eine Menge zusammenhängender Artikelversionen ausgewählt werden. Dies eignet sich hervorragend für die Abbildung von Querschnittanalysen.

Werden hingegen nur wenige Artikel mit einer großen Zahl an Einzelversionen abgefragt, reduziert sich die Parallelität der Abfragen und damit deren Performance. Grund dafür ist das Zusammenwirken von Map-Reduce und HBase. Standardmäßig verarbeitet ein *Mapper*, pro Aufruf der Map-Funktion, eine *Row* aus dem Datenbestand der *Data-Node* seiner zugeordneten *Region*. Im ungünstigsten Fall liegen alle Artikel bzw. *Rows* in der selben Region. Dies würde zu einer komplett sequentiellen Verarbeitung der Einzelversionen führen. Der beschriebene Effekt relativiert sich über die Menge der abgefragten Artikel und wird zusätzlich über die Wahl des *Row Keys* kompensiert. Kommt es dennoch zu einer Konzentration aller relevanten *Rows* in einer *Region*, kann die *getSplits* Methode der Klasse *TableInputFormat*[249] überschrieben werden. Sie beinhaltet den Verteilungsalgorithmus und ermöglicht eine individuelle Anpassung an den vorliegenden Anwendungsfall.

Pro Tabelle wird eine *Column Family* „markup“ angelegt. Der Bezeichner ist mit sechs Zeichen relativ kurz und die Zugriffscharakteristik der beinhalteten Spalten ist ähnlich. Werden die Wikiartikel als unkomprimiertes WOM-JSON gespeichert, müssen die Spalten auf Grund ihrer mittleren Größe von 308 KB (siehe Kapitel 3.1.2 Tabelle 2) als *medium-sized* deklariert werden.

Neben der funktionalen Eignung für das aktuelle Einsatzszenario ist die grundsätzliche Erweiterbarkeit des Schemas für zukünftige Anwendungsfälle ein wichtiges Qualitätsmerkmal. Wollte man beispielsweise das bestehende Backend einer Mediawiki Installation auf HBase migrieren und übergangsweise, sowohl die WOM-Repräsentation als auch das ursprüngliche Wikimarkup speichern, kann der „markup“ *Column Family*, pro *Row*, ein weiterer *Column Qualifier* „wm“ (für Wiki-Markup) hinzugefügt werden. Die Verwaltung der Versionen geschieht analog zum bisherigen Ansatz über den *Timestamp*.

Die Bereitstellung weiterer artikel-externer Informationen, wie z.B. den Autor einer Artikelversion oder die Anzahl der Artikelzugriffe, wäre ebenfalls denkbar. Hierfür ist eine weitere *Column Family*, „meta“, notwendig, die z.B. die *Column Qualifier* „author “, zur Ablage der Autoreninformation und „views “, zur Speicherung der Artikelzugriffe, enthält. Die Verwaltung und Abbildung der unterschiedlichen Autoren auf die einzelnen Artikelversionen geschieht über den Zeitstempel. Die Konsistenzsicherung ist dabei Aufgabe der Anwendung. Bei den Zugriffen pro Artikel handelt es sich prinzipiell um eine artikelglobale Metrik. Verzichtet man auf die Erhebung der zeitlichen Entwicklung der Zugriffe, ist für diese Spalte keine Versionierung erforderlich.

Auch die redundante Ablage häufig benötigter Informationen, wie z.B. die Artikel oder Webseiten auf die eine bestimmte Artikelversion verweist, wäre möglich. Dazu bedarf es einer weiteren *Column Family*, „lout“, zur Verwaltung ausgehender Links. Die Einträge verwenden dabei das Ziel, auf das verwiesen wird, als *Column Qualifier* und speichern z.B.

[249] http://hbase.apache.org/apidocs/org/apache/hadoop/hbase/mapreduce/TableInputFormat.html (besucht am 07.06.2016)

zwei Strings, „int“ oder „ext “, zur Unterscheidung interner oder externer Links als *value*. Bei externen Links wird das invertierte Linkziel (tld.domain.subdomain ...) als *Column Qualifier* verwendet. Die Zuordnung zu den jeweiligen Artikelversionen geschieht, analog zu den vorherigen Anwendungsfällen, über die Versionsnummer.

4.5.3 Funktionale Eignung

Die funktionale Eignung von HBase ergibt sich in Kombination mit Map-Reduce. Unter Verwendung des in Abschnitt 4.5.2 skizzierten Schemas können alle Versionen eines Artikels gezielt über *Row Key Scans* abgefragt werden. Die Auswahl einer bestimmten Artikelversion bzw. der Versionen, die in einem bestimmten Zeitfenster erstellt wurden, erfolgt über den *Timestamp*. Auf diese Weise lassen sich die verschiedenen zeitlichen Dimensionen der Queries abbilden und gezielt Daten vorfiltern. Die erreichte Selektivität entspricht nicht der klassischer SQL-artiger Abfragesprachen, trotzdem lassen sich die zu verarbeitenden Datenmengen damit drastisch reduzieren. Für eine Analyse der aktuellen Versionen aller Artikel der englischen Wikipedia, reduziert sich der Datenbestand von ca. 800 Millionen Artikelversionen auf etwa 5 Millionen. Bei historischen Queries ist die Datenreduktion noch höher. Hier müssen im schlimmsten Fall pro Artikel rund 46.000 Versionen ausgewertet werden.

Die eigentliche Datenanalyse erfolgt mittels in Java geschriebener Map- und Reduce-Funktionen. Diese greifen intern auf die Referenzimplementierung[250] von WOM zurück und können dessen vollen Funktionsumfang für die Umsetzung der unter 3.2 geforderten Abfragen verwenden. Die Mapper- und Reducer-Schnittstellen werden vom Hadoop-Framework vorgegeben[251].

Der Zugriff auf HBase erfolgt über ein von Hadoop definiertes *Scan*-Objekt[252], dessen Möglichkeiten zur Auswahl von Spalten durch die Angabe von *Filtern*[253] erweitert werden können. Das Ergebnis eines Table-Scans ist eine Menge von *Results*[254]. Jedes *Result* beinhaltet eine *Row*, deren Spaltenmenge auf Basis der Scanner-Einstellungen gefiltert bzw. eingegrenzt wurde.

Für eine einfache Kombination von Map-Reduce und HBase stellt Hadoop die Klasse *TableMapReduceUtil*[255] zur Verfügung. Über die Methoden *initTableMapperJob* und *initTa-*

[250] http://search.maven.org/#search|ga|1|org.sweble.wom3 (besucht am 08.06.2016)
[251] https://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html (besucht am 07.06.2016)
[252] http://www.cloudera.com/documentation/enterprise/5-4-x/topics/admin_hbase_scanning.html (besucht am 07.06.2016)
[253] http://www.cloudera.com/documentation/enterprise/5-6-x/topics/admin_hbase_filtering.html (besucht am 07.06.2016)
[254] https://hbase.apache.org/apidocs/org/apache/hadoop/hbase/client/Result.html (besucht am 07.06.2016)
[255] https://hbase.apache.org/apidocs/org/apache/hadoop/hbase/mapreduce/TableMapReduceUtil.html (besucht am 07.06.2016)

bleReducerJob können Map- und Reduce-Jobs angelegt werden, die Daten aus HBase lesen oder bei Bedarf in eine HBase-Tabelle zurückschreiben[256]. Das Ausgabeziel des Analyseergebnisses ist frei konfigurierbar. Das Zurückschreiben in HBase ist nur eine von vielen Optionen. Der Einfachheit halber bietet es sich an, Analyseergebnisse direkt in HDFS zu speichern. Ein Zurückschreiben in HBase ergibt unter Berücksichtigung des aktuell vorgeschlagenen Schemas nur dann Sinn, wenn das Verarbeitungsergebnis neue Artikelversionen beinhaltet. Einen solchen Anwendungsfall stellen die von Dorn et al. beschriebenen Wiki Content Transformations[DR13] dar. Als Eingabe für Map-Reducejobs können, neben HBase, auch beliebige andere Datenquellen dienen und stattdessen die Berechnungsergebnisse in HBase gespeichert werden[256]. Ein möglicher Anwendungsfall dafür ist das Einparsen der Wikipedia-XML-Dumps und die Umwandlung des beinhalteten Wikitexts in WOM.

Wichtig bei der Kombination von Map-Reduce und HBase ist die Verteilung der Daten auf die *Mapper* und die Anzahl der gestarteten Map-Tasks. Bei jedem Aufruf der Map-Funktion eines Map-Tasks wird genau eine *Row* übergeben. Standardmäßig bzw. bei Zuhilfenahme der *TableMapReduceUtil*, welche intern wiederum auf die Klasse *TableInputFormat*[257] zurückgreift, wird pro *Region* pro *Table* ein Map-Task gestartet. Ist dieses Verhalten nicht gewünscht, kann die *getSplits* Methode der *TableInputFormat* Klasse überschrieben werden. Die Anzahl der Reduce-Tasks wird manuell pro Map-Reduce-Job festgelegt[258].

Bei dieser Art der Datenverarbeitung liegt der Schlüssel zum Erfolg in einer möglichst gleichmäßigen Verteilung der Daten im Cluster. Im Idealfall ist die Anzahl der Spalten pro Zeile relativ gleich verteilt. Logisch zusammengehörende Daten liegen in der selben *Region*- oder zumindest auf dem selben *Region Server*- und die Gesamtheit aller *Rows* einer Tabelle verteilt sich gleichmäßig über alle *Regions* und somit *Region Server* des Clusters. Auf diese Weise muss bei gezielten Key-Value-Zugriffen nur ein *Region Server* angesprochen werden und komplexe Map-Reduce-Jobs, die Analysen oder Aggregationen über alle *Rows* durchführen, können stark parallelisiert werden (ein *Mapper-Task* pro *Region*) und meist auf lokalen Datenbeständen operieren. Hadoop versucht bei der Verteilung der *Rows* auf die *Mapper* stets sicher zu stellen, dass die verarbeiteten Daten direkt lokal in der *Data Node* des zugehörigen *Region Servers* liegen.

Die Funktionsweise der *Mapper* und *Reducer* wird durch die Implementierung der gleich-

[256] http://archive.cloudera.com/cdh5/cdh/5/hbase-0.98.1-cdh5.1.4/book/mapreduce.example.html (besucht am 07.06.2016)
[257] http://hbase.apache.org/apidocs/org/apache/hadoop/hbase/mapreduce/TableInputFormat.html (besucht am 07.06.2016)
[258] http://archive.cloudera.com/cdh5/cdh/5/hbase-0.98.1-cdh5.1.4/book/mapreduce.example.html (besucht am 07.06.2016)

namigen Klassen[259,260] bzw. der spezialisierten Varianten *TableMapper*[261] und *TableReducer*[262] festgelegt. Ausgehend von der komprimierten Speicherung der Wikipedia-Artikel als WOM-JSON, beinhaltet eine Mapper-Implementierung im Wesentlichen folgende Schritte:

Eine Iteration über alle Spalten der übergebenen Zeile. Jede Spalte wird

- dekomprimiert,
- deserialisiert
- und anschließend, mit Hilfe der Java-Implementierung von WOM, analysiert.
- Das Analyseergebnis kann optional, für einen anschießenden Reduce-Schritt, in den *Mapper Context* eingefügt werden.

Da die Queries bei diesem Ansatz nicht in einer deklarativen Abfragesprache formuliert, sondern direkt in Java programmiert werden und dabei auf die Referenzimplementierung von WOM 3.0 zurückgegriffen werden kann, sind den funktionalen Möglichkeiten, was die Art der Suchparameter oder die Granularität des Abfrageergebnis angeht, keine Grenzen gesetzt. Die zeitlichen Dimensionen der Abfragen lassen sich direkt auf *Table-Scans* und den range-basierten Zugriff über die *Timestamps* abbilden. Einzig was die Formulierung komplexer Aggregationen angeht ist das Map-Reduce-Modell limitiert. Hier könnten, je nach Komplexität der gewünschten Aggregation, mehrere Map-Reduce Schritte notwendig sein. Die *Reducer* müssten dabei Zwischenergebnisse in HDFS speichern, die selber wiederum als Eingabe für neue Map-Tasks einer weiteren Iteration dienen. Die *Mapper* des zweiten Durchlaufs können ihre Ergebnisse erneut an *Reducer* weitergeben, die ihrerseits ein noch weiter verfeinertes Zwischenergebnis in HDFS speichern usw.

4.5.4 Nicht-funktionale Eignung

Eines der wesentlichen Ziele bei der Entwicklung von HBase, bzw. dessen Vorbild Google Big Table, ist die verteilte Verarbeitung großer Datenmengen auf Commodity Hardware [Cha+08, S. 1]. Die HBase-Architektur ist daher grundsätzlich auf horizontale Skalierbarkeit ausgelegt. Ob HBase in der Lage ist, die geforderten Datenmengen zu verwalten, hängt stark von dem verwendeten Schema und der Art, wie sich die Daten im Cluster verteilen, ab.

[259] https://hadoop.apache.org/docs/r2.6.2/api/org/apache/hadoop/mapreduce/Mapper.html (besucht am 07.06.2016)
[260] https://hadoop.apache.org/docs/r2.6.2/api/org/apache/hadoop/mapreduce/Reducer.html (besucht am 07.06.2016)
[261] http://hbase.apache.org/apidocs/org/apache/hadoop/hbase/mapreduce/TableMapper.html (besucht am 07.06.2016)
[262] https://hbase.apache.org/apidocs/org/apache/hadoop/hbase/mapreduce/TableReducer.html (besucht am 07.06.2016)

Je nach dem, ob die Artikel komprimiert oder unkomprimiert gespeichert werden, ergeben sich folgende Kenngrößen:

Kenngröße	Unkomprimiert	LZMA2-komprimiert
Versionshistorie der englischen Wikipedia	800 Millionen Artikelversionen	
Durchschnittliche Artikelgröße (hierarchisches WOM-JSON)	308 KB	24 KB
Gesamtgröße englische Wikipedia	247 Terabyte	19 Terabyte

Tabelle 11: HBase – Kenngrößen für die Berechnung der Hardwareressourcen

Wendet man das unter 4.5.2 vorgeschlagene Schema, unter Berücksichtigung der gerade erhobenen Kenngrößen, auf den Datenbestand der englischen Wikipedia an, ergibt sich folgender Soll-Ist-Vergleich:

Kenngröße	Soll-Wert	Ist-Wert	
		unkomprimiert	LZMA2-kompr.
Cell size	x < 100 KB oder 100 KB < x < 10 MB[263]	308 KB^{m}	24 KB^{m}
Region size	5 - 50 GB[247,264]	40 GB^{r}	20 GB^{r}
Column families per table	1 - 3[247]	1^{a}	1^{a}
Regions per region server	20 - 200[264]	40	50
Data per region server[265]	ca. 6 TB[246]	1,6 TB^{r}	1 TB^{r}
Number of region servers	-	155^{e}	19^{e}
Data to block cache ratio	-	5%^{s}	5%^{s}
Block cache size[266]	-	80 GB^{e}	50 GB^{e}
Memstore cache size per column family per region	128 - 256 MB[267]	128 MB^{r}	128 MB^{r}
Total memstore cache size per region server	-	5,12 GB^{e}	6,4 GB^{e}
JVM heap memory[268]	x < 24 GB[267]	12 GB^{e}	16 GB^{e}
Total RAM per region server	-	ca. 100 GB^{e}	ca. 74 GB^{e}

m) Messwert (siehe 3.1.2); a) Aus Schema abgeleitet (siehe 4.5.2); e) Errechneter Wert; r) Anhand von Richtwert festgelegt; s) Schätzwert

Tabelle 12: HBase – Ressourcenbedarf für die Verwaltung der englischen Wikipedia

[263] http://hbase.apache.org/book.html#hbase_mob (besucht am 09.06.2016)
[264] http://archive.cloudera.com/cdh5/cdh/5/hbase-0.98.6-cdh5.2.3/book/regions.arch.html (besucht am 03.06.2016)

Alle Konfigurationsparameter liegen, sowohl im komprimierten als auch im unkomprimierten Fall, im Rahmen der empfohlenen Richtwerte. Auf Grund der guten horizontalen Skalierbarkeit und der Veröffentlichung unter der Apache-Lizenz, Version 2.0[269], wird HBase auch aus nicht-funktionaler Sicht als geeignet eingestuft.

4.5.5 Fazit

HBase ist sowohl funktional als auch nicht-funktional geeignet. Die funktionale Eignung ergibt sich in Kombination mit Map-Reduce, da *Get* und *Scan* ausschließlich für eine Vorauswahl der Artikel, basierend auf ihrem Namen und ihrer Version, genutzt werden können. Die eigentlichen inhaltsbasierten Abfragen werden mit Hilfe von Map-Reduce-Funktionen realisiert. Jede Abfrage wird dabei auf einen- oder eine Folge mehrerer- Map-Reduce-Jobs abgebildet. Durch den Einsatz von Java, in Kombination mit der Referenzimplementierung von WOM, sind den Abfragen funktional prinzipiell keine Grenzen gesetzt. Der hohe Funktionsumfang geht jedoch zu Lasten der Benutzbarkeit, da jede Anfrage vom User manuell programmiert werden muss. Dies setzt Java-Kenntnisse, ein grundlegendes Verständnis des Map-Reduce-Modells, sowie Basiswissen über die Traversierung von Bäumen voraus. Für die Parallelisierung der Abfragen sorgt das Hadoop-Framework. In den meisten Anwendungsfällen reicht die eingebaute Verteilungsstrategie aus. Konzentrieren sich jedoch alle relevanten *Rows* in einer *Region*, muss der Anwender auch hier manuell eingreifen.

Im nicht-funktionalen Bereich überzeugt HBase durch hervorragende horizontale Skalierbarkeit und den Einsatz beliebiger, clientseitiger Kompressionsalgorithmen. Letzteres bietet enorme Möglichkeiten zur Einsparung von Hardwareressourcen. Die Hochrechnung im Abschnitt nicht-funktionale Eignung hat gezeigt, dass der Datenbestand der englischen Wikipedia, bei Anwendung von LZMA2-Kompression, mit ca. 19 statt 155 *Region Servern* und 74 GB RAM statt 100 GB RAM pro Server verwaltet werden kann. Allgemein gilt; je höher die Kompressionsraten der Daten, desto geringer die Anzahl der benötigten Server, sowie deren Festplatten- und RAM-Kapazität. Welchen Einfluss die Datenkompression auf die Rechenleistung der benötigten CPUs oder die Kompressions- und Dekompressiondauer hat, hängt von dem verwendeten Kompressionsalgorithmus ab. Basierend auf dem erwarteten Workload wird zur Verwendung eines asymmetrischen Kompressionsalgorithmus mit relativ hohem Berechnungsaufwand bei der Datenkompression und

[265] Die effektive Festplattenauslastung ergibt sich durch Multiplikation der Nutzdaten pro Region Server mit dem HDFS-Replikationsfaktor (standardmäßig drei).

[266] Off-Heap Bucket Cache http://www.cloudera.com/documentation/enterprise/5-5-x/topics/admin_hbase_blockcache_configure.html (besucht am 10.06.2016)

[267] http://archive.cloudera.com/cdh5/cdh/5/hbase-0.98.6-cdh5.2.3/book/regions.arch.html (besucht am 09.06.2016)

[268] Standardmäßig ist die *total memstore cache size per region server* auf 40% des JVM Heaps begrenzt http://hadoop-hbase.blogspot.de/2013/01/hbase-region-server-memory-sizing.html (besucht am 10.06.2016).

[269] https://hbase.apache.org/license.html (besucht am 10.06.2016)

geringem Rechenaufwand bei der Dekompression geraten.

5 Zusammenfassung

Die Kombination von WOM und NoSQL-Systemen zur Erschließung des Datenbestands der Wikipedia ist eine vielschichtige und komplexe Aufgabe. Diese Arbeit ist die Erste in einer Serie weiterer Arbeiten. Sie konkretisiert im ersten Teil die Anforderungen an die zu realisierende Analysedatenbank und dokumentiert bis dahin unveröffentlichte Änderungen zwischen den Versionen 1.0 und 3.0 des WOM-Formats. Die Datenbanksystemanalyse untersucht diverse, kostenlose Systeme aus den Bereichen nativer XML Datenbanken (Sedna, BaseX, eXist-db, X-Hive, Oracle Berkley DB XML und Apache Xindice), XML-enabled-Databases (PostgrSQL, MySQL, MariaDB), JSON-basierter Document Stores (Couchbase Server, MongoDB), Multi-Model-Datenbanken (ArangoDB, OrientDB), sowie Wide Column Stores (Apache HBase) auf ihre Eignung hinsichtlich der zuvor definierten Anforderungen.

Die Anforderungen sind dabei bestimmt durch die zu verarbeitende Datenmenge, die Queries, sowie den charakteristischen Workload. Die durchschnittliche Artikelgröße aller Artikel der englischen Wikipedia beträgt 42,69 KB (vgl. Kapitel 3.1.2). Der Speicherverbrauch des XML-Outputs von WOM (vgl. Listing 2) ist bei deaktiviertem „Pretty Printing“[270] etwa 2,52-mal höher als der des zu erzeugenden Wiki-Markups. Der Faktor für WOM-JSON/hierarchisches WOM-JSON (vgl. Listing 3) liegt bei 7,22. Die durchschnittliche Artikelgröße für WOM-XML beträgt etwa 107,5 KB, bei Einsatz von WOM-JSON/hierarchische WOM-JSON werden 308,06 KB erwartet (vgl. Tabelle 2). Für die Verwaltung der englischen Wikipedia ergeben sich damit Datenvolumina von 86 TB (WOM-XML) und 246,45 TB (WOM-JSON/hierarchisches WOM-JSON).

Das Query-Profil ist in aktuelle Queries, Längs- und Querschnitte unterteilt, wobei die Artikel in jeder zeitlichen Achse nach vorgegebenen Knoteninhalten, Knoten mit einem bestimmten Typ oder speziellen Knotenattributen, sowie nach bestimmten Knotenkonstellationen / Knotenkontexten durchsucht werden können. Die Granularität der Abfrageergebnisse reicht von einzelnen Artikel-Identifiern, über Artikelabschnitte (Teilbäume), bis hin zu ganzen Artikeln (vgl. Kapitel 3.2).

Der Workload umfasst hauptsächlich lesende Zugriffe und ist durch komplexe Analysen, bestehend aus beliebigen Kombinationen der zuvor diskutierten Queries, geprägt. Die lesenden Zugriffe werden von einem Strom aus 1,4 Einfügeoperationen pro Sekunde für die englische Wikipedia, bzw. 5,2 Einfügeoperationen pro Sekunde bei Verwaltung der gesamten Wikipedia begleitet (vgl. Kapitel 3.3).

[270] „Pretty Printing“ bezeichnet das zusätzliche Einfügen von nicht inhaltstragenden Leerzeichen, Tabstops oder Absätzen zur Steigerung der Lesbarkeit und Vereinfachung der Wahrnehmung von Sinnzusammenhängen für das menschliche Auge.

Als Ergebnis der Systemanalyse (vgl. Kapitel 4) verbleiben Couchbase Server und Apache HBase. Beide Systeme sind grundlegend unterschiedlich (JSON-basierter Document Store vs. Wide Column Store) und markieren zwei gänzlich verschiedene Ansätze zur Lösung der Aufgabenstellung. Couchbase Server bietet mit N1QL eine deklarative, SQL-artige Querylanguage zur direkten Ausführung inhaltsbezogener Suchen innerhalb der Datenbank an, deren Performance zusätzlich durch das Setzen globaler Sekundärindizes optimiert werden kann. Weiterhin steht mit den MapReduce-Views ein Mechanismus zur Definition benutzerdefinierter Inidizes zur Verfügung, mit dessen Hilfe direkt auf besonders relevante Dokument-IDs oder extrahierte Aggregate zugegriffen werden kann. Die inhaltsbezogene Suche und die daraus resultierende Verwaltung der Artikel als JSON-Dokumente zieht jedoch, je nach Replikationsfaktor der Daten, einen Ressourcenbedarf von 53, 99 oder 149 Servern, mit jeweils 128 GB Arbeitsspeicher, etwa acht virtuellen CPUs und im Mittel 4,4 TB persistentem Speicher nach sich. Das genaue Datenvolumen des persistenten Speichers kann auf Grund der verwendeten Snappy-Kompression nicht exakt bestimmt werden und variiert zwischen 2,68 TB und 6,13 TB (vgl. Tabelle 7).

Apache HBase geht einen anderen Weg und speichert die Nutzdaten als uninterpretierte Bytefolge. Für den Einsatz von HBase ist die Definition eines Schemas erforderlich. Das Schema bestimmt welche Abfragen formuliert werden können und wie sich die Daten im Cluster verteilen. Die Schemadefinition befindet sich in Kapitel 4.5.2. Das dort modellierte Schema verteilt die Artikel gemäß ihres Titels im Cluster und gleicht den HBase-internen Versionszähler mit dem Erstellungsdatum der Artikel ab. Auf diese Weise wird eine Vorauswahl der Artikelversionen entlang der unterschiedlichen zeitlichen Achsen ermöglicht. Die Fähigkeit zur Formulierung inhaltsbezogener Abfragen ergibt sich erst in Kombination mit Hadoop-MapReduce, wobei auch hier das Schema den Umfang der Parallelisierbarkeit und damit die Abfrageperformance festlegt. Für die Implementierung der Abfragen kommt Hadoop-MapReduce zum Einsatz. Die *Mapper* und *Reducer* können direkt auf die Referenzimplementierung von WOM zurückgreifen, was ein hohes Maß an Flexibilität bei der Formulierung der Abfragen ermöglicht. Durch die interne Datenrepräsentation als uninterpretierte Bytefolge und die framework-gestützte Parallelisierung der MapReduce-Jobs entsteht zusätzlicher Raum für den Einsatz clientseitiger Kompressionsalgorithmen. Für die unkomprimierte Verwaltung der englischen Wikipedia ergibt sich ein Ressourcenbedarf von 155 Servern, mit jeweils 100 GB, exklusiv für HBase reserviertem Arbeitsspeicher und 1,6 TB, 3,2 TB oder 4,8 TB persistentem Speicher. Der genaue Wert hängt von dem Replikationsfaktor der Daten ab (vgl. Tabelle 12). Für die Ausführung der MapReduce-Jobs muss zusätzlicher Haupt- und Festplattenspeicher eingeplant werden, sodass der geschätzte effektive Ressourcenbedarf pro Server bei 2 TB, 4 TB oder 6 TB persistentem Speicher und 128 GB Arbeitsspeicher liegt. Bei Einsatz von LZMA2-Kompression reduziert sich die Anzahl der *Region Server* auf 19 mit jeweils 74 GB, exklusiv für HBase reserviertem, Arbeitsspeicher und 1 TB, 2 TB oder 4 TB persistentem Speicher (vgl. Tabelle 12). Kalkuliert man die Ausführung der MapReduce-Jobs mit ein, steigt der Ressourcenbedarf auf geschätzte 100 GB Arbeitsspeicher und 1,5 TB, 3 TB oder 4,5 TB persistentem Speicher pro Server.

Welches der Systeme für den vorliegenden Anwendungsfall das Geeignetere ist, konnte

im Verlauf dieser Arbeit nicht geklärt werden und hängt letztlich von den verfügbaren Hardwareressourcen, sowie von den technischen Fähigkeiten der Enduser ab. Basierend auf den gesammelten Daten haben sowohl die Community-Edition von Couchbase Server, als auch die Kombination aus Apache HBase und Hadoop-MapReduce das Potential zur Verwaltung und Analyse der WOM-Repräsentation des Datenbestands der Wikipedia.

6 Ausblick

Der letzte offene Punkt vor der Inbetriebnahme des ersten Systemprototypen ist die Reduktion des Datenvolumens. Je nachdem, welches Datenbanksystem eingesetzt wird, ergeben sich zwei unterschiedliche Herangehensweisen.

Couchbase Server Bei Einsatz von Couchbase Server sollte eine Größenreduktion des (hierarchischen) WOM-JSON-Outputs in Betracht gezogen werden. Aktuell umfasst die hochgerechnete mittlere Artikelgröße der (hierarchischen) WOM-JSON-Repräsentation 308,6 KB. Der Wert für WOM-XML liegt bei 107,5 KB (vgl. Tabelle 2). Dieser Messwert ist untypisch, im Normalfall ist die JSON-Repräsentation, auf Grund der fehlenden öffnenden und schließenden XML-Tags, kleiner als die zugehörige XML-Variante. Im vorliegenden Fall trifft das nicht zu. Die Ursache liegt in der Wiederholung der künstlich eingeführten *!type* und *!children / !value* Schlüssel (vgl. Abbildung 3). Eine Möglichkeit mit dem Problem umzugehen ist die Reduktion der künstlich eingefügten Schlüssel auf die kürzestmögliche Zeichenfolge (aus *!type* wird *!t*, *!children* wird in *!c* umgewandelt etc.). Ein offensichtlicher Nachteil bei dieser Vorgehensweise ist die Reduktion der Lesbarkeit der Daten. Dieser Aspekt hat beim Einsatz von Couchbase jedoch einen hohen Stellenwert. Die JSON-Schlüssel tauchen direkt in den N1QL-Queries auf und beeinflussen damit maßgeblich die Lesbarkeit der Abfragen (vgl. Listing 6). Sollte die Wahl auf Couchbase Server fallen, muss eine Grundlagenentscheidung getroffen werden, ob Enduser direkt auf die Datenbank zugreifen dürfen oder ob der Zugriff über ein Query-Repository abstrahiert wird. Ersteres erfordert ein gewisses, intuitives Verständnis der Daten und damit längere Schlüssel. Beim Einsatz eines Query-Repositories können die Schlüsselbezeichner stärker gekürzt und der Bedienkomfort über die API zurückgewonnen werden.

HBase speichert die Daten als uniterprätierte Bytefolge, die eigentliche Datenverarbeitung wird durch die Java-API von WOM abstrahiert. Die Priorität liegt hier auf der Reduktion des Datenvolumens und möglichst kurzen De- und Serialisierungszeiten. Als erster Zwischenschritt kann WOM-XML zur Ablage der Daten innerhalb von HBase verwendet werden. Damit reduziert sich das Datenvolumen von geschätzten 246 TB auf 86 TB (vgl. Tabelle 2). Der nächste, bereits exemplarisch für den WOM-JSON-Output beschriebene, Schritt ist die Anwendung clientseitiger Datenkompression. Die dadurch erreichte Kompressionsrate (vgl. Tabelle 1) und Reduktion der Hardwareressourcen (vgl. Tabelle 12) ist

immens. Der zusätzliche Rechenaufwand zur (De-)Komprimierung verlängert jedoch die De- und Serialisierungsdauer der Daten. Langfristig bietet sich daher der Einsatz eines Binärformats wie Apache Avro[271] zur Ablage der Daten in HBase an. Auf diese Weise wird die Reduktion des Datenvolumens mit kurzen De- und Serialisierungszeiten verknüpft. Im Fall von Apache Avro können die bestehenden XML- und JSON-Formate direkt weiterverwendet und als String in ein Avro-Schema eingebettet werden[272]. Das selbe Vorgehen hat sich bereits bei der Vorverarbeitung der Daten bewährt (vgl. Kapitel 3.1.2) und kann beim Einsatz von Apache HBase direkt adaptiert werden.

[271] https://avro.apache.org/docs/current/index.html (besucht am 13.07.2016)
[272] https://avro.apache.org/docs/current/gettingstartedjava.html (besucht am 13.07.2016)

Abbildungsverzeichnis

1 WOM Knotenmenge . . . 7
2 WOM-Element-Groups . . . 12
3 WOM-Element-Content-Groups . . . 13
4 Demo Seite – HTML Rendering. . . . 16
5 Messung der Artikelgrößen in Abhängigkeit vom Datenformat . . . 21
6 Couchbase – Teilschritte beim Erstellen einer View . . . 39
7 Couchbase – Die Map- und Reduce-Operationen . . . 41
(a) Map-Schritt . . . 41
(b) (Re)reduce-Schritt . . . 41
8 MongoDB – Beispielbaumstruktur zur Veranschaulichung der Verwaltung von Bäumen . . . 47
9 HBase – Datenmodell . . . 72
10 HBase – Cluster-Architektur . . . 73
(a) Zusammenhang zwischen Rows und Regions . . . 73
(b) Systemkomponenten . . . 73

Auflistungsverzeichnis

1 Demo Seite – Wiki Markup. . . . 16
2 Demo Seite – WOM-XML-Repräsentation. . . . 16
3 Demo Seite – WOM-JSON-Repräsentation . . . 17
4 Couchbase – N1QL IN- und WITHIN-Operator . . . 36
5 Couchbase – N1QL Ergebnisobjekt . . . 37
6 Couchbase – Verarbeitung einer komplexen Abfrage mit N1QL . . . 37
7 Couchbase – Rückgabe von Teilbäumen mit N1QL . . . 38
8 Couchbase – Rückgabe ganzer Artikel mit N1QL . . . 38
9 Couchbase – Querschnittanalysen mit N1QL . . . 38
10 Couchbase – Längsschnittanalysen mit N1QL . . . 38
11 Couchbase – Signatur der Map-Funktion . . . 39
12 Couchbase – Beispiel für eine einfache Map-Only-View . . . 40
13 Couchbase – Signatur der Reduce-Funktion . . . 40
14 Couchbase – Beispiel für eine View mit Reduce-Schritt – Map-Funktion . . . 41
15 Couchbase – Beispiel für eine View mit Reduce-Schritt – Reduce-Funktion 41
16 Couchbase – Aufbau der URL für den Zugriff auf eine MapReduce-View . 43
17 Couchbase – Querschnittanalysen mit MapReduce-Views – Teil 1.1 . . . 43
18 Couchbase – Querschnittanalysen mit MapReduce-Views – Teil 1.2 . . . 43
19 Couchbase – Längsschnittanalysen mit MapReduce-Views – Teil 1 . . . 43
20 Couchbase – Längsschnittanalysen mit MapReduce-Views – Teil 2 . . . 43
21 MongoDB – Signatur des Aggregate-Kommandos . . . 44
22 MongoDB – Signatur der Pipeline-Stufe lookup . . . 45

23 Child References – Methode zur Umwandlung von Bäumen in Listen . . . 47
24 Parent References – Methode zur Umwandlung von Bäumen in Listen . . 47
25 Array of Ancestors – Methode zur Umwandlung von Bäumen in Listen . . 47
26 Materialized Paths – Methode zur Umwandlung von Bäumen in Listen . . 47
27 Nested Sets – Methode zur Umwandlung von Bäumen in Listen 47
28 Demo Seite – flache WOM-JSON-Repräsentation 48
29 MongoDB – Verarbeitung einer komplexen Abfrage mit der Aggregation Pipeline – Teil 1 49
30 MongoDB – Verarbeitung einer komplexen Abfrage mit der Aggregation Pipeline – Teil 2 50
31 MongoDB – Bestimmung der Granularität des Abfrageergebnisses mit der Aggregation-Pipeline 51
32 MongoDB – Querschnittanalysen mit der Aggregation-Pipeline 52
33 MongoDB – Längsschnittanalysen mit der Aggregation-Pipeline 52
34 MongoDB – Signatur des MapReduce-Kommandos 52
35 MongoDB – Signatur der Map-Funktion 53
36 MongoDB – Signatur der Reduce-Funktion 54
37 MongoDB – Eigenschaften der Reduce-Funktion 54
38 MongoDB – Signatur der Finalize-Funktion 54
39 MongoDB – Längsschnittanalysen mit dem MapReduce-Kommando . . . 55
40 MongoDB – Querschnittanalysen mit dem MapReduce-Kommando 55

Tabellenverzeichnis

1 Messung der Artikelgrößen in Abhängigkeit vom Datenformat 20
2 Hochrechnung des erwarteten Datenvolumens der englischen Wikipedia in Abhängigkeit vom Datenformat 22
3 Couchbase – Commodity Hardware 57
4 Couchbase – Variablen für die Berechnung der Hardwareressourcen des Core Data Service 58
5 Couchbase – Konstanten für die Berechnung der Hardwareressourcen des Core Data Service 59
6 Couchbase – Ressourcenbedarf für die Verwaltung der englischen Wikipedia – Multidimensional Scaling 60
7 Couchbase – Ressourcenbedarf für die Verwaltung der englischen Wikipedia – Symmertic Scaling 61
8 MongoDB – Kenngrößen für die Berechnung der Hardwareressourcen . . 64
9 MongoDB – Ressourcenbedarf für die Verwaltung der englischen Wikipedia in Abhängigkeit vom Replikationsfaktor der Dokumente 65
10 HBase – Schemaentwurf 75
11 HBase – Kenngrößen für die Berechnung der Hardwareressourcen 80
12 HBase – Ressourcenbedarf für die Verwaltung der englischen Wikipedia . 80

Literatur

[Bou99] Ronald Bourret. *XML and Databases*. 1999. URL: http://ece.ut.ac.ir/dbrg/seminars/AdvancedDB/2006/Sanamrad-Hoseininasab/Refrences/6.pdf (besucht am 27. 05. 2016).

[Cha+08] Fay Chang u. a. „Bigtable: A Distributed Storage System for Structured Data“. In: *ACM Trans. Comput. Syst.* 26.2 (Juni 2008), 4:1–4:26. ISSN: 0734-2071. DOI: 10.1145/1365815.1365816. URL: http://doi.acm.org/10.1145/1365815.1365816 (besucht am 31. 05. 2016).

[DG08] Jeffrey Dean und Sanjay Ghemawat. „MapReduce: Simplified Data Processing on Large Clusters“. In: *Commun. ACM* 51.1 (Jan. 2008), S. 107–113. ISSN: 0001-0782. DOI: 10.1145/1327452.1327492. URL: http://doi.acm.org/10.1145/1327452.1327492 (besucht am 29. 05. 2016).

[DR11a] Hannes Dohrn und Dirk Riehle. „Department Informatik“. In: (2011). URL: http://www.opus4.kobv.de/opus4-fau/oai/container/index/docId/1845 (besucht am 18. 01. 2016).

[DR11b] Hannes Dohrn und Dirk Riehle. „Design and implementation of the sweble wikitext parser: unlocking the structured data of wikipedia“. In: *Proceedings of the 7th International Symposium on Wikis and Open Collaboration*. ACM, 2011, S. 72–81. URL: http://dl.acm.org/citation.cfm?id=2038571 (besucht am 18. 01. 2016).

[DR13] Hannes Dohrn und Dirk Riehle. „Design and implementation of wiki content transformations and refactorings“. In: *Proceedings of the 9th international symposium on open collaboration*. ACM, 2013, S. 2. URL: http://dl.acm.org/citation.cfm?id=2491057 (besucht am 18. 01. 2016).

[DR14] Hannes Dohrn und Dirk Riehle. „Fine-grained change detection in structured text documents“. In: ACM Press, 2014, S. 87–96. ISBN: 978-1-4503-2949-1. DOI: 10.1145/2644866.2644880. URL: http://dl.acm.org/citation.cfm?doid=2644866.2644880 (besucht am 18. 01. 2016).

[Fre+12] Sergio Miranda Freire u. a. „Performance of XML Databases for Epidemiological Queries in Archetype-Based EHRs“. In: Scandinavian Conference on Health Informatics 2012, October 2-3, Linköping, Sweden. Linköping University Electronic Press, 2012, S. 51–57. URL: http://www.diva-portal.org/smash/record.jsf?pid=diva2:558990&dswid=9762 (besucht am 05. 04. 2016).

[Fre+16] Sergio Miranda Freire u. a. „Comparing the Performance of NoSQL Approaches for Managing Archetype-Based Electronic Health Record Data“. In: *PLOS ONE* 11.3 (9. März 2016), e0150069. ISSN: 1932-6203. DOI: 10.1371/journal.pone.0150069. URL: http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0150069 (besucht am 05. 04. 2016).

[GHS07] Christian Grün, Alexander Holupirek und Marc H. Scholl. „Visually Exploring and Querying XML with BaseX“. In: Datenbanksysteme in Business, Technologie und Web (BTW). 2007, S. 629–632. ISBN: 978-3-88579-197-3. URL:

https://kops.uni-konstanz.de/handle/123456789/3007 (besucht am 05. 04. 2016).

[Grü+06] Christian Grün u. a. „Pushing XPath Accelerator to its Limits“. In: *Evaluation of Data Management Systems*. 2006.

[KN16] Marcus Klang und Pierre Nugues. „WikiParq: A Tabulated Wikipedia Resource Using the Parquet Format“. In: *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC 2016)*. 2016, S. 4141–4141. URL: http://lup.lub.lu.se/record/5e2a7be2-9bc2-4d72-bec3-618939d0b729 (besucht am 25. 05. 2016).

[NM] Tom Neumerkel und Robert Manthey. „Funktionsumfang und Eignung von XML-Datenbanken für Multimedia-und Metadaten“. In: (). URL: http://www.minf.uni-bamberg.de/lwa2013/FinalPapers/lwa2013_submission_51.pdf (besucht am 07. 04. 2016).

[Sen+14] Shilad Sen u. a. „WikiBrain: Democratizing computation on Wikipedia“. In: ACM Press, 2014, S. 1–10. ISBN: 978-1-4503-3016-9. DOI: 10.1145/2641580.2641615. URL: http://dl.acm.org/citation.cfm?doid=2641580.2641615 (besucht am 11. 01. 2016).

[SMJ13] Pavel Strnad, Ondrej Macek und Pavel Jira. „Mapping XML to Key-Value Database“. In: DBKDA 2013, The Fifth International Conference on Advances in Databases, Knowledge, and Data Applications. 27. Jan. 2013, S. 121–127. ISBN: 978-1-61208-247-9. URL: http://www.thinkmind.org/index.php?view=article&articleid=dbkda_2013_5_30_30098 (besucht am 07. 04. 2016).

[Zah+10] Matei Zaharia u. a. „Spark: cluster computing with working sets.“ In: *HotCloud* 10 (2010), S. 10–10. URL: http://static.usenix.org/legacy/events/hotcloud10/tech/full_papers/Zaharia.pdf (besucht am 16. 07. 2016).