Duale Hochschule Baden-Württemberg

Stuttgart, Campus Horb

NewSQL-Datenbanken

Eine Evaluierung führender Open Source NewSQL Datenbanken bezüglich moderner Anwendungszwecke

---

Eingereicht von:	Robin Schmidt
Matrikelnummer:	5749191
Kurs:	TINF2015
Studiengang:	Informatik
Hochschule:	DHBW Stuttgart Campus Horb
Betreuer:	Prof. Dr.-Ing. Herden, Olaf
Bearbeitungszeitraum:	01.10.2017 - 11.06.2018

Eingereicht am 11.06.2018

Ehrenwörtliche Erklärung

Gemäß § 5 (3) der „Studien- und Prüfungsordnung DHBW Technik“ vom 29. September 2015. Ich versichere hiermit, dass ich meine Studienarbeit mit dem Thema „NewSQL-Datenbanken: Eine Evaluierung führender Open Source NewSQL Datenbanken bezüglich moderner Anwendungszwecke“ selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Böblingen, 6. Juni 2018 Robin Schmidt

Zusammenfassung

Über einen großen Zeitraum hinweg waren relationale Datenbanken die führende Datenbanktechnologie, wenn es um die Datenhaltung, sowie Datenabfragen und generelles Datenmanagement ging. Dies änderte sich mit den wachsenden Anforderungen in den Bereichen „*Skalierbarkeit*“ und „*Performanz*“, aufgrund der stetig wachsenden Datenvolumina in modernen Applikationen und Cloud-Anwendungen. Alternative Systeme, die den Namen „*NewSQL*“ tragen, versuchen diese bestehenden Problemfelder erfolgreich zu bewältigen. NewSQL Datenbanken sind moderne, relationale Datenbankmanagementsysteme, welche die Skalierbarkeit und Performanz von NoSQL Datenbanken mit den stark konsistenten ACID Transaktionen traditioneller Datenbanksysteme, vereinigen.

In dieser Arbeit werden die Vorteile, Klassifikationen und Charakteristiken verschiedener NewSQL Datenbankmanagementsysteme herausgearbeitet und anhand von ausgewählten Kriterien methodisch evaluiert. Diese Kriterien umfassen „*Funktionalität*“, „*Nutzbarkeit*“, „*Qualität*“, „*Sicherheit*“, „*Performanz*“, „*Skalierbarkeit*“ und „*Dokumentation*“. Diese und diverse alternative Kriterien werden in ihrem Inhalt abgegrenzt und begründet gewichtet. Innerhalb dieser Arbeit werden besonders die drei NewSQL Datenbankmanagementsysteme „*NuoDB*“, „*VoltDB*“ und „*CockroachDB*“ bezüglich ihrer Implementierungen bewertet und gegenübergestellt. Hierbei kristallisiert sich VoltDB als führende NewSQL Datenbank heraus und wird anschließend in den Bereichen „*Anfragemöglichkeiten*“, „*Performanz*“, „*Concurrency Control*“, „*Replikate*“, „*Partitionierung und Konsistenz*“ und „*Sicherheit*“ zu entsprechenden Repräsentanten der traditionellen, sowie der NoSQL Datenbanksysteme verglichen. Diese Repräsentanten sind „*MySQL*“ für die traditionellen SQL Datenbanken und „*MongoDB*“, sowie „*Neo4j*“ für die NoSQL Bewegung. Abschließend wurde die führende NewSQL Datenbank in der Kategorie „*Nutzbarkeit*“ in einer virtuellen Umgebung aufgesetzt, sodass anhand dieses Prototyps die Funktionsweise und Methodiken dieser Art von Datenbanken veranschaulicht werden kann.

Abstract

For a long time relational databases were the leading database technology for data storage, retrieval and management. This changed with the growing demand for scalability and performance due to the constantly growing volumes of data in modern applications and cloud computing. Alternative systems called "$NewSQL$" try to successfully overcome these existing problems. NewSQL databases are modern, relational database management systems that combine the scalability and performance of NoSQL databases with the highly consistent ACID transactions of traditional database systems.

In this work the advantages, classifications and characteristics of various NewSQL database management systems will be explained and evaluated based on chosen criteria. These criteria include *"$functionality$"*, *"$usability$"*, *"$quality$"*, *"$security$"*, *"$performance$"*, *"$scalability$"* and *"$documentation$"*. These and various alternative criteria are defined in their content and weighted accordingly. Within this work especially the three NewSQL database management systems *"$NuoDB$"*, *"$VoltDB$"* and *"$CockroachDB$"* are evaluated and compared within their implementations. This process indicates VoltDB as the leading NewSQL database, and subsequently gets compared in the areas of *"$query\ possibilities$"*, *"$performance$"*, *"$concurrency\ control$"*, *"$replica$"*, *"$partitioning\ and\ consistency$"* and *"$security$"* to representatives of traditional and NoSQL database systems. These representatives are *"$MySQL$"* for the traditional SQL databases and *"$MongoDB$"*, as well as *"$Neo4j$"* for the NoSQL movement. Finally, the leading NewSQL database in the usability category got implemented in a virtual environment, so based on this prototype the functionality and methodology of NewSQL databases can be illustrated.

Keywords: *NewSQL databases, NuoDB, VoltDB, CockroachDB, Big Data, Open Source Business Readiness Rating, MySQL, MongoDB, Neo4j*

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

ACID	Atomic, Consistent, Isolated, Durable
AKID	Atomarität, Konsistenz, Isolation, Dauerhaftigkeit
API	Application Programming Interface
BI	Business Intelligence
BRR	Business Readiness Rating
CLI	Call Level Interface
CPU	Central Processing Unit
CRM	Customer-Relationship-Management
CRUD	Create, Read, Update, Delete
DBMS	Datenbankmanagementsysteme
DCL	Data Control Language
DDL	Data Description Language
DML	Data Manipulation Language
DQL	Data Query Language
ETL	Extract, Transform, Load
HPC	High-Performance Computing
IDMS	Integrated Database Management System
IMS	Information Management System
ISAM	Index Sequential Access Method
JDBC	Java Database Connectivity
MVCC	Multi-Version Concurrency Control

OLAP	Online Analytic Processing
OLTP	Online Transaction Processing
OO	Object-oriented
OODBMS	Object Oriented Database Management Systems
ORM	Object-Relational Mapping
OSS	Open Source Software
P2P	Peer-to-Peer
RDBM	Relational Database Management System
REST	Representational State Transfer
SM	Storage Manager
SQL	Structured Query Language
SSD	Solid-State-Drive
SSL	Secure Sockets Layer
TE	Transaction Engine
TLS	Transport Layer Security
TO	Timestamp Ordering
VM	Virtuelle Maschine

Kapitel 1

1 Einleitung

„We're still in the first minutes of the first day of the Internet Revolution"
- Scott Cook[1]

Heutzutage werden in Bereichen wie „*Management*", „*Verwaltung*", „*Bankenwesen*", „*Soziale Netzwerke*" oder „*Sensoren*" enorme Datenvolumen, ausgehend von den verschiedensten Datenquellen, generiert. Dieser Vorgang wird meist als „*Big Data*" beschrieben und kann von den üblichen Datenverarbeitungswerkzeugen, den relationalen Datenbankmanagementsystemen (RDBMS), nicht bewältigt werden. Aus diesem Grund haben sich neue Datenbankarchitekturen und -technologien entwickelt, die darauf ausgelegt sind, diese Anwendungsszenarien zu erfüllen. Das Eingangszitat von Scott Cook beschreibt ebenfalls diese Entwicklung und suggeriert zusätzlich, dass diese Tendenz auch in den kommenden Jahren Bestand haben wird. Aus diesem Grund ist es notwendig diese Aspekte im Entwicklungsprozess moderner Datenbanken zu berücksichtigen. Ein Aspekt, der diese Entwicklung stark vorangetrieben hat, sind Anforderungen im Bereich *„Business Intelligence"*, welche die finanziellen, strategischen und produktionsorientierten Bereiche positiv prägten. Dieses Konzept umfasst das Design, die Entwicklung, die Überwachung und das Management von Daten, um Prozesse innerhalb des Unternehmens zu optimieren und neu auszurichten.

Um diese modernen Anforderungen zu bewältigen, verwenden manche Unternehmen Datencenter und -farmen, die Cluster aus mehreren tausend Maschinen zusammenfassen. RDBMS stellen in diesem Szenario durch das normalisierte Datenmodell und ihr Transaktionsschema keine gute Alternative dar. Hinzu kommt, dass viele Unternehmen eine individuelle Lösung auswählen, welche die Anforderungen ihrer Applikation an die Datenbank optimal erfüllen kann. Der Ansatz „*One size fits all*" im Rahmen der Datenhaltung ist nicht mehr praktikabel.

1.1 Motivation der Aufgabe

Viele Unternehmen stehen vor dem zuvor beschriebenen Problem, dass sie eine individuelle Lösung für ihre Applikation benötigen, um optimale Performanz und Antwortzeiten zu erhalten. Dies kann wirtschaftlich betrachtet langfristig die Kundenzufriedenheit

[1] Geboren im Jahr 1952, Mitgründer von Intuit, Direktor von eBay und Procter & Gamble

erhöhen, was zu einer dauerhaften Erhöhung der monetären Einkünfte führen kann. Allerdings kann es problematisch sein, einen ausreichenden Überblick über alle Datenbanktypen mit ihren Vor- und Nachteilen und mögliche Implementierungen der verschiedenen Datenbankmanagementsysteme zu erhalten.

Dieses Problem soll innerhalb dieser Arbeit gelöst werden, indem verschiedene Datenbanktypen bezüglich unterschiedlichster Kategorien verglichen werden und so eine optimale Lösung für jeden Anwendungszweck gefunden werden kann. Im Rahmen der moderneren Datenbankansätze, explizit innerhalb der NoSQL und NewSQL Bewegung, werden ebenfalls verschiedene Konzepte vorgestellt, welche in bestimmten Anwendungsszenarien große Vorteile bieten. Hier werden außerdem konkrete Datenbankmanagementsysteme miteinander verglichen, um neben dem optimalen Datenbanktyp auch die optimale Realisierung zu wählen.

1.2 Gegenstand und Ziele der Arbeit

Ziel der Studienarbeit ist es, einen umfassenden Überblick über Konzepte und Revolutionen zu geben, die unter dem Namen „$NewSQL$“ zusammengefasst werden. Hierzu sollen zunächst mehrere NewSQL Datenbanken ausgewählt werden, die im Rahmen dieser Studienarbeit verglichen werden sollen. Für diese sollen die Möglichkeiten und Ansätze der Implementierung herausgearbeitet und mit passenden Abbildungen und Übersichten veranschaulicht werden. Darauf folgt eine ausgiebige Evaluation der verschiedenen Repräsentanten der NewSQL Datenbanken. Hierbei sollen diese mittels eines methodischen Evaluierungsprozesses anhand verschiedener Kategorien bewertet werden, sodass die Vor- und Nachteile der einzelnen Datenbankmanagementsysteme deutlich werden. Eines der betrachteten NewSQL Datenbankmanagementsysteme soll ebenfalls praktisch realisiert werden und so als Prototyp einer solchen Datenbank dienen. Abschließend sollen Unterschiede zur NoSQL Bewegung, sowie den traditionellen, rein relationalen Systemen, herausgearbeitet und erläutert werden.

Um diese Ziele möglichst fundiert umzusetzen, müssen anfänglich Grundlagen der Datenbankentwicklung in den letzten Jahrzehnten, sowie Implementierungsstrategien verschiedener Konzepte in NewSQL Datenbanken erarbeitet werden. Gleichzeit sollen die bestehenden Kenntnisse aus verschiedenen Datenbankvorlesungen aufgefrischt und erweitert werden. Diese Kenntnisse sollen zur Auswahl und Festlegung von passenden Kriterien und deren Gewichtung in der Evaluation der NewSQL Datenbanken, sowie dem Vergleich der verschiedenen Datenbanktypen, positiv beitragen.

Kapitel 2

2 Datenbank Generationen

Datenbanken haben in ihrer Entwicklung hin zu den modernen Datenbanksystemen, wie wir sie heute kennen, einige Veränderungen und Probleme bewältigen müssen. Hierbei lässt sich diese Entwicklung grob in drei Phasen einteilen. Die „*Pre-Relationale Phase*“, die „*Relationale Phase*“ und die „*Next Generation Phase*“.

Abbildung 2.1 zeigt diese drei übergeordneten Datenbankgenerationen, welche in den vergangenen Jahren stattgefunden haben. In den Anfängen der Verbreitung von elektrischen Computern und den darauf folgenden 20 Jahren etablierten sich immer aufwendigere Datenbanksysteme. Nachdem im Jahr 1970 das relationale Modell entwickelt wurde, stellten sich alle bedeutsamen Datenbanksysteme auf dieses um und besaßen damit eine ähnliche Architekturbasis [Har15]. Die Bestandteile dieser Basis umfassten das zuvor genannte relationale Modell, ACID Transaktionen, sowie die SQL Anfragesprache.

Ab dem Jahr 2005 wurden immer mehr Datenbanksysteme entwickelt, die sich nicht an das traditionelle, relationale Konzept der bereits bestehenden Datenbanksysteme hielten. Dies ist unter anderem auf die steigenden Anforderungen im Umfeld der *„Big Data“* zurückzuführen. Im diesem Unterkapitel soll verdeutlicht werden, wie und in welchem Umfang die vorangegangenen Datenbankgenerationen zu dieser neuen Generation von Datenbanksystemen geführt haben.

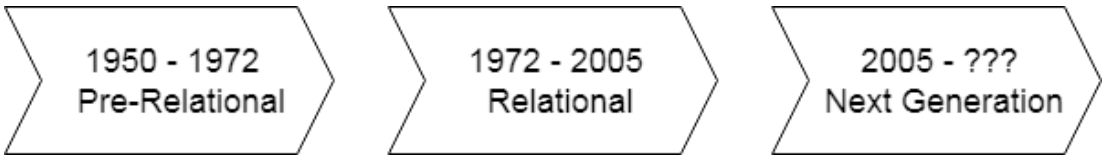


Abb. 2.1: Zeitlicher Verlauf der Datenbankgenerationen angelehnt an: [Har15]

2.1 Frühe Datenbanksysteme

Obwohl der Begriff „*Datenbank*“ erst in den späten 1960er Jahren in das allgemein anerkannte Vokabular aufgenommen wurde, war das Sammeln und Aufbereiten von Daten bereits in vorherigen Jahrzehnten ein wichtiger Bestandteil der Entwicklung der menschlichen Zivilisation und deren Technologie [Har15]. Laut verschiedener Quellen handelt es bei einer Datenbank lediglich um eine „*organisierte Kollektion von*

$Daten$“. Bereits vor der Einführung von Datenbanken, wie wir sie heute kennen, gab beziehungsweise gibt es einige Strukturen, die formal dieser Definition einer Datenbank entsprechen:

Bücher: Vor allem Bücher, die eine klare Struktur aufweisen, wie Lexika oder Wörterbücher, repräsentieren Datensätze in physischer Form.

Büchereien: Oder andere Informationsarchive stellen somit das nicht technische Äquivalent zu modernen Datenbanksystemen dar.

Lochkarten: Oder auch ähnliche physische Medien mit Informationen, die maschinell verarbeitet werden können.

Frühe Datenbanksysteme benutzten ursprünglich einen Lochstreifen und im späteren Verlauf ein Magnetband, um Daten sequentiell abzuspeichern. Auf diesen war es möglich vorwärts oder auch rückwärts durch die Datensätze zu iterieren, allerdings benötigte es noch bis Mitte der 1950er Jahre bis ein schneller Zugriff auf einzelne Datensätze möglich war. Dies wurde durch die Entwicklung von Index Mechanismen, wie der *Index Sequential Access Method* (ISAM) unterstützt und bildete die Grundlage für die ersten *Online Transaction Processing* (OLTP) Computersysteme [Har15].

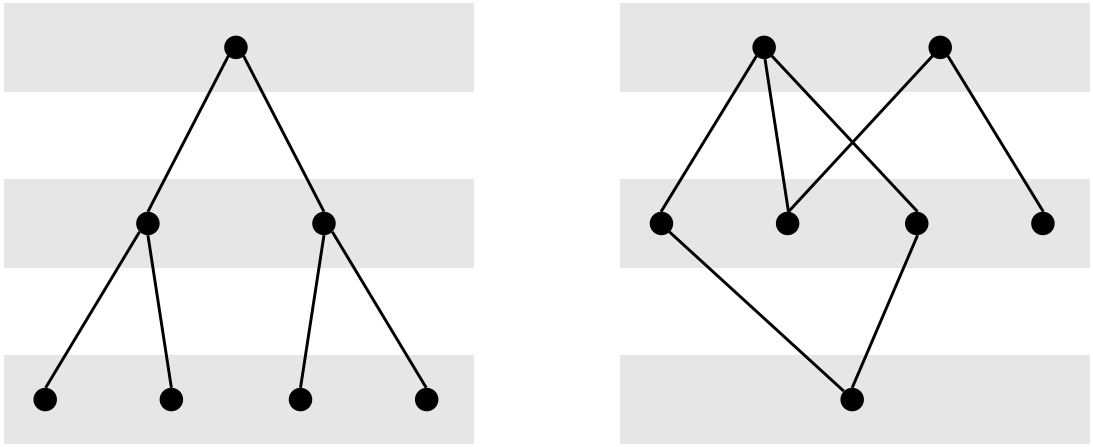
Obwohl die ersten elektronischen Datenbanksysteme durch ISAM oder ähnliche Indexmechanismen betrieben wurden, gab es noch keine Datenbankmanagementsysteme (DBMS). Die Verwaltung der Indizes lag jeweils in der Verantwortung der Applikationen.

2.2 Die erste Datenbankgeneration

Die Problematik, dass jede Applikation die Indizes selbständig verwalten musste, stellte sich sehr schnell als Produktivitätsengpass heraus. Jede Applikation musste ihren eigenen Programmcode zur Datenverwaltung schreiben und somit gab es keine einheitliche Verwaltung. Hinzu kommt, dass sobald die selbstgeschriebene Datenverwaltung Programmierfehler aufwies, unweigerlich beschädigte Daten entstanden. Schlussendlich sorgten Konzepte, wie „$Mehrbenutzerbetrieb$“, „$Caching$“ oder „$Prefetching$“ für ein enormes Maß an sich wiederholendem Programmieraufwand in verschiedenen Applikationen und eine Trennung der Datenbanklogik von der eigentlichen Applikationslogik schien wünschenswert [Har15]. Diese Schicht des Datenbankmanagementsystems sollte den bestehenden Programmiermehraufwand und die Konsistenz der Datenbankmethoden sicher stellen.

In den Anfängen der 1970er Jahre stellten sich zwei Modelle des Datenbankmanagementsysteme als Vorreiter in diesem Bereich heraus. Das Netzwerk-Datenbankmodell, welches in Datenbanken wie dem *Integrated Database Management System* (IDMS) implementiert wurde, sowie das hierarchische Datenbankmodell, welches vor allem im *Information Management System* (IMS) von IBM wieder zu finden war [Har15].

Abbildung 2.2 zeigt hierbei die unterschiedliche Struktur dieser beiden Ansätze. Auffällig ist, dass das hierarchische Datenbankmodell einen vergleichsweise einfacheren Aufbau darstellt.



Hierarchisches Datenbankmodell

Netzwerk Datenbankmodell

Abb. 2.2: Hierarchisches und Netzwerk Datenbankmodell

Oftmals werden diese Systeme als „*navigierende*“ Systeme beschrieben, da man durch Zeiger von einem Objekt zum nächsten navigieren muss. Diese Struktur stellt auch einen der Nachteile dieser Ansätze dar. Einerseits galten sie als sehr unflexibel bezüglich der Datenstruktur und den Anfragemöglichkeiten. Andererseits waren sie nur auf die grundlegenden Datenbankoperationen ausgelegt. Diese umfassten Operationen, die heutzutage als *CRUD* beschrieben werden, also *Create*, *Read*, *Update* und *Delete*. Da Computersysteme immer mehr mit Unternehmensprozessen vernetzt wurden, stieg auch die Anforderungen an analytische Auswertungen der Daten.

2.3 Die zweite Datenbankgeneration

Diese Nachteile führten zur zweiten großen Datenbankgeneration, in der besonders relationale Datenbanken die bestehenden Nachteile beheben sollten. Die relationale Theorie sorgte schnell für eine verbesserte Bedienbarkeit, sowie eine verbesserte Datenstruktur. Diese Optimierungen ermöglichten eine effektivere Benutzung der Datenbank. Grundbestandteile dieses Ansatzes waren unter anderem:

Tupel: Tupel sind eine ungeordnete Menge von Attributwerten. In einem richtigen Datenbanksystem entspricht dies einer Zeile und das Attribut einer Zeile.

Relationen: Relationen sind eine Menge von Tupeln, die in einem richtigen Datenbanksystem eine Tabelle repräsentieren.

Bedingungen: Bedingungen sind für die Konsistenz innerhalb der Datenbank zuständig. Schlüsselbedingungen werden benutzt, um Tupel oder die Beziehung zwischen mehreren Tupeln zu identifizieren.

Operationen: Anfragen auf Relationen, wie Verbünde oder Projektionen. Diese liefern in der Realität Daten in Tabellenform zurück.

Das relationale Modell allein sorgte nicht für einen explizit definierten Aufbau von konkurrierenden Datenänderungsanfragen, später auch Transaktionen genannt. Dieses Problem der Konsistenz und Integrität der Daten bestand trotz dem relationalen Modell immer noch. In den späten 1970er Jahren definierte Jim Gray ein Transaktionskonzept, welches genau an dieser Stelle ansetzte und später zum Standard in diesem Bereich werden sollten. Gray beschrieb eine Transaktion wie folgt:

> „*A transaction is a transformation of state which has the properties of atomicity (all or nothing), durability (effects survive failures) and consistency (a correct transformation).*“ - Jim Gray

Dies bildete die Grundlage für AKID (*engl.* ACID) Transaktionen, welche sich aus den Bestandteilen „*Atomarität*“, „*Konsistenz*“, „*Isolation*“ und „*Dauerhaftigkeit*“ zusammensetzen.

Atomarität: Die Transaktion ist nicht teilbar, entweder werden alle Anfragen innerhalb der Transaktion angewendet oder sie werden in ihrer Gesamtheit verworfen.

Konsistenz: Bevor und nach einer Transaktion befindet sich die Datenbank in einem konsistenten Zustand.

Isolation: Eine Transaktion sollte nicht die Ergebnisse einer anderen Transaktion beeinflussen, während diese noch nicht abgeschlossen ist.

Dauerhaftigkeit: Sobald eine Transaktion abgeschlossen ist, wird davon ausgegangen, dass die Daten persistent gehalten werden (auch bei Hardware- oder Betriebssystemfehlern).

Mit den ersten relationalen Datenbanken, wurde auch die Syntax für eine passende Anfragesprache ausgearbeitet. Hierbei handelt es sich um die SQL Syntax. Bis Mitte der 1980er Jahre breiteten sich relationale Datenbanken weiter aus und die Vorteile dieses Ansatzes sorgten für einen steigenden Adaptionsgrad. Während der erfolgreichen Jahre wurden immer mehr Datenbanksysteme veröffentlicht. Besonders *DB2*, *MySQL* oder *Microsoft SQL Server* wurden populär. Obwohl diese Datenbanksysteme versuchten sich in Verfügbarkeit, Performance oder auch Funktionalität von den anderen Marktteilnehmern zu unterscheiden, stützten sich alle Datenbanksysteme auf die drei Grundprinzipien: Codd's relationales Modell, die SQL Sprache und ACID Transaktionen [Har15]. Wie wir im späteren Verlauf dieser Entwicklung sehen werden, sind ACID Transaktionen einer der Hauptgründe für die Entwicklung neuer Datenbankarchitekturen. Dies ist durch die Limitierung der Skalierbarkeit auf einen Datenknoten begründet [Har15].

Zwischen den 1980er bis 1990er Jahren kam besonders objektorientierte ($engl.$ Object-oriented OO) Programmierung auf, welche die prozeduralen Programmiersprachen ablösen sollten. So kann ein Objekt beispielsweise die Struktur eines Mitarbeiters, mit möglichen Aktionen repräsentieren [Har15]. Anfang der 1990er Jahre konvertierten die meisten Programmiersprachen zu einem objektorientierten Model und viele neue objektorientierte Programmiersprachen kamen auf [Har15]. Dies stellte sich als die erste Herausforderung für die relationalen Datenbanken heraus. Entwickler sahen die objektorientierte Darstellung der Daten innerhalb ihrer Applikationen und die relationale Darstellung innerhalb der Datenbank als problematisch an, da zur Konvertierung mehrere SQL Anfragen notwendig waren [LeeoJ, Har15].

Durch diese Entwicklung wurden sehr schnell objektorientierte Datenbankmanagementsysteme ($engl.$ Object Oriented Database Management Systems OODBMS), native XML-Datenbanken und Multidimensionale Datenbanken als die besseren Alternativen für die auftretenden Anforderungen angesehen. Diese OODBMS erlaubten es Applikationen ihre Programmobjekte leicht zu laden und zu speichern, indem Programmobjekte ohne Normalisierung gespeichert wurden [LeeoJ]. Viele sahen die OODBMS als den direkten Nachfolger zu den relationalen Datenbankmanagementsystemen an und so implementierten auch große relationale Datenbanken verschiedene OODBMS Methoden. Dies gelang auch erfolgreich und so konnten native OODBMS am Ende dieses Epoche keinen Marktanteil gewinnen. Trotzdem wurden diese Methoden in den relationalen Datenbanken von Programmierern, die mit objektorientierten Programmiersprachen arbeiteten, kaum bis gar nicht genutzt. Viele haben sich mit der Nutzung von relationalen Datenbankmanagementsystemen abgefunden und nutzten ein Objekt-Relationales Mapping ($engl.$ Object-Relational Mapping ORM), welches Teile der Überführung automatisierte [LeeoJ, Har15].

2.4 Die dritte Datenbankgeneration

Bis zur Mitte der 2000er Jahre waren relationale Datenbanken fest als Architekturmodell etabliert. Obwohl verschiedene Teilaspekte weiterentwickelt und an manchen Stellen auch verbessert wurden, war keine größere Generation des Architekturmodells in naher Zukunft abzusehen. Trotzdem stiegen, durch die Weiterentwicklung der Applikationen hin zu breit skalierenden Webanwendungen, die Anforderungen an die eingesetzten relationalen Datenbanken soweit, dass ein Engpass entstand, der nicht durch inkrementelles Verbessern des bisher bestehenden Architekturmodells bewältigt werden konnte. Diese Entwicklung musste unweigerlich zur bisher letzten Generation im Bereich Datenbankarchitekturen führen, der nächsten Generation von Datenbanken.

Ab dem Jahr 2004 stellten mehr und mehr Webanwendungen interaktivere Benutzeroberflächen zur Verfügung. Dies wurde durch $Asynchronous\ JavaScript\ and\ XML$

(AJAX) ermöglicht, bei dem JavaScript innerhalb des Browsers mit dem Backend über XML kommuniziert. XML wurde hierbei schnell durch die *Javascript Object Notation* (JSON) abgelöst. Dies entwickelte sich sehr schnell zum Standard in diesem Bereich und manche Webseiten fingen an, ihre JSON Dokumente direkt in relationale Datenbanktabellen zu speichern. Ab hier war es nur noch eine Frage der Zeit, bis dieser relationale Mittelsmann durch eine passendere Alternative abgelöst wurde. Hierbei handelt es sich um dokumentbasierte Datenbanken, in denen JSON Dokumente direkt abgespeichert werden konnten [HKGH16]. Zwei Vorreiter in diesem NoSQL Bereich sind *MongoDB* und *CouchBase*.

Im Jahr 2007 veröffentlichte Michale Stonebraker eine Arbeit mit dem Titel „*The End of an Architectual Era (It's Time for a Complete Rewrite)*“ [SMA+07]. In dieser thematisiert Stonebreaker die Probleme der relationalen Architektur und die Tatsache, dass die Varianz der mordernen Datenbankbelastungen mehr als nur eine Art der Datenbankarchitektur suggerieren. Hierzu wurden von Stonebreakers Team mehrere Varianten vorgestellt, von denen zwei besonders hervorstachen, *H-Store* und *C-Store*. H-Store beschreibt hierbei eine Datenbank, die vollständig im Hauptspeicher agiert, während C-Store eine säulenartige Datenbankarchitektur beschreibt. Beide Designarten waren wegweisend zu einer Entwicklung, die heute als „*NewSQL Datenbanken*“ beschrieben werden. Hierbei handelt es sich um Datenbanken, die eine Abwandlung oder auch Modifizierung der Hauptprinzipien des relationalen Models implementierten. Die Funktionsweise dieser neuen Art von Datenbanken soll in Kapitel 4 näher erläutert werden.

Kapitel 3

3 Business Intelligence und Reporting

Zwischen 1980 und 1990 mussten relationale Datenbanken mehr und mehr Entscheidungsunterstützende Applikationen unterstützen, welche einen besonderen Fokus auf interaktive Reaktionszeiten legten. Hieraus entwickelten sich Systeme, die als Data Warehouses bekannt wurden, und der Begriff Online Analytic Processing (OLAP) wurde eingeführt, um sie von OLTP zu differenzieren. Edgar Codd[1] hat hierbei eine tragende Rolle in der Begriffsdefinition gespielt.

In diesem Kapitel soll das nötige Basiswissen vermittelt werden, um die Entwicklung hin zu modernen Datenbankmodellen nachvollziehen zu können. Es wird besonderen Wert auf die Begriffsdefinitionen von Business Intelligence und den Ablauf der Reporterstellung gelegt, um vor allem die generelle Funktionsweise von Business Intelligence Funktionalitäten zu erklären.

3.1 Begriffsdefinitionen Business Intelligence

Zum Terminus „*Business Intelligence*“ gibt es keine einheitliche wissenschaftliche Definition aufgrund des praktischen Hintergrundes der Begrifflichkeit. In frühen Stadien der Begriffsdefinitionen wurden Softwarewerkzeuge zur Managementunterstützung, die von externen Dienstleistern bereitgestellt wurden, unter dem Überbegriff „*Business Intelligence*“ zusammengefasst. Dieser Sammelbegriff wandelte sich in seiner Definition, als er vermehrt in wissenschaftlichen Diskussionen verwendet wurde [Dou08]. In verschiedenen Fachliteraturen wird oftmals versucht eine aktuelle Definition für diesen Begriff zu erfassen:

> *„Business Intelligence ist [...] ein IT-basierter Prozess der Informationsgenerierung, der die Entscheidungsfindung innerhalb der unternehmensrelevanten Planung, Steuerung und Kontrolle unterstützt.“* - Chamoni u. Linden (2007), [LC07]

> „*Unter Business Intelligence wird ein integriertes Gesamtkonzept verstanden, das für die unterschiedlichen Ausprägungen der Anforderungen*

[1] 1923 - 2003, war ein britischer Mathematiker und Datenbanktheoretiker

an geeignete Systeme zur Entscheidungsunterstützung tragfähige und miteinander verknüpfte Lösungen anbietet.“ - Gluchowski u. Kemper (2006), [Lin16]

„*Business Intelligence (BI) is an umbrella term that combines architectures, tools, databases, analytical tools, applications and methodologies.*“ - Turban, Sharda u. Delen (2011), [Lin16]

Die aufgezeigten Definitionen unterscheiden sich teilweise in ihrem Umfang, trotzdem teilen sie die Aspekte des analytischen Hintergrundes und der Entscheidungsunterstützung. Betrachtet man verschiedene Definitionen genauer, lassen sich drei verschiedene Auffassungsmöglichkeiten erkennen. Hierbei handelt es sich um „*enge*“, „*analyseorientierte*“ und „*weite*“ Auffassungen des Business Intelligence Begriffes [Dou08].

Innerhalb der **engen Auffassung** zählen alle Produkte und Dienste zu Business Intelligence, die die Aufbereitung von organisierten Daten unterstützen. Lösungen die hier eingesetzt werden, umfassen zum Beispiel Briefing-Books oder Excel-Add-ins, um Daten visuell hervorzuheben. Jegliche Front-End Produkte, die beispielsweise Reporting ermöglichen, gehören nicht in dieses Segment.

Zur **analyseorientierten Auffassung** werden alle Methoden und Werkzeuge zugeordnet, die aus vorhandenen Daten eine Analyse ermöglichen. Hierunter fallen beispielsweise Kennzahlen, Business Activity Monitoring oder auch Ad-Hoc-Reporting.

Die **weite Auffassung** beinhaltet alle Komponenten, die Datenmaterial zur Informations- und Wissensgenerierung aufbereiten und gleichzeitig Auswertungs- und Präsentationsfunktionen bereitstellen [GDG08]. Daraus folgt, dass hierunter unter anderem das Data Warehouse, sowie das Standard-Reporting fallen.

Im folgenden Unterkapitel soll auf Basis der weiten Auffassung die Architektur von Business Intelligence Systemen näher erläutert werden, um ein Gesamtverständnis über die Prozesszusammenhänge zu erreichen.

3.2 Architekturbeschreibung

Eine idealistische Architektur von BI-Systemen besteht aus fünf Kernkomponenten, die in direktem Zusammenhang stehen: Operative Prozesse, Quellsysteme, Extract Transform Load (ETL), Data Warehouse und Dispositive Prozesse. Diese bauen sequentiell aufeinander auf und bilden so eine Schichtenarchitektur, welche in Abbildung 3.1 illustriert wird [Dou08].

Aus den verschiedenen **Quellsystemen**, wie beispielsweise CRM oder ERP, werden die Daten mittels dem **ETL-Prozess** im **Data Warehouse** zusammengefasst. Daraufhin kann auf Basis der gesammelten Daten im Data Warehouse, mithilfe der **dispositiven**

Prozesse, Wissen generiert werden, welches eingesetzt wird, um die **operativen Prozesse** zu optimieren [SSC15, LCA+17]. Aus diesem Grund müssen auch die operativen Prozesse im Kontext der Business Intelligence vorhanden sein, damit aus dem generierten Wissen ein Rückschluss auf diese ermöglicht wird und das Wissen effektiv genutzt werden kann [Lei15]. Die dispositiven Prozesse umfassen Reporting, Planung, Analyse und Steuerung und bilden somit die Kernwerkzeuge zur Wissensgenerierung.

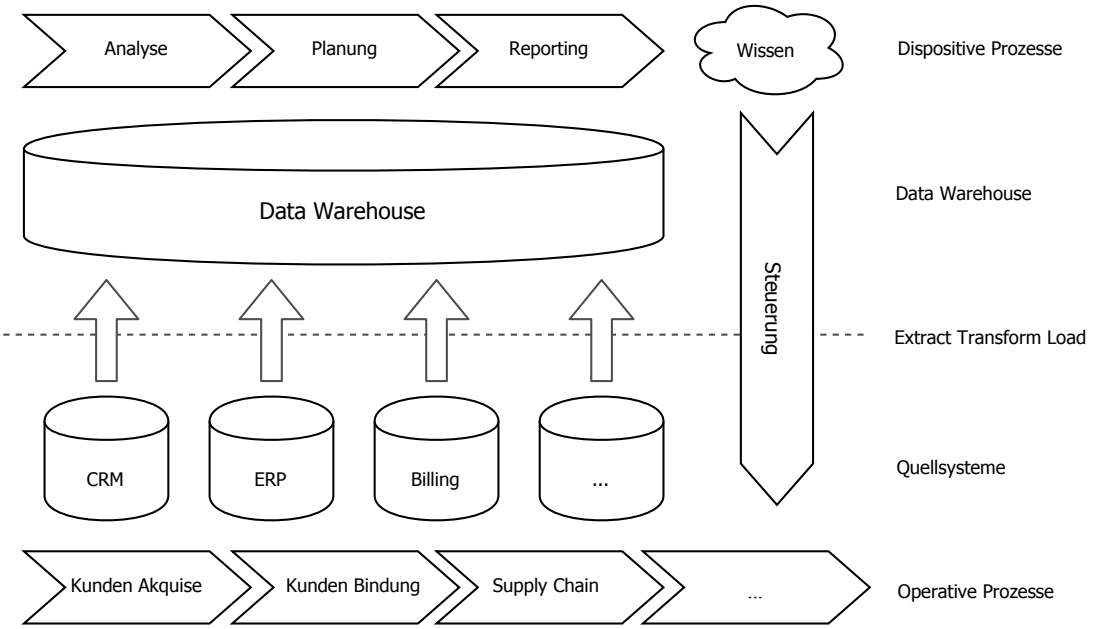


Abb. 3.1: Schichtenarchitektur der Business Intelligence angelehnt an: [SM10, Lei15]

Diese dispositiven Prozesse, speziell der Prozess des Reportings, soll im nachfolgenden Unterkapitel näher betrachtet werden.

3.3 Einordnung des Reportings

Unter Reporting wird allgemein die Erstellung und Bereitstellung von Berichten (*engl.* Reports) verstanden. Diese „*Berichte*“ sind hierbei Dokumente, welche Informationen in aufbereiteter und kombinierter Form enthalten [KMU06, Mio16]. Zum Begriff des Reportings zählen außerdem alle Daten, Personen, Organisationseinheiten, Vorschriften oder Prozesse, die an der Erstellung oder Bereitstellung der Berichte beteiligt sind [Dou08]. Solche Prozesse sind beispielsweise die Berichtsgestaltung und -erstellung, die Berichtsverteilung, die Berichtsverwaltung oder die Berichtsaufnahme und -diskussion. Alle diese Prozesse tragen auf mindestens eine Weise zur Erstellung und Bereitstellung von Berichten bei.

Generell kann in zwei Arten der Bereitstellung von Berichten unterschieden werden. Hierbei handelt es sich einerseits um bedarfsorientierte Berichte (*engl.* Ad-Hoc-Reporting)

und andererseits um zeitlich orientierte Berichte. Teilweise werden die zeitlich orientierten Berichte nochmals in Standard-Reporting und Dashboard-Reporting unterschieden [Lei15]. Diese zwei Unterkategorien der zeitlich orientierten Berichte unterscheiden sich nochmals in Art der Darstellung und der jeweils vorgesehenen Zielgruppe.

3.3.1 Standard-Reporting

Beim Standard-Reporting werden Berichte periodisch zu vordefinierten Zeitpunkten automatisch generiert und gleichzeitig archiviert, beziehungsweise den Endanwendern zur Verfügung gestellt. Notwendig hierzu ist es, jeweils einen Zeitpunkt oder einen Trigger zu definieren, wann der Bericht erzeugt werden soll. Berichte können beispielsweise am zweiten Montag jedes Monats um 8:00 Uhr oder im Falle eines Triggers, sobald die Daten des Vortages verfügbar sind, generiert werden [Lei15].

Hier werden standardisierte Layouts und Inhalte verwendet, sodass einheitlich alle vordefinierten Kennzahlen aufbereitet werden können. Außerdem müssen Output-Format und Transportmedium ebenfalls eine konstante Größe über alle Berichte sein.

3.3.2 Ad-Hoc-Reporting

Beim Ad-Hoc-Reporting wird ein Bericht für den Benutzer individualisiert und wird von ihm unmittelbar benötigt [Lei15]. Dies bedeutet, dass beim Benutzer ein Bedarf vorhanden ist, welcher nicht von den zeitlich orientierten Reporting-Möglichkeiten abgedeckt wird. Für diese sollten, wie bei den zeitlich orientierten Berichten auch, bestimmte Standards definiert werden, die bei der Berichtserstellung berücksichtigt werden. Solche Standards könnten beispielsweise das Layout, den Umfang oder die Dimensionen näher beschreiben, sodass auch hier einheitlich vorgegangen wird. Während die Umsetzung des Standardreportings in der Regel über Mitarbeiter mit guten technischen Kenntnissen erfolgt, muss das Ad-hoc Reporting auch einem trainierten Fachanwender möglich sein [GC15].

An dieser Stelle sei auf die T2_2000 Projektarbeit mit dem Titel *„Berechnung und Auswertung von Leistungskennzahlen der Produktion innerhalb eines Manufacturing Execution Systems“* [Sch17] verwiesen, da dort besonders auf diese Art des Reportings eingegangen wird.

3.3.3 Dashboard-Reporting

Das Dashboard-Reporting ist im Grunde ein aggregiertes und komprimiertes Standard-Reporting, welches meistens zum Reporting auf Management-Ebene eingesetzt wird [Lei15]. Ein solches Dashboard besitzt ein intuitives Design, welches alle relevanten

Kennzahlen auf einen Blick einsehbar macht. Dies sorgt dafür, dass in kurzer Zeit ein grober Gesamtüberblick über die derzeitige Ausgangssituation erreicht werden kann. Dies eignet sich optimal für Managementaufgaben, allerdings sollten für ein tieferes Verständnis ergänzend die Standard-Reports herangezogen werden. Meistens werden Cockpit ähnliche Designs eingesetzt, um intuitiv bedienbare Layouts zu erreichen.

3.4 Variablen des Reportings

Beim Reporting müssen jeweils fünf verschiedene Gesichtspunkte berücksichtigt werden, um qualitativ hochwertige Berichte zu erstellen: „*Berichtsinhalt*“, „*Berichtszweck*“, „*Berichtsform*“, „*Berichtsinstanz*“ und „*Berichtszeit*“ [CG06]. Diese Parameter hängen jeweils von den Anforderungen des Kunden ab und müssen dementsprechend angepasst werden.

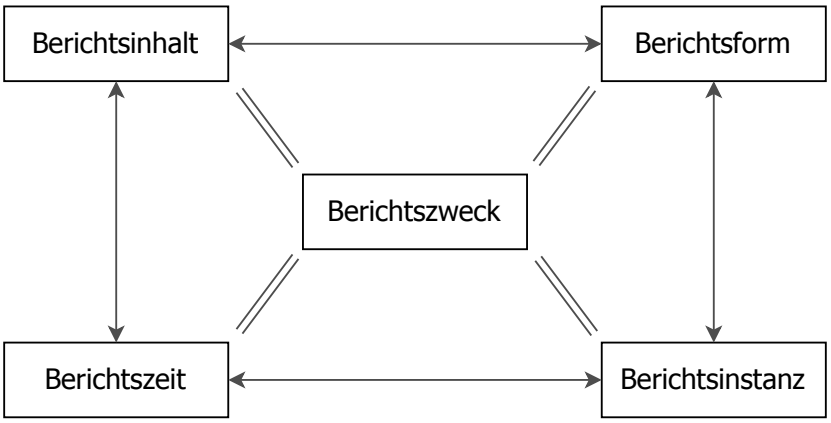


Abb. 3.2: Merkmale des Reportings angelehnt an: [CG06, GDG08]

Berichtsinhalt umfasst den Umfang, also den Detaillierungsgrad, Breite und Gegenstand jedes Berichtes. Hierbei handelt es sich um die Varianz der dargestellten Informationen.

Berichtszweck beinhaltet den Hintergrund des jeweiligen Berichtes. Er beschreibt, ob dieser zur Dokumentation, zur Entscheidungsvorbereitung, zur Kontrolle oder zum Auslösen eines Bearbeitungsvorganges eingesetzt werden soll.

Berichtsform repräsentiert die visuelle Komponente jedes Berichtes. Dazu gehören Darstellungsmedium, Struktur, Präsentationsmedium und Übertragungsweg.

Berichtsinstanz behandelt die Instanzen, die bei der Berichterstellung mitwirken. Beispielsweise könnten dies der Empfänger, der Ersteller oder der Verantwortliche sein.

Berichtszeit beschreibt die zeitliche Komponente jedes Berichtes. Dies umfasst die Länge und die Regelmäßigkeit der Berichtsintervalle [GDG08].

Abbildung 3.2 zeigt, wie diese Merkmale in Zusammenhang stehen. Der Berichtszweck bildet hierbei eine zentrale Einheit, an der sich die Ausprägungen der anderen Merkmale orientieren. Zwischen den aufgeführten Kriterien befinden sich mehrfach konkurrierende und komplementäre Beziehungen. Eine bestehende Konkurrenz befindet sich zwischen der Aktualität und der Genauigkeit eines Berichtes [CG06]. Hierbei muss für jeden Einzelfall mit Hintergrund des Berichtzweckes abgewogen werden, welchem Kriterium die höhere Priorität zugewiesen wird [GDG08].

3.5 Big Data

In diesem Bereich ist ebenfalls der Begriff „$Big\ Data$“ zu erwähnen. Hierbei handelt es sich um die moderne Entwicklung der Daten, die vor allem durch Cloud Applikationen, Mobile Applikationen und Soziale Medien erreicht wurde. Dieser Terminus lässt sich nach [IBM18] in die „$vier\ V's$“ aufteilen. Hierbei handelt es sich um das Volumen ($engl.$ Volume), die Geschwindigkeit ($engl.$ Velocity), die Vielfalt ($engl.$ Variety) und die Richtigkeit ($engl.$ Veracity):

Volume beschreibt die immer steigenden Ausmaße der Datenmengen in zukünftigen Unternehmen und Applikationen.

Velocity beschreibt die notwendige Geschwindigkeit zur Verarbeitung dieser großen Datenmengen.

Variety beschreibt die unterschiedlichen Formate, in denen die Daten vorliegen (Tweets, Texte, Videos, JSON, Graphen, etc.).

Veracity beschreibt die Notwendigkeit von validen Daten für moderne Applikationen und Unternehmensführungen.

Diese Entwicklungen forderten neue, moderne Datenbankmodelle, welche mit den neu entstandenen Anforderungen umgehen konnten und eine optimale Lösung für den zukünftigen Anstieg aller Unteraspekte von Big Data darstellen.

Kapitel 4

4 Moderne Datenbankmodelle

Wie bereits in Kapitel 2.4 angedeutet, bildeten sich vor allem zwei Varianten moderner Datenbankmodelle heraus. Die Funktionsweise dieser Alternativen und deren Unterschiede sollen in diesem Kapitel anhand passender Beispiele näher erläutert werden.

4.1 Spaltenbasierte Datenbanken

Durch eine Datendarstellung in Zeilen von verschiedenen Werkzeugen, ist es für die meisten Menschen ein natürlicher Reflex einen Datensatz als eine Zeile zu visualisieren. So wird durch Tabellen, Bücher oder sogar die Satzstruktur europäischer Sprachen, welche von links nach rechts und von oben nach unten verlaufen, eine zeilenbasierte Datenstruktur suggeriert [Har15]. Auch wenn dieses Layout sich als ein einfaches Format zur Datenvisualisierung etabliert hat, ist es oftmals nicht die optimale Strukturierung von Daten innerhalb einer Datenbank.

Die ersten Datenbanksysteme implementierten eine solche zeilenorientierte Speicherung der Daten und haben damit eine gute schreiboptimierte Performanz für die damals dominanten OLTP Anforderungen geboten [SAB+05, Har15]. Als sich die Anforderungen allerdings weg von OLTP und hin zu mehr und mehr analytischen Anforderungen und Data Warehousing entwickelten, wurde dieses Modell verworfen und eine spaltenorientierte Datenspeicherung wurde geschaffen, um eine leseoptimierte Architektur zu erreichen [SAB+05]. Das geschah hauptsächlich mit dem Hintergrund, dass in Data Warehouses selten ein Datensatz über alle Spalten abgefragt wird, sondern öfters eine Spalte über alle Datensätze. Hier werden lediglich periodisch mittels eines Bulk Loads große Mengen an Daten geschrieben [SAB+05].

Das Prinzip der spaltenbasierten Datenhaltungsmethode basiert auf einer gruppierten Speicherung der Spaltenwerte. Dieses Prinzip und dessen Unterschiede zu typischen zeilenbasierten Alternativen wird in Abbildung 4.1 näher erläutert. Hierbei zeigt sich, dass ein Datensatz innerhalb dieser Implementierung von einer Spalte, anstatt einer Zeile, repräsentiert wird. Tatsächliche Implementierungen dieser Architektur unterscheiden sich meistens noch in weiteren Aspekten, die hier allerdings nicht weiter aufgezeigt werden sollen.

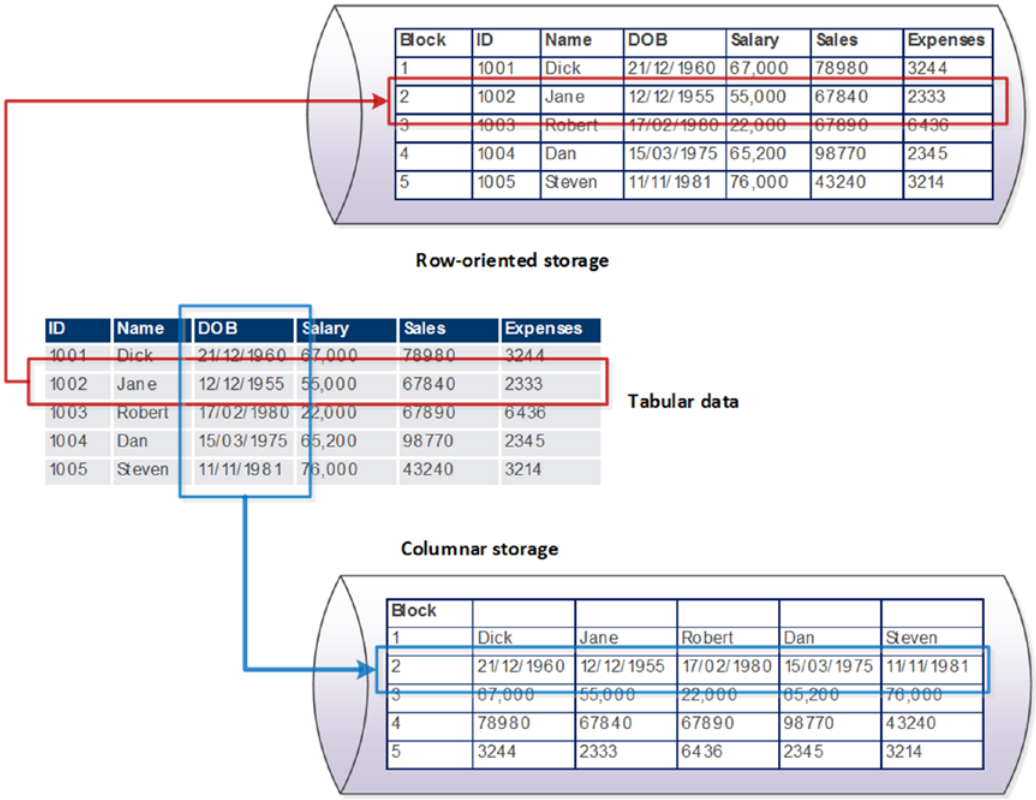


Abb. 4.1: Vergleich zwischen zeilen- und spaltenbasierten Datenhaltung angelehnt an: [Har15]

Vorteile dieser Methodik sind die generelle Performanzverbesserung für OLAP Transaktionen. Aggregationsanfragen werden in einer spaltenbasierten Architektur deutlich optimiert, da sich die Daten im selben Speicherblock befinden. Ein Beispiel dafür ist eine typische Aggregierung in Form einer Summe der Gehälter jeder einzelnen Person. Angenommen die Tabelle im gezeigten Beispiel aus Abbildung 4.1 trägt den Namen „*Mitarbeiter*“, dann wäre die passende SQL Anfrage für eine solche Aggregation in Programmlisting 4.1 dargestellt.

```
1 SELECT SUM(Salary) FROM Mitarbeiter
```

Listing 4.1: Summenaggregation der Gehälter

Bei der zeilenorientierten Datenstruktur müsste eine solche Anfrage den Wert für das Gehalt aus Block 1 bis 5 herauslesen, während in der spaltenorientierten Variante nur Block 3 gelesen werden muss. An dieser Stelle sei nochmals auf das Whitepaper „*C-Store: A Column-oriented DBMS*“ von Michael Stonebreaker [SAB+05] verwiesen, indem besonderen Wert auf eine vergleichende Performanzevaluation einer solchen Architektur gelegt wurde.

Nachteile dieser Variante sind vor allem die schlechtere Verarbeitung von OLTP Anfragen, da es sich hierbei um eine inverse Anforderung zum oben genannten Beispiel handelt.

Der Lesenachteil, der hierbei entsteht, lässt sich bis zu einem gewissen Punkt durch Caching und mehrspaltige Projektionen ausgleichen. Andere Performanznachteile für Operationen, die sich nur auf einen Datensatz beziehen, lassen sich nur schwer bis gar nicht verringern. In der Praxis verwenden solche Datenbanken einen schreiboptimierten Bereich, welcher *schreiboptimierter Delta Store* (*engl.* write-optimized delta store) genannt wird. Einfachheitshalber kann man sich die Datensätze im Delta Store als Zeilenformat vorstellen, auch wenn dieser in heutigen Systemen teilweise alternierend implementiert ist. Abbildung 4.2 illustriert hierbei die Beziehungen und Vorgänge zwischen dem spaltenbasierten Speicherbereich (*engl.* Column Store) und dem Delta Store.

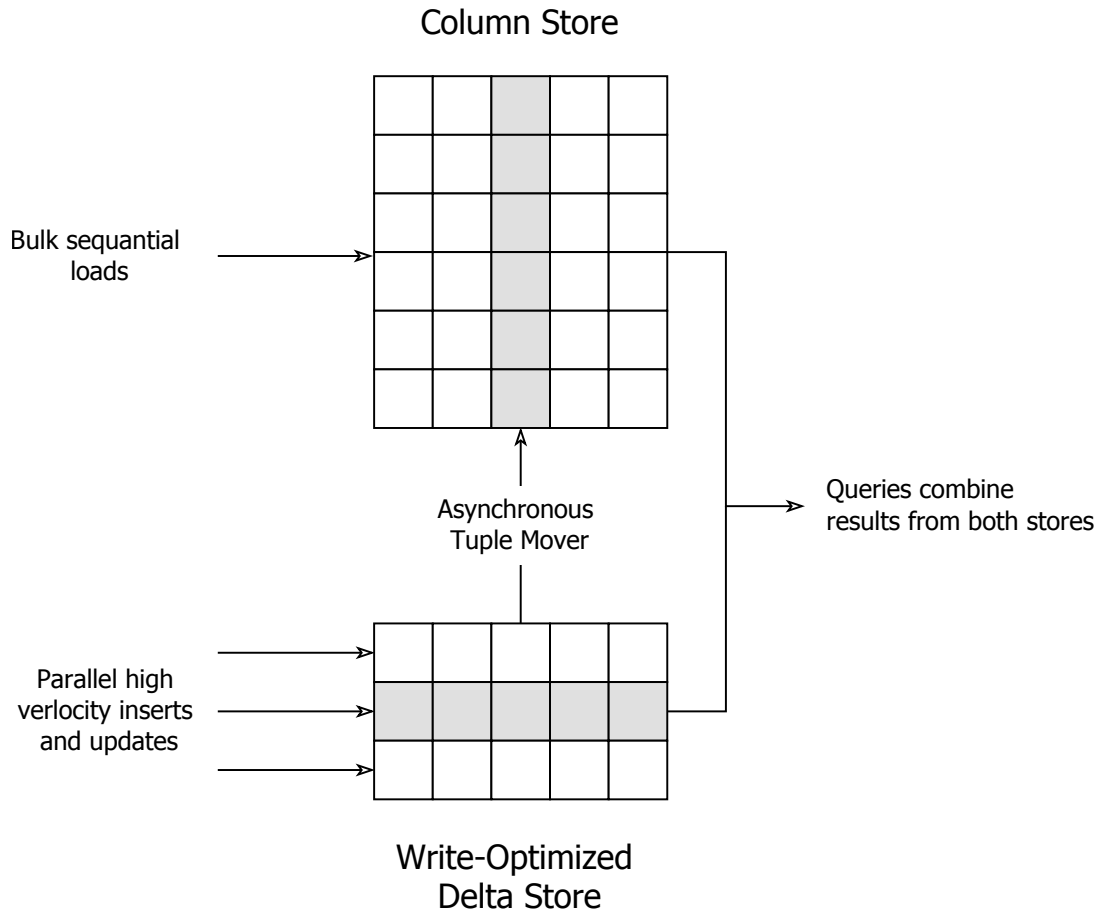


Abb. 4.2: Delta Store in spaltenbasierten Datenbanken angelehnt an: [Har15]

Die Daten des Delta Stores werden in periodischen oder triggerbasierten Abständen mit dem spaltenbasierten Speicher zusammengeführt. Trotzdem liefern Anfragen auf Daten sowohl Ergebnisse aus dem spaltenbasierten Speicher, als auch aus dem Delta Store zurück, damit von einem konsistenten Datenbankzustand ausgegangen werden kann. Bulk Loads, wie beispielsweise ETL Prozesse, werden direkt auf den Column Store ausgeführt, während Einfüge- bzw. Updateoperationen auf den Delta Store gelenkt werden.

Dieses Architekturprinzip besitzt einen sehr großen Einfluss auf die Entwicklung von relationalen Datenbanken, die Data Warehousing Rollen übernehmen wollen, sowie „$NewSQL$“ Systemen. Auch viele hauptspeicherbasierten Datenbanken implementieren die hier erklärte Architektur, um die Datenverwaltung zu optimieren. Auf diese soll im folgenden Unterkapitel näher eingegangen werden.

4.2 Hauptspeicherbasierte Datenbanken

Seit den ersten Datenbanksystem wird versucht den Platteninput bzw. -output so gering wie möglich zu halten, denn der Zugriff auf diesen ist schon immer deutlich langsamer gewesen, als der Zugriff auf den Hauptspeicher oder die CPU. Mit den Entwicklungen nach dem Moore'schen Gesetz hat sich dieser Zustand noch weiter in diese Richtung entwickelt, da hierdurch sowohl Hauptspeicher, als auch CPU stark an Performanz gewonnen haben, während die Plattenspeicherperformanz zurückgeblieben ist [SMA+07, Har15].

Mit dem Aufstieg und der Etablierung von Solid-State-Drives (SSDs) wurde dieser Entwicklung mit hohem Grad entgegengewirkt. Allerdings ist die Hauptspeicherkapazität von Servern bereits so groß, dass heutzutage viele kleinere Datenbanken vollständig in den Hauptspeicher eines Servers bzw. in den eines Clusters passen. So kann der Platteninput bzw. -output vollständig umgangen werden. Aus diesem Grund ist die hauptspeicherbasierte Datenbank ($engl.$ In-Memory Database) für ein solches Anwendungsszenario die effizientere Alternative [SMA+07, Har15]. Hinzu kommt, dass der Hauptspeicherpreis und die Hauptspeicherkapazität eines Servers sich in den vergangen Jahrzehnten exponentiell zugunsten dieses Konzeptes entwickelt haben [Sta17, SMA+07]. Abbildung 4.3 illustriert diese Entwicklung.

Es gibt generell **zwei Änderungen**, die eine solche In-Memory Datenbank im Vergleich zu traditionellen Datenbanken behandeln muss:

In traditionellen Datenbanken werden typischerweise Daten innerhalb des Hauptspeichers gecacht, um den Platteninput bzw. -output zu verringern. Dieses Vorgehen ist allerdings belanglos für In-Memory Datenbanken, denn die gesamte Datenmenge befindet sich bereits im Hauptspeicher. So muss eine solche Datenbank **ohne Cache** im traditionellen Sinn auskommen.

Zusätzlich muss ein **alternatives Persistenzmodell** zur Sicherung der Daten implementiert werden, da die Daten im Hauptspeicher bei Stromverlust verloren gehen. Hierzu werden üblicherweise eine oder mehrere der folgenden Methoden implementiert [Har15]:

- Übertragen der Daten zu den übrigen Mitgliedern des Clusters

- Speicherung von kompletten Datenbank Abbildungen (*engl.* Snapshots oder Checkpoints) auf die Platte
- Schreiben der Transaktionen bzw. Operationen in eine auf der Platte gespeicherte Datei (*engl.* Transaction Log oder Journal)

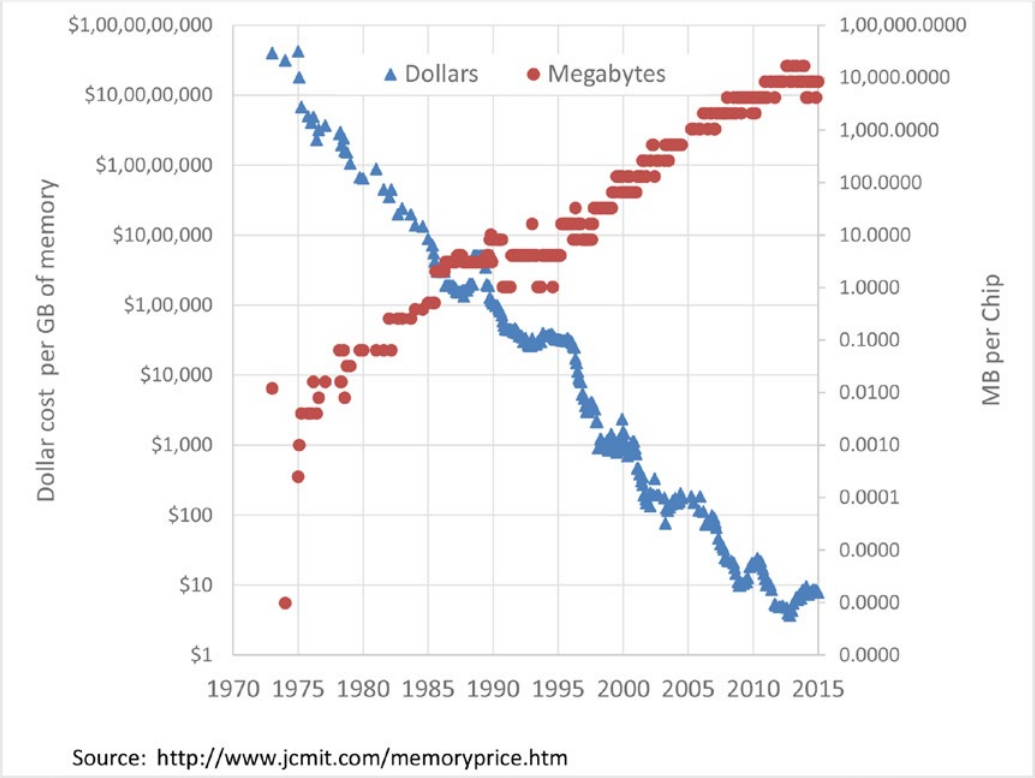


Abb. 4.3: Hauptspeicherkosten und -kapazität in den letzten Jahrzenten angelehnt an: [Har15]

Mit der Konfiguration, die im Whitepaper „*The End of an Architectual Era (It's Time for an Complete Rewrite)*“ [SMA+07] beschrieben wird, erreicht die In-Memory Datenbank „*H-Store*“ 70.416 Transaktionen pro Sekunde im Vergleich zu den 850 Transaktionen pro Sekunde einer relationalen Datenbank. Somit ist H-Store 82 mal schneller als das Vergleichsprodukt. In den letzten Jahren wurden In-Memory immer beliebter im Umfeld von Big Data Applikationen und High-Performance Computing (HPC) [Pok15].

Im folgenden Kapitel sollen mehrere Möglichkeiten zur Datenbankskalierbarkeit aufgezeigt und erläutert werden. Diese nehmen vor allem im Rahmen von NewSQL Datenbanken eine elementare Rolle ein.

Kapitel 5

5 Skalierbarkeit in Datenbanken

Im modernen Datenbankumfeld und dem Zeitalter von Big Data ist es notwendig, dass Datenbanken, bei wandelnden Anfragestrukturen und Datenmengen, auch weiterhin eine hohe Verfügbarkeit, sowie eine optimale Performanz bereitstellen [Mul17]. In diesem Umfeld gilt natürlich, sobald entsprechende Daten nicht verfügbar sind, zieht dies eine unmittelbare Verfügbarkeitsstörung der jeweiligen Applikation mit sich. Eine nicht verfügbare, beziehungsweise langsame, Applikation resultiert unmittelbar im indirekten Verlust von Geld oder anderen monetären Größen. Dieser Zusammenhang sollte den Grad der Wichtigkeit einer verfügbaren und vor allem betriebsbereiten Datenbank verdeutlichen.

In diesem Kapitel sollen zunächst notwendige Terminologien definiert, sowie verschiedene Skalierbarkeitstypen erläutert werden. Daraufhin werden verschiedene traditionelle Skalierbarkeitsmethoden mit passenden Diagrammen illustriert und vermittelt. Abschließend werden moderne Skalierbarkeitsmethoden aufgezeigt und mit passenden Beispielen differenziert.

5.1 Begriffsdefinitionen Skalierbarkeit vs Elastizität

Die Begriffe „*Skalierbarkeit*“ und „*Elastizität*“ werden oftmals falsch, oder auch als Synonyme füreinander eingesetzt. Aus diesem Grund sollen hier passende Definitionen eingeführt werden, um eine konsistente Begriffsstruktur zu gewährleisten.

Skalierbarkeit beschreibt die Eigenschaft eines Systems wachsenden Arbeitsumfang mit geringem Modifikationsaufwand performant zu bewältigen.

> „*Scalability refers to the capability of a system to handle growing amount of work, or its potential to perform more total work in the same elapsed time when processing power is expanded to accomodate growth*“ - [Mul17]

Auch die Verfügbarkeit während Administrationsänderungen ist ein wichtiger Aspekt innerhalb der Skalierbarkeit, da bei wachsender Nachfrage Schema-Änderungen, Upgrades oder Wartungsarbeiten durchgeführt werden müssen, ohne die Endnutzererfahrung zu vernachlässigen [Mul17]. Generell lässt sich sagen: Ein skalierendes System muss auch bei Modifikationen des DBMS bezüglich der wachsenden Nachfrage abgewandelt werden können, ohne die Verfügbarkeit zu verringern.

> „*Elasticity is the degree to which a system can adapt to workload changes by provisioning and deprovisioning resources in an on-demand manner, such that each point in time the available resources match the current demand as closely as possible*“ - [Mul17]

Das Ziel der Elastizität ist es zu jedem Zeitpunkt t die Ressourcen R_t des Systems so anzupassen, dass R_t nur die tatsächlich benötigten Ressourcen sind. Sowohl eine Überbelegung, als auch eine Unterbelegung von Ressourcen, können hierdurch beseitigt werden [Mul17]. Beide Varianten sorgen für den Verlust von Kapital eines Unternehmens. Bei Überbelegung werden zu viele Ressourcen bereitgestellt, was höhere Kosten in der Bereitstellung fordert, während bei Unterbelegung eventuelle Performanzprobleme auftreten, was z.B. zu Kundenverlust führen kann.

Im Datenbankumfeld wird die Elastizität mittels Clustering und flexiblen Datenmodellen implementiert [Mul17]. Hier gilt: Je mehr Änderungen toleriert werden und je einfacher Clustering umsetzbar ist, desto elastischer ist das DBMS [Mul17].

5.2 Typen der Datenbankskalierbarkeit

Die Skalierbarkeit einer Datenbank lässt sich in zwei Typen klassifizieren, die „*Vertikale Skalierbarkeit*“ und die „*Horizontale Skalierbarkeit*“. Beide Typen werden in diesem Unterkapitel näher erläutert und Vorteile bzw. Nachteile werden anhand passender Beispiele herausgearbeitet.

5.2.1 Vertikale Skalierbarkeit

Die vertikale Skalierbarkeit (*engl.* auch Scale up) ist das Hinzufügen von Ressourcen, wie beispielsweise Hauptspeicher oder bessere CPUs. Die inverse Operation hierzu ist bekannt als Scale Down und ist prinzipiell das Entfernen von Ressourcen, beispielsweise durch einen Speicheraustausch oder das Einbauen schlechterer CPUs.

Scale Up resultiert üblicherweise in einer Performanzoptimierung und ist die standardmäßige Methode der Skalierbarkeit für relationale Datenbankmanagementsysteme (RDBMs), die für einen Server entwickelt wurden [Mul17]. Diese Methode zieht allerdings eine Neukonfiguration, sowie Downtime mit sich und ist limitiert durch das System und die Software des Systems [Mul17].

5.2.2 Horizontale Skalierbarkeit

Horizontale Skalierbarkeit (*engl.* auch Scale Out) ist das Hinzufügen von Hardware zu einem System. Die inverse Operation hierzu, also das Entfernen von Hardware, wird als Scale In bezeichnet [Mul17].

Da sich der Preis von Hardware in den vergangen Jahrzehnten verringert hat, wie in Abbildung 4.3 gezeigt wurde, macht es mehr Sinn diese Methode der Skalierbarkeit zu adaptieren. Limitiert ist diese Art der Skalierbarkeit durch die Möglichkeiten der Software, die in einem Netzwerk bereitgestellten, Computerressourcen zu nutzen. Natürlich bringt dieser Ansatz auch mehr Konfigurations- bzw. Verwaltungsaufwand mit sich und es besteht eine mögliche Latenz zwischen einzelnen Knoten [Mul17].

Abbildung 5.1 stellt den Unterschied zwischen vertikaler und horizontaler Skalierbarkeit noch einmal übersichtlich dar.

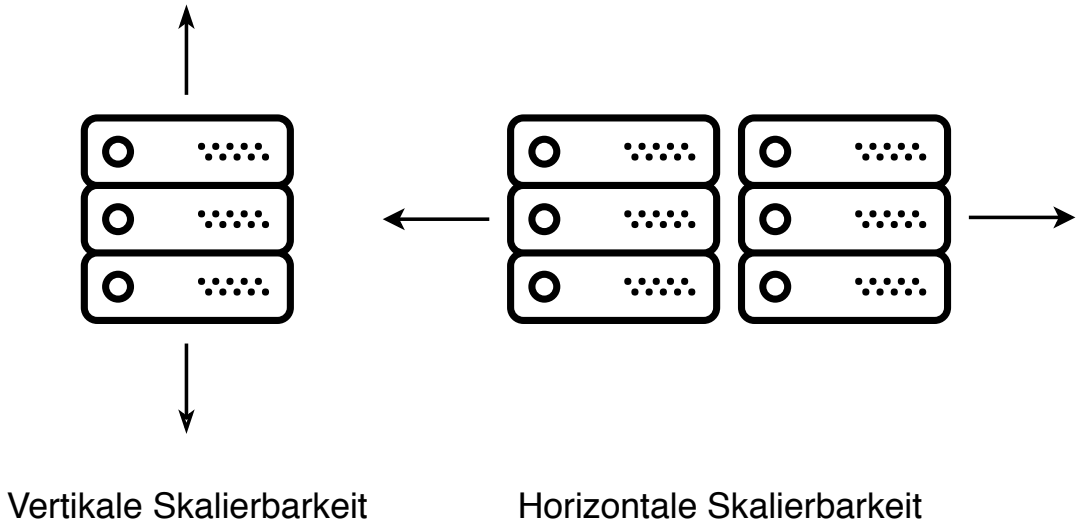


Abb. 5.1: Vergleich zwischen vertikaler und horizontaler Skalierbarkeit

Es besteht auch die Möglichkeit ein System sowohl vertikal, als auch horizontal zu skalieren und so eine Kombination der beiden Varianten zu realisieren. Hierzu werden Ressourcen zu bestehenden Servern hinzugefügt, um vertikal zu skalieren, und zusätzliche Server eingebunden, um horizontal zu skalieren [Mul17].

5.3 Traditionelle Skalierbarkeitsmethoden in Datenbanken

Traditionell wird in Datenbanken Clustering eingesetzt, um Skalierbarkeit in Datenbanksystemen zu gewährleisten. Damit werden mehrere Server eingesetzt, um Datenbankanfragen abzuarbeiten und die Arbeitsmenge aufzuteilen. Hier gibt es üblicherweise zwei Ansätze, namentlich „*Shared-Disk*“ und „*Shared-Nothing*“. Diese beiden Ansätze zur horizontalen Skalierbarkeit sollen in folgenden Unterkapiteln näher erläutert werden.

5.3.1 Shared-Disk

In einer Shared-Disk Architektur besitzen alle Server einen gemeinsamen Plattenspeicher, haben aber trotzdem noch eigene CPUs und Hauptspeicher [Lee11]. Es kann lediglich auf diesen kollektiven Plattenspeicher zugegriffen werden. Abbildung 5.2 zeigt das Schema einer solchen Architektur.

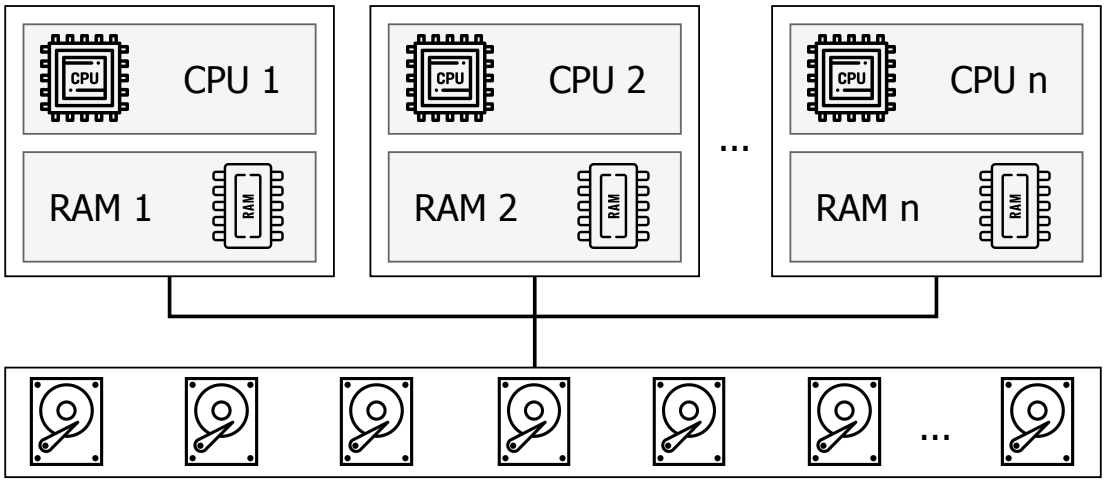


Abb. 5.2: Schema einer Shared-Disk Architektur angelehnt an: [Mul17]

Wichtig zu nennen ist hierbei, dass durch den separaten Hauptspeicher und die Möglichkeit Daten zu cachen, bei Änderungen des Masterdatensatzes die jeweiligen Caches aktualisiert werden müssen, um auch weiterhin eine Datenkonsistenz zu gewährleisten [Mul17]. Außerdem muss ein angemessenes blockieren bzw. freigeben von Ressourcen stattfinden, um Änderungen von mehreren Knoten konsistent zu bearbeiten.

Vorteile von Shared-Disk sind unter anderem eventuell geringere Kosten, Erweiterbarkeit, höhere Verfügbarkeit und generell Load Balancing [Lee11, Mul17]. Eine solche Shared-Disk Implementierung findet sich unter anderem in den DBMS DB2 und Oracle RAC wieder.

5.3.2 Shared-Nothing

Im Shared-Nothing Ansatz besitzt jeder Server einen privaten Hauptspeicher, sowie einen privaten Plattenspeicher [Lee11]. Abbildung 5.3 zeigt dieses Schema einer solchen Shared-Nothing Architektur und differenziert hierbei die einzelnen Bereiche.

Die einzelnen Server kommunizieren über Nachrichten über ein aufgespanntes Netzwerk. So werden Anfragen von Clients automatisch an das System mit den benötigten Daten weitergeleitet und von diesem bearbeitet. Wichtig ist hier, dass Daten zu jedem Zeitpunkt t nur jeweils zu einem System gehören und von diesem bearbeitet werden können [Lee11]. Beim Ausfall eines Systems können die Daten allerdings dynamisch einem verfügbaren System zugeordnet werden.

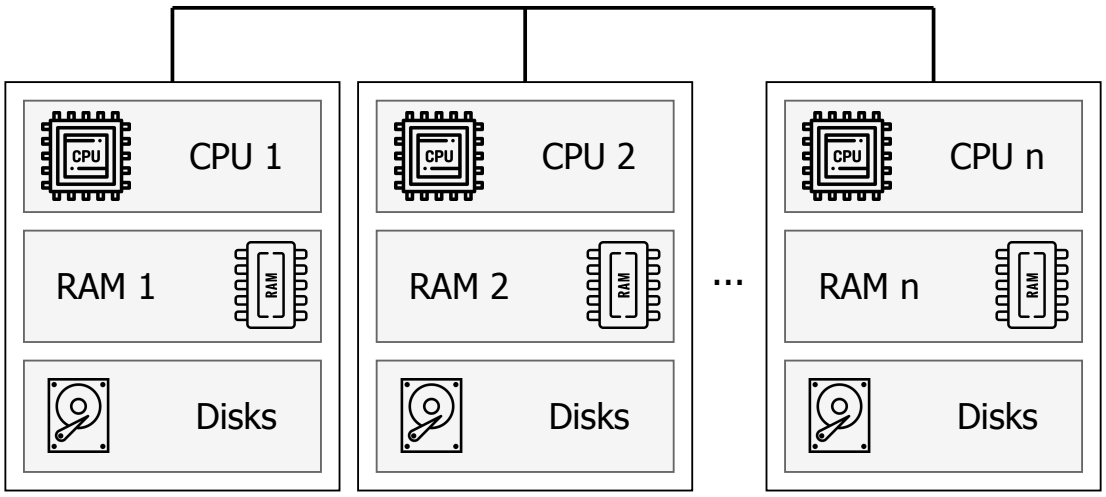


Abb. 5.3: Schema einer Shared-Nothing Architektur angelehnt an: [Mul17]

Der größte Vorteil einer solchen Architektur ist die verbesserte Skalierbarkeit, die sie mit sich bringt [Lee11, Mul17]. Shared-Nothing Architekturen sind aufgrund der isolierten Ressourcen stark erweiterbar und im Vergleich zu Shared-Disk Architekturen besser skalierbar. Dadurch bietet es sich an eine solche Architektur für die leseintensive OLAP Anfragen eines Data Warehouses einzusetzen.

Ein großer Nachteil einer solchen Architektur ist vor allem das Partitionierungsschema, welches zur Distribution der Daten auf die verschiedenen Knoten der Datenbank entwickelt werden muss [Sto86]. Hierbei gibt es nämlich kein optimales Partitionierungsschema, sondern dieses muss individuell auf die jeweilige Applikation abgestimmt werden. So sollte dieses bei Änderungen der typischen Anforderungen oder Datenvolumina hinsichtlich der Effizienz überdacht werden. Üblicherweise werden Daten horizontal nach einer Zeile partitioniert. Hierbei erhält jede Tabelle eine Untermenge der Zeilen in der ursprünglichen Tabelle. Beispielsweise enthält Tabelle 1 Daten über europäische Verkaufszahlen und Tabelle 2 Daten über nicht europäische Verkaufszahlen. Auch eine vertikale Partitionierung ist allerdings möglich.

Ein Problem, welches hierbei auftritt, ist das Modifizieren oder Abrufen von Daten über mehrere Partitionen. Solange Anfragen sich nur auf eine Partition beziehen funktioniert Shared-Nothing am Besten, wenn sich dieses auf mehrere Partitionen ausweitet, besteht eine höhere Komplexität und ACID Transaktionen könnten verloren gehen [Sto86].

Datenbanken, die Shared-Nothing implementierten, sind unter anderem „$MySQL$“, „$Teradata$“ und viele NoSQL oder auch NewSQL Varianten [Mul17].

5.3.3 Datenbank Sharding

Sharding wird oftmals in Shared-Nothing Architekturen eingesetzt, um automatisch die Datenbank zu partitionieren und die Daten in kleine Untermengen ($engl.$ auch

Shards) einzuteilen [Har15]. Generell teilt Sharding, ähnlich wie bei der horizontalen Partitionierung, die Daten zeilenweise. Der Unterschied ist hierbei, dass Daten auf mehrere Datenbankinstanzen verteilt werden, während bei horizontalen Partitionierung meistens innerhalb einer Instanz gearbeitet wird [Pok15]. Jeder Shard stellt hierbei eine separate logische Instanz dar, diese können über mehrere Datenbankserver, Datencenter oder sogar Kontinente verteilt sein. Es gibt verschiedene Methoden, wie diese Shards erstellt werden [Gur15]:

Bereichspartitionierung ($engl.$ range partitioning) verteilt die Datensätze auf Partitionen basierend auf den Bereichen eines Partitionierungsschlüssels. Initial werden alle Datensätze vom routenden Server anhand dieser Bereiche aufgeteilt. Anschließend ist jeder Knoten für die Haltung der Daten und das Lesen bzw. Schreiben auf diesen selbst zuständig.

Konsistentes Hashing ($engl.$ consistent hashing) funktioniert, indem die Daten als Ring betrachtet werden. Dieser Ring wird in n Bereiche eingeteilt, wobei n die Anzahl der verfügbaren Knoten ist. Jeder Knoten wird anschließend auf eine Position im Ring gesetzt. Um nun die Zugehörigkeit eines Datenobjekts zu erhalten, wird der entsprechende Hashwert berechnet und auf dem Ring platziert. Dieses Datenobjekt besitzt die Zugehörigkeit zu dem nächsten Knoten auf dem Ring im Uhrzeigersinn.

Datenbanken, die automatisches Sharding implementieren sind unter anderem Apache HBase, Couchbase und Informix [Mul17].

Sharding kann allerdings auch einen großen negativen Einfluss auf die Fehlertoleranz einer Datenbank haben. Sobald ein Shard nicht mehr erreichbar ist ($engl.$ offline Shard), sind die dort gespeicherten Daten nicht mehr verfügbar. Um dies zu verhindern, werden oftmals Replikate der Daten eingesetzt.

5.3.4 Replikation

Replikate werden, wie bereits erwähnt, meistens in Kombination mit Sharding bei Shared-Nothing Architekturen eingesetzt, um die Fehlertoleranz zu verbessern [Har15, Mul17]. Hierbei ist ein Replikat eine Kopie der Daten, die verteilt im Cluster abgelegt wird. Es lässt sich generell in drei Varianten von Replikaten unterscheiden [Pok15, SK15]:

Master-Slave Replikation legt einen Knoten als bestimmende Kopie fest, die Schreiboperationen bearbeitet, während die anderen Kopien sich damit synchronisieren und Leseoperationen bearbeiten [Gur15].

Multi-Master Replikate besitzen keine Aufteilung von Operationen, welche nur bestimmte Replikate bearbeiten dürfen, sondern jedes Replikat kann Änderungen an den Daten vornehmen. Dies setzt natürlich eine Synchronisierung der Daten verschiedener Replikate voraus. Alle Instanzen arbeiten hierbei synchron, sodass keine Diskrepanzen zwischen den einzelnen Knoten bestehen.

Peer-to-Peer (P2P) besitzen ebenfalls keine Aufteilung von Operationen. Sie unterscheiden sich zu Multi-Master Replikaten dadurch, dass die Schreiboperation zu den anderen Knoten übertragen wird. Das bedeutet, dass es zu einer Verzögerung der Synchronisierung kommen kann. Eine Anfrage zur selben Zeit an zwei unterschiedliche Knoten kann hierbei ein unterschiedliches Resultat liefern.

Zwei Beispiele, welche den Unterschied und die Benutzung zwischen Multi-Master Replikaten und Peer-to-Peer Replikaten verdeutlichen, sind Datenbanken für Kontotransaktionen und Datenbanken für Social Media Posts. Bei der Datenbank für Kontotransaktionen ist es wichtig, dass eine Person nicht zweimal die gleiche Menge Geld abheben kann und so mehr abhebt, als sich auf seinem Konto befindet. Bei der Datenbank für Social Media Posts könnte diese geographisch verteilt aufgesetzt sein, um eine große Menge Anfragen zu bearbeiten. Hierbei ist es egal, ob ein Post aus Deutschland sofort den Personen in Australien angezeigt wird, oder erst einige Minuten verzögert.

In dem Whitepaper „$What's\ really\ New\ with\ NewSQL$“ von Andrew Pavlo und Matthew Aslett [PA16] wird die synchrone Arbeitsweise von Multi-Master Replika als aktiv-aktiv und die frühzeitige Bearbeitung von Peer-to-Peer Replika innerhalb eines Knotens als aktiv-passiv beschrieben. Ebenfalls wird hier die Eigenschaft von stark konsistenten ($engl.$ strongly consistent) DBMS aufgezeigt, in der Transaktionen von allen Knoten bestätigt und angewendet werden müssen, bevor die Transaktion als abgeschlossen angesehen wird. Das Gegenteil hierzu wird letztendlich konsistente ($engl.$ eventual consistent) DBMS genannt. Innerhalb dieser Arbeit werden diese Begrifflichkeiten im späteren Verlauf ebenfalls mit dieser Definition verwendet.

Je nachdem, ob Replikate aktiv-aktiv oder aktiv-passiv geschrieben werden, besitzen diese Ansätze verschiedene Vor- und Nachteile. Bei der asynchronen aktiv-passiv Variante kann nicht sichergestellt werden, dass die Daten erfolgreich zu allen Knoten repliziert wurden. Dies bedeutet, dass sehr einfach Daten verloren gehen können. Allerdings sind diese einfacher zu implementieren als aktiv-aktiv Replikate.

5.4 Moderne Skalierbarkeitsmethoden in Datenbanken

In den vorangegangen Unterkapiteln wurden traditionelle Methoden vorgestellt, die zur Skalierbarkeit bzw. Elastizität in Datenbanken eingesetzt werden. Sowohl Shared-Disk, Shared-Nothing, als auch NoSQL besitzen Nachteile, die bereits genannt wurden. Im realen Unternehmensumfeld wird ein DBMS System benötigt, welches weiterhin SQL und ACID Transaktionen unterstützt, aber die Möglichkeit besitzt elastisch Scale Up bzw. Scale Down zu betreiben [Mul17]. Hierfür gibt es elastische SQL Datenbanksysteme,

die eine serviceorientierte Alternative zu den bisherigen traditionellen Architekturen darstellen [Mul17]. Diesbezüglich gibt es mehrere Ansätze, im folgenden soll besonders auf die zwei Schichten Architektur mit Multi-Version Concurrency Control (MVCC) eingegangen werden.

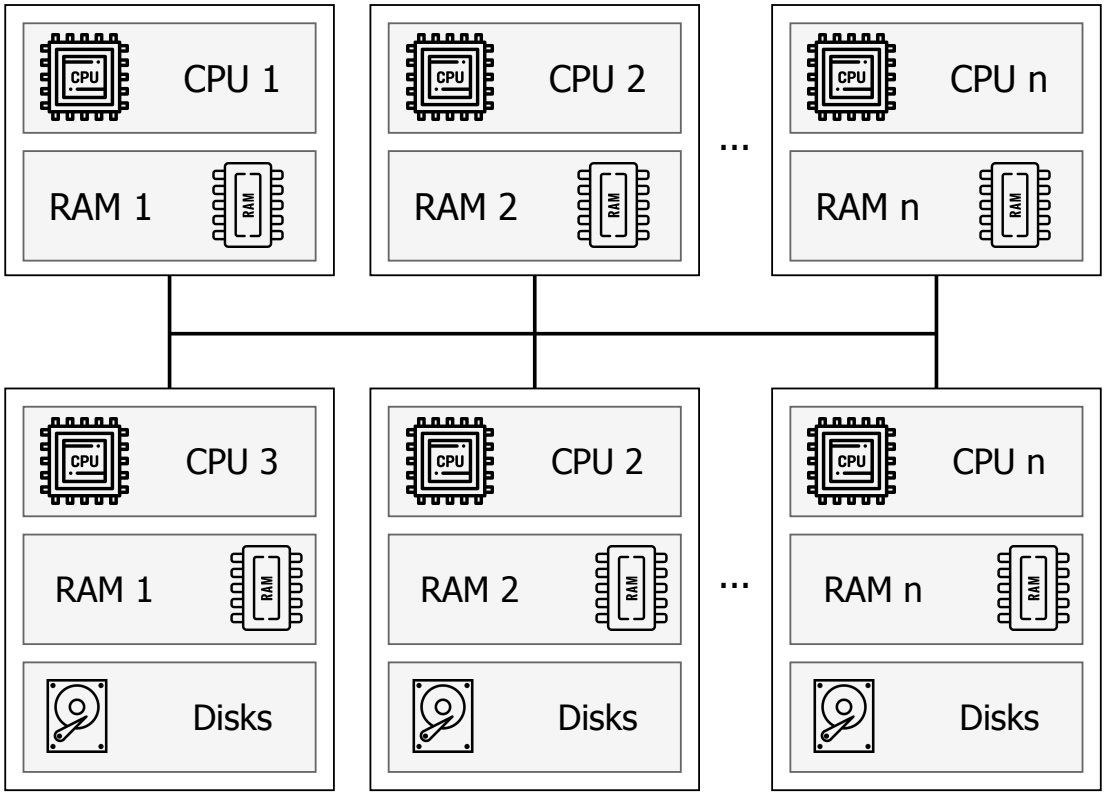


Abb. 5.4: Schema einer elastischen SQL Architektur angelehnt an: [Mul17]

In einem solchen DBMS wird die transaktionale Schicht von der Speicherungsschicht separiert. Der transaktionale Teil unterstützt hier einen In-Memory Cache verteilt über mehrere Server oder sogar geographisch verteilte Datencenter. Während die Speicherungsschicht Peer-to-Peer Nachrichten verwendet, um Datenbankcommits zu verwalten und Daten, die nicht im transaktionalen Cache vorliegen, zu liefern [Mul17]. Da beide Schichten unabhängig voneinander entwickelt werden, können diese auch individuell skaliert werden. So können bei höherem oder niedrigerem Durchsatz mehrere transaktionale Prozesse gestartet bzw. gestoppt werden und das DBMS passt die Aufgabenverteilung automatisch an den bestehenden Umfang an. Gleichermaßen kann mehr als ein Speicherungsprozess gestartet werden. Diese Architektur von elastischem SQL zeigt sich in Abbildung 5.4.

Da die Daten an mehreren Stellen gecacht werden können, müssen diese durch Replika konsistent in beiden Schichten gehalten werden [Mul17]. Plattenzugriffe sind generell nur notwendig, wenn kein Knoten die benötigten Daten hält, oder beim Aktualisieren, Einfügen oder Löschen von Datensätzen. Der Lese- bzw. Schreibzugriff wird mittels Multi-Version Concurrency Control sichergestellt. Hierbei werden alle Daten als Versio-

nen betrachtet, wobei eine neue Version bei jeglichen Modifikationen entsteht. Jeder Transaktionsprozess cachet mehrere Versionen eines Objektes, solange bis die entsprechende Transaktion comitted wird. Bei einem Rollback wird die entsprechende Version einfach aus dem Cache gelöscht.

Alternativ werden zur Concurrency Control teilweise auch zeitstempelbasierte Verfahren ($engl.$ Timestamp Ordering TO) eingesetzt. Hierbei nimmt das Datenbankmanagementsystem an, dass Transaktionen keine Operationen ausführen, welche die Serialisierbarkeit beeinträchtigen [PA16].

In einer solchen Architektur sind die einzelnen Knoten in ihrer Gesamtheit unabhängig voneinander, keiner ist alleine zuständig für Aufgaben oder Daten. So kann jeder Knoten heruntergefahren werden, ohne dass die Verfügbarkeit des Gesamtsystems darunter leidet [Mul17]. Der größte Vorteil einer solchen elastischen SQL Struktur liegt in der Verbindung von konsistenten Transaktionen mit standardisiertem SQL und einer einfachen Scale-Out Realisierung. So liegen die Stärken dieser Architektur in der Bearbeitung von OLTP Anfragen [Mul17]. Beispiele für elastische SQL Systeme sind NuoDB und CockroachDB, die im späteren Verlauf dieser Arbeit explizit beleuchtet werden.

5.5 Abschließender Vergleich der verschiedenen Skalierbarkeitsmethoden

Stellt man Shared-Disk, Shared-Nothing und elastisches SQL gegenüber, zeigen sich die einzelnen Vorteile bzw. Nachteile noch einmal deutlicher. Tabelle 5.1 stellt genau diese Unterschiede, die bereits in vorherigen Kapiteln aufgezeigt wurden, übersichtlich dar.

Zusammenfassend lässt sich sagen, dass jede dieser Skalierbarkeitsmethoden in verschiedenen Bereichen die bestmögliche Alternative darstellt und deswegen, je nach Anforderungen, die passende DBMS Architektur gewählt werden sollte. Hierzu sollten die einzelnen Vor- bzw. Nachteile bekannt sein, um sich einen groben Überblick verschaffen zu können und so effektiv eine fundierte Entscheidung treffen zu können.

Sowohl die Skalierbarkeit, als auch die Elastizität, sind Eigenschaften, die für ein Datenbanksystem im heutigen Zeitalter unabdingbar sind. Die jeweiligen Implementierungen dieser Konzepte in kommerziellen DBMS Systemen wird im späteren Verlauf dieser Arbeit näher betrachtet und vorgestellt. Dies bildet einen wesentlichen Unterschied zwischen den verschiedenen NewSQL Datenbanksystemen. Zunächst soll allerdings ein passendes Konzept zur Evaluierung verschiedener NewSQL Datenbankmanagementsystemen aufgestellt werden.

Shared-Disk	Shared-Nothing	Elastisches SQL
Anpassung an wandelnden Arbeitsumfang, Load Balancing	Kann günstige Hardware ausnutzen	Benutzt günstigere Hardware, um sich an den Arbeitsumfang anzupassen. Automatisiertes Load Balancing
Hohe Verfügbarkeit	Hohe Skalierbarkeit, mehr Downtime	Durchgehende Verfügbarkeit, hohe Skalierbarkeit
Einfaches Scale-In	Datenpartitionierung, kein Scale-In	Einfaches Scale-In
Gute Performanz bei Leseoperationen	Gute Performanz bei vielen Lese- und Schreiboperationen	Sowohl bei Lese-, als auch Schreiboperationen sehr gute Performanz
Daten müssen partitioniert werden	Daten müssen über den Cluster partitioniert werden	Daten müssen partitioniert werden

Tab. 5.1: Vergleich zwischen Shared-Disk, Shared-Nothing und elastischem SQL angelehnt an: [Mul17]

Kapitel 6

6 Vorbereitung der Evaluierung

Zu entscheiden, welche Software in einem Unternehmen eingesetzt werden soll, kann eine besonders fordernde Aufgabe sein. Neben den Vorteilen einer Software bringt diese immer bestimmte Risiken mit sich. Dabei kann es sich beispielsweise um Kompatibilität, Nutzbarkeit, Skalierbarkeit oder auch Sicherheit handeln [Ope05]. Diese und noch mehr Aspekte gilt es bei der Auswahl einer passenden Softwarelösung zu beachten und abzuwägen.

Dieses Kapitel soll die Methodiken und Vorgehensweisen bei der Evaluierung der einzelnen NewSQL Datenbanken erklären und somit die Grundlage der Evaluierung darstellen. Auf Basis der Kriterien, die in diesem Kapitel ausgewählt werden, wird eine Vorbewertung der einzelnen Optionen vorgenommen und so ein Satz NewSQL Datenbanken ausgewählt. Des Weiteren bilden die hier beschriebenen Hauptbewertungskriterien das Fundament für die abschließende Evaluierung der einzelnen Datenbanken.

6.1 Bedeutung von Open Source Software

Open Source Software (OSS) ist Software, dessen Quellcode für die Öffentlichkeit frei verfügbar ist. Oftmals werden die Begriffe *„Free Software“* und *„Open Software“* als Synonyme füreinander eingesetzt, obwohl diese sich in ihrer Philosophie und Lizenzierung unterscheiden [Hah14]. Eine Open Source Software muss per se nicht kostenlos sein, sondern kann auch anderweitig lizenziert sein. Ein Großteil der heutigen Software ist allerdings beides, sowohl offen, als auch frei verfügbar.

> *„The term open source generally refers to software that is made readily available by an individual or group for others to use, modify, or redistribute under a licensing agreement with very few restrictions.“* - [Hah14]

In heutigen Unternehmen werden immer mehr Open Source Softwarekomponenten implementiert, da diese im Vergleich zu passenden Closed Source Alternativen diverse Vorteile mit sich bringen. Die wohl offensichtlichste Verbesserung ist die Senkung der Lizenzkosten, da diese bei den meisten Open Source Anwendungen wegfallen. Zudem bieten sie einen geringeren Lock-In-Effekt und eine reduzierte Abhängigkeit von einem Entwickler, was ebenfalls die Zukunftssicherheit der ausgewählten Software erhöht

[Kes13]. Darüber hinaus wird durch die Offenheit des Quellcodes, die Sicherheit von OSS stetig verbessert, da es allen Beteiligten offen steht, Sicherheitslücken zu finden und zu beheben [Kes13]. Natürlich bringt Open Source Software auch einige Nachteile mit sich, wie mangelnder Support, unsichere Weiterentwicklung der Komponenten oder eingeschränkte Benutzerfreundlichkeit.

Im Rahmen dieser Studienarbeit wird die Evaluierung auf Open Source NewSQL Datenbanken beschränkt. Dies ist in besonderem Maße durch die oben genannten Vorteile und die Möglichkeit zur Analyse der Implementierung begründet. Das für die Bewertung angewandte Evaluierungsmodel wird im folgenden Unterkapitel näher erläutert.

6.2 Open Source Evaluierungsmodel

Damit ein methodisches Vorgehen innerhalb der Bewertung von verschiedenen NewSQL Datenbanken gewährleistet ist, wurde das *Business Readiness Rating Model* (BRR) [Ope05, Dou08] zur Bewertung von Open Source Anwendungen herangezogen. Hiermit lässt sich das breite Feld der Open Source Anwendungen, im Bereich der NewSQL Datenbanken, auf einen Satz möglicher Alternativen beschränken. Das angewandte Model lässt sich in vier Phasen strukturieren.

Die nachfolgenden Phasen sind im Wesentlichen aus der Beschreibung „*Business Readiness Rating for Open Source - A Proposed Open Standard to Facilitate Assessment and Adoption of Open Source Software*“ [Ope05] entnommen worden, welche aus der Zusammenarbeit von SpikeSource und der Intel Corporation® entstanden ist.

Phase 1 - Vorbewertung In dieser Phase werden verschiedene Komponenten ausgewählt, die anhand diverser Kategorien evaluiert werden sollen. Daraufhin wird eine Liste aus Vorkriterien, oder auch K.o.-Kriterien, ausgearbeitet, auf deren Basis die ausgewählten Komponenten geprüft werden. Jede Komponente, die hierbei eines der Vorkriterien nicht erfüllt, wird aus der Liste genommen und fällt damit durch die Bewertung.

Phase 2 - Bewertung des Einsatzumfeldes Innerhalb des Business Readiness Rating gibt es zwölf Kategorien, anhand derer eine Open Source Anwendung bewertet werden kann. Diese sollen in dieser Phase nach ihrer Relevanz sortiert werden, wobei 1 die höchste Platzierung darstellt und 12 die niedrigste. Den sieben höchsten Kategorien (weniger sind ebenfalls möglich) soll nun eine Prozentzahl zugewiesen werden, welche die Wichtigkeit des jeweiligen Aspektes repräsentiert. Die Summe aller Prozentzahlen über die sieben ausgewählten Kategorien soll 100% ergeben.

Jede Kategorie besitzt eine oder mehrere Metriken. Eine Metrik ist in diesem Umfeld ein Kriterium zur Bewertung innerhalb der jeweiligen Kategorie. Alle Metriken sollen nun analog zu den Kategorien nach ihrer Wichtigkeit geordnet und mit einer Prozentzahl versehen werden. Die Summe aller Metriken innerhalb einer Kategorie soll ebenfalls 100% ergeben.

Phase 3 - Datensammlung und -verarbeitung Nun werden Daten über die ausgewählten Komponenten gesammelt und auf die Metriken innerhalb der Kategorien angewendet. Eine Metrik besitzt hierbei eine Bewertungsskala von 1 (nicht akzeptabel) bis 5 (exzellent), wobei 1 die schlechteste und 5 die beste Bewertung ist. Die Metriken sollen auf Grundlage der gesammelten Informationen bewertet werden und anschließend wird ein gewichteter Wert anhand der zuvor vergebenen Prozentzahl errechnet. Falls eine Metrik nur zwei Alternativen zulässt, also zum Beispiel Ja oder $Nein$, soll eine negative Antwort mit 1 (nicht akzeptabel) und eine positive Antwort entweder mit 3 (akzeptabel) oder mit 5 (exzellent) bewertet werden. Welche der beiden Optionen für eine positive Antwort gewählt wird, soll anhand des Verbesserungsgrades durch diese Metrik festgemacht werden.

Phase 4 - Datenübertragung Abschließend werden aus den gewichteten Werten der Metriken die gewichteten Werte der Kategorien ermittelt. Die Summe aller gewichteten Werte der Kategorien ergibt daraufhin die endgültige BRR-Punktzahl. Die BRR-Punktzahl lässt sich also nach Formel 6.1 berechnen.

Sei n die Anzahl der gewichteten Kategorien, x_i die Gewichtung einer Kategorie ($\sum_{i=1}^{n} x_i = 1$), p_i die Anzahl der Metriken innerhalb einer Kategorie, m_{ij} die Gewichtung einer Metrik ($\forall i : \sum_{j=1}^{p_i} m_{ij} = 1$) und b_{ij} die Bewertungspunktzahl einer Metrik. Dann ergibt sich für die BRR-Punktzahl B folgende Formel mit ihrem Wertebereich W_B:

$$B = \sum_{i=1}^{n} x_i \cdot \sum_{j=1}^{p_i} m_{ij} \ \cdot \ b_{ij} \qquad\qquad W_B \longmapsto [0;\ 5] \tag{6.1}$$

6.3 Kriterienauswahl

In diesem Unterkapitel sollen die Kriterien für die Bewertung der NewSQL Datenbanken aufgezeigt werden und die jeweils enthaltenen Metriken bestimmt werden. Die Kriterien umfassen einerseits die Vorbewertungskriterien, die besonders in Phase 1 verwendet werden, andererseits die Hauptkriterien, die in den übrigen Phasen eine größere Rolle spielen.

Die hier aufgelisteten Kriterien und Metriken sind in ihrer Struktur aus der Beschreibung „*Business Readiness Rating for Open Source - A Proposed Open Standard to*

Facilitate Assessment and Adoption of Open Source Software“ [Ope05] entnommen worden.

6.3.1 Kriterien der Vorbewertung

Innerhalb der Phase 1 - Vorbewertung wurden folgende Vorkriterien ausgewählt, die für alle NewSQL Datenbanken gelten müssen, damit sie in der Hauptbewertung berücksichtigt werden können:

- Das Projekt besitzt eine Open Source Lizenzierung.
- Das Projekt soll als NewSQL Datenbank klassifiziert werden.
- Die Software wurde erstmalig vor mindestens 6 Monaten veröffentlicht.
- Die Updateintervalle suggerieren eine stetige Aktualisierung.

6.3.2 Kriterien der Hauptbewertung

Die 12 Kriterien, anhand derer die Hauptbewertung stattfindet, umfassen Funktionalität, Nutzbarkeit, Qualität, Sicherheit, Performanz, Skalierbarkeit, Architektur, Support, Dokumentation, Annahme, Community und Professionalität. Diese sollen innerhalb dieses Unterkapitels näher erläutert werden.

Funktionalität

Das Kriterium der Funktionalität wird alternierend zu den übrigen Kriterien der Hauptbewertung evaluiert, da sich jede Applikation hinsichtlich ihres Umfanges unterscheidet. Jede Open Source Software besitzt einen Satz von Funktionalitäten, die erfüllt werden müssen. Dabei handelt es sich um die notwendigen Standardfunktionalitäten, die eine Applikation dieser Art anbieten muss. Manche Applikationen bieten darüber hinaus Zusatzfunktionalitäten an, die sie von übrigen Marktalternativen differenzieren sollen.

Die Bewertung der Funktionalität setzt sich aus zwei Teilen zusammen. Einerseits die **Bewertung der Basisfunktionalitäten**, in der eine Liste von Basisfunktionalitäten aufgestellt wird und diese auf einer 3-Punkte-Skala nach ihrer Relevanz gewichtet werden. Hierbei ist 1 weniger wichtig und 3 besonders wichtig. Sofern eine Basisfunktionalität bei einer Software vorhanden ist, wird diese Punktzahl zur Gesamtsumme addiert. Ist die Basisfunktionalität hingegen nicht vorhanden, wird die Punktzahl von der Gesamtsumme subtrahiert. Dieses Vorgehen bestraft Open Source Software für mangelnde Basisfunktionalitäten. Andererseits die **Bewertung der Zusatzfunktionalitäten**, bei der die Definition verschiedener Zusatzfunktionalitäten und die Gewichtung dieser auf der 3-Punkte-Skala analog zur Bewertung der Basisfunktionalitäten abläuft. Einziger

Unterschied hierbei ist, dass bei mangelnden Zusatzfunktionalitäten innerhalb einer Software die Gewichtung **nicht** von der Gesamtsumme subtrahiert, sondern lediglich mit 0 bewertet wird.

Sei b_i die Bewertung einer Standardfunktionalität, z_j die Bewertung einer Zusatzfunktionalität, s_i die maximal erreichbare Punktzahl einer Standardfunktionalität, n die Anzahl der Standardfunktionalitäten und m die Anzahl der Zusatzfunktionalitäten. Dann ergibt sich für den Feature-Score F folgende Formel 6.2 mit ihrer Wertemenge W_F:

$$F = \frac{\sum_{i=1}^{n} b_i + \sum_{j=1}^{m} z_j}{\sum_{i=1}^{n} s_i} \qquad W_F \longmapsto]-\infty;\ \infty[\tag{6.2}$$

Durch die Einbeziehung von Zusatzfunktionalitäten können hierbei Werte über 100% und unter 0% erreicht werden. Tabelle 6.1 zeigt hierbei das endgültige Bewertungsschema des Kriteriums der Funktionalität.

Punktzahl	Beschreibung	Prozentintervall
1	nicht akzeptabel	weniger als 65%
2	schlecht	$65\% - 80\%$
3	akzeptabel	$80\% - 90\%$
4	sehr gut	$90\% - 96\%$
5	exzellent	mehr als 96%

Tab. 6.1: Bewertungsschema der Funktionalitäten angelehnt an: [Ope05]

Nutzbarkeit

Im Rahmen der Nutzbarkeit wird bewertet, wie benutzerfreundlich und intuitiv die Oberfläche der Software ist. Hierbei wird geprüft, inwiefern es dem Benutzer leicht fällt die Basis- und Zusatzfunktionalitäten zu bedienen. Außerdem fallen die Installations-, Konfigurierungs- und Wartungszeiten in diese Kategorie, diese sollten nach Möglichkeit sehr gering sein. Daraus ergeben sich folgende Metriken:

Metrik 1.1: Endanwender UI-Erfahrung

Metrik 1.2: Zeit für die Installation der Komponenten, die die OSS voraussetzt

Metrik 1.3: Zeit für eine initiale Installation oder Konfiguration

Qualität

Die Qualität der Software wird auf Basis verschiedener Metriken zur Quellcode-, Design- und Testqualität bewertet. Zusätzlich spielt die Vollständigkeit und Fehlerfreiheit der

Software eine tragende Rolle. Damit ergeben sich folgende drei Metriken:

Metrik 2.1: Anzahl der Updates und Patches in den letzten 12 Monaten

Metrik 2.2: Anzahl der offenen Bugs in den letzten 6 Monaten

Metrik 2.3: Prozentzahl der behobenen Fehler

Sicherheit

Das Sicherheitskriterium befasst sich mit den möglichen Sicherheitslücken der Software. Mittels verschiedener Metriken wird hier versucht festzustellen, in welchem Umfang das Projekt anfällig für Sicherheitslücken ist und wie mit bestehenden Sicherheitslücken umgegangen wird. Hierzu wurden ebenfalls drei Metriken aufgestellt:

Metrik 3.1: Anzahl der kritischen Sicherheitslücken in den letzten 6 Monaten

Metrik 3.2: Anzahl der immer noch offenen Sicherheitslücken

Metrik 3.3: Gibt es noch weitere Informationen bezüglich der Sicherheit? (Website, Wiki, etc.)

Performanz

Die Performanz einer Open Source Software trifft Aussagen über die Geschwindigkeit verschiedener Operationen. Zusätzlich wird hierbei überprüft, ob Optimierungsmöglichkeiten innerhalb der Software vorhanden sind und inwiefern diese konfigurierbar sind. Zu diesem Zweck wurden zwei Metriken aufgestellt:

Metrik 4.1: Performanz- und Testberichte verfügbar?

Metrik 4.2: Performanzverbesserung und -konfiguration möglich?

Skalierbarkeit

Im Rahmen der Skalierbarkeit wird bewertet, inwiefern die Software in umfassenderen Umgebungen skaliert. Hierbei wird besonders darauf geachtet, ob die Software bereits in der Praxis getestet und skaliert wurde. Außerdem spielen Aspekte wie Thread-Sicherheit oder Clustering eine Bedeutung. Um dieses Kriterium effektiv bewerten zu können, wurden zwei Metriken aufgestellt:

Metrik 5.1: Referenz-Implementierung

Metrik 5.2: Zur Skalierbarkeit entworfen?

Architektur

Das Architekturkriterium thematisiert die Modularität, Portabilität, Flexibilität, Erweiterbarkeit, Offenheit und Integration der Software. Hierzu werden die Erweiterbarkeit durch Drittanbieter Plug-ins und eine öffentliche API bewertet. Dies ergibt folgende Metriken:

Metrik 6.1: Sind Drittanbieter Plug-Ins vorhanden?

Metrik 6.2: Öffentliche API oder externe Dienste vorhanden?

Support

Der Support einer OSS wird einerseits definiert durch die Qualität des professionellen Supportes, welcher sich im optimalen Fall in Installations-, Problemlösungs- und Integrationssupport gliedert. Andererseits durch das Volumen der Hauptmailingliste, welche die erste Instanz des kostenlosen Supports darstellt. Dadurch ergeben sich folgende Metriken:

Metrik 7.1: Qualität des professionellen Supports

Metrik 7.2: Durchschnittliches Volumen der Hauptmailingliste in den letzten 6 Monaten

Dokumentation

Das Dokumentationskriterium wird anhand der verschiedenen Dokumentationsformen und deren Qualität bewertet. Ein gutes Dokumentationskonzept sollte mehrere Dokumentationsformate für verschiedene Nutzergruppen beinhalten. Außerdem wird die offizielle Dokumentation oftmals durch explizite Benutzerbeiträge positiv ergänzt. Daraus ergeben sich folgende zwei Metriken:

Metrik 8.1: Existenz verschiedener Dokumentationsformen

Metrik 8.2: Framework für Benutzerbeträge

Annahme

Die Annahme innerhalb des Marktes, der Industrie und der Community wird an den veröffentlichten Büchern und passenden Referenz-Implementierungen festgemacht. Es ist wichtig eine standardisierte Vorgehensweise beim Zählen der Bücher zu definieren, in diesem Fall werden Bücher gezählt, die über *Amazon.com* gefunden werden. Diese Plattform wurde als repräsentativ angesehen, da Amazon rund 28 Millionen Bücher anbietet[1] [Sel17]. Hier ergeben sich folgende zwei Metriken:

[1] Stand: Ende 2016

Metrik 9.1: Wie viele Bücher lassen sich auf Amazon.com finden?

Metrik 9.2: Referenz-Implementierung

Community

Da die Community ein wichtiger Bestandteil jeder Open Source Software ist, wird anhand dieses Kriteriums die Aktivität und die Lebhaftigkeit der Community bewertet. Hierbei werden vor allem die erste Anlaufstelle der Community, die Hauptmailingliste, und einzigartige Code-Beitragende mit einbezogen. Deswegen ergeben sich folgende zwei Metriken:

Metrik 10.1: Durchschnittliches Volumen der Hauptmailingliste in den letzten 6 Monaten

Metrik 10.2: Anzahl der einzigartigen Code-Beitragenden in den letzten 6 Monaten

Professionalität

Der Grad der Professionalität innerhalb des Entwicklungsprozesses und der generellen Projektorganisation wird innerhalb dieses Kriteriums festgehalten. Hierzu werden der Projekttreiber, der das Projektmanagement und Ressourcenverteilung übernimmt, und die Aufnahmeschwierigkeit in das Kernentwicklerteam herangezogen. Dies ergibt die folgenden Metriken:

Metrik 11.1: Projekttreiber

Metrik 11.2: Schwierigkeit, um in das Kernentwicklerteam aufgenommen zu werden

6.4 Kriteriengewichtung

Nach den Phasen, die in Kapitel 6.2 beschrieben wurden, sollen innerhalb *Phase 2 - Bewertung des Einsatzumfeldes* die Kriterien und Metriken nach ihrer Relevanz geordnet und ihnen jeweils Prozentzahlen zugeordnet werden. Diese Aufteilung und die passenden Begründungen werden in diesem Unterkapitel näher erläutert.

6.4.1 Gewichtung der Kriterien für die Hauptbewertung

Aus den zwölf Kategorien, die innerhalb des BRR-Models eingeführt werden, sollen sieben oder weniger Kategorien ausgewählt werden, die gewichtet werden. Tabelle 6.2 stellt hierbei die Priorisierung der einzelnen Kriterien dar, wobei 1 die beste Platzierung und 12 die schlechteste Platzierung ist.

#	Kriterium	Gewichtung
1	Performanz	20%
2	Skalierbarkeit	20%
3	Nutzbarkeit	15%
4	Dokumentation	15%
5	Funktionalität	10%
6	Qualität	10%
7	Sicherheit	10%
8	Support	0%
9	Architektur	0%
10	Annahme	0%
11	Community	0%
12	Professionalität	0%

Tab. 6.2: Gewichtung der Kriterien für die Hauptbewertung

Die Kriterien *„Support“*, *„Architektur“*, *„Annahme“*, *„Community“* und *„Professionalität“* werden aus anwendungsbezogenen Gründen innerhalb der Bewertung nicht berücksichtigt, da sie weniger relevant für NewSQL Datenbanken sind. Aus diesem Grund werden sie mit 0% bewertet. Damit ergeben sich *„Performanz“*, *„Skalierbarkeit“*, *„Nutzbarkeit“*, *„Dokumentation“*, *„Funktionalität“*, *„Qualität“* und *„Sicherheit“* als die ausschlaggebenden Kriterien bei der Bewertung.

Besonders bei der Bewertung von NewSQL Datenbanken spielen die Performanz und die Skalierbarkeit eine wichtige Rolle. Aus diesem Grund wurden beide mit 20% bewertet.

Sowohl die Nutzbarkeit, als auch die Dokumentation, wurden mit 15% bewertet. Die Dokumentation eines Open Source Projektes unterstützt die Unternehmen maßgeblich in der Implementierung und der Verwendung der Software. Ist diese nur teilweise oder gar nicht vorhanden, so erhöht sich der Zeitaufwand der Implementierung und der Einarbeitung auf ein Vielfaches [Dou08]. Die Nutzbarkeit einer Software spielt hierbei ebenfalls eine wichtige Rolle, da besonders die Konfiguration und das Aufsetzen der Lösung für NewSQL Datenbanken relevant sind.

Die Funktionalität, Qualität und die Sicherheit wurden abschließend mit 10% bewertet, da verschiedene Funktionalitäten von einer NewSQL Datenbank unterstützt werden müssen, damit diese als funktionsfähige Datenbank im kommerziellen Umfeld eingesetzt werden kann. Im Bezug auf kommerzielle Nutzung ist die Sicherheit einer NewSQL Datenbank essentiell und ein wichtiger Aspekt, der hier ebenfalls berücksichtigt werden sollte. Auch die Qualität einer NewSQL Datenbank bezüglich bestehender Bugs und der

damit verbundenen Behebungsrate sind maßgeblich für die Nutzung im kommerziellen Umfeld.

6.4.2 Gewichtung der Basis- und Zusatzfunktionalitäten

#	Funktion	Kategorie	Gewichtung
1	Tabelle anlegen	DDL	3
2	Tabelle löschen	DDL	3
3	Tabelle editieren	DDL	3
4	Datenbank editieren	DDL	3
5	Datenbank erstellen	DDL	3
6	Datenbank löschen	DDL	3
7	Index erstellen	DDL	2
8	Index löschen	DDL	2
9	Ansicht erstellen	DDL	2
10	Ansicht löschen	DDL	2
11	Datensatz erstellen	DML	3
12	Datensatz löschen	DML	3
13	Datensatz editieren	DML	3
14	Datensatz abrufen	DQL	3
15	Rolle erstellen	DCL	2
16	Rolle löschen	DCL	2
17	Berechtigung geben	DCL	2
18	Berechtigung entfernen	DCL	2

Tab. 6.3: Gewichtung der Basisfunktionalitäten

Die hier aufgezeigten Funktionalitäten werden für alle betrachteten NewSQL Datenbanken bewertet. Tabelle 6.3 zeigt hierbei die Basisfunktionalitäten und Tabelle 6.4 die Zusatzfunktionalitäten. Die Basisfunktionalitäten setzen sich aus den elementaren Operationen zusammen, die jede Datenbank unterstützen muss. Hierbei wird nochmals in die Kategorien Data Description Language (DDL), Data Manipulation Language (DML), Data Control Language (DCL) und Data Query Language (DQL) unterschieden. Die Zusatzfunktionen sind hierbei Funktionen, die von einem guten DBMS implementiert werden.

#	Funktion	Gewichtung
1	Replikation	3
2	Main Memory Storage	2
3	Partitionierung	2
4	Crash Recovery	3
5	Metadaten	2
6	Datenbank Constraints	3
7	Datentypen	2
8	Concurrency Control	3

Tab. 6.4: Gewichtung der Zusatzfunktionalitäten

6.4.3 Gewichtung der Nutzbarkeitsmetriken

Die in Kapitel 6.3.2 aufgestellten Metriken zur Nutzbarkeit werden, wie in Tabelle 6.5 gezeigt, bewertet. Hierbei ist Metrik 1.3 - Zeit für eine initiale Installation oder Konfiguration der wichtigste Aspekt innerhalb dieser Kategorie. Aus diesem Grund wurde diese auch am stärksten gewichtet, während die übrigen zwei Metriken schwächer bewertet wurden. Metrik 1.1 - Endanwender UI-Erfahrung und Metrik 1.2 - Zeit für die Installation der Komponenten, die die OSS voraussetzt wurden schwächer bewertet, da sie innerhalb NewSQL Datenbanken eine unrelevantere Rolle spielen.

#	Metrik	Gewichtung
1	1.1: Endanwender UI-Erfahrung	40%
2	1.2: Zeit für die Installation der Komponenten, die die OSS voraussetzt	10%
3	1.3: Zeit für eine initiale Installation oder Konfiguration	50%

Tab. 6.5: Gewichtung der Nutzbarkeitsmetriken

6.4.4 Gewichtung der Qualitätsmetriken

Bei den Qualitätsmetriken wurden alle drei Metriken mit einer ähnlichen Gewichtung versehen. Lediglich die Prozentzahl der behobenen Fehler wurde aufgrund der Tatsache, dass sich daran die Effektivität des Entwicklungsteams bewerten lässt, mit 40% etwas höher als die übrigen Metriken angesiedelt. Diese Aufteilung der Gewichtungen wird in Tabelle 6.6 illustriert.

#	Metrik	Gewichtung
1	2.1: Anzahl der Updates / Patches in den letzten 12 Monaten	30%
2	2.2: Anzahl der offenen Bugs in den letzten 6 Monaten	30%
3	2.3: Prozentzahl der behobenen Fehler	40%

Tab. 6.6: Gewichtung der Qualitätsmetriken

6.4.5 Gewichtung der Sicherheitsmetriken

Sowohl Metrik 3.1 - Anzahl der kritischen Sicherheitslücken in den letzten 6 Monaten, als auch Metrik 3.2 - Anzahl der immer noch offenen Sicherheitslücken, stellen hierbei einen wichtigen Aspekt für die Gesamtsicherheit dar. Deswegen wurden beide gleichermaßen hoch bewertet. Metrik 3.3 - Gibt es noch weitere Informationen bezüglich der Sicherheit? wurde weniger stark gewichtet, da dies nicht essentiell für die Sicherheit einer Software ist. Die Gewichtung der einzelnen Metriken wird in Tabelle 6.7 aufgezeigt.

#	Metrik	Gewichtung
1	3.1: Anzahl der kritischen Sicherheitslücken in den letzten 6 Monaten	40%
2	3.2: Anzahl der immer noch offenen Sicherheitslücken	40%
3	3.3: Gibt es noch weitere Informationen bezüglich der Sicherheit?	20%

Tab. 6.7: Gewichtung der Sicherheitsmetriken

6.4.6 Gewichtung der Performanzmetriken

Die Gewichtung der Performanzmetriken ist mit 20% Unterschied zwischen den beiden Metriken nahezu ausgeglichen. Sofern keine eigene Performanzevaluation durchgeführt wird, ist es etwas wichtiger, dass ausführliche und vollständige Performanz- und Testberichte verfügbar sind. Tabelle 6.8 zeigt die prozentuale Aufteilung der beiden Metriken.

#	Metrik	Gewichtung
1	4.1: Performanz- und Testberichte verfügbar?	60%
2	4.2: Performanzverbesserung und -konfiguration möglich?	40%

Tab. 6.8: Gewichtung der Performanzmetriken

6.4.7 Gewichtung der Skalierbarkeitsmetriken

Beide Metriken im Rahmen der Skalierbarkeit besitzen eine hohe Relevanz. Ist eine Anwendung nicht zur Skalierbarkeit entworfen, können in der Praxis bei vielen Benutzern kritische Probleme auftreten. Solche Probleme könnten beispielsweise Prozessorauslastung oder enormer Speicherverbrauch sein. Eine passende Referenz-Implementierung sollte zeigen, dass diese Probleme nicht in der Praxis auftreten. Aus diesem Grund wurde Metrik 5.1 - Referenz-Implementierung etwas höher gewichtet. Metrik 5.2 - Zur Skalierbarkeit entworfen? sollte für alle NewSQL Datenbanken standardmäßig gegeben sein, deswegen wurde diese Metrik deutlich geringer bewertet. Tabelle 6.9 illustriert die Gewichtung der beiden Metriken.

#	Metrik	Gewichtung
1	5.1: Referenz-Implementierung	70%
2	5.2: Zur Skalierbarkeit entworfen?	30%

Tab. 6.9: Gewichtung der Skalierbarkeitsmetriken

6.4.8 Gewichtung der Dokumentationsmetriken

Die Metrik 8.1 - Existenz verschiedener Dokumentationsformen wurde mit 75% bewertet, da es sich dabei um die maßgebende Dimension der Dokumentationsqualität handelt. Ohne eine entsprechende Dokumentation ist es schwierig eine passende Konfiguration und eine richtige Verwendung sicher zu stellen. Metrik 8.2 - Framework für Benutzerbeträge wurde hierbei als zweitrangig eingestuft, da von Benutzern erstellte Tutorials oder sonstige Beiträge, eine ausführliche Dokumentation nicht ersetzen, sondern lediglich um explizite, anwendungsbezogene Beispiele ergänzen sollten. Tabelle 6.10 illustriert diese Gewichtung der beiden Metriken.

#	Metrik	Gewichtung
1	8.1: Existenz verschiedener Dokumentationsformen	75%
2	8.2: Framework für Benutzerbeträge	25%

Tab. 6.10: Gewichtung der Dokumentationsmetriken

Kapitel 7

7 Umsetzung

Nachdem eine methodische Vorgehensweise für die Bewertung in Kapitel 6 aufgestellt wurde, sollen in diesem Kapitel verschiedene NewSQL Datenbanken vorgestellt und anschließend evaluiert werden. Hierzu werden sowohl die Kriterien zur Vorbewertung, als auch die Kriterien zur Hauptbewertung mit einbezogen und bilden damit die Basis der nachfolgenden Bewertung. Ziel ist es die führenden NewSQL Systeme vergleichend gegenüber zu stellen und die Vor- bzw. Nachteile der verschiedenen Alternativen herauszuarbeiten.

7.1 Vorstellung verschiedener NewSQL Datenbankmanagementsysteme

Im Rahmen dieser Arbeit wurden, unter Berücksichtigung der Vorbedingungen aus Kapitel 6.3.1, die NewSQL DBMS „$NuoDB$“, „$VoltDB$“ und „$CockroachDB$“ aufgrund ihrer Popularität und dem generellen Webauftritt ausgewählt.

7.1.1 NuoDB

NuoDB ist eine verteilte Datenbankanwendung, welche ACID Transaktionen, SQL und relationale Logik unterstützt [Nuo13]. Im Rahmen der Skalierbarkeit setzt NuoDB auf elastisches SQL, welches bereits in Kapitel 5.4 erläutert wurde. Hierbei ist die verteilte Architektur von NuoDB in eine drei Schichten Architektur eingeteilt [Mur13, Nuo13]. Es wird eine „$Administrative\ Schicht$“, eine „$Transaktionelle\ Schicht$“ und eine „$Speicherungsschicht$“ implementiert [Nuo13]. Die Vorteile dieses Ansatzes wurden ebenfalls in Kapitel 5.4 umrissen.

Die Schichten werden durch einen ausführbaren Prozess definiert, der in zwei Modi gestartet werden kann. Je nachdem, was benötigt wird, kann dieser als Transaction Engine (TE) oder Storage Manager (SM) fungieren. Eine minimale Konfiguration wäre hierbei eine Transaction Engine und ein Storage Manager, welche parallel auf einem Knoten laufen. Sobald eine zusätzliche Transaction Engine oder Storage Manager auf einem neuen Knoten gestartet wird, authentifiziert sich dieser automatisch mit den bestehenden Prozessen. So wird eine durchgehende Verfügbarkeit sichergestellt.

Um die Konsistenz der Transaktionen sicherzustellen wird bei NuoDB Multi-Version Concurrency Control eingesetzt.

Über den zwei Datenbankschichten läuft die Administrative Schicht, die ebenfalls aus einer Menge an Prozessen besteht. Diese Prozesse werden „$Agents$“ genannt und werden auf jedem Knoten ausgeführt, auf dem eine Datenbank aktiv sein könnte [Nuo13]. Dies bereitet den Knoten darauf vor, Datenbankprozesse zu starten und ermöglicht den Zugriff innerhalb der „$Management\ Domain$“. Diese Domain stellt die Menge an bereitgestellten Datenbankknoten dar und erlaubt die Verbindung mehrerer Knoten.

Der Vergleich wird auf Basis der Version „$nuodb\text{-}ce\text{-}3.1.0.0.linux.x86\_64.tar.gz$“ durchgeführt.

7.1.2 VoltDB

Innerhalb dieser Studienarbeit wird VoltDB als zweite NewSQL Datenbank betrachtet. Hierbei handelt es sich um eine In-Memory Datenbank, welche von verschiedenen bekannten Datenbankanalysten entwickelt wurde. Einer diese Datenbankexperten ist Michael Stonebreaker, dessen Ansichten und Papers auch innerhalb dieser Arbeit an mehreren Stellen eingeflossen sind. Laut eigenen Angaben soll VoltDB $50 - 100$ mal schneller als relationale und NoSQL Datenbanksysteme sein [Kar17, Har15, Vol17]. Auch hier werden SQL als Datenmanipulationssprache und ACID Transaktionen unterstützt [Ber15]. VoltDB implementiert Shared-Nothing als Clusteringverfahren und automatisiertes Sharding [Vol17].

Der Vergleich wird auf Basis der Version „$voltdb\text{-}community\text{-}8.0.tar$“ durchgeführt.

7.1.3 CockroachDB

Das letzte NewSQL Datenbanksystem, welches im Rahmen dieser Studienarbeit betrachtet wird, ist CockroachDB. Das Unternehmen „$Cockroach\ Labs$“, welches für die Entwicklung dieses Datenbankmanagementsystems zuständig ist, wurde im Jahr 2015, mit dem Ziel Daten einfach zu verarbeiten, gegründet [Kar17]. Ihren Sitz hat Cockroach Labs in New York City.

CockroachDB stellt eine Reihe an Funktionen bereit, wie beispielsweise SQL, automatisiertes Skalieren oder stark konsistente Replika. Es wurde außerdem auf Basis einer Schlüssel-Wert Speicherung ($engl.$ key-value store) implementiert.

Der Vergleich wird auf Basis der Version „$cockroach\text{-}v2.0.0.linux\text{-}amd64.tgz$“ durchgeführt.

7.2 Vergleich der implementierten NewSQL Datenbanken

In diesem Kapitel werden die vorgestellten NewSQL Datenbanksysteme aus dem vorangegangenen Unterkapitel anhand des Business Readiness Ratings aus Kapitel 6 bewertet.

7.2.1 Bewertung der Funktionalitätsmetriken

Durch die vollständige Implementierung von SQL in allen drei NewSQL Datenbanksystemen werden alle Basisfunktionalitäten, sowie zwei der Zusatzfunktionalitäten (Datenbank Constraints und Datentypen), abgedeckt und unterstützt. Aus diesem Grund erhalten sowohl NuoDB, VoltDB, als auch CockroachDB die volle Punktzahl von $46/46$ bei den Basisfunktionalitäten.

Funktion (Punktzahl)	NuoDB	VoltDB	CockroachDB
Replikation (3)	Stark konsistent + passiv	Stark konsistent + aktiv	Stark konsistent + passiv
Main Memory Storage (2)	Ja	Ja	Nein
Partitionierung (2)	Ja	Ja	Ja
Crash Recovery (3)	Ja	Ja	Ja
Metadaten (2)	Ja	Ja	Ja
Datenbank Constraints (3)	Ja	Ja	Ja
Datentypen (2)	Ja	Ja	Ja
Concurrency Control (3)	MVCC	TO	MVCC
Summe (max.: 20)	**20**	**20**	**18**

Tab. 7.1: Bewertung der Zusatzfunktionalitäten auf Basis von: [Vol18, Nuo18, Coc18]

In Tabelle 7.1 werden die Zusatzfunktionalitäten der vorgestellten NewSQL Datenbanksysteme bewertet und gegenübergestellt. Wichtig ist hier, dass Begrifflichkeiten und Abkürzungen aus Kapitel 5 verwendet werden. Da alle drei führende NewSQL Datenbanken sind, werden nahezu alle Funktionalitäten unterstützt. Sie unterscheiden sich lediglich in ihren Replika, Speicherzugriffen und Concurrency Control Schemata. Mit den erfüllten Funktionalitäten werden folgende Feature-Scores erreicht:

NuoDB erreicht einen Feature-Score von $143,48\%$ und damit eine Punktzahl in der Kategorie Funktionalität von 5

VoltDB erreicht ebenfalls einen Feature-Score von $143,48\%$ und damit eine Punktzahl in der Kategorie Funktionalität von 5

CockroachDB erreicht einen Feature-Score von $139,13\%$ und damit eine Punktzahl in der Kategorie Funktionalität von 5

7.2.2 Bewertung der Nutzbarkeitsmetriken

Tabelle 7.2 zeigt die Bewertung der NewSQL Datenbanken auf Basis der Nutzbarkeitsmetriken. CocroachDB schneidet innerhalb dieses Kriteriums als Erster, VoltDB als Zweiter und NuoDB als Letzter ab.

Im Rahmen der Nutzbarkeit wurde die Metrik 1.1 - Endanwender UI-Erfahrung bei NuoDB aufgrund der nicht intuitiven Weboberfläche und der notwendigen Konfiguration am schlechtesten bewertet. Die Weboberfläche von VoltDB stellt wichtige Funktionen bereit und überzeugt mit einer intuitiven Organisation. Am Besten ist jedoch die Weboberfläche von CockroachDB. Diese ist herausstehend gut gestaltet und erleichtert dem Nutzer alle wichtigen Funktionen zu finden.

Metrik	NuoDB	VoltDB	CockroachDB
1.1: UI-Erfahrung	2	4	5
1.2: Installation der Komponenten, die die OSS voraussetzt	4	4	5
1.3: Initiale Installation oder Konfiguration	5	5	5
Summe (gewichtet)	**3,7**	**4,5**	**5**

Tab. 7.2: Bewertung der Nutzbarkeitsmetriken

Innerhalb der Installation wird bei NuoDB mindestens die Java JRE 1.7 oder 1.8 und bei NuoDB Java 7 oder 8, NTP und Python 2.6 benötigt. Deswegen erhalte diese geringfügige Abzüge innerhalb Metrik 1.2 - Installation der Komponenten, die die OSS voraussetzt. Bei der initialen Installation und Konfiguration bekommen alle NewSQL Datenbanken die volle Punktzahl von 5 Punkten, da keine weiteren Konfigurationsschritte notwendig sind und die Installation reibungslos abläuft.

7.2.3 Bewertung der Qualitätsmetriken

Die Ergebnisse der NewSQL Datenbanksysteme innerhalb des Kriteriums der Qualität werden in Tabelle 7.3 illustriert. Innerhalb Metrik 2.1 - Anzahl der Patches in den letzten 12 Monaten schneidet NuoDB am schlechtesten ab, da diese im aufgestellten Zeitraum nur zwei Releases mit vergleichsweise mittlerem Umfang veröffentlicht haben. CockroachDB hat hier 19 Patches bereitgestellt und VoltDB stellt wöchentliche Releases zur Verfügung. VoltDB hat derzeitig 1198 offene und 3903 geschlossene Bugs bzw. Probleme, dies führt zu einer Prozentzahl behobener Fehler von $76,51\%$. CockroachDB hat derzeitig 1524 offene und 10332 geschlossene Fehler mit einer Prozentzahl behobener Fehler von $87,15\%$. NuoDB stellt keinen Zugriff auf die Bugs bzw. Fehler bereit, da die entsprechende Seite nicht erreichbar ist, somit müssen Metrik 2.2 - Anzahl der

offenen Bugs in den letzten 6 Monate und Metrik 2.3 - Prozentzahl der behobenen Fehler mit der schlechtesten Note von 1 bewertet werden.

Metrik	NuoDB	VoltDB	CockroachDB
2.1: Anzahl der Patches in den letzten 12 Monaten	3	5	4
2.2: Anzahl der offenen Bugs in den letzten 6 Monaten	1	4	4
2.3: Prozentzahl der behobenen Fehler	1	3	4
Summe (gewichtet)	**1,6**	**3,9**	**4**

Tab. 7.3: Bewertung der Qualitätsmetriken

7.2.4 Bewertung der Sicherheitsmetriken

Im Rahmen der Sicherheitskategorie wurde CockroachDB am Besten bewertet, da laut eigenen Angaben keine offenen Sicherheitslücken bestehen und in den letzten sechs Monaten keine kritischen Sicherheitslücken entstanden sind. Außerdem stellt CockroachDB eine E-Mail Adresse für die Einsendung bestehender Sicherheitslücken und einen Link zu behobenen Sicherheitslücken bereit. VoltDB hat acht Bugs in den letzten sechs Monaten als „*Critical*“ oder „*Blocker*“ klassifiziert, von denen zwei noch offen und einer in Bearbeitung ist. Sie stellen außerdem eine sehr gute Dokumentation zu Themen wie Authentifizierung, Rollen, Benutzerrechten oder Verschlüsselungen bereit. NuoDB besitzt leider keine Angaben bezüglich Metrik 3.1 - Kritischen Sicherheitslücken in den letzten 6 Monaten oder Metrik 3.2 - Anzahl der immer noch offenen Sicherheitslücken bereit. Aus diesem Grund müssen beide Metriken mit der Punktzahl 1 bewertet werden. Auch weitere Sicherheitsinformationen sind kaum bis gar nicht vorhanden.

Metrik	NuoDB	VoltDB	CockroachDB
3.1: Kritischen Sicherheitslücken in den letzten 6 Monaten	1	4	5
3.2: Anzahl der immer noch offenen Sicherheitslücken	1	4	5
3.3: Gibt es noch weitere Sicherheitsinformationen?	2	5	4
Summe (gewichtet)	**1,2**	**4,2**	**4,8**

Tab. 7.4: Bewertung der Sicherheitsmetriken

Tabelle 7.4 fasst die Bewertungen innerhalb der einzelnen Metriken übersichtlich zusammen und zeigt die gewichtete Summe der einzelnen Metriken für die betrachteten Datenbankmanagementsysteme innerhalb der Sicherheitskategorie.

7.2.5 Bewertung der Performanzmetriken

Da im Rahmen dieser Studienarbeit keine ausführliche eigene Performanzevaluation möglich ist, muss innerhalb dieser Kategorie auf verschiedene Literatur, sowie Performanz- und Testberichte verwiesen werden. Dies dient generell dazu, einen groben Überblick im Vergleich der verschiedenen Datenbankmöglichkeiten zu erhalten. Hierzu wird sich vorallem auf das Whitepaper „*Performance Evaluation of NewSQL Databases*“ [Kar17] und den Artikel „*Benchmarking Google Cloud Spanner, CockroachDB, and NuoDB*“ [Bul17] bezogen.

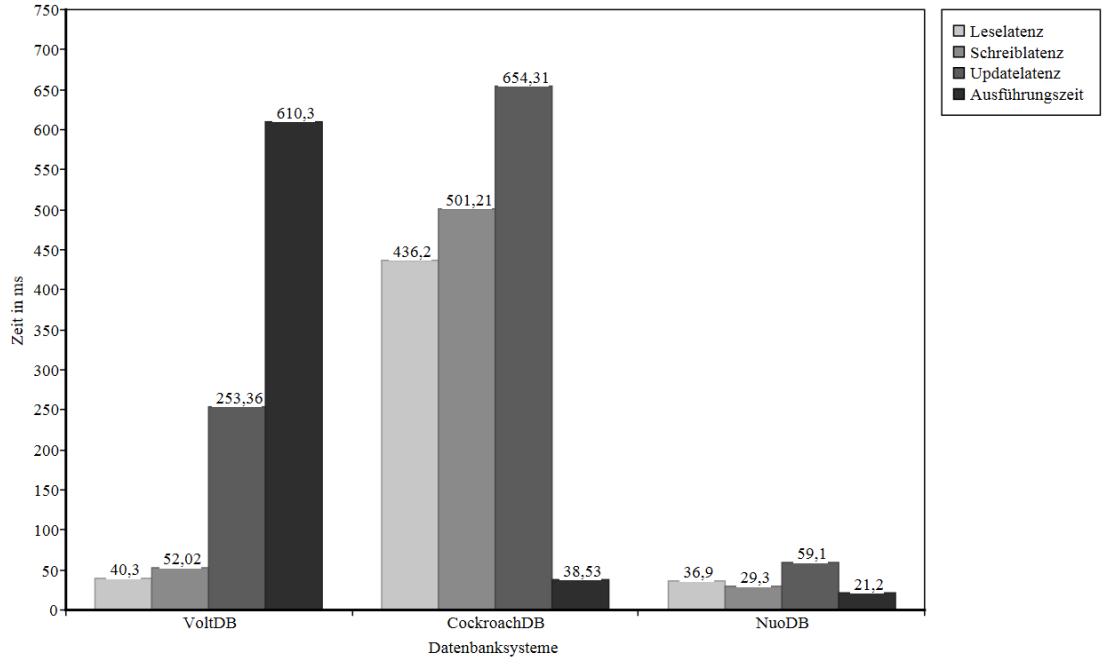


Abb. 7.1: Performanzevaluation der NewSQL Datenbanken angelehnt an: [Kar17]

Alle betrachteten Performanzevaluationen kommen auf ähnliche Ergebnisse in den Kategorien „*Leselatenzen*“, „*Schreiblatenzen*“, „*Updatelatenzen*“ und „*Ausführungszeiten*“. Abbildung 7.1 zeigt beispielsweise diese Ergebnisse der Kategorien aus [Kar17]. Hierbei ist deutlich erkennbar, dass NuoDB die beste Performanz aufweist, während VoltDB und CockroachDB deutliche Latenzen aufzeigen. Mit diesem Hintergrund wurde in Tabelle 7.5 NuoDB mit 5 und CockroachDB, als auch VoltDB mit 3 bewertet.

Eine Performanzverbesserung und -konfiguration ist in den verschiedenen Datenbankmanagementsystemen in unterschiedlichem Umfang dokumentiert und umsetzbar. Für NuoDB werden in der Dokumentation verschiedene Optionen der Performanzverbesserung beschrieben und näher erläutert. Diese Optionen fallen unter anderem in die Bereiche „*Systemkonfiguration*“, „*NuoDB-Konfiguration*“ oder „*Datenbank Design*“. Hierfür werden verschiedene Möglichkeiten und Schritte aufgezeigt, die umgesetzt werden können, um die Performanz zu verbessern. VoltDB zeigt diesbezüglich Ansätze der Performanzverbesserung in den Bereichen „*Datenbank Schema*“, „*Stored Procedures*“

und „*Applikationsdesign*“. Auch CockroachDB verfolgt hierzu einige Methoden, welche in die selbe Richtung gehen.

Metrik	NuoDB	VoltDB	CockroachDB
4.1: Performanz- und Testberichte verfügbar?	5	3	3
4.2: Performanzverbesserung und -konfiguration möglich?	5	4	4
Summe (gewichtet)	**5**	**3,4**	**3,4**

Tab. 7.5: Bewertung der Performanzmetriken

Tabelle 7.5 zeigt hier nochmals alle Bewertungen der einzelnen Metriken und die abschließende gewichtete Summe der Datenbankmanagementsysteme in der Kategorie der Performanz.

7.2.6 Bewertung der Skalierbarkeitsmetriken

Alle drei Datenbanksysteme besitzen viele Referenz-Implementierungen in den verschiedensten Unternehmen. So hat NuoDB Kunden, wie beispielsweise „*Forbes*“, „*Informix*“, „*Movemedical*“ oder „*Wideorbit*“ und noch viele weitere. Insgesamt werden auf der Unternehmenswebsite von NuoDB 15 namentliche Referenz-Unternehmen genannt, in denen NuoDB erfolgreich eingesetzt wird [Nuo18]. Daneben listet VoltDB 62 Unternehmen als Referenz-Kunden auf, unter denen sich Namen wie „*T-Mobile*“, „*Nokia*“, „*Huawei*“, „*Hewlett Packard Enterprise*“ oder „*Mitsubishi Electric*“ wieder finden lassen [Vol18]. CockroachDB besitzt noch die wenigsten Referenz-Implementierungen mit 5 Unternehmen, laut eigenen Angaben [Coc18]. Dabei handelt es sich um „*Baidu*“, „*Kindred*“, „*Tierion*“, „*Heroic Labs*“ und „*Gorgias*“ [Coc18]. Die Bewertungen der $\text{Metrik } 5.1 \text{ - Referenz-Implementierung}$ zeigen sich in Tabelle 7.6.

Metrik	NuoDB	VoltDB	CockroachDB
5.1: Referenz-Implementierung	4	5	3
5.2: Zur Skalierbarkeit entworfen?	5	5	5
Summe (gewichtet)	**4,3**	**5**	**3,6**

Tab. 7.6: Bewertung der Skalierbarkeitsmetriken

Da es sich bei den betrachteten Datenbanksystemen um NewSQL Datenbanksystemen handelt, ist davon auszugehen, dass alle drei zur Skalierbarkeit entworfen wurden. So skaliert NuoDB durch elastisches SQL und VoltD, sowie CockroachDB durch eine Shared-Nothing Architektur. Dies wurde bereits in vorangegangenen Kapiteln näher

beleuchtet. Aus diesem Grund wurden alle Datenbanksysteme mit der besten Punktzahl von 5 innerhalb der $\mathrm{Metrik}\ 5.2$ - $\mathrm{Zur\ Skalierbarkeit\ entworfen?}$ bewertet.

Insgesamt schließt damit VoltDB mit voller Punkzahl von 5, NuoDB mit $4,3$ und CockroachDB mit einer Punktzahl von $3,6$ innerhalb des Skalierbarkeitskriteriums ab.

7.2.7 Bewertung der Dokumentationsmetriken

Die Dokumentationen sind bei allen drei Datenbankmanagementsystemen mehr als ausreichend vorhanden. Hierbei sticht besonders VoltDB mit einer besonders ausführlichen Dokumentation heraus, die alle gewünschten Bereiche umfassend abdeckt. Trotzdem sind die Dokumentationen von NuoDB und CockroachDB ebenfalls stark ausgeprägt und ausreichend, um beide Datenbankmanagementsysteme im kommerziellen Umfeld zu realisieren.

Außerdem stellt jedes der drei Datenbankmanagementsysteme ein Framework für Benutzereinträge zur Verfügung. Hier können Nutzer ihre Probleme oder Fragen rund um die Architektur, die Installation, verschiedene Schnittstellen oder die Implementierung stellen. Generell unterscheiden sich diese kaum bis gar nicht voneinander, deswegen erhalten alle betrachteten Datenbanksysteme die volle Punktzahl. Die Bewertungen der einzelnen Metriken innerhalb der Dokumentation werden in Tabelle 7.7 dargestellt.

Metrik	NuoDB	VoltDB	CockroachDB
8.1: Existenz verschiedener Dokumentationsformen	4	5	4
8.2: Framework für Benutzerbeträge	5	5	5
Summe (gewichtet)	**4,25**	**5**	**4,25**

Tab. 7.7: Bewertung der Dokumentationsmetriken

Abschließend zeigt sich, dass innerhalb des Dokumentationskriteriums alle drei NewSQL Datenbanken sehr gute Ergebnisse erzielen und so mit $4,25$ für NuoDB und CockroachDB und mit 5 für VoltDB mehr als zufrieden stellende Ergebnisse liefern.

7.2.8 Zusammenfassende Übersicht der Ergebnisse

Zusammenfassend zeigt sich in Tabelle 7.8, dass alle betrachteten Datenbankmanagementsysteme relativ nah beieinander in den Endergebnissen des Business Readiness Ratings liegen. Trotzdem kristallisiert sich VoltDB als führende NewSQL Datenbank heraus. Dies wurde auch von vielen anderen Unternehemen so angesehen, wie bereits in Kapitel 7.2.6 aufgezeigt. NuoDB besitzt besonders schlechte Werte innerhalb der

Kategorien „*Qualität*“ und „*Skalierbarkeit*“. Dies ist hauptsächlich durch nicht bewertbare Metriken begründet, die aufgrund von Serverproblemen seitens NuoDB auftreten. Sobald diese erfolgreich behoben wurde, würde NuoDB sich CockroachDB annähern und diese eventuell sogar überholen. Solange diese allerdings anhalten, muss NuoDB diese schlechte Bewertungen in Kauf nehmen.

Kategorie	NuoDB	VoltDB	CockroachDB
Funktionalität (10%)	5	5	5
Nutzbarkeit (15%)	3,7	4,5	5
Qualität (10%)	1,6	3,9	4
Sicherheit (10%)	1,2	4,2	4,8
Performanz (20%)	5	3,4	3,4
Skalierbarkeit (20%)	4,3	5	3,6
Dokumentation (15%)	4,25	5	4,25
Summe (gewichtet)	**3,83**	**4,67**	**4,17**

Tab. 7.8: Zusammenfassung des Open Source Business Readiness Ratings

Für eine kommerzielle Implementierung einer NewSQL Datenbank, würde sich aufgrund dieser Ergebnisse VoltDB als flächendeckend beste Möglichkeit anbieten. Falls allerdings besonderen Wert auf eine herausstehende Performanz oder Nutzbarkeit gelegt wird, sollte anhand dieser Ergebnisse neu evaluiert werden und die passende Lösung ausgewählt werden. VoltDB besitzt nämlich über alle Kategorien gute bis sehr gute Werte, allerdings werden sie in manchen Kategorien von den Konkurrenzsystemen geschlagen.

7.3 Implementierung einer prototypischen NewSQL Lösung

Im Rahmen dieser Studienarbeit soll ein NewSQL Datenbanksystem prototypisch auf einem USB-Stick mittels virtuellen Maschinen (VMs) implementiert werden, sodass diese Implementierung als Plug & Play Prototyp die Funktionsweise und die Möglichkeiten einer modernen NewSQL Datenbank veranschaulicht. Als Datenbanksystem wurde hierzu CockroachDB ausgewählt, aufgrund der Bestnote in der Kategorie Nutzbarkeit. Dadurch soll sichergestellt werden, dass Funktionalitäten wie Replika oder das Hinzufügen und Entfernen von Knoten anschaulich visualisiert sind.

Zunächst sollte ein Knoten der Datenbank durch jeweils eine virtuellen Maschine repräsentiert werden. Diese sollten über das jeweils bereitgestellte Netzwerk kommunizieren

und so einen Datenbankcluster formen. Dieses Konzept wurde allerdings aufgrund mehrerer Gründe verworfen:

Inkompatibilität verschiedener Windows Versionen mit dem Host-Only Netzwerk von Virtualbox.

IP-Konfiguration muss bei verschiedenen Netzwerken teilweise neu konfiguriert werden.

Netzwerkadapter von Virtualbox muss auf dem Hostsystem durch die Virtualbox installiert werden.

Alternativ wurde ein lokaler Cluster auf einer virtuellen Maschine aufgesetzt, in der jeweils ein Terminal als ein Knoten fungiert und sich über eine Weboberfläche überwachen lassen. Zugangsdaten für die virtuelle Maschine „$CockroachDB\_Node\_One$“ lauten $user = i15027$, $password = test1234$ und $root = test1234$. Diese lassen sich ebenfalls aus der Datei „$Login.txt$“ auf dem USB-Datenträger im Ordner „$Debian$“ entnehmen.

Um den lokalen Cluster auf der virtuellen Maschine zu starten, muss das passende Bash-Skript „$LocalClusterSetup.bsh$“ im Verzeichnis $/home/i15027/Schreibtisch$ ausgeführt werden. Dieses Bash-Skript wird in Programmlisting 7.1 dargestellt.

Das Bash-Skript fordert den Benutzer auf, die Anzahl der Knoten zu spezifizieren und speichert diese in der Variable $number$. Daraufhin wird der erste Knoten in einem neuen Terminal und anschließend die übrigen Knoten gestartet. Diesen weiteren Knoten werden freie Ports sowie HTTP-Ports zugewiesen und sie werden dem Cluster des ersten Knotens auf Port 26257 hinzugefügt. Der Befehl $unset\ SESSION\_MANAGER$ wird verwendet, da $xterm$ versucht die Umgebungsvariable $\$SESSION\_MANAGER$ zu verwenden und diese Lognachricht nicht auf der Konsole ausgegeben werden soll.

```
1 #!/bin/bash
2 echo 'Please enter number of nodes'
3 read number
4 unset SESSION_MANAGER
5 xterm -e 'cockroach start --insecure --host=localhost' &
6 for ((i = 1; i <= number; i++))
7 do
8 xterm -e "cockroach start --insecure --store="node${i}" --host=localhost
      --port=0 -http-port="80${i}" --join=localhost:26257" &
9 done
```

Listing 7.1: Bash-Skript lokaler Cluster

--insecure Führt den Knoten im unsicheren Modus aus. Ist dies nicht gesetzt, muss *--certs-dir* auf valide Zertifikate gesetzt werden.

--host Spezifiziert den Hostnamen oder IP-Adresse auf den gehört wird, für Kommunikation innerhalb des Clusters.

--store Stellt den Pfad zur Datenhaltung dar.

--port Ist der Port, welcher benutzt werden soll, zur internen oder Client Kommunikation. Sofern ein unbenutzter Port verwendet werden soll, muss 0 übergeben werden.

--http-port Ist der Port für die HTTP-Anfragen der Admin Weboberfläche.

--join Spezifiziert die Adresse, um die Knoten miteinander zu verknüpfen.

Die bereitgestellte Weboberfläche ist unter *localhost:8080* zu erreichen. Hier lassen sich für den Cluster unter anderem die einzelnen Knoten, den Zustand der Replika, Verzögerungen, Aufwand des Clusters und auch die getätigten SQL Anfragen überwachen.

Möchte man nun eine standardmäßige SQL Anfrage im Cluster ausführen, so kann dies an jedem Knoten des Clusters passieren. Um dies darzustellen, kann man ein neues Cockroach SQL Terminal mit dem Kommando *cockroach sql --insecure* öffnen. Dies ermöglicht die Eingabe von typischen SQL Anfragen. Wird zusätzlich noch ein Port mittels *cockroach sql --insecure --port=26258* angegeben, so handelt es sich hierbei um den Knoten, an dem diese SQL Anfrage ausgeführt werden soll. Dieses SQL Terminal lässt sich mit $\backslash q$ wieder beenden.

Solche SQL Anfragen können, wie in Programmlisting 7.2 dargestellt, aussehen. Für weitere Informationen bezüglich der SQL Möglichkeiten sei hier auf die CockroachDB Dokumentation [Coc18] verwiesen. Es besteht außerdem die Möglichkeit, innerhalb des SQL Terminals verschiedene SQL Befehle anzeigen zu lassen:

\h Listet alle verfügbaren SQL Befehle kategorisiert auf.

\hf Listet alle verfügbaren SQL Funktionen in alphabetischer Reihenfolge auf.

\h<Befehl> Zeigt weitere Informationen zu einem bestimmten Befehl an.

\hf<Funktion> Zeigt weitere Informationen zu einer bestimmten SQL Funktion an.

```
1  CREATE DATABASE IF NOT EXISTS bank;
2  DROP DATABASE bank;
3  SHOW DATABASES;
4  CREATE TABLE accounts (id INT PRIMARY KEY, balance DECIMAL);
5  SHOW COLUMNS FROM accounts;
6  DROP TABLE accounts;
7  SHOW TABLES;
8  INSERT INTO accounts (balance, id) VALUES (10000.3, 1);
9  CREATE INDEX balance_idx ON accounts (balance DESC);
10 SHOW INDEX FROM accounts;
11 SELECT id, balance FROM accounts ORDER BY balance DESC;
```

Listing 7.2: Beispiele der CockroachDB SQL Anfragen angelehnt an: [Coc18]

Diese SQL Anfragen liefern Resultate, welche die in Programmlisting 7.3 und Programmlisting 7.4 dargestellte Form besitzen. Programmlisting 7.3 zeigt hierbei alle Spalten der Tabelle *accounts* mit ihren Typen und zusätzlichen Eigenschaften. Während Programmlisting 7.4 das Resultat eines Select-Befehls illustriert.

```
> SHOW COLUMNS FROM accounts;

+---------+---------+-------+---------+-----------+
|  Field  |  Type   | Null  | Default | Indices |
+---------+---------+-------+---------+-----------+
| id      | INT     | false | NULL    | {primary} |
| balance | DECIMAL | true  | NULL    | {}        |
+---------+---------+-------+---------+-----------+
(2 rows)
```

Listing 7.3: Beispiel 1 der CockroachDB SQL Anfragen

Sowohl bei Programmlisting 7.3, als auch bei Programmlisting 7.4, handelt es sich um beispielhafte Ergebnisse zweier SQL Anfragen. Diese sind in dieser Form nur replizierbar, sofern die abgefragten Tabellen und Datensätze zuvor mit entsprechendem SQL angelegt wurden.

```
> SELECT id, balance FROM accounts ORDER BY balance DESC;

+----+---------+
| id | balance |
+----+---------+
|  2 |   25000 |
|  1 | 10000.5 |
|  4 |  9400.1 |
|  3 | 8100.73 |
|  5 | NULL    |
|  6 | NULL    |
+----+---------+
(6 rows)
```

Listing 7.4: Beispiel 2 der CockroachDB SQL Anfragen

Ein Datenbankknoten wird abgeschaltet, indem im jeweiligen Terminal mittels $STRG + C$ das Herunterfahren eingeleitet wird. Dies kann durch erneutes Eingeben beschleunigt werden. Möchte man einen bestimmten Cluster nicht wieder hochfahren, macht es ebenfalls Sinn, die jeweiligen Daten der Knoten zu löschen. Dies kann beispielsweise durch folgenden Kommandozeilenbefehl geschehen: *rm -rf cockroach-data node2 node3*.

Kapitel 8

8 Vergleich zwischen SQL, NoSQL und NewSQL

Nachdem in den vorherigen Kapiteln die Datenbankgenerationen und die generelle Entwicklung hin zu NewSQL Datenbanksystemen, typische Skalierbarkeitsmethoden, sowie verschiedene NewSQL DBMS anhand des BRR bewertet wurden, soll in diesem Kapitel ein Vergleich zwischen SQL, NoSQL und NewSQL erfolgen. Hierzu wird dies explizit an ausgewählten Vertretern der jeweiligen Kategorien geschehen. Diese Vertreter sind hierbei „$MySQL$“ [MyS18], für SQL, „$MongoDB$“ [Mon18], „$Neo4j$“ [Neo18] für die die NoSQL Bewegung und „$VoltDB$“ [Vol18] für die NewSQL Datenbanksysteme [Rud16]. Es wurden zwei Datenbanksysteme für die NoSQL Bewegung ausgewählt, da jeweils eine dokumentenbasierte Datenbank (MongoDB) und eine Graph-Datenbank (Neo4j) verglichen werden sollen [Rud16].

Merkmal	SQL	NoSQL	NewSQL
Relational	Ja	Nein	Ja
SQL	Ja	Nein	Ja
ACID Transaktionen	Ja	Nein	Ja
Horizontale Skalierbarkeit	Nein	Ja	Ja
Performanz	Nein	Ja	Ja
Schemalos	Nein	Ja	Nein

Tab. 8.1: Übersicht der Unterschiede zwischen SQL, NoSQL und NewSQL angelehnt an: [Gur15]

Tabelle 8.1 stellt nochmals einen zusammenfassenden Vergleich der Hauptcharakteristiken von SQL, NoSQL und NewSQL dar. So werden beispielsweise nur NoSQL Datenbanken als schemalos bezeichnet, da sie alleine erlauben, unstrukturierte Daten zu speichern, ohne ein Schema zu kennen [YWJH15]. Diese Charakteristiken wurden bereits in vorangegangen Kapiteln, in ihrer Terminologie erklärt und näher beschrieben.

Um die Unterschiede zwischen den Datenbanktypen im Detail aufzuzeigen, sollen diese anhand der jeweiligen Vertreter bezüglich ihrer „$Anfragemöglichkeiten$“, „$Performanz$“,

„*Concurrency Control*“, „*Replikate*“, „*Partitionierung und Konsistenz*“ und „*Sicherheit*“ verglichen werden.

8.1 Anfragemöglichkeiten

Generell sind die Anfragemöglichkeiten verschiedener Datenbankmanagementsysteme direkt mit dem eingesetzten Datenmodell verbunden [Gur15]. Aus diesem Grund ist es notwendig, diese genauer zu betrachten, um eine passende Datenbanklösung für einen bestimmten Anwendungszweck zu finden. Die verschiedenen Datenbankalternativen unterscheiden sich nicht nur im eingesetzten Datenmodell, sondern auch in den implementierten Schnittstellen zu den Daten.

So bieten beispielsweise dokumentbasierte Datenbanken, wie MongoDB, umfassende APIs an, die eine Reihe an mächtigen Anfragen ermöglichen. Solche Anfragen gehen über Bereichsanfragen, Sekundärindizes, oder Unterdokumente [Gur15]. MongoDB bietet zudem noch die Möglichkeit, Anfragen durch reguläre Ausdrücke zu erweitern [Mon18]. Die meisten Datenbanken dieser Art, implementieren ebenfalls eine Representational State Transfer (REST) API.

Graphdatenbanken stellen im Vergleich zu den anderen Datenbanktypen Anfragemöglichkeiten zweier Arten zur Verfügung:

Pattern-Matching sucht Teile des ursprünglichen Graphen nach dem entsprechenden Muster ab.

Graph-Traversierung durchläuft den Graphen ausgehend von dem gewählten Knoten. Hier können verschiedene Strategien, wie Breiten- bzw. Tiefensuche verfolgt werden. .

Auch hier stellen die meisten Datenbanken eine REST API bereit. Zudem gibt es ebenfalls verschiedene Anfragesprachen namens „*SparQL*“, „*Gremlin*“ oder „*Cypher*“, die in mehreren Graph Datenbanken zur Anfrage unterstützt werden. So werden diese auch in Neo4j implementiert. An dieser Stelle sei auf die Studienarbeiten „*Benchmarking von Graphdatenbanken*“ [Str16] von Mark Strempel und „*Benchmarking von Graphdatenbanken*“ [Met17] von Hannes Mettman verwiesen, in der besonderen Wert darauf gelegt wurde, die Funktionsweise und Grenzen von Graphdatenbanken herauszuarbeiten und strukturiert darzustellen.

Tabelle 8.2 fasst die Anfragemöglichkeiten der einzelnen Datenbanktypen übersichtlich zusammen und zeigt deren Gemeinsamkeiten und Unterschiede auf. Hier zeigt sich, dass besonders die NoSQL Datenbanken, im Bereich der Anfragemöglichkeiten Unterschiede im Vergleich zu den übrigen Datenbanktypen aufweisen. So implementieren diese nicht das standardisierte SQL als Anfragesprache, sondern greifen hier auf passende Alternativen zurück.

Anfragemöglichkeit	MySQL	MongoDB	Neo4j	VoltDB
Rest API	Ja	Ja	Ja	Ja
Query Language	SQL	Nein	Cypher, Gremlin und SparQL	SQL
Andere APIs	CLI und andere APIs in verschiedenen Sprachen	CLI und andere APIs in verschiedenen Sprachen	CLI und andere APIs in verschiedenen Sprachen	CLI und andere APIs in verschiedenen Sprachen

Tab. 8.2: Anfragemöglichkeiten der Datenbanktypen

8.2 Performanz

Die Performanz der verschiedenen Datenbanktypen soll innerhalb dieses Unterkapitels gegenüber gestellt und bewertet werden. Da es im Rahmen dieser Arbeit nicht möglich ist, eine ausgiebige Performanzevaluation selbst durchzuführen, muss sich in diesem Kapitel auf Ergebnisse ähnlicher Arbeiten bezogen werden. So sind die Ergebnisse der Lese-, Schreib- und Löschlatenzen innerhalb dieses Kapitels aus dem Whitepaper „*Comparison of SQL, NoSQL and NewSQL Databases for Internet of Things*“ [HK16] von Haleemunnisa und Kumud entnommen. Die hier aufgeführten Ergebnisse wurden zusätzlich noch anhand anderer wissenschaftlicher Arbeiten validiert, die an dieser Stelle nicht explizit genannt werden müssen. Alle Quellen zeigen ähnliche Tendenzen innerhalb der Performanzunterschiede zwischen den Datenbanktypen.

Im oben genannten Whitepaper wird die Performanzevaluation anhand vier verschiedener Kategorien betrachtet:

Single Client Single Operations (Lesen, Schreiben, Löschen)

Single Client Multi Operations (Schreiben)

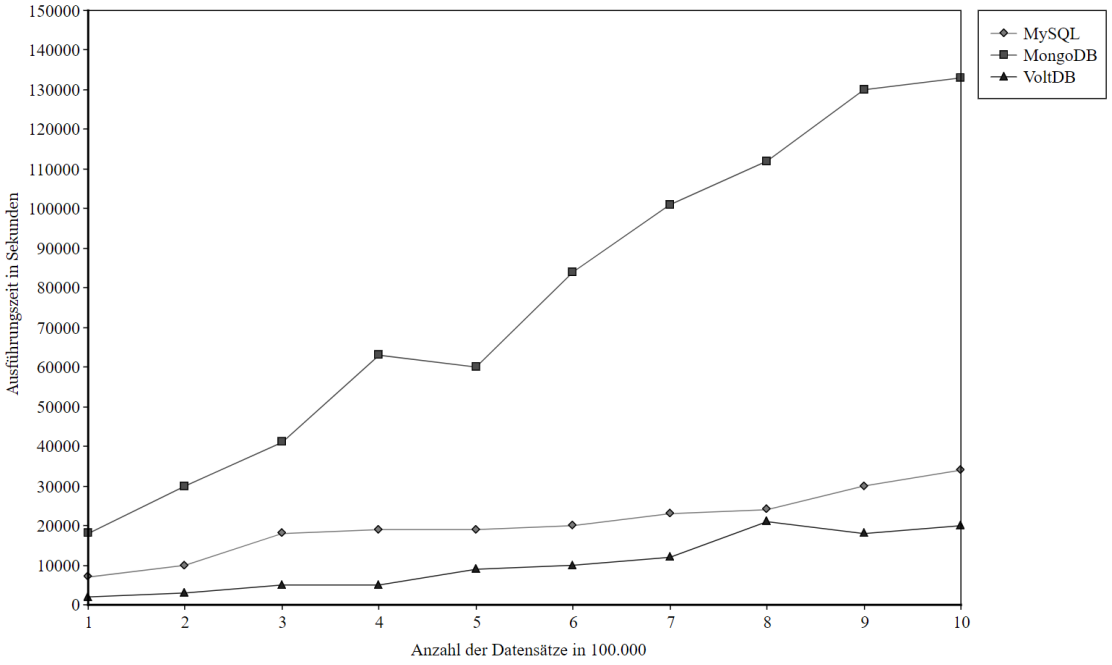
Multi Client Single Operations (Lesen, Schreiben)

Multi Client Multi Operations (Schreiben)

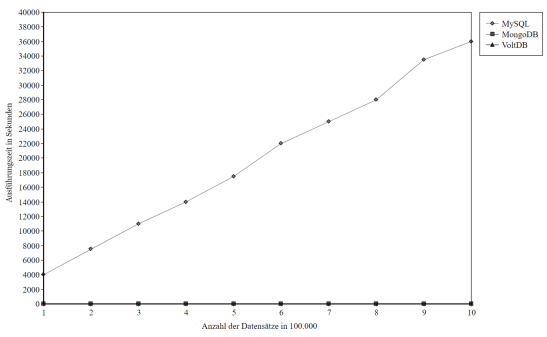
Abbildung 8.1 zeigt die Ergebnisse der „*Single Client Single Operations*“ für das Lesen, Schreiben und Löschen von Datensätzen. Hier zeigt sich eine sehr große Differenz zwischen MySQL und MongoDB bzw. VoltDB. Im Vergleich der Datenbankmanagementsysteme der verschiedenen Datenbanktypen, schneidet bei Leseoperationen NoSQL mit MongoDB am schlechtesten, traditionelles SQL mit MySQL als zweiter, und NewSQL mit VoltDB am Besten ab. Ähnlich große Differenzen der Datenbanktypen lassen sich auch für Schreib- und Löschoperationen erkennen.

Für die übrigen drei Kategorien innerhalb der Performanzevaluation schneidet das traditionelle SQL meist als schlechtester Datenbanktyp, NoSQL als zweitbester und NewSQL

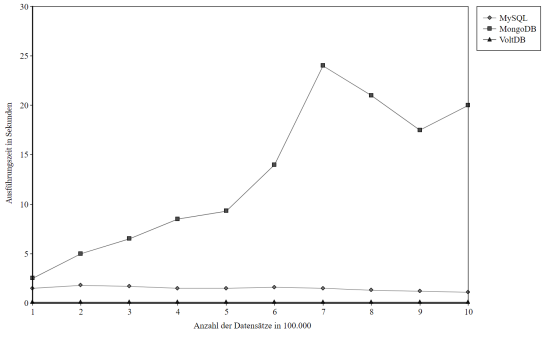
als führender Datenbanktyp ab. Die exakten Ergebnisse der einzelnen Operationstypen lassen sich im oben genannten Whitepaper nachlesen. VoltDB ist am schnellsten in fast allen Fällen. Trotzdem kann MongoDB durch Performanzoptimierung dank des flexiblen Datenmodells sehr gut mithalten. Auch hier zeigt sich erneut, dass es keine „*One fits all*“ Lösung gibt, sondern je nach Anwendungszweck entschieden werden muss, welcher Datenbanktyp für die vorhandenen Anforderungen am besten geeignet ist.



(a) Single Read Operations



(b) Single Write Operations



(c) Single Delete Operations

Abb. 8.1: Single Client Performanzevaluation angelehnt an: [HK16]

Diese Performanzunterschiede werden zusätzlich durch die Möglichkeiten zur vertikalen oder horizontalen Skalierbarkeit der einzelnen Datenbankmanagementsystemen ausgebaut. Wie bereits in vorherigen Kapiteln herausgearbeitet, handelt es sich bei der horizontalen Skalierbarkeit, um eine der Hauptfunktionalitäten von NoSQL oder auch NewSQL Datenbanksystemen. Traditionelles SQL besitzt, im Rahmen der Skalierbarkeit, hauptsächlich nur Möglichkeiten zur vertikalen Skalierbarkeit, indem Ressourcen innerhalb eines Knotens verbessert werden.

8.3 Concurrency Control

Concurrency Control ist ein wichtiger Bestandteil von NewSQL und NoSQL Systemen, da sie eine große Menge von parallelen Nutzerzugriffen verwalten müssen. Traditionelle SQL Datenbanksysteme verwenden hierzu eine pessimistische Methode, welche exklusiven Zugriff auf Daten gewährt. Möchte ein anderer Nutzer Zugriff auf ein blockiertes Datenobjekt erhalten, muss dieser warten, bis der Datensatz wieder freigegeben ist. Bei Datenbankclustern ist es notwendig eine Alternative zu finden, da dies sehr schnell zu einer Verschlechterung der Performanz führen kann [Gur15].

Neben den pessimistischen Verfahren gibt es ebenfalls optimistische Nebenläufigkeitsverfahren. Diese nehmen an, dass es zu Konflikten kommen kann, diese allerdings nur selten auftreten. Bevor Daten comitted werden, überprüft jede Transaktion, ob eine andere Transaktion den betreffenden Datensatz verändert hat. Sollte dies der Fall sein, wird ein Rollback ausgeführt [Gur15]. Diese Art der Concurrency Control kann sehr gut funktionieren, wenn Daten nur selten modifiziert werden.

Viele der NoSQL und NewSQL Datenbanken verwenden zu diesem Zweck Multi-Version Concurrency Control, welche bereits in Kapitel 5.4 erläutert wurde. Manche der NoSQL Datenbanksysteme erlauben den Applikationen ebenfalls, optimistische Concurrency Control Mechanismen zu implementieren [Gur15]. Außerdem gibt es auch eine kleine Untermenge von NewSQL Datenbanken, welche wie beispielsweise VoltDB, innovativere Ansätze verfolgen. Diese gehen davon aus, dass die gesamten Daten in den Hauptspeicher einer Maschine passen. So können Transaktionen sequentiell ohne Sperren von Datensätzen ausgeführt werden. Tabelle 8.3 fasst die Concurrency Control Ansätze der Datenbankmanagementsysteme nochmals zusammen.

Datenbank	Concurrency Control
MySQL	Tabellen- bzw. Reihen-Locking
MongoDB	Lese-Schreib Locks
Neo4j	Schreib Locks auf Knoten und Beziehungen bis Comitted
VoltDB	Keine Concurrency Control, ein Thread Modell

Tab. 8.3: Concurrency Control der Datenbanktypen angelehnt an: [Gur15]

8.4 Replikate

Replikate innerhalb eines Datenbanksystems bewirken neben der besseren Skalierbarkeit und Performanz auch eine optmierte Fehlertoleranz und Verfügbarkeit der Daten [Gur15]. Die verschiedenen Konzepte und Ansätze der Replikation wurden bereits in Kapitel

5.3.4 anhand eines Beispiels aufgezeigt und für verschiedene NewSQL Datenbanken in Kapitel 7.2.1 betrachtet. Diese Konzepte finden in sich in verschiedenen Varianten, bei traditionellem SQL, NoSQL und NewSQL Datenbanken wieder.

Da sowohl Vor- und Nachteile der einzelnen Methoden bereits herausgearbeitet wurden, werden hier nur die verschiedenen Umsetzungen aufgezeigt, um die Unterschiede der Datenbanktypen zusammenzufassen:

MySQL bietet die Option an das passive Master-Slave Prinzip zu nutzen.

MongoDB verwendet ebenfalls die Replikation im passiven Master-Slave Prinzip.

Neo4j verwendet Replikation im aktiven Master-Slave Prinzip.

VoltDB bietet sowohl das aktive Master-Slave Prinzip, als auch das Multi-Master Prinzip an.

Wie sich hier zeigt, finden sich die Vorteile von Replikaten über alle Datenbanktypen hinweg. Diese werden allerdings jeweils verschiedenen implementiert.

8.5 Partitionierung und Konsistenz

Im Bereich Big Data mit dem großen Umfang an Daten und Lese- bzw. Schreibanfragen, welche die Kapazitäten einer Maschine überschreiten, ist es notwendig Daten zu partitionieren. Traditionelle RDBMS unterstützen normalerweise keine horizontale Skalierbarkeit, aufgrund des normalisierten Datenmodells und der ACID Transaktionen [Gur15]. Die meisten NoSQL und NewSQL Datenbanken implementieren jedoch horizontale Partitionierung oder Sharding. Diese Konzepte wurden bereits in Kapitel 5.3.3 näher erklärt.

Partitionierung in Graphdatenbanken gestaltet sich allerdings schwieriger, als in den übrigen NoSQL Datenbanken. Hier gibt es generell zwei widersprüchliche Anforderungen, die eingehalten werden müssen [Gur15]. Alle verbunden Knoten eines Graphens sollten auf einem Datenbankknoten gehalten werden, um effizient durch den Graph traversieren zu können. Trotzdem dürfen nicht zu viele Knoten eines Graphen auf einem Datenbankknoten gehalten werden, da dies zu Performanzproblemen führen kann. Es gibt verschiedene Ansätze für Partitionierungsalgorithmen in Graphdatenbanken, die allerdings wenig Anwendung in tatsächlichen Graphdatenbanken finden [Son12]. Neo4j implementiert so ebenfalls keine Partitionierungsmethode, alternativ bieten sie Cache Sharding an. NewSQL Datenbanken verwenden ebenfalls verschiedene Partitionierungsmethoden. So verwendet VoltDB konsistentes Hashing, bei dem durch einen Schlüssel die Datenobjekte über mehrere Datenbankknoten verteilt werden.

Die Konsistenz eines Datenbanksystems ist hierbei stark an die Möglichkeiten zur Partitionierung gebunden. Das Konzept von starker Konsistenz und letztendlicher

Konsistenz wurde bereits in Kapitel 5.3.4 aufgezeigt. Die meisten der NoSQL Systeme implementieren das Konzept der letztendlichen Konsistenz. Trotzdem ermöglichen viele die Möglichkeit, diese zu konfigurieren, indem Performanz für eine bessere Konsistenz geopfert wird. So ermöglicht dies auch MongoDB, indem zwei Parameter gesetzt werden [Gur15]. Der erste Parameter ist „*read only*“ vom Master zu setzen, sodass nur auf einen Knoten für Leseanfragen zugegriffen wird. Die zweite Möglichkeit ist es „*write concern*“ auf „*replica acknowledged*“ zu setzen, sodass der Schreibzugriff sicher auf allen Replika ausgeführt wird.

Datenbanken	Partitionierung	Konsistenz
MySQL	Keine Partitionierung	Stark Konsistent
MongoDB	Bereichspartitionierung basierend auf einem Partitionierungsschlüssel	Konfigurierbar
Neo4j	Keine Partitionierung, Cache Sharding als alternative	Starke Konsistenz
VoltDB	Konsistentes Hashing	Starke Konsistenz

Tab. 8.4: Partitionierung und Konsistenz der Datenbanktypen

Tabelle 8.4 zeigt eine Übersicht über die verschiedenen Ansätze der Partitionierung und Konsistenzen von den betrachteten Repräsentanten der verschiedenen Datenbanktypen.

8.6 Sicherheit

Sicherheit ist ein essentieller Bestandteil von Datenbanken jeglicher Art, aufgrund der Möglichkeit, personenbezogene oder sensible Daten abzuspeichern. Aus diesem Grund hat es sich in den letzten Jahren zu einem der wichtigsten Kriterien für die kommerzielle Implementierung von Datenbankmanagementsystemen entwickelt. Kategorien, die hier betrachtet werden sollen, sind „*Authentifizierung*“, „*Autorisierung*“ und „*Verschlüsselung*“.

Die Verschlüsselung der Daten, sodass diese nicht von unautorisierten Dritten gelesen werden können, unterteilt sich nochmals in drei Unterkategorien. Diese sind die „*Verschlüsselung der Daten im Ruhezustand*“, „*Verschlüsselung der Daten von Client zu Server*“ und „*Verschlüsselung der Daten von Server zu Server*“. Tabelle 8.5 zeigt für diese Kategorien die Ergebnisse der verschiedenen Datenbankmanagementsysteme. Hier wird deutlich, dass sowohl NoSQL Datenbanken, als auch NewSQL Datenbanken noch nicht alle Sicherheitsstandards implementieren, wie es traditionelle

RDBMS tun. Diese sind teilweise nicht vorhanden, oder durch die passende Lizenzierung, wie bei MongoDB, beschränkt.

Sicherheitsverfahren	MySQL	MongoDB	Neo4j	VoltDB
Authentifizierung	Ja	Ja	Ja	Ja, Nutzer werden in einem Deployment Datei definiert
Autorisierung	Ja	Ja, die Rechte beinhalten „*Lesen*“, „*Schreiben*“, „*dbAdmin*“ und „*userAdmin*“	Ja, die Rechte beinhalten „*reader*“, „*publisher*“, „*architect*“ und „*admin*“	Ja
Ruhezustand (Verschlüsselung)	Ja	Nur Enterprise Edition, Logs werden nicht verschlüsselt	Nein	Nein
Client-Server (Verschlüsselung)	Ja - TLS/SSL	Ja - TLS/SSL	Ja - TLS/SSL	Ja - TLS/SSL
Server-Server (Verschlüsselung)	Ja	Ja	Nein	Ja

Tab. 8.5: Sicherheit der Datenbanktypen

Diese Sicherheitslücken können bei der Auswahl eines Datenbanktyps für den Einsatz in einem Unternehmen ein entscheidender Faktor sein und sollten deswegen umgehend geschlossen werden. Andernfalls bleiben traditionelle RDBMS den neueren Lösungen der NoSQL und NewSQL Bewegungen, im Bereich der Sicherheit, klar überlegen.

9 Kapitel 9
Fazit und Ausblick

„Every revolution has its counterrevolution -that is a sign the revolution is for real“ - C. Wright Mills[1]

In der Vergangenheit haben sich durch verteilte Web- und Mobilapplikationen neue Anforderungen im Bereich der Datenhaltung und -verarbeitung von Datenbanken ergeben. Diese Anforderungen konnten nicht von den traditionellen Datenbanksystemen erfüllt werden und sorgten so für die Entwicklung von neuen Architekturen. Hieraus haben sich zwei Kategorien entwickelt, die sich als Alternative zu den bestehenden Architekturen klassifizieren. Hierbei handelt es sich um NoSQL und NewSQL Datenbankmanagementsysteme.

Innerhalb dieser Arbeit wurde die Entwicklung von frühen Datenbanksystemen über traditionelle, relationale Datenbankmanagementsystemen und NoSQL Ansätze bis hin zu NewSQL Datenbanksystemen aufgezeigt. Hierbei wurden mehrere moderne Konzepte, wie In-Memory Datenbanken oder verschiedene Clustering Methodiken aufgezeigt und anhand von verschiedenen Beispielen, die Vor- und Nachteile der verschiedenen Methodiken herausgearbeitet. Zusätzlich wurde eine umfassende Evaluierung expliziter Realisierungen der NewSQL Bewegung mittels den Grundzügen des Open Source Business Readiness Ratings vorgenommen, um ein methodisches Vorgehen sicher zu stellen. Hierbei wurden die Stärken und Schwächen der Datenbankmanagementsysteme herausgearbeitet und gegenübergestellt. Anhand passender Repräsentanten und Kriterien wurde die NewSQL Bewegung ebenfalls zu traditionellem SQL und NoSQL Datenbanken abgegrenzt. Diese Vergleiche sollen dazu dienen, einen Überblick über die verschiedenen Möglichkeiten zu gewinnen und den passenden Datenbanktyp für die Anforderungen einer Applikation wählen zu können.

Als praktischer Teil dieser Studienarbeit wurde CockroachDB, als führendes NewSQL Datenbankmanagementsystem in der Kategorie „$Nutzbarkeit$“, prototypisch in einer virtuellen Umgebung implementiert. Diese Implementierung verfolgt den Hintergrund, die Funktionalitäten und das Aufsetzen einer NewSQL Lösung aufzuzeigen und dient damit als Plug & Play Prototyp für zukünftige Arbeiten in diesem Bereich.

Die Datenbanksysteme, wie sie innerhalb dieser Arbeit aufgezeigt wurden, sind noch lange nicht in ihrer Entwicklung abgeschlossen. Diese werden sich auch zukünftig in ihren

[1] 28. August 1916 - 20. März 1962, US-amerikanischer Soziologe

Konzepten weiterentwickeln, um den modernen Anwendungszwecken zu genügen. Es könnte sogar zu einer Konterrevolution kommen, wie es das einleitende Zitat zu diesem Kapitel bereits suggeriert. Obwohl es unwahrscheinlich klingt, dass sich diese Art der modernen Datenbankmanagementsysteme in ihrer Gesamtheit zurück zu traditionellen RDBMS entwickeln, ist ein Kompromiss durchaus in der Zukunft eine Möglichkeit. Jetzige NewSQL Lösungen sind nicht dazu entworfen RDBMS zu ersetzen, sondern in bestimmten Szenarien, eine erfolgreiche Lösung darzustellen. Ohne Zweifel optimieren NewSQL Datenbanken Applikationen mit Business Intelligence Funktionalitäten oder Applikationen, die ACID Transaktionen voraussetzen. Aufgrund der Vielfältigkeit von heutigen Anwendungszwecken, werden zukünftige Datenbanksysteme wahrscheinlich mehrere dieser Anwendungsszenarien durch passende Konfiguration optimal bearbeiten können und so in einem breiteren Anwendungsbereich eingesetzt werden.

Weitere Arbeiten, die innerhalb dieses Bereiches angestellt werden können, sind besonders ein eigener Performanzvergleich und eine Realisierung in einer tatsächlichen Applikation. Die damit erhobenen Werte können mit den Ergebnissen, der in dieser Arbeit zitierten Quellen, verglichen und so zusätzlich validiert werden. Durch die Realisierung in einer tatsächlichen Applikation, können ebenfalls noch einmal Vor- und Nachteile aufgearbeitet und in einer realen Umgebung getestet werden. In diesem Bereich kann außerdem die prototypische Implementierung von CockroachDB von einem lokalen Cluster auf mehrere Server aufgeteilt werden, um auch hier realitätsnahe Beobachtungen zu machen. Eine Betrachtung der Veränderungen im Bereich NoSQL und NewSQL Datenbanken in den kommenden Jahren könnte ebenfalls interessante Ansätze und Implementierungen aufzeigen.

Ein Aspekt, der besonders hier noch zukünftige Relevanz haben könnte, ist eine Technologie, welche besonders bei Kryptowährungen Anwendung findet. Bei der Kryptowährung „$Bitcoin$“ beispielsweise, kann ein Benutzer mittels seinem privaten Schlüssel Eigentum nachweisen und Transaktionen autorisieren. Die anderen Benutzer können über einen öffentlichen Schlüssel diese Transaktion validieren. Das verwendete Peer-to-Peer System verwendet keinen zentralen Server, sondern ein verteiltes öffentliches Transaktionsbuch namens „$Blockchain$“. Hierbei erlaubt die Blockchain eine Validierung der Transaktionen durch mehrere Knoten, mittels des öffentlichen Schlüssels, bevor die Transaktion abgeschlossen wird. Dieses Prinzip stellt eine neue Art der verteilten Datenbank dar, da sich hier die Berechtigungen auf Daten anders verhalten. In der Blockchain wird das Eigentumsrecht der Daten vom Ersteller der Daten und nicht vom Inhaber der Datenbank verwaltet, wie in bisherigen Datenbanken. So könnten im Umfeld der sozialen Medien, Beiträge nur vom Urheber und nicht von dem zugehörigen Unternehmen modifiziert beziehungsweise gelöscht werden [Har15]. Obwohl die meisten Datenbankinhaber heutzutage diese Kontrolle über die Daten beibehalten wollen, besteht die Möglichkeit, dass zukünftig bestehende Datenbanksysteme ähnliche Authentifizierungs- und Autorisierungsmethoden implementieren.

Literaturverzeichnis

[Ber15] Bernstein, David: *IEEE Cloud Computing*. Bd. 2, Issue: 4, S. 90–92: *Cloud Tidbits Column: Today's Tidbit: VoltDB*. 2015

[Bul17] Bulanov, Boris ; DZone / Database Zone (Hrsg.): *Benchmarking Google Cloud Spanner, CockroachDB, and NuoDB: Take a deep dive into three elastic SQL databases, all of which are pretty promising options that meet the needs of the modern data center*. https://dzone.com/articles/benchmarking-google-cloud-spanner-cockroachdb-and. Version: 2017. – zuletzt abgerufen am: 06.05.2018

[CG06] Chamoni, Peter ; Gluchowski, Peter: *Analytische Informationssysteme: Business Intelligence-Technologien und -Anwendungen*. Berlin and Heidelberg : Springer Gabler, 2006. – ISBN 978–3–540–33752–2

[Coc18] Cockroach Labs: *CockroachDB: The SQL database for global cloud services: Unternehmenswebsite*. www.cockroachlabs.com. Version: 2018. – zuletzt abgerufen am: 24.04.2018

[Dou08] Doumack, Andrej: *Evaluierung von Reporting-Frameworks am Beispiel der ausgewählten Open-Source-Anwendungen*. Gummersbach, Fachhochschule Köln, Diplomarbeit, Oktober 2008

[GC15] Gluchowski, Peter ; Chamoni, Peter: *Analytische Informationssysteme: Business Intelligence-Technologien und -Anwendungen*. 5., vollständig überarbeitete Auflage. Berlin and Heidelberg : Springer Gabler, 2015. – ISBN 978–3–662–47762–5

[GDG08] Gluchowski, Peter ; Dittmar, Carsten ; Gabriel, Roland: *Management Support Systeme und Business Intelligence: Computergestützte Informationssysteme für Fach- und Führungskräfte*. 2, vollst. überarb. Aufl. Berlin and Heidelberg : Springer, 2008. – ISBN 978–3–540–23543–9

[Gur15] Gurevich, Yuri: *Comparative Survey of NoSQL / NewSQL DB Systems*. Israel, Open University of Israel, Masterarbeit, 2015

[Hah14] Hahn, Erin N. ; Johns Hopkins APL Technical Digest, Volume 32, Number 4 (Hrsg.): *An Overview of Open-Source Software Licenses and the Value of Open-Source Software to Public Health Initiatives*. http://www.

jhuapl.edu/techdigest/TD/td3204/32_04-Hahn.pdf. Version: 2014. – zuletzt abgerufen am: 17.01.2018

[Har15] Harrison, Guy: *Next generation databases: NoSQL, NewSQL, and Big Data*. New York : Apress, 2015. – ISBN 978–1–4842–1330–8

[HK16] Haleemunnisa, Fatima ; Kumud, Wasnik: *Comparison of SQL, NoSQL and NewSQL Databases for Internet of Things*. Piscataway, NJ : IEEE, 2016. – ISBN 978–1–5090–2730–9

[HKGH16] Hutchison, Dylan ; Kepner, Jeremy ; Gadepally, Vijay ; Howe, Bill: *2016 IEEE High Performance Extreme Computing Conference (HPEC): The Westin Hotel Wltham-Boston (70 Third Avenue, Waltham, MA 02451) : 13-15 September 2016*. Piscataway, NJ : IEEE, 2016. – ISBN 978–1–5090–3525–0

[IBM18] IBM: *The Four V's of Big Data*. www.ibmbigdatahub.com/infographic/four-vs-big-data. Version: 2018

[Kar17] Karambir, Kaur: *Performance Evaluation of NewSQL Databases*. Punjab, Punjab Institute of Technology, International Conference of Inventive Systems and Control (ICISC-2017), 2017

[Kes13] Kessler, Steffen: *Anpassung von Open-Source-Software in Anwenderunternehmen: Philipps-Universität Marburg, Dissertation*. Wiesbaden : Springer Vieweg, 2013 (Entwicklung und Management von Informationssystemen und intelligenter Datenauswertung). – ISBN 978–3–658–01954–9

[KMU06] Kemper, Hans-Georg ; Mehanna, Walid ; Unger, Carsten: *Business Intelligence - Grundlagen und praktische Anwendungen: Eine Einführung in die IT-basierte Managementunterstützung*. 2., ergänzte Auflage. Wiesbaden : Friedr. Vieweg & Sohn Verlag/GWV Fachverlage GmbH Wiesbaden, 2006 (IT erfolgreich lernen). – ISBN 978–3–8348–9056–6

[LC07] Linden, Markus ; Chamoni, Peter: *Das Wirtschaftsstudium: wisu Zeitschrift für Ausbildung, Prüfung, Berufseinstieg und Fortbildung*. Bd. 36: *Business Intelligence*. 2007

[LCA+17] Leonard, Andy ; Currie, Scott ; Alley, Jacob ; Andersson, Martin ; Avenant, Peter ; Fellows, Bill ; Peck, Simon ; Smith, Reeves ; Sondak, Raymond ; Weissman, Benjamin ; Wilhelmsen, Cathrine: *The Biml Book: Business Intelligence and Data Warehouse Automation*. Berkeley, CA : Apress, 2017. – ISBN 978–1–4842–3134–0

[Lee11] Lee, Sunguk: *International Journal of Energy*. Bd. 2 Issue 4: *Shared-Nothing vs. Shared-Disk Cloud Database Architecture: Research Institute of Industrial Science and Technology*. 2011 www.sersc.org/journals/IJEIC/vol2_Is4/15.pdf

[LeeoJ] Lee, Sin-Min: *Object oriented Database*. slideplayer.com/slide/5183466/#. Version: o.J.. – zuletzt abgerufen am: 04.06.2018

[Lei15] Leipert, Ralph: *Business Intelligence Wissensportal*. http://www.business-intelligence24.com. Version: 2015. – zuletzt abgerufen am: 11.01.2018

[Lin16] Linden, Markus: *Geschäftsmodellbasierte Unternehmenssteuerung mit Business-Intelligence-Technologien: Unternehmensmodell - Architekturmodell - Datenmodell*. 1. Aufl. 2016. Wiesbaden : Springer Fachmedien Wiesbaden, 2016. – ISBN 978–3–658–11800–6

[Met17] Mettmann, Hannes: *Benchmarking von Graphdatenbanken*. Horb, DHBW Stuttgart, Studienarbeit, 2017

[Mio16] Mio, Chiara: *Integrated Reporting: A New Accounting Disclosure*. London : Palgrave Macmillan UK, 2016. – ISBN 978–1–137–55148–1

[Mon18] MongoDB: *Unternehmenswebsite*. https://www.mongodb.com/. Version: 2018. – zuletzt abgerufen am: 16.05.2018

[Mul17] Mullins, Craig S.: *The Pros and Cons of Database Scaling Options*. http://go.nuodb.com/rs/139-YPK-485/images/Database-Scaling-Options.pdf. Version: 2017. – zuletzt abgerufen am: 22.03.2018

[Mur13] Murugan, Ananda Sentraya P.: *A Study of NoSQL and NewSQL databases for data aggregation on Big Data*. Stockholm, Sweden, KTH Royal Institute of Technology, Masters Degree Project, 2013. http://www.diva-portal.org/smash/get/diva2:706302/FULLTEXT01.pdf

[MyS18] MySQL: *Unternehmenswebsite*. https://www.mysql.com/de/. Version: 2018. – zuletzt abgerufen am: 23.05.2018

[Neo18] Neo4j: *Unternehmenswebsite*. https://neo4j.com. Version: 2018. – zuletzt abgerufen am: 23.05.2018

[Nuo13] NuoDB, Inc.: *A Technical Whitepaper: The Architecture & Motivation for NuoDB: Version 1*. Cambridge, MA 02142, 2013

[Nuo18] NuoDB, Inc.: *The SQL Database for modern Data Centers: Unternehmenswebsite*. https://www.nuodb.com. Version: 2018. – zuletzt abgerufen am: 24.04.2018

[Ope05] OpenBRR: *Business Readiness Rating for Open Source: A Proposed Open Standard to Facilitate Assessment and Adoption of Open Source Software: RFC 1*. https://www.immagic.com/eLibrary/ARCHIVES/GENERAL/CMU_US/C050728W.pdf. Version: 2005. – zuletzt abgerufen am: 16.01.2018

[PA16] Pavlo, Andrew ; Aslett, Matthew: *SIGMOD Record*. Bd. 45, No. 2: *What's Really New with NewSQL?* 2016

[Pok15] Pokorný, Jaroslav: *ACM international conference proceedings series*. Bd. Vol. 1008: *Database Technologies in the World of Big Data: 16th International Conference, CompSysTech'15 proceedings Dublin, Ireland, June 25-26, 2015: CompSysTech (Conference)*. 2015. – ISBN 978–1–4503–3357–3

[Rud16] Rudolf, Christoph: *SQL, noSQL or newSQL - comparison and applicability for Smart Spaces*. München, Technische Universität München, Diss., 2016

[SAB+05] Stonebreaker, Michael ; Abadi, Daniel J. ; Batkin, Adam ; Chen, Xuedong ; Cherniack, Mitch ; Ferreira, Miguel ; Lau, Edmond ; Lin, Amerson ; Madden, Sam ; O'Neil, Elizabeth ; O'Neil, Pat ; Rasin, Alex ; Tran, Nga ; Zdonik, Stan: *C-Store: A Column-oriented DBMS: Proceedings of the 31st VLDB Conference*. Trondheim, Norway, 2005

[Sch17] Schmidt, Robin: *Berechnung und Auswertung von Leistungskennzahlen der Produktion innerhalb eines Manufacturing Execution Systems*. Horb, DHBW Stuttgart, Projektarbeit T2_2000, 02.10.2017

[Sel17] Sellics: *Amazon verdoppelt Sortiment seit 2014 auf 229 Mio. Produkte!* https://sellics.com/de/blog-amazon-verdoppelt-sortiment-seit-2014-auf-229-mio-produkte. Version: 2017. – zuletzt abgerufen am: 01.03.2018

[SK15] Sharma, Gitanjali ; Kaur, Pankaj D.: *Architecting Solutions for Scalable Databases in Cloud: Proceedings of the Third International Symposium on Women in Computing and Informatics (WCI-2015): International Symposium on Women in Computing and Informatics / International Conference on Advances in Computing, Communications and Informatics*. 2015 (ICPS). – ISBN 978–1–4503–3361–0

[SM10] Savoska, Snezana ; Manevska, Violeta: *Business Intelligence Tools for Statistical Data Analysis: Proceedings of the ITI 2010 32nd Int. Conf. on Information Technology Interfaces*. Bitola, Partizanska, 2010

[SMA+07] Stonebreaker, Michael ; Madden, Samuel ; Abadi, Daniel J. ; Harizopoulos, Stavros ; Hachem, Nabil ; Helland, Pat: *The End of an Architectual Era: (It's time for a Complete Rewrite): 33rd International Conference on Very Large Data Bases, University of Vienna, Austria, September 23-27 2007 : conference proceedings*. New York, N.Y. : Association for Computing Machinery, 2007. – ISBN 978–1–59593–649–3

[Son12] Soni Madhulatha, T.: Graph Partitioning Advance Clustering Technique. In: *International Journal of Computer Science & Engineering Survey* 3

(2012), Nr. 1, S. 91–104. http://dx.doi.org/10.5121/ijcses.2012.3109. – DOI 10.5121/ijcses.2012.3109. – ISSN 09763252

[SSC15] Smuts, Martin ; Scholtz, Brenda ; Calitz, Andre: *Design Guidelines for Business Intelligence Tools for Novice Users: Proceedings of the 2015 Annual Research Conference of the South African Institute of Computer Scientists and Information*. Stellenbosch, South Africa, 2015 (ACM international conference proceedings series). – ISBN 978–1–4503–3683–3

[Sta17] Statistic Brain: *Average Historic Price of RAM*. https://www.statisticbrain.com/average-historic-price-of-ram/. Version: 2017

[Sto86] Stonebreaker, Michael: *Database Engineering*. Bd. 9, Nummer 1: *The Case for Shared Nothing*. 1986 db.cs.berkeley.edu/papers/hpts85-nothing.pdf

[Str16] Strempel, Mark: *Benchmarking von Graphdatenbanken*. Horb, DHBW Stuttgart, Studienarbeit, 2016

[Vol17] VoltDB: *Technical Overview*. https://www.voltdb.com/wp-content/uploads/2017/03/hv-white-paper-voltdb-technical-overview.pdf. Version: 2017. – zuletzt abgerufen am: 18.04.2018

[Vol18] VoltDB: *Real-time Analytics & Real-time Decisions: Accelerate Business Impact with VoltDB's Translytical Database: Unternehmenswebsite*. https://www.voltdb.com. Version: 2018. – zuletzt abgerufen am: 24.04.2018

[YWJH15] Yuan, Li-Yan ; Wu, Lengdong W. ; Jia-Huai, You: *A Demonstration of Rubato DB: A Highly Scalable NewSQL Database System for OLTP and Big Data Applications: Compilation proceedings of the 2015 ACM Symposium on Principles of Database Systems, ACM SIGMOD International Conference on Management of Data, and SIGMOD/PODS 2015 PhD symposium, May 31 - June 4, 2015, Melbourne, VIC, Australia*. New York, NY : ACM, 2015. – ISBN 978–1–4503–2758–9

Anhang A

A Konfiguration der virtuellen Maschine und Installation CockroachDB

In diesem Kapitel werden alle Hardware- und Softwarekonfigurationen aufgelistet, die für die virtuelle Maschine gewählt wurden. Diese können je nach Belieben angepasst werden:

Betriebssystem: Debian 9.4

Hauptspeicher: 3000MB

Bootreihenfolge: Diskette, DVD, Platte

Beschleunigung: VT-x/AMD-V, Nested Paging, KVM-Paravirtualisierung

Grafikspeicher: 16MB

Festplattenspeicher: 8GB

Das CockroachDB Archiv muss zunächst mit folgendem Befehl heruntergeladen und ausgeführt werden:

```
1 wget -qO- https://binaries.cockroachdb.com/cockroach-v2.0.2.linux-amd64.tgz |
      tar xvz
```

Anschließend muss das binary der Umgebungsvariable hinzugefügt werden, damit Cockroach Befehle aus dem Terminal ausgeführt werden können. Dies geschieht am Besten mit den nötigen Berechtigungen, die mittels dem Präfix $sudo$ oder durch den Befehl su erreicht werden können:

```
1 cp -i cockroach-v2.0.2.linux-amd64/cockroach /usr/local/bin
```

Anschließend lassen sich die CockroachDB Befehle im Terminal ausführen und es bieten sich alle Möglichkeiten.

Anhang B

B Hardware- und Softwareumgebung des Performanzvergleiches

Tabelle B.1 zeigt die Hardware- bzw. Softwareumgebung des Clients und des Server, in welcher der Performanzvergleich aus Kapitel 8.2 stattgefunden hat.

Parameter	Server	Client
OS	64-bit Ubuntu 14.04	64-bit Ubuntu 14.04
Prozessor	Intel Core i7-3770 CPU @ 3.40 GHz x 8	Intel Core i5-2410M CPU @ 2.30GHz x 4
MySQL	5.5.49	JDBC Connector Version 5.1.39
MongoDB	3.2.7	Java Driver 3.2.0
VoltDB	6.3	Java Driver 6.3
JDK	1.8.0	1.8.0
PHP	5.5	5.5

Tab. B.1: Hardware- und Softwareumgebung [HK16]