Fachhochschule der Wirtschaft
-FHDW-
Bielefeld

Bachelor Thesis

Thema:

Problemstellungen der Datenhaltung in global verteilten Systemen und deren Auswirkungen auf die Anwendungsarchitektur

Prüfer:
Prof. Dr. Wilhelm Nüßer
Hr. Jörg Pommeranz

Verfasser:

Andreas Gerlach
Listerstr. 6
45147 Essen

8. Semester
Studiengang Bachelor of Science
Schwerpunkt: Wirtschaftsinformatik

Eingereicht am:

16. Februar 2012

Inhaltsverzeichnis **Seite**

Abbildungsverzeichnis

Seite

Tabellenverzeichnis

Seite

Abkürzungsverzeichnis

ACID	Atomic, Consistent, Isolated, Durable
AJAX	Asynchronous JavaScript and XML
BASE	Basically available, soft state, eventually consistent
bspw.	beispielsweise
bzgl.	bezüglich
bzw.	beziehungsweise
CDN	Content-Delivery-Network
CL	Cache Layer
CSS	Cascading Stylesheets
DAS	Direct Attached Storage
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
EDV	elektronische Datenverarbeitung
ETL	Extract, Transform, Load
ggf.	gegebenenfalls
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
i.d.R.	in der Regel
inkl.	inklusive
I/O	Input/Output
ISP	Internet Service Provider
IP	Internet Protocol
IT	Informationstechnik
LAN	Local Area Network
LB	Load Balancer
NAS	Network Attached Storage
NoSQL	Not-Only-SQL
o.g.	oben genannten
OLAP	Online Analytic Processing

OLTP	Online Transcation Processing
RDBMS	relationales Datenbank-Management-System
REST	Representational State Transfer
SAN	Storage Area Network
SOA	Service-orientierte Architektur
SOAP	Simple Object Access Protocol
SPOF	Single Point Of Failure
SQL	Structured Query Language
TCP	Transmission Control Protocol
u.a.	unter anderem
UDP	User Datagram Protocol
usw.	und so weiter
URL	Uniform Resource Locator
u.U.	unter Umständen
WAN	Wide Area Network
WSDL	Web Service Description Language
XML	Extensible Markup Language
z.Zt.	zurzeit

1 Einleitung

1.1 Ausgangslage

"Wir leben in einer Zeit vollkommener Mittel und verworrener Ziele"

– Albert Einstein

Jenes Zitat von Albert Einstein spiegelt auf bizarre Weise die Problematik der Datenhaltung in global verteilten Systemen wider: dem Nutzer suggeriert das Internet ein hohes Maß an Mobilität, während die Datenflut in einem scheinbar unübersichtlichen Geflecht mündet.

Aus diesem Gewirr von Servern die benötigten Informationen allen möglichen Anwendern zeitnah zur Verfügung zu stellen, bedeutet einen enormen technischen Aufwand – nicht nur aus Verwaltungssicht sondern auch im Hinblick auf die Konzeption eines derart hochgradig verteilten Systems. Dabei steht die Forschung in diesem Bereich noch am Anfang. Grundlegende Theorien zu diesem Thema wurden vor allem durch die großen Internetdienstleister wie Google, Amazon oder Facebook in den letzten zehn Jahren veröffentlicht.

Allerdings stellt sich beim Sichten dieser Dokumente[1] schnell heraus, dass jedes Unternehmen dabei ohne Rücksicht auf einen Status Quo vorgegangen ist, wodurch verschiedene Ansätze für den Aufbau eines derartigen Systems entstanden sind. Größtenteils beschäftigen sich jedoch alle Studien mit der Problematik, wie in einem global verteilten System mit der umfangreichen Menge der täglich anfallenden Daten umgegangen werden soll. Daten, die zudem in einem Kontext zu bringen sind, um daraus gewinnbringende Informationen für ein Unternehmen abzuleiten.

Die Erkenntnisse aus diesen Studien sollen hier aufgegriffen und mit den bewährten Methoden zur Datenhaltung in ein gemeinsames, schlüssiges Konzept überführt werden, welches einen Leitfaden für die Handhabung von Daten unter Berücksichtigung der verschiedenen Arten von Informationen, die in einem global verteilten System auftreten können, darstellt.

---

[1] siehe hierzu Google (2006), Amazon (2007) und Facebook (2009)

1.2 Zielsetzung der Arbeit

Diese Arbeit analysiert und bewertet die verschiedenen Mechanismen der Datenhaltung in global verteilten Systemen. Dabei werden mögliche Einsatzfelder und bewährte Praktiken und Problemfelder erörtert als auch deren Auswirkung auf die Anwendungsarchitektur unter den Aspekten der Skalierbarkeit, Verfügbarkeit, Wartbarkeit und Konsistenz kritisch diskutiert. Bilanzierend ersucht diese Arbeit eine Architektur-Blaupause für ein global verteiltes System aufzuzeigen, in der die verschiedenen Ablagesysteme für die Daten mit ihrem etwaigen Einsatzzweck beurteilt werden.

1.3 Aufbau und Struktur der Arbeit

Den Auftakt der Arbeit bildet die Betrachtung jener Daten, die beim Betrieb einer Softwareanwendung auftreten. Diese werden nach verschiedenen Kriterien klassifiziert, um in der weiteren Betrachtung Akzente setzen zu können. Hieraus soll eine Zuweisung resultieren, welche Arten von Daten einer besonderen Aufmerksamkeit beim Design der Anwendungsarchitektur bedürfen.

Die Resultate bzw. Einordnung der Daten dienen in den weiteren Kapiteln der Arbeit dazu, passende Mechanismen und Praktiken für einen speziellen Typ an Daten herauszuarbeiten. Hierzu müssen jedoch zunächst die zur Verfügung stehenden Optionen zur Ablage von Daten in einem Softwaresystem analysiert werden. Diese Vorbetrachtung ist unabdingbar, da es nicht ein alleiniges Verfahren gibt, welches alle Einsatzzwecke unter Berücksichtigung der Rahmenparameter zufriedenstellend abdeckt.

Um ein tieferes Verständnis für die Anforderungen von global verteilten Systemen zu schaffen, widmet sich das vierte Kapitel dem allgemeinen Aufbau eines solchen Anwendungssystems und bestimmt damit die Rahmenparameter, die in der folgenden Betrachtung eine zentrale Rolle einnehmen werden. Im Anschluss daran werden die für einen globalen Betrieb der Anwendung verfügbaren Mechanismen berücksichtigt.

Auf Basis dieser breit gefächerten Analyse erfolgt abschließend die Auswertung in Hinblick auf die Zielsetzung dieser Arbeit, um bilanzierend die Datenhaltung in global verteilten Systemen und deren Auswirkungen auf die Anwendungsarchitektur kritisch zu diskutieren.

2 Daten

2.1 Bedeutung von Daten

Daten symbolisieren Fakten zu Objekten oder Sachen in der realen Welt, die uns interessieren und deren Charakteristika wir für erinnerungswürdig erachten. Dabei sind es in den meisten Fällen mehrere Sachverhalte, die wir zu einem Gegenstand erfassen und festhalten möchten. Die Objekte oder Sachen der realen Welt werden in diesem Zusammenhang auch als Entitäten bezeichnet; Fakten werden in diesen Entitäten mit Hilfe von Attributen repräsentiert.

Daten haben für die Informationsgewinnung nicht nur in der Wissenschaft und Forschung sondern auch für die Wirtschaft einen signifikanten Stellenwert. Ihre Relevanz existierte bereits vor der Einführung der elektronischen Datenverarbeitung mittels eines IT Systems, dessen Einsatz jedoch die Aufnahme, Speicherung und Auswertung von Daten wesentlich vereinfacht hat. Die Erfassung der Daten kann hierbei entweder manuell durch eine Person aufgrund einer Beobachtung oder automatisch durch die Interaktion mit einem Anwendungssystem erfolgen.[2] Um aus den reinen Daten Informationen zu gewinnen, müssen diese eine Semantik bekommen. D.h. die Daten müssen in dem ihnen zugehörigen Kontext betrachtet werden, um daraus eine Aussagekraft ableiten zu können. In einem Informationssystem werden diese Zusammenhänge über zusätzliche Daten – den sogenannten Metadaten – hergestellt.

“Die Information ist das Öl des 21. Jahrhundert”[3] – diese Aussage wurde vom Senior Vice President des Beratungshauses Gartner bewusst gewählt, um die anstehenden Herausforderungen bei der Verwaltung der in einem Unternehmen anfallenden Daten zu verdeutlichen. So gehen Analysten der Firma IDC davon aus, dass die Datenmenge “in den zurückliegenden fünf Jahren um den Faktor fünf gewachsen”[4] sei und der “IBM zufolge produzieren wir derzeit weltweit täglich 2,5 Trillionen Byte Daten [...]. 90 Prozent des derzeitigen globalen Datenbestands seien in den beiden zurückliegenden Jahren entstanden”[5].

---

[2] vgl. Gillenson, M. (2005), S. 10ff

[3] Sondergaard, P. in Bayer, M. (2011b), S. 14

[4] Bayer, M. (2011b), S. 14

[5] Bayer, M. (2011b), S. 14

Die Ursache für diese Datenexplosion liegt darin begründet, dass im Gegensatz zur frühen IT-gestützten Datenverarbeitung, wo die zumeist strukturierten Daten i.d.R. durch Mitarbeiter gepflegt und durch wenige Transaktionssysteme produziert wurden, dank der Verbreitung von mobilen Endgeräten und der zunehmenden individuellen Beteiligung im Web 2.0 heute sehr verschiedene Datenquellen existieren, welche große Datenmengen mit Informationen unterschiedlicher Qualität und Beschaffenheit zur Verfügung stellen.[6] Diese zum Großteil unstrukturierten Daten wie bspw. Texte, Bilder oder Videos müssen in die vorhandenen Informationsstrukturen bestmöglich integriert und zugänglich gemacht werden, schätzen doch $^2/_3$ der IT-Spezialisten in einer Umfrage des Magazins Computerwoche deren Anteil auf mittlerweile bis zu 60 Prozent des Gesamtdatenbestand in ihrem Unternehmen ein.[7]

Zusätzlich sind auch die Ansprüche an die Datenqualität gestiegen. Es geht hierbei in erster Linie nicht mehr alleine darum, möglichst viele Daten im Unternehmen zu sammeln, sondern diese Daten zum richtigen Zeitpunkt an der richtigen Stelle analysieren zu können. Stand dabei früher eher die periodisch wiederholte Aufbereitung historischer Daten in Berichtsform im Vordergrund, so "geht es heute verstärkt darum, neue Daten in Echtzeit auszuwerten und künftige Entwicklungen zu simulieren, um eine möglichst valide Grundlage für [unternehmerische] Entscheidungen zu erhalten".[8] Diese Analysen müssen sich an den Ansprüchen der Adressaten ausrichten, wobei zeitgleich die Anzahl der Nutzer solcher Datenanalysen im Unternehmen stark angestiegen ist.[9]

In den letzten Jahren wird daher immer häufiger von einer unternehmerischen Ressource im Zusammenhang mit den Daten einer Organisation gesprochen, weil durch diese versucht wird, die Objekte und deren Beziehungen in einem Unternehmen umfassend zu beschreiben. Dadurch können die erfassten Daten neben den Mitarbeitern und den Patenten einen weiteren Wettbewerbsvorteil für das einzelne Unternehmen im Informationszeitalter ausmachen.[10]

---

[6] vgl. Bayer, M. (2011b), S. 14ff

[7] vgl. Bayer, M. (2011c), S. 19

[8] Bayer, M. (2011b), S. 15f

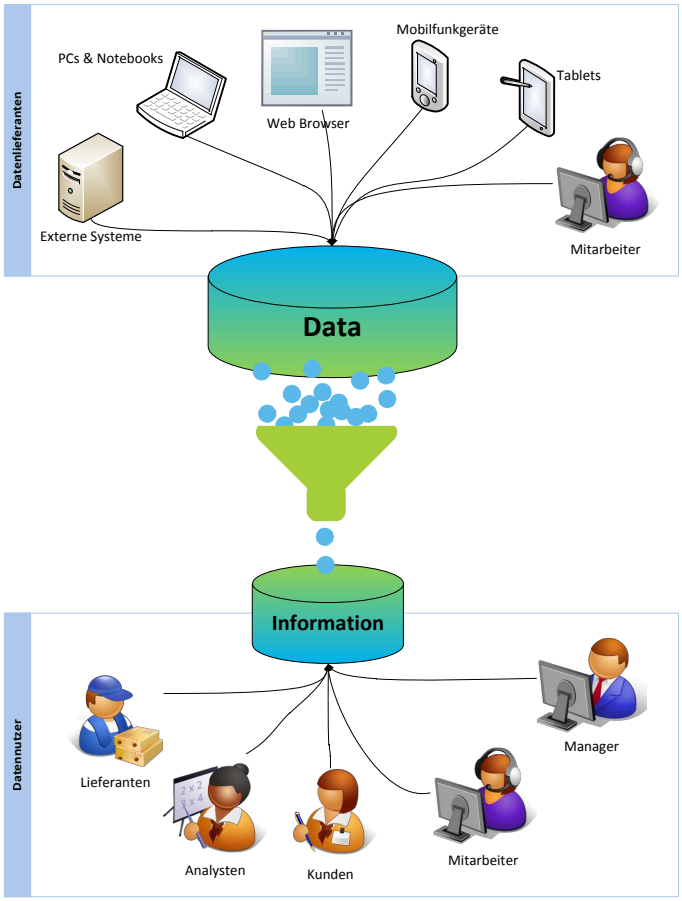
[9] vgl. Bayer, M. (2011b), S. 14ff

[10] vgl. Gillenson, M. (2005), S. 10

Ähnlich wie bei den anderen unternehmerischen Ressourcen müssen daher auch die Daten in der Organisation optimal verwaltet und den jeweiligen Anspruchsstellern rechtzeitig zur Verfügung gestellt werden. Im Gegensatz zu den anderen Arbeitsmitteln eines Unternehmens muss jedoch der Verwaltung und Bereitstellung der Daten eine besondere strategische Betrachtung zugrunde liegen, da sie als Einzige über ein enormes Größenvolumen und einem kontinuierlichen Wandel verfügen.[11]

Teile dieser Strategie betreffen dabei auch die Art und Weise, wie die Daten in einem EDV System abgelegt und über welche Mechanismen sie zugänglich gemacht werden. Hierbei sind u.U. spezialisierte Werkzeuge notwendig, damit die "Menge der Daten [uns nicht] um die Ohren [fliegt]".[12]

Abbildung 1: Informationen als Ressource eines Unternehmens



Quelle: eigener Entwurf nach Bodendorf, F. (2006), S. 1ff

[11] vgl. Gillenson, M. (2005), S. 12

[12] Spies, R. in Bayer, M. (2011b), S. 17

2.2 Kategorisierung von Daten

Um eine optimale Speicher- und Zugriffsmethodik für die Daten zu erarbeiten, müssen diese zunächst in eine Struktur gebracht werden. Dazu dient die folgende Kategorisierung der Daten, die in einem Informationssystem auftreten können. Die Einteilung der Informationen erfolgt dabei anhand verschiedener Dimensionen, die zunächst losgelöst vom Gesamtzusammenhang aufgestellt werden. Sie bilden gemeinsam einen mehrdimensionalen Würfel mit den möglichen Einteilungsgraden und werden in der praktischen Betrachtung herangezogen, um für einen spezifischen Typus an Daten eine optimale Strategie für deren Verwaltung aufzuzeigen.

2.2.1 Dimension: Format

Daten können entweder in einem Textformat oder einem binären Format gespeichert sein. Das Textformat hat den Vorteil, dass es vom Menschen einsehbar und lesbar ist. Als Beispiel dient hier eine Komma-separierte Textdatei mit Adressdaten von Kunden. Im Gegensatz dazu handelt es sich bei der binären Darstellung um eine für den Rechner optimierte Speicherung der Daten. Diese können nur unter Zuhilfenahme eines Programms, welches das binäre Format lesen kann, für den Menschen zugänglich gemacht werden.

Nachteilig an der text-orientierten Darstellung der Daten ist die Größe der Dateien, da eine derartige Speicherung ineffizienter und Ressourcen-intensiver als das Binärformat ist. Gleichfalls muss ein Rechner, der die Daten in Textform auswerten möchte, diese zunächst analysieren und in eine maschinenkonforme Datenstruktur konvertieren. Fehlt den Daten in der Textdatei die Semantik, wird die maschinelle Auswertung hierdurch noch zusätzlich erschwert.[13]

2.2.2 Dimension: Beschaffenheit

Daten lassen sich anhand ihrer Semantik in die Klassen strukturiert, semi-strukturiert und unstrukturiert einordnen. Strukturierte Daten folgen dabei einem festen, wiederkehrenden Schema – bspw. die Adresskartei einer Arztpraxis oder die Bestelldaten eines Online Händlers. Unstrukturierte Daten sind hingegen schemalos und haben keine Semantik. Dazu zählen u.a. Binärdateien mit Bilder-, Video- oder Musik-

---

[13] vgl. Hunt A. / Thomas, D. (2000), Kapitel 14

daten als auch reine Textdateien in Form einer e-Mail, einer Web-Seite oder eines Text-Dokuments. Um nicht strukturierte Daten maschinell auswerten zu können, müssen diese ihre Semantik in Form von Meta-Attributen mitführen. Die damit angereicherten Daten werden daher als semi-strukturiert bezeichnet. Sie folgen zwar keinem festen, wiederkehrenden Schema, liefern jedoch alle notwendigen Informationen mit, damit ihre Rohdaten dem jeweiligen Kontext zugeordnet werden können. Beispiele dafür sind u.a. die Übermittlung von Textdaten in Form von XML oder die zusätzlichen Meta-Attribute im Kopf von Bilddaten, die Aufschluss über Aufnahmezeitpunkt und -ort geben sowie benutzerspezifische Schlagwörter beinhalten können.[14]

2.2.3 Dimension: Rolle

Daten können in einem betrieblichen Informationssystem in unterschiedlichen Rollen auftreten: den Anwendungs-, Konfigurations- und Laufzeitdaten. Als Anwendungsdaten werden alle Daten bezeichnet, die zur Ausführung des Informationssystems notwendig sind. Die Daten dazu können entweder in binärer Form (bspw. als maschinen-ausführbare Datei) oder in Textform (bspw. als Script einer Web Seite) abgelegt sein und liegen in dieser Gestalt meist unstrukturiert vor.

Die Konfigurationsdaten beeinflussen das Verhalten der Anwendung, indem sie die Ausführung der Anwendung mit bestimmten, vorher definierten Parametern steuern. Dazu können technische Einstellungen bspw. zur Datenbankverbindung und dem Log-Verhalten der Anwendung als auch betriebswirtschaftliche Einstellungen wie bspw. der für eine Altersprüfung gültige Wertebereich gezählt werden. Die Konfigurationsdaten können sowohl in Textform oder in binärer Form abgelegt sein, müssen aber für den Menschen einsehbar und änderbar gestaltet werden. Da auch die Anwendung den Kontext der Daten kennen muss, sind sie zudem in strukturierter oder semi-strukturierter Form zu speichern. Als Textform kommen sie daher i.d.R. als XML Dateien oder Textdateien mit Schlüssel-Wert Paaren – der Schlüssel spiegelt dabei den Kontext wider – vor. Die binäre Form der Ablage der Konfigurationsdaten umfasst in diesem Zusammenhang die Speicherung der Informationen in einer Datenbank.

---

[14] vgl. Florescu, D. (2005), S. 20ff

Als Laufzeitdaten werden alle Daten bezeichnet, die durch die Nutzung und den Betrieb der Anwendung anfallen bzw. entstehen. Das Format und die Beschaffenheit der Laufzeitdaten ist somit stark abhängig von der Anwendung selbst und es können alle Permutationen der o.g. Kriterien Beschaffenheit und Format auftreten.

2.2.4 Dimension: Verwendungszweck

Inhaltlich unterscheidet man die Laufzeitdaten in einem Informationssystem zwischen Stammdaten, Bestandsdaten und Bewegungsdaten. Stammdaten klassifizieren dabei Dinge aus der realen Welt – bspw. Produkte, Konsumenten, Lieferanten, Filialen, Anwender usw. Nach ihnen kann recherchiert werden – bspw. welches Produkt erfüllt die Eigenschaften x,y und z. Sie werden zur Erzeugung der Bewegungsdaten benutzt und in diesen referenziert. Die Bewegungsdaten entstehen durch Transaktionen, die der Nutzer mit dem Informationssystem ausführt. Durch die Verarbeitung der Bewegungsdaten werden u.U. die Bestandsdaten – dabei handelt es sich bspw. um den Lagerbestand eines Rohstoffs – innerhalb des IT Systems verändert.[15]

Stammdaten und Bestandsdaten sind i.d.R. strukturiert abgelegt. Sie unterscheiden sich durch die Häufigkeit und Art ihrer Änderungen. Diese sind bei den Stammdaten seltener der Fall als bei den Bestandsdaten und werden nur durch bestimmte Rollen im Unternehmen durchgeführt. Bestandsdaten ändern sich hingegen häufiger; was durch die Verarbeitung von Bewegungsdaten hervorgerufen wird. Bewegungsdaten können sowohl strukturiert, semi-strukturiert als auch unstrukturiert sein. Sie sind stetigen Änderungen unterworfen, da sie die Arbeitsmasse des Programms repräsentieren.

2.2.5 Dimension: Dynamik

Mittels der Dynamik der Daten wird ausgedrückt, wie häufig diese Daten Änderungen unterworfen sind. Statt hierfür feste Größen vorzugeben, kann auf eine Klassifizierung wie sehr oft, oft, manchmal, selten und nie zurückgegriffen werden. Eine Beurteilung der Werte muss abhängig vom Einsatzszenario des Anwendungssystems erfolgen. Denkbar wäre eine Einstufung nach dem Änderungsintervall der Daten – was zudem Rückschlüsse auf die Zeitdauer ermöglicht, in der ein Datensatz voraussichtlich unverändert sein wird.

---

[15] vgl. Bayer, M. (2011a), S. 14

Tabelle 1: Bewertung der Dynamik anhand des Änderungsintervalls

Kriterium	Änderungsintervall
sehr oft	stündlich
oft	täglich
manchmal	monatlich
selten	jährlich
nie	–

eigener Entwurf

2.2.6 Dimension: Größe

Ähnlich ist mit der Größendimension der Daten zu verfahren. Hierbei bieten sich die Klassen: sehr groß, groß, klein und sehr klein als Beschreibung an. Die Spezifizierung kann dabei entweder nach dem Umfang der Daten in Bytes oder nach deren Übertragungsdauer über eine Standard-Netzwerkverbindung erfolgen. Da die Einstufung der Werte auch von der eingesetzten Technik bzw. vom technischen Fortschritt abhängt, muss hierbei ebenfalls eine zum Zielsystem passende Beurteilung der Klassen stattfinden.

Tabelle 2: Definition des Größenkriteriums mittels Umfang und Übertragungsdauer

Kriterium	Umfang in Bytes	Übertragungsdauer
sehr groß	10^9 (Gigabytes)	Stunden
groß	10^6 (Megabytes)	Minuten
klein	10^3 (Kilobytes)	Sekunden
sehr klein	10^0 (Bytes)	Millisekunden

eigener Entwurf

2.2.7 Dimension: Reichweite

Viele Informationen sind durch einen örtlichen Faktor bestimmt. So sind bspw. Transaktionsdaten eines Abverkaufssystems mit dem Ort ihrer Entstehung verbunden. Abhängig von der geographischen Reichweite der Informationen, kann daher zwischen lokalen, regionalen, überregionalen und globalen Daten unterschieden werden.

Lokale Daten sind dabei auf ihren Entstehungsort respektive einem kleinen Umkreis darum beschränkt – bspw. Abteilungsdaten, Daten auf einem Einzelplatz-PC. Die regionalen Daten beziehen sich bereits auf einen größeren Umkreis um ihren Ursprungsort herum – bspw. die zugehörige Stadt oder das jeweilige Bundesland. Informationen, die für ein ganzes Land oder einen Kontinent Gültigkeit besitzen, werden auch als überregionale Daten bezeichnet. Und im Falle, dass die Informationen weltweite Gültigkeit haben, gehören sie in die Kategorie der globalen Daten.
Die auf lokaler Ebene entstandenen Daten lassen sich u.U. zu Informationen auf den höheren Ebenen über Summenfunktionen verdichten und können somit für ein breiter gefächertes Gebiet in Gänze betrachtet werden.

2.2.8 Dimension: Langlebigkeit

Viele Informationen werden ebenfalls durch einen zeitlichen Faktor bestimmt. So sind bspw. die Transaktionsdaten eines Abverkaufssystems immer mit dem jeweiligen Verkaufsdatum versehen. Dabei können diese Daten – u.a. auch durch gesetzliche Vorgaben – eine minimale Aufbewahrungsdauer besitzen, in der diese Informationen in detaillierter Granularität vorhanden sein müssen. Es kann daher nach kurzfristigen, mittelfristigen und langfristigen Informationen unterschieden werden, wobei i.d.R. mit zunehmender Dauer der Verwahrung die Auflösung der Daten abnimmt und diese nur noch in sinnvollen Schritten konsolidiert abgelegt werden; so stehen bspw. mittelfristig nur noch die Umsätze pro Quartal innerhalb einer Warengruppe zur Verfügung.

2.2.9 Dimension: Wertigkeit

Die Wertigkeit der Daten gibt an, inwieweit der erfolgreiche Betrieb der unternehmerischen Tätigkeit von diesen Daten abhängt. Es kann daher nach kritischen oder unkritischen Daten unterschieden werden. Das Hauptaugenmerk beim Konzipieren eines Anwendungssystems ist natürlich auf die Ablage der kritischen Daten zu richten.

2.2.10 Dimension: Zugriffsform

Der Zugriff auf die strukturierten Daten innerhalb eines Anwendungssystems erfolgt i.d.R. zufällig. D.h. es ist im Vorfeld nicht bekannt, in welcher Reihenfolge die Daten vom System angefordert werden. Beim sequentiellen Zugriff auf die Informationen werden die Daten jedoch in einer bestimmten Reihenfolge für die Verarbeitung benötigt.

Daher muss für den Zugriff auf diese Daten eine kontinuierliche Datenübertragungsrate sichergestellt werden, mit der die Informationen zwischen den Systemen ausgetauscht werden – bspw. bei der Übertragung von Multimedia Dateien im Internet. Diese Tatsache hat nicht nur Auswirkungen auf der Ablage der Informationen in einem System, sondern stellt auch eine Herausforderung für deren spätere Bereitstellung dar.

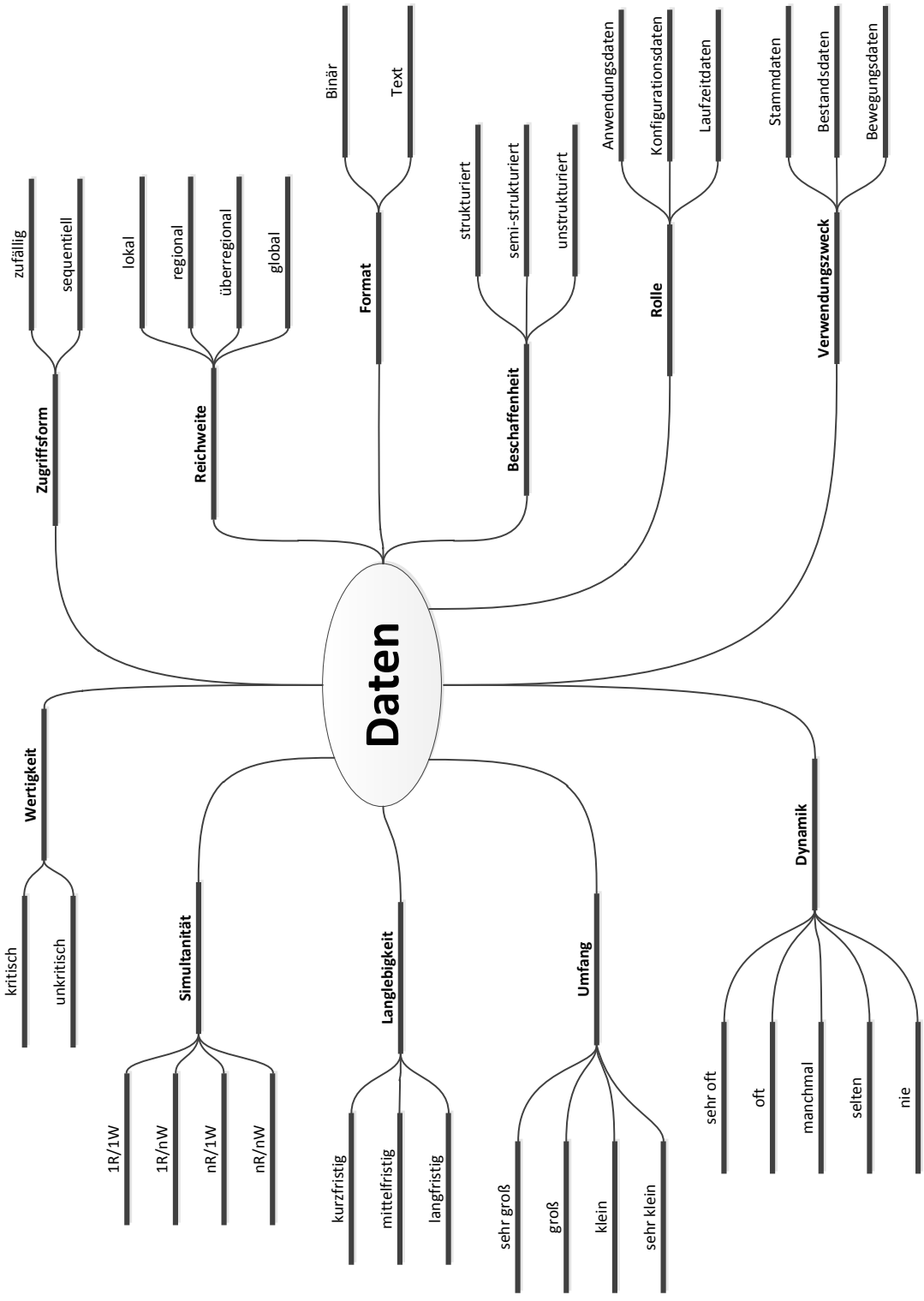
2.2.11 Dimension: Simultanität

Die Informationen in einem Anwendungssystem werden von verschiedenen Nutzergruppen gelesen und / oder verändert. Dabei spielt es eine Rolle, inwiefern auf die Daten von mehreren Nutzern gleichzeitig zugegriffen wird und welche Art des Zugriffs dabei erfolgt. Es lässt sich daher zwischen den Klassen einer lesend und schreibend, einer lesend und mehrere schreibend, mehrere lesend und einer schreibend sowie mehrere lesend und schreibend unterscheiden.

Dabei stellt die Einordnung in die Klasse einer lesend und schreibend i.d.R. kein Problem für die Ablage der Daten und dem Betrieb der Anwendung dar, da die Informationen einer einzelnen Person zugeordnet und von dieser verwaltet werden – bspw. die persönlichen Einstellungen innerhalb eines Programms. Ähnlich unkompliziert verhält es sich mit der Einstufung in mehrere lesend und einer schreibend. Hierbei werden Änderungen an den Daten ebenfalls nur von einer Person initiiert, wobei jedoch mehrere Nutzer lesend auf die Informationen zugreifen – bspw. ein persönliches Nutzerprofil in einer Social Web Anwendung.

Kritischer ist der Umgang mit den Daten der Klassen einer lesend und mehrere schreibend sowie mehrere lesend und schreibend anzusehen, da es hierbei zu überschneidenden Änderungen an den Informationen kommen kann. Zugleich steigt in diesen Datenklassen die Frequenz der Schreibvorgänge, was eine erhöhte Last auf dem Ablagesystem der Daten zur Folge hat. Es ist daher wichtig, dort einen hohen Datendurchsatz sicherzustellen, um mit der rasanten Taktung der Schreibzugriffe umgehen zu können. Gleichfalls gilt es die Konsistenz der Daten zu gewährleisten. Beispiele für diese Klassen sind u.a. die Pinnwand in einer Social Web Anwendung oder die Verarbeitung von Verkaufstransaktionen in einem zentralen System.

Abbildung 2: Überblick der Dimensionen zur Kategorisierung von Daten



Quelle: eigener Entwurf

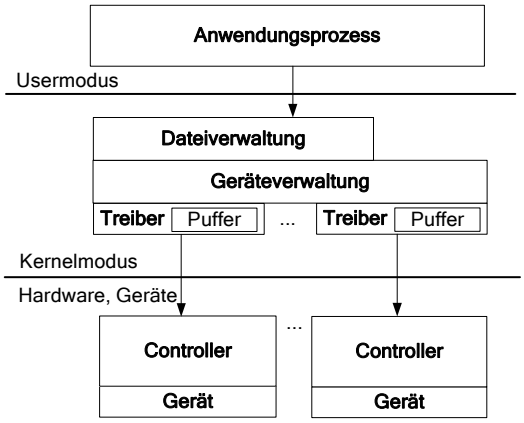
3 Mechanismen zur Datenspeicherung

3.1 Dateisysteme

3.1.1 Aufbau und Funktionalität

Daten, die in der aktuellen Verarbeitung sind, hält das Anwendungsprogramm im schnellen Hauptspeicher des Systems vor. Jedoch handelt es hierbei um flüchtigen Speicher. Daher müssen die Daten vor dem Ausschalten des Computers oder dem Beenden des Programms auf einem dauerhaften Datenspeicher abgelegt werden, um für spätere Verarbeitungszwecke erneut zur Verfügung zu stehen. Die Ablage und Verwaltung von Daten auf einem Magnetspeicher gehört zu den fundamentalen Aufgaben eines Betriebssystems. Es nimmt die Anfragen und Befehle des Anwendungsprogramms entgegen und setzt diese in Steuerungsbefehle der Magnetspeichereinheit um.[16]

Abbildung 3: Dateiverwaltung als integraler Bestandteil des Betriebssystems



Quelle: Mandl, P. (2010), S. 254

Als Speichermedium für die Daten werden heute i.d.R. Festplatten verwendet. Sie bestehen aus mehreren übereinander angeordneten Magnetscheiben und deren zugehörigen Schreib-/Leseköpfen. Der Arm, an dem die Schreib-/Leseköpfe befestigt sind, ist beweglich. Dadurch werden die Köpfe über den Magnetscheiben positioniert. Diese drehen sich mit einer konstanten Geschwindigkeit, die je nach Ausführung zwischen 5.400 und 15.000 Umdrehungen pro Minute beträgt.

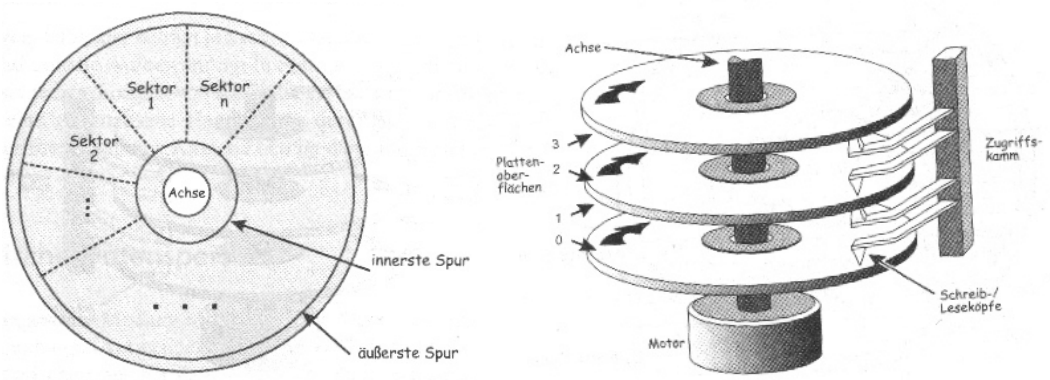
---

[16] vgl. Mandl, P. (2010), S. 254ff

Aus physikalischer Sicht werden die Daten auf dem Medium als Blöcke fester Größe in Sektoren und Spuren untergebracht. Die Sektoren teilen eine Magnetscheibe dabei in mehrere Kuchenstücke. Gleichfalls ist eine Magnetscheibe in mehrere Spuren aufgeteilt, die entlang der Drehrichtung verlaufen.

Um die Daten auf der Festplatte zu lokalisieren, muss das Betriebssystem wissen, auf welcher Spur, in welchem Sektor und auf welcher Magnetscheibe – repräsentiert durch die Nummer des Schreib-/Lesekopfes – die Daten zu finden sind.[17]

Abbildung 4: physikalischer Aufbau einer Festplatte



Quelle: Stahlknecht, P. / Hasenkamp, U. (2005), S. 57f

Die logische Sicht auf die Struktur der Festplatte wird über das Dateisystem des Betriebssystems realisiert. Es existieren diverse Dateisysteme, die sich je nach System funktional stark voneinander unterscheiden können. Ihnen gemein sind jedoch die folgenden grundlegenden Eigenschaften: Verwaltung von Dateien, Verzeichnissen und Verzeichnishierarchien und die Verwaltung von freien und beschädigten Blöcken auf dem Magnetspeicher.

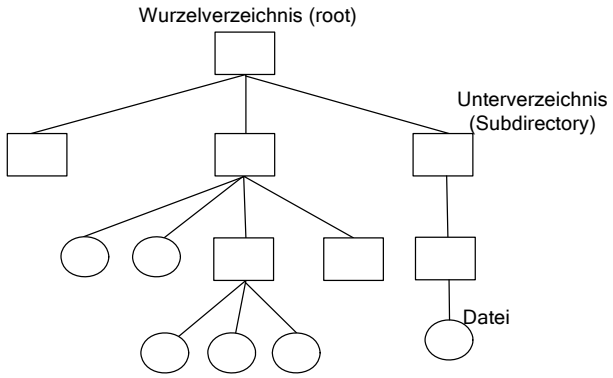
Die Ablage von Daten in einem Dateisystem erfolgt mit Hilfe der Dateien. Die Dateien liegen in einem ihnen zugewiesenen Verzeichnis und sind darin über ihren eindeutigen Namen adressierbar. Verzeichnisse können wiederum nicht nur Dateien sondern auch weitere Verzeichnisse beinhalten. Dadurch entsteht eine hierarchische Sicht auf die Daten des Speichermediums.[18]

---

[17] vgl. Mandl, P. (2010), S. 258ff

[18] vgl. Mandl, P. (2010), S. 264f

Abbildung 5: logische Sicht auf die Festplatte



Quelle: Mandl, P. (2010), S. 265

Die Dateien repräsentieren jedoch nicht nur die eigentlichen Rohdaten, die in ihnen abgelegt sind. Zu jeder Datei werden auch Meta-Informationen geführt, die Auskunft über den Namen und die Größe der Datei, die ihr zugewiesenen Blöcke auf dem Speichermedium, den Besitzer der Datei sowie möglicherweise weitere Zugriffsrechte als auch statistische Informationen wie bspw. das Datum der Erstellung, der letzten Änderung und des letzten Zugriffs auf die Datei geben. Die Meta-Informationen aller Dateien speichert das Betriebssystem in einem reservierten Bereich des Speichermediums – dieser wird allgemein als File Table bezeichnet.[19]

Beim Zugriff auf die Datei über ihren vollständigen Namen ermittelt das Betriebssystem den passenden Eintrag in der File Table, um mit dessen Informationen herauszufinden, in welchen Blöcken sich die Daten der Datei befinden. Da die Datei blockweise eingelesen wird, muss das Betriebssystem alle Blöcke der Datei ausfindig machen und der Reihe nach anspringen, um den Inhalt der Datei zu restaurieren. Daraus resultiert auch die Zeitdauer, die das System benötigt, um die Datei vom physikalischen Speichermedium zu lesen. Weil die Blöcke je nach Intelligenz und Auslastung des Dateisystems über die gesamte physikalische Struktur verteilt sein können, kommt es u.U. zu enormen Verzögerungen beim Wiederherstellen der Daten, was durch die jeweilige Neupositionierung der Schreib-/Leseköpfe der Festplatte verursacht wird. Diese sind jedoch erforderlich, um die Blöcke auf dem Medium zu erreichen.[20]

---

[19] vgl. Horn, C. / Kerner, I. / Forbrig, P. (2003), S. 152ff

[20] vgl. Mandl, P. (2010), S. 260f

Um den Aufwand für das Einlesen der Daten möglichst klein zu halten, muss die Datei daher bestenfalls in aufeinander folgenden Blöcken auf dem Speichermedium abgelegt sein. Damit wird es dem System ermöglicht, die Daten sequentiell von der Festplatte zu lesen, was einen bis zu fünf Zehnerpotenzen höheren Datendurchsatz als beim zufälligen Zugriff auf die Datenblöcke bedeuten kann.[21] Neuere Entwicklungen ersetzen die Festplatten mit ihren beweglichen Bauteilen immer häufiger durch auf Flash-Speicher basierende Solid-State-Drives (SSDs). Diese verzichten gänzlich auf mechanische Komponenten und haben dadurch eine mehr als zwei mal höhere Datenübertragungsrate und eine fast doppelt so lange Haltbarkeitsdauer (gemessen anhand der "mean time between failures") als aktuelle Festplatten.[22]

Moderne Dateisysteme bieten verschiedene Sicherheitsmechanismen an, um eine Zerstörung von Dateien oder dem Dateisystem zu verhindern. Dazu gehören bspw. die bereits genannten Zugriffsrechte in den Meta-Informationen einer Datei. Hierüber kann der Besitzer einer Datei einstellen, welche anderen Nutzer respektive welche Gruppe von Nutzern auf die Datei lesend, schreibend und / oder ausführend zugreifen darf. Zusätzlich dazu protokollieren einige Dateisysteme alle Änderungen an der Dateisystemstruktur mit, um im Falle eines Systemabsturzes einen konsistenten Zustand wiederherstellen zu können – dies wird auch als Journaling bezeichnet. Um zusätzlich gegen physikalische Beschädigungen der File Table geschützt zu sein, pflegen manche Dateisysteme des Weiteren eine oder mehrere Kopien dieser Meta-Informationen, die dafür jeweils in unterschiedlichen Bereichen der Festplatte abgelegt werden.[23]

Diese Maßnahmen bieten jedoch nur einen unzureichenden Schutz gegen den Total-Ausfall des Magnetspeichers. In diesem Fall wären alle Informationen auf der Festplatte verloren. Um sich dagegen abzusichern, müssen die Daten der Festplatte in regelmäßigen Abständen auf einem separaten Laufwerk gesichert werden. Dies kann durch den Einsatz von Backup-Medien sowie durch das Aufsetzen eines RAID Systems mit mehr als einer Festplatte erfolgen.[24]

---

[21] vgl. Jacobs, A. (2009), S. 6

[22] vgl. Intel (2011)

[23] vgl. Mandl, P. (2010), S. 270ff

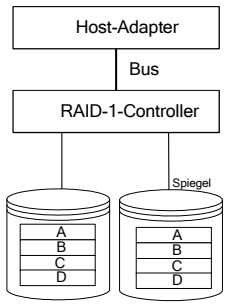
[24] vgl. Stahlknecht, P. / Hasenkamp, U. (2005), S. 485f

3.1.2 RAID-Systeme

Bei einem RAID System handelt es sich um einen Verbund von mehreren Festplatten, die aus der Sicht des Betriebssystems als eine logische Einheit angesprochen werden. Je nach RAID-Level – hier werden die Level 1, 1+0 und 5 betrachtet – lassen sich damit die Ausfallsicherheit und / oder die Geschwindigkeit beim Datenzugriff steigern.

Bei einer RAID-1 Konfiguration werden alle Daten auf eine zweite Festplatte gespiegelt, was die Ausfallsicherheit des Gesamtsystems erhöht, da im Störfall einer Festplatte sofort mit der Kopie auf der Spiegelplatte weitergearbeitet werden kann. Diese Konfiguration wirkt sich jedoch negativ auf die Geschwindigkeit aus, da sämtliche Schreibvorgänge hierbei redundant auf beiden Datenträgern durchgeführt werden müssen.

Abbildung 6: RAID-1 Konfiguration

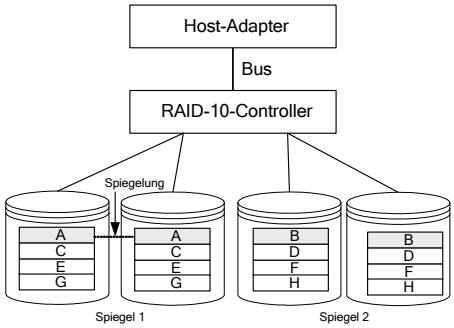


Quelle: Mandl, P. (2010), S. 275

Die RAID-1+0 Konfiguration wird z.Zt. häufig in der Praxis verwendet, da sie einen Kompromiss zwischen Geschwindigkeit und Sicherheit darstellt. Bei dieser Konfiguration müssen wenigstens vier Festplatten im System vorhanden sein, wobei zwischen jeweils zwei von ihnen eine redundante Spiegelung existiert und die Daten des Systems blockweise auf den anderen beiden Festplatten verteilt werden. Durch diese Aufteilung erhöht sich die Geschwindigkeit des Datentransfers, da die Festplattenzugriffe parallel abgehandelt werden können. Aufgrund der zusätzlichen Spiegelung der Daten wird gleichzeitig die Datensicherheit gewährleistet.[25]

[25] vgl. Mandl, P. (2010), S. 275f

Abbildung 7: RAID-1+0 Konfiguration

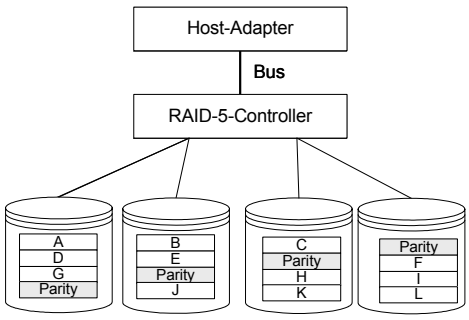


Quelle: Mandl, P. (2010), S. 276

Sowohl die RAID-1 als auch die RAID-1+0 Konfiguration sind jedoch Ressourcen-intensiv, da die Festplatten für die Spiegelung jeweils doppelt vorhanden sein und die Speicherzugriffe den Datensatz redundant ablegen müssen. So sind in einer RAID-1+0 Konfiguration mit vier Festplatten á 200 GB von den 800 GB Speicherplatz lediglich 400 GB nutzbar, da der Rest für die Spiegelung der Daten reserviert ist.

Eine Optimierung von RAID-1+0 stellt daher RAID-5 dar. Hierbei werden nicht die kompletten Daten aus Gründen der Ausfallsicherheit und zwecks Wiederherstellung der Daten redundant gehalten, sondern es werden in regelmäßigen Abständen Fehlerkorrekturcodes – sogenannte Parity Bits – auf den Festplatten abgelegt. Diese ermöglichen eine komplette Restaurierung der Daten, solange lediglich eine Festplatte des Verbunds ausfällt.[26]

Abbildung 8: RAID-5 Konfiguration



Quelle: Mandl, P. (2010), S. 277

[26] vgl. Mandl, P. (2010), S. 277

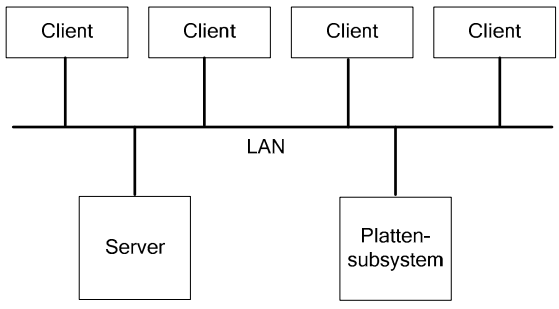
Dadurch können die vorhandenen Ressourcen effizienter genutzt werden. Bei einer Konfiguration wie im obigen Beispiel stehen von den 4 x 200 GB Nettokapazität beim Einsatz von RAID-5 insgesamt 600 GB für Nutzdaten zur Verfügung. RAID-5 zeichnet sich zudem durch eine sehr gute Lesegeschwindigkeit aus. Die Schreibgeschwindigkeit fällt aber aufgrund der zusätzlichen Fehlerkorrekturcodes etwas geringer als der Normwert aus; sie hat sich jedoch im Vergleich zur kompletten Spiegelung der Daten wie in der RAID-1 oder RAID-1+0 Konfiguration wesentlich gesteigert.[27]

Die bisherige Betrachtung umfasste die Struktur von Datenträgern und Festplatten im Allgemeinen, wie sie bspw. in einem einzelnen Rechnern im Einsatz sein kann – auch Direct Attached Storage (DAS) genannt. Um Dateien in einem Netzwerk für verschiedene Nutzer mühelos verfügbar zu machen und zusätzlich eine zentrale Verwaltung der Medien offerieren zu können, existieren spezialisierte Lösungen in Form von Network Attached Storage (NAS) und Storage Area Network (SAN).

3.1.3 Netzwerk Lösungen

Bei der NAS Lösung werden die Speichermedien direkt ins das lokale Netzwerk (LAN) eingebunden und hierüber nutzbar gemacht. Dieses Vorgehen hat jedoch den Nachteil, dass die vorhandene Netzwerk-Umgebung für den Datentransfer zwischen den Rechnern und den Speichereinheiten genutzt und dadurch das Netzwerk mit zusätzlichen Datenpaketen, die durch den Datenaustausch mit dem Speichermedium entstehen, belastet wird.

Abbildung 9: Network Attached Storage Lösung

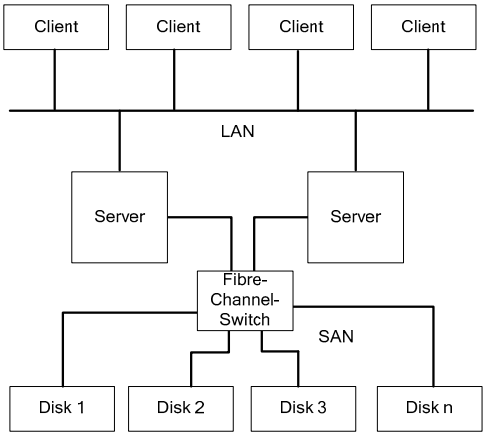


Quelle: Mandl, P. (2010), S. 278

[27] vgl. Mandl, P. (2010), S. 273ff

Diesen Schwachpunkt umgehen die SAN-Lösungen, indem sie ein dediziertes Netzwerk zwischen den Rechnern und den Speichereinheiten aufbauen. Dieses Netzwerk dient dabei alleinig der schnellen Datenübertragung zwischen den einzelnen Speichermedien und den an ihnen angeschlossenen Rechnern und wird i.d.R. über Glasfaserkabel (Fibre Channel) zur Verfügung gestellt.

Abbildung 10: Storage Area Network Lösung



Quelle: Mandl, P. (2010), S. 279

Ein weiterer Vorteil des SAN ist die mögliche Virtualisierung der vorhandenen Struktur, wodurch eine örtliche Verteilung der einzelnen Speichermedien ermöglicht wird. Für den Nutzer des Systems stellt sich das SAN weiterhin als eine logische Speichereinheit dar. Die Anbindung über Glasfaserkabel verspricht hohe Bandbreiten und mittels eines Fibre-Channel-Switches können zwischen einzelnen Rechnern und Speichermedien Punkt-zu-Punkt Übertragungswege aufgebaut werden, was einen hohen Datendurchsatz gewährleistet.

Als Alternative zu den Glasfaserkabeln existiert in diesem Bereich auch noch die iSCSI Technologie, die die Übertragung im SAN über die Standard Internet-Protokolle TCP/ IP ermöglicht. Dadurch kann der kostenintensive Aufbau eines Glasfaserkabel-Netzes umgangen und evtl. vorhandene Netzwerkkomponenten wiederverwendet werden.[28]

[28] vgl. Mandl, P. (2010), S. 278f

3.2 Relationale Datenbanken

3.2.1 Relationenmodell

Das Konzept der relationalen Datenbanken geht auf ein Relationenmodell von Hrn. Codd zurück, das dieser in den 1970er Jahren bei IBM entwickelt hat. Dieses Modell beruht auf der Mengentheorie aus der Mathematik und fasst Entitäten eines Typs zu einer Menge – einer Relation – zusammen. Jede Entität hat dabei einen festen Satz von Attributen und jedes Attribut einen definierten Wertebereich – auch Domäne genannt.[29]
Eine Relation lässt sich in mathematischer Form auch als "Teilmenge des kartesischen Produkts (Kreuzprodukts) der n Domänen: $R \subseteq D_1 \times \ldots \times D_n$"[30] ausdrücken.

Die Attribute einer Relation werden zudem näher spezifiziert, indem jedes Attribut eine Definition erhält, die den eindeutigen Namen, einen Datentyp sowie den bereits genannten Wertebereich umfasst. Die Summe dieser Definitionen wird auch als Schema der Relation bezeichnet.[31] Besondere Bedeutung erlangt in diesem Zusammenhang der Schlüssel einer Relation, welcher die "minimale Menge von Attributen [beschreibt], deren Werte das zugeordnete Entity eindeutig innerhalb aller Entities seines Typs identifiziert".[32] Dieser Schlüssel kann auch künstlich geschaffen werden – bspw. durch das Hinzufügen einer eindeutigen, fortlaufenden Nummer. Einzelne Ausprägungen dieser Entitätstypen, für die gilt: $(e_1, e_2, e_3, \ldots, e_n) \in R$, sind daher Elemente der Relation und werden allgemein auch als Tupel bezeichnet.

Zusätzlich zu den Relationen können in dem Modell von Codd auch die Beziehungen zwischen den Relationen erfasst werden. Die Beziehungen lassen sich als kartesisches Produkt der an der Beziehung beteiligten Entitätstypen darstellen: $B \subseteq E_1 \times E_2 \times \ldots \times E_n$, wobei n den Grad der Beziehung angibt. Diese sind in der Praxis meistens binär – also als Beziehung zwischen zwei Relationen definiert. Die Beziehung zwischen diesen beiden Typen lässt sich über ihre Funktionalität charakterisieren und ist entweder eine:

- **1:1 Beziehung**: jeder Entität aus E_1 wird höchstens eine Entität aus E_2 zugeordnet und umgekehrt jeder Entität aus E_2 ist höchstens eine Entität aus E_1 zugewiesen,

---

[29] vgl. Stahlknecht, P. / Hasenkamp, U. (2005), S. 172f

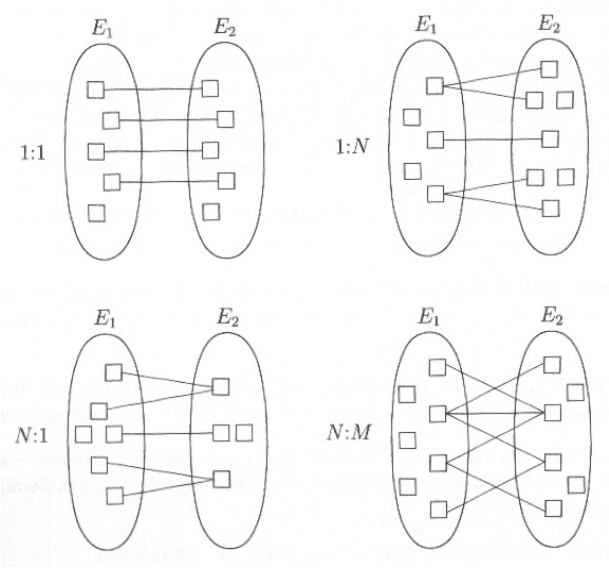
[30] Kemper, A. / Eickler, A. (2011), S. 71

[31] vgl. Kemper, A. / Eickler, A. (2011), S. 71f

[32] Kemper, A. / Eickler, A. (2011), S. 39

- **1:N Beziehung**: jeder Entität aus E_1 wird keine oder beliebig viele Entitäten aus E_2 zugewiesen, jedoch ist einer Entität aus E_2 höchstens eine Entität aus E_1 zugeordnet,
- **N:1 Beziehung**: entsprechend der 1:N Beziehung jedoch spiegelverkehrt oder
- **N:M Beziehung**: es gelten keine Restriktionen zwischen E_1 und E_2, jedes Element der beiden Entitätstypen kann mit keinem oder beliebig vielen Elementen des anderen Typs verbunden sein.

Abbildung 11: mögliche Beziehungen im Relationenmodell



Quelle: Kemper, A. / Eickler, A. (2011), S. 41

Diese Funktionalität stellt somit eine Integritätsbedingung dar, die für alle Entitäten der beteiligten Typen gilt und deren Einhaltung vom Datenbank System erzwungen werden muss.[33]

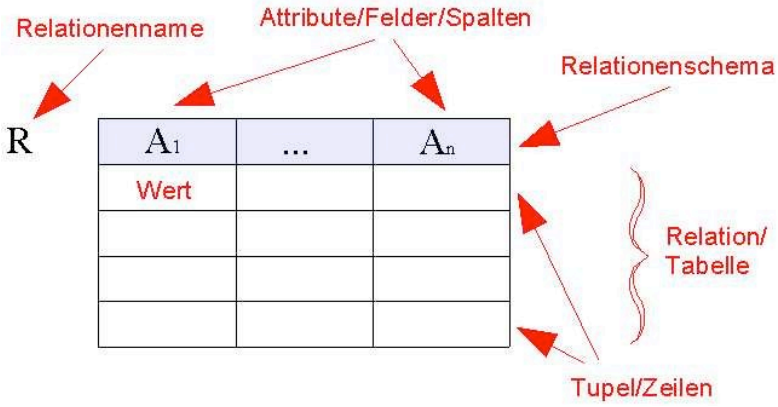
3.2.2 Abbildung des Modells

Nach dem Modell von Codd werden diese mathematischen Konstrukte auf zweidimensionale Tabellen mit Spalten und Zeilen abgebildet. Eine Tabelle stellt den Typ einer Entität und somit eine Relation dar. Die Spalten der Tabelle entsprechen den Attributen und eine Zeile in der Tabelle repräsentiert ein Tupel respektive eine Entität mit ihren spezifischen Attributwerten innerhalb der Relation.

---

[33] vgl. Kemper, A. / Eickler, A. (2011), S. 39f

Abbildung 12: Abbildung einer Relation auf eine Tabelle



Quelle: Manhard, K. (2008a)

Doch bevor das Relationenmodell auf Tabellen abgebildet werden kann, muss es einen Normalisierungsprozess durchlaufen.[34] "Dabei werden

- zuerst die in der Regel in unnormalisierter Form vorliegenden Relationen in die 1. Normalform überführt und
- dann schrittweise in weiteren Normalformen Relationen entwickelt, in denen Redundanzen verringert werden."[35]

Um die erste Normalform zu erhalten, müssen die eventuell vorhandenen Wiederholungsgruppen in den Attributwerten einer Entität beseitigt werden. Dazu sind diese Entitäten mehrfach zu duplizieren und dabei jeweils mit einem der Attributwerte zu versehen. Dieses Vorgehen führt unweigerlich zu Redundanzen in den Tupeln dieser Relation. Um diese zu vermeiden und das Modell in nachfolgende Formen zu transferieren, müssen zusätzliche Relationen und Beziehungen zum Modell hinzugefügt werden.

Befindet sich das Modell in der ersten Normalform wird für jeden Typ überprüft, inwieweit alle Nichtschlüssel-Attribute des Typs funktional alleinig und in Gesamtheit vom Schlüssel der Relation abhängig sind. Dies sind die Vorraussetzungen für das Erreichen der zweiten Normalform des Modells. Das Modell befindet sich anschließend in der dritten Normalform, wenn auch zwischen den Nichtschlüssel-Attributen einer Relation keine weiteren funktionalen Abhängigkeiten existieren.

---

[34] vgl. Stahlknecht, P. / Hasenkamp, U. (2005), S. 172f

[35] Stahlknecht, P. / Hasenkamp, U. (2005), S. 173

Evtl. Verstöße gegen diese Regeln sind durch das Hinzufügen von neuen Relationen und Beziehungen zum bereits bestehenden Modell aufzulösen.[36]

In der Praxis wird generell die dritte Normalform benutzt, um die Entitätstypen mit ihren Attributen und Beziehungen in ein Tabellenmodell zu überführen. Zur Abbildung der Beziehungen zwischen zwei Relationen E_1 und E_2 ist dabei abhängig von der Funktionalität der Beziehung folgendes Vorgehen zu wählen:

- **1:1 Beziehung**: da der Schlüssel der beiden Relationen gleich sein muss, werden die zum Schlüssel gehörenden Attribute zwischen beiden Relationen abgeglichen, so dass im Anschluss beide Relationen über dieselben Schlüsselattribute verfügen,
- **1:N Beziehung**: die Schlüsselattribute von E_1 werden als zusätzliche Attribute in das Schema der Relation E_2 aufgenommen, gehören jedoch nicht zu deren Schlüssel; diese Attribute werden auch Fremdschlüssel genannt,
- **N:1 Beziehung**: analog der 1:N Beziehung jedoch seitenverkehrt – die Schlüsselattribute von E_2 werden Bestandteil des Schemas von E_1, gehören jedoch nicht zu dessen Schlüssel; diese Attribute werden auch Fremdschlüssel genannt,
- **N:M Beziehung**: es wird eine neue Relation E_3 geschaffen, die sowohl die Schlüsselattribute von E_1 als auch die von E_2 übernimmt und sie zu ihren Schlüsselattributen macht. E_1 und E_2 referenzieren sich anschließend über E_3.

Auf der Basis der dritten Normalform und den zur Abbildung der bestehenden Beziehungen zwischen den einzelnen Relationen getätigten Erweiterungen kann nun die Umwandlung des Modells auf ein Tabellenmodell erfolgen. Hierbei werden aus den einzelnen Relationen und deren Attributen Tabellen mit den entsprechenden Spalten erstellt. Der Datentyp der Spalten muss den Wertebereich der Attribute aufnehmen können. Attribute, die zum Schlüssel der Relation gehören, werden als Primärschlüssel-Spalten in die Tabellenabbildung übernommen. Beziehungen zwischen zwei Relationen, die über eine Einschränkung der möglichen Attribut-Ausprägungen mit Hilfe von Fremdschlüssel-Attributen in einer der beteiligten Relationen modelliert wurden, sind in die tabellarischen Form mit Hilfe von Fremdschlüssel-Einschränkungen auf den zugehörigen Spalten zu überführen.[37]

---

[36] vgl. Bodendorf, F. (2006), S. 21f

[37] vgl. Kemper, A. / Eickler, A. (2011), S. 73ff

3.2.3 Relationale Datenbank-Management-Systeme

Um die Entwicklung von Anwendungen auf Basis des relationalen Tabellenmodells zu vereinfachen, sind die grundlegenden Funktionen zum Verwalten, Abrufen und Manipulieren von Tabellen sowie zur Sicherstellung von Datenschutz und -konsistenz innerhalb der Tabellen in einem relationalen Datenbank Management System (RDBMS) vereint worden. Dieses setzt sich aus mehreren Schichten zusammen. Die physische Schicht beschreibt, in welchem Format die Daten auf den Datenspeichern abgelegt werden. Hierbei werden die Tabellen eines Tabellenmodells i.d.R. zu einer Datenbank zusammengefasst und in einer oder mehreren zusammengehörigen Dateien auf der Festplatte des Systems gespeichert. Die Verwaltung dieser Dateien inkl. deren Speicherort und Größe auf dem Speichermedium obliegt der Hoheit des RDBMS.

Da das RDBMS als einheitliches Datenbankverwaltungssystem zum Einsatz kommt, kann es mehr als nur eine Datenbank zeitgleich beherbergen. Ein entsprechender Zugriffsschutz auf die Datenbank und auf einzelne Tabellen einer Datenbank wird dabei durch das RDBMS sichergestellt. Das System ermöglicht und steuert außerdem den parallelen Zugriff auf die Daten von verschiedenen Anwendungen heraus – dies geschieht vorrangig zur Gewährleistung der Datenintegrität und -konsistenz.

Aus Endnutzer-Sicht bietet das RDBMS eine logische Sicht auf die Tabellenstruktur sowie eine flexible Abfragesprache, um die Daten aus den Tabellen auszulesen. Diese Structured Query Language (SQL) ist als ISO Standard normiert worden.[38] Sie erlaubt als deklarative Sprache die Formulierung von Anweisungen an das RDBMS, um die gängigen Operationen:

- **CREATE**: zum Hinzufügen neuer Zeilen in eine Tabelle,
- **READ**: zum Abfragen von Daten aus ein oder mehreren Tabellen,
- **UPDATE**: zum Verändern von existierenden Daten in einer Tabelle und
- **DELETE**: zum Löschen vorhandener Daten aus einer Tabelle

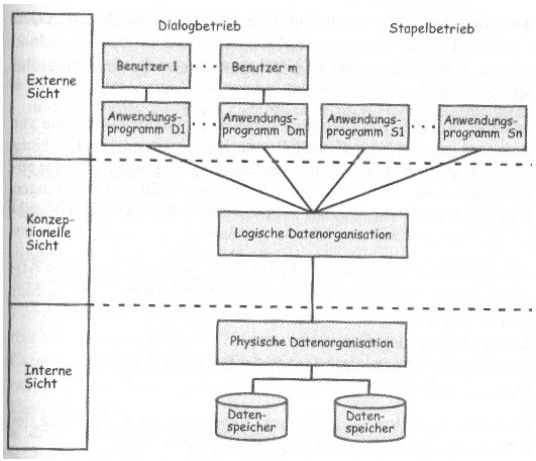
auf den Datenbestand einer Datenbank auszuführen.[39]

---

[38] vgl. Stahlknecht, P. / Hasenkamp, U. (2005), S. 183ff

[39] vgl. Redmond, E. / Wilson, J. (2011), S. 15

Abbildung 13: Aufbau eines relationalen Datenbank Management Systems



Quelle: Stahlknecht, P. / Hasenkamp, U. (2005), S. 187

Zudem ist im SQL Standard die Möglichkeit vorgesehen, einfache Gruppierungen und Summenfunktionen auf den Daten des Systems auszuführen. Mit Ersterem lassen sich die Daten ad hoc unter Einsatz der Summenfunktionen nach verschiedenen Kriterien konsolidieren. Ohne Anwendung einer Gruppierung arbeiten die Summenfunktionen auf der gesamten Datenmenge einer Abfrage und liefern als Ergebnis immer einen Einzelwert zurück.[40]

Bei umfassenden Datenabfragen muss das RDBMS die zugehörigen Daten in den Tabellen ausfindig machen. Dazu werden die Daten der Tabellen von der Festplatte in den Hauptspeicher geladen und u.U. zeilenweise durchsucht. Um hier eine Optimierung der Abfragegeschwindigkeit zu erreichen, erlaubt das RDMBS es, zusätzliche Indizes für einzelne Attribute einer Tabelle anzulegen. Für den Primär-Schlüssel einer Tabelle wird bereits beim Anlegen ein Index eingerichtet. Für die Nichtschlüssel-Attribute der Tabelle muss je nach Datentyp und Abfragekriterium entschieden werden, ob zusätzliche Indizes notwendig sind und welche Art von Index für einen schnelleren Zugriff auf die Daten sinnvoll erscheint. Hierbei helfen die RDBMS, indem sie Statistiken für die Ausführung einer Abfrage zur Verfügung stellen. Diese geben Aufschluss darüber, mit welchen Mechanismen, in welchen Schritten und der dazu jeweils benötigten Zeit die Abfrage vom RDBMS ausgeführt wurde.[41]

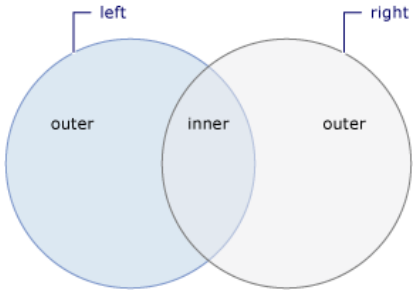
---

[40] vgl. Redmond, E. / Wilson, J. (2011), S. 23

[41] vgl. Redmond, E. / Wilson, J. (2011), S. 20

Eine Besonderheit der RDBMS ist die Zusammenführung von Daten aus zwei Tabellen mit Hilfe von sogenannten Joins zu einer neuen, den Daten aus beiden Tabellen umfassenden Ansicht. Hierbei muss dem RDBMS mitgeteilt werden, über welche Spalten die beiden Tabellen miteinander in Verbindung gesetzt werden sollen. Werden keine Spalten für die Übereinstimmungsprüfung spezifiziert, so bildet das RDBMS ein Kreuzprodukt aus den Zeilen der beiden Tabellen (CROSS JOIN). Abhängig vom Anwendungsfall können die beiden Tabellen auch per INNER JOIN so miteinander verbunden werden, dass nur die Zeilen mit inhaltlich passenden Informationen aus den beiden Tabellen zurückgeliefert werden. Alternativ kann über ein OUTER JOIN eine Ergebnismenge produziert werden, die alle Daten einer der beiden Tabellen enthält und fehlende Informationen aus der zweiten Tabelle mit NULL Werten auffüllt. Je nach Richtung des Joins spricht man hierbei von LEFT JOIN oder RIGHT JOIN.[42]

Abbildung 14: mögliche JOIN Verbindungen zwischen 2 Tabellen



Quelle: MSDN (2012)

Um die Datenintegrität und -konsistenz in der Datenbank sicherzustellen, werden alle Änderungsanweisungen vom RDBMS über eine Datenbanktransaktion abgesichert. Diese kann entweder implizit durch das RDBMS oder explizit durch den Anwender vorgegeben werden. Daten, die sich in einer Änderungstransaktion befinden, sind für andere Anwender des Systems noch nicht sichtbar. Erst bei erfolgreichem Abschluss der Transaktion werden die modifizierten Daten auf dem Datenträger persistiert und anderen Nutzern zur Verfügung gestellt. Tritt bei der Verarbeitung eines der Kommandos in der Transaktion ein Fehler auf, so werden alle vorher ausgeführten Änderungen an den Daten und Strukturen in der Datenbank rückgängig gemacht.

---

[42] vgl. Redmond, E. / Wilson, J. (2011), S. 17ff

Durch diese Maßnahme werden die ACID Anforderungen von geschäftlichen Anwendungssystemen durch das RDBMS erfüllt. ACID steht hierbei für:

- **Atomic**: entweder alle Anweisung sind erfolgreich oder keine davon,
- **Consistent**: die Datenkonsistenz wird in jedem Augenblick der Transaktion gewährleistet,
- **Isolated**: Anweisungen in unterschiedlichen Transaktionen können sich nicht gegenseitig beeinflussen und
- **Durable**: sobald die Transaktion erfolgreich abgeschlossen wird, sind die geänderten Daten physikalisch auf dem Datenspeicher abgelegt und selbst im Falle eines Systemausfalls noch rekonstruierbar.[43]

Sämtliche Änderungsoperationen werden dabei vom System in einer Log-Datei protokolliert, um im Falle eines Systemausfalls auf einen konsistenten Zustand zurückzukehren. Diese Log-Datei besteht parallel zu den eigentlichen Datendateien einer jeden Datenbank und ist häufigen Schreiboperationen ausgesetzt, da jede Transaktion mit ihren Aktionen hierin aufgezeichnet wird.[44]

Aufgrund der ACID Eigenschaften, die jedes aktuelle RDBMS erfüllt, werden die Systeme auch vorrangig im Umfeld von transaktionsbasierten Anwendungssystemen (OLTP) eingesetzt – sei es für die Auftragsverwaltung, die Personalabrechnung oder als Bestandteil eines Abverkaufssystems.[45]

Da auch die Schemata der einzelnen Tabellen einer Datenbank in den Datenstrukturen des Datenbank-Management-Systems verwaltet werden, können über spezifische SQL Anweisungen nachträgliche Änderungen am Aufbau der Datenbank und an den Tabellen vorgenommen werden. Die Umsetzung dieses Teilbereiches der SQL Sprache ist jedoch teilweise abhängig vom Hersteller und den Fähigkeiten des RDBMS.[46] Der Funktionsumfang der Datenbank-Management-Systeme kann sich dabei durchaus gravierend voneinander unterscheiden, da aktuelle RDBMS auf einer Vielzahl von Geräten und in diversen Umgebungen – angefangen vom Smartphone bis zu verteilten

---

[43] vgl. Hewitt, E. (2010), S. 7

[44] vgl. Kemper, A. / Eickler, A. (2011), S. 298f

[45] vgl. Kemper, A. / Eickler, A. (2011), S. 523

[46] vgl. Kemper, A. / Eickler, A. (2011), S. 22ff

Unternehmensanwendungen – im Einsatz sind. Das erklärt auch die weite Verbreitung dieser Systeme. Wurde die Wahl für ein System getroffen, sollten jedoch auch dessen herstellerspezifische Eigenheiten genutzt werden, um eine bestmögliche Performance für die Anwendung von der Datenbankseite zu erhalten.[47]

3.3 Multi-dimensionale Datenbanken

3.3.1 Einsatzgebiete eines Data Warehouse

Während sich die relationalen Datenbanken gut für den Einsatz in transaktionsgesteuerten Umgebungen eignen, sind sie jedoch für analytische Problemstellungen, die auf einer großen Datenbasis basieren, weniger sinnvoll.[48]

Ein wesentlicher Grundsatz beim Einsatz von relationalen Datenbanken im OLTP Umfeld ist, dass Transaktionen und deren Änderungen an vorhandenen Daten möglichst klein gehalten werden, um die Verarbeitungsgeschwindigkeit beim gleichzeitigen Eintreffen von vielen Anfragen zu gewährleisten.[49]

Um aus dieser großen Menge von gesammelten Transaktionsdaten mit deren feiner Granularität jedoch wertvolle Informationen und Zusammenhänge ablesen zu können, müssen diese Daten nach Mustern durchsucht und sinnvoll zusammengefasst werden. Zwar bietet die SQL Abfragesprache der RDBMS entsprechende Unterstützung für Gruppierungen und Summenfunktionen, doch wächst deren Ausführungsdauer und Ressourcenbeanspruchung mit dem Umfang der in den Tabellen gespeicherten Daten exponentiell an. Dadurch könnten andere Bereiche des RDBMS in Mitleidenschaft gezogen werden – bspw. eine mögliche Verlangsamung der Transaktionsverarbeitung. Daher wurden spezielle Datenbanksysteme entwickelt, die auf die schnelle Analyse und Auswertung von großen Datenbeständen spezialisiert sind. Diese werden auch mit dem Namen Online Analytical Processing (OLAP) Systeme und die von ihnen verwaltete, aus den Daten verschiedener OLTP Systemen aufgebaute Datenbank auch als Data Warehouse bezeichnet.[50]

---

[47] vgl. Redmond, E. / Wilson, J. (2011), S. 29

[48] vgl. Kemper, A. / Eickler, A. (2011), S. 529f

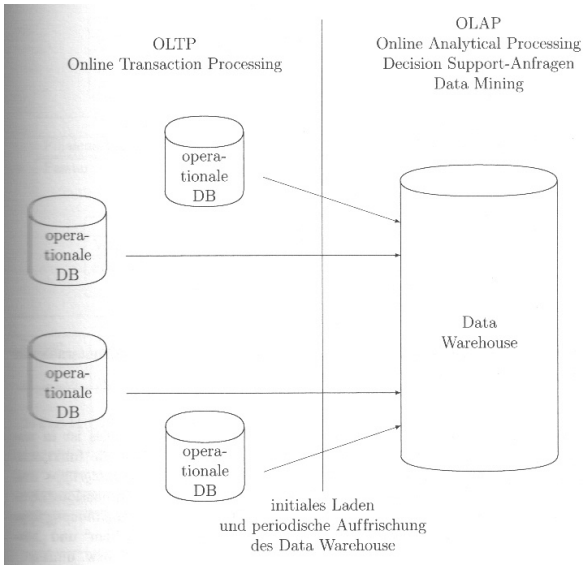
[49] vgl. Hewitt, E. (2010), S. 7f

[50] vgl. Kemper, A. / Eickler, A. (2011), S. 529f

3.3.2 ETL Prozess

Um die Daten aus den operativen Datenbanken in ein Data Warehouse zu überführen, werden diese durch einen ETL Prozess geschleust. ETL steht hierbei für Extract, Transform, Load und beschreibt in groben Zügen bereits die einzelnen Schritte, die dabei ausgeführt werden. Ziel des Prozesses ist es daher, die in verschiedenen Datenquellen vorkommenden Daten aus dem operativen Geschäft periodisch wiederkehrend zu extrahieren, zusammenzuführen und in das neue Schema des Data Warehouse zu überführen. Die Ansprüche an die Datenqualität in einem Data Warehouse sind dementsprechend hoch angesiedelt, da es die teilweise unternehmensweit verstreuten Daten in eine gemeinsame Datenstruktur überführen und für Auswertungen auch über einen längeren Zeitraum zur Verfügung stellen soll.[51]

Abbildung 15: Extraktion der Daten in ein Data Warehouse



Quelle: Kemper, A. / Eickler, A. (2011), S. 531

3.3.3 Das Sternschema

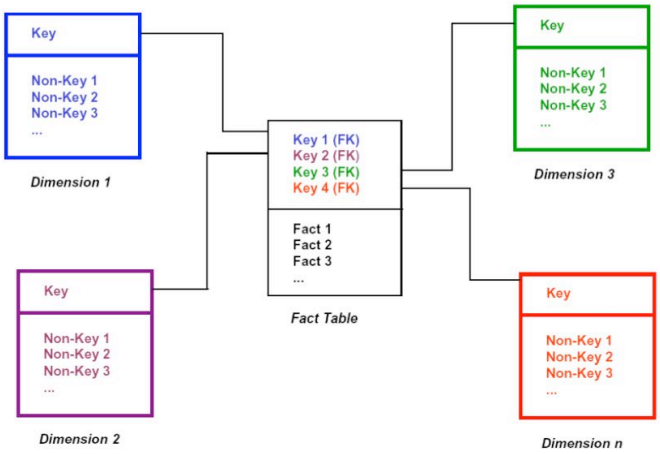
“Als Datenbankschema für Data Warehouse-Anwendungen hat sich das sogenannte Sternschema (engl. star schema) durchgesetzt. Dieses Schema besteht aus einer Faktentabelle und mehreren Dimensionstabellen, die über Fremdschlüsselbeziehungen mit der Faktentabelle verbunden sind.”[52]

---

[51] vgl. Bodendorf, F. (2006), S. 36ff

[52] Kemper, A. / Eickler, A. (2011), S. 530

Abbildung 16: Aufbau des Sternschemas



Quelle: Manhard, K. (2008b)

Die Bewegungsdaten (Transaktionsdaten) aus den operativen Datenbanken werden über den ETL Prozess in die Faktentabelle überspielt, die damit über einen enormen Umfang an Datensätzen verfügt. Die Dimensionstabellen enthalten hingegen Stammdaten und Meta-Informationen. Sie sind i.d.R. weniger umfangreich bemessen als die Faktentabelle und nicht normalisiert. Über die Verbindung zur Faktentabelle geben sie Aufschluss über die Bedeutung verschiedener Attribute in der Faktentabelle – bspw. den Produktnamen zur Produkt-Kennung. Dieses Datenbankschema ließe sich auch noch in einem RDBMS ablegen, doch würde die Geschwindigkeit der Abfragen mit zunehmendem Umfang der Faktentabelle stark abnehmen. Daher haben sich hier am Markt alternative, für die Abfrage komplexer Informationen optimierte Datenstrukturen etabliert.[53]

3.3.4 Mehrdimensionale Räume

Die Datenstrukturen gehen dabei von einer mehrdimensionalen Sicht auf die Daten aus. Diese werden nach abhängigen Attributen, den sogenannten Kennzahlen oder Fakten (engl. Facts), und nach unabhängigen Attributen – den Dimensionen – klassifiziert. Die unabhängigen Attribute spannen dabei einen Vektorraum auf, auf dessen jeweiligen Achsen die Werte eines Attributes zu finden sind. Diese Werte entstammen einer Dimensionstabelle aus dem Sternschema.

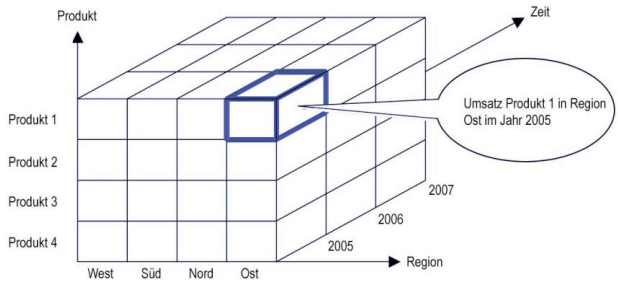
---

[53] vgl. Engels, C. (2009), S. 129ff

Den abhängigen Attributen kann über ihre vorhandenen Zuordnungen zu den einzelnen Dimensionen eine bestimmte Position in diesem Vektorraum zugewiesen werden. Diese Daten kommen aus der Faktentabelle des Sternschemas. Durch diese Abbildung ist ein schneller, lesender Zugriff auf die Daten in den jeweiligen Kennzahlen möglich, da sie sich leicht über die in der Abfragebedingung angegebenen Dimensionen im Raum ausfindig machen lassen.

In den Dimensionen können zudem Abhängigkeiten in den einzelnen Werten mit Hilfe von Hierarchien dargestellt werden – bspw. in der Kalenderdimension: Jahr - Quartal - Monat - Woche - Tag. Die Daten in einer mehrdimensionalen Datenbank werden dabei automatisch anhand der Dimensionen und Hierarchien konsolidiert abgelegt und können daher mittels der integrierten Drill-Down oder Roll-Up Operationen rasch aufgebrochen oder zusammengefasst werden.[54]

Abbildung 17: Daten in einem 3-dimensionalen Raum



Quelle: Manhard, K. (2008c)

Abbildung 18: Drill-Down/ Roll-Up Operationen

	Jahresumsatz
Produkt A	345 Mio. €
Produkt B	113 Mio.€
Produkt C	232 Mio. €

Drill-down ⇑ ⇓ Roll-up

	Jahresumsatz
Produktgruppe ABC	690 Mio. €

Quelle: Manhard, K. (2008c)

[54] vgl. Bodendorf, F. (2006), S. 40ff

3.3.5 Data Mining

Während die bisherige Betrachtung der OLAP Systeme die Analyse und Auswertung der existierenden, bekannten Datenbasis eines Unternehmens zum Ziel hatte, gehen die Data Mining Verfahren einen Schritt weiter und suchen in den gesammelten Daten nach wiederkehrenden Mustern, um neue, noch unbekannte Zusammenhänge aus den Daten ableiten zu können. Zu ihren Methoden zählen diverse Verfahren aus der Statistik – bspw. Regressions-, Faktoren- und Clusteranalysen.[55] Ihre Einsatzzwecke sind somit vielfältig: so können über ein vorher definiertes und erprobtes Schema Annahmen oder Vorhersagen über Objekte bzw. deren Verhalten getroffen werden. Ebenfalls lassen sich hierüber gleichartige Objekte in den Daten finden und zu Gruppen zusammenfassen.[56]

3.4 NoSQL Datenbanken

Wie auch bereits bei der Auswertung und Analyse der operativen Daten eines Unternehmens neue Wege und Strukturen für den optimierten Datenzugriff notwendig geworden sind, haben sich auch mit der zunehmenden Verbreitung des Internets und den darauf laufenden Web Anwendungen spezialisierte Datenbanken etabliert. Diese lassen sich grob in die vier Kategorien: Key-Value-Stores, spaltenorientierte Datenbanken, dokumentorientierte Datenbanken und Graph-Datenbanken einteilen und werden unter dem Schlagwort NoSQL Datenbanken zusammengefasst.

3.4.1 Key-Value-Store

Die Datenbanken aus dieser Kategorie haben ein sehr einfaches Datenmodell, da sie zu einem eindeutigen Zugriffsschlüssel lediglich einen einzigen Wert speichern können. Der Inhalt dieses Wertes kann aus strukturierten, semi-strukturierten oder auch unstrukturierten Daten bestehen. Ein alphanumerischer Schlüssel wird für den Zugriff auf die dahinter liegenden Daten benötigt. Man kann sich daher ein Dateisystem auch als Key-Value-Store vorstellen, wobei über den kompletten Dateinamen (Key) auf den Dateiinhalt (Value) zugegriffen wird.[57] Wie im Dateisystem können auch in einem Key-Value-Store die Schlüssel hierarchisch aufgebaut sein.

---

[55] vgl. Bodendorf, F. (2006), S. 46ff

[56] vgl. Kemper, A. / Eickler, A. (2011), S. 555ff

[57] vgl. Redmond, E. / Wilson, J. (2011), S. 6

Je nach Implementierung werden dafür Buckets oder Namespaces verwendet – beide Techniken beschreiben jedoch einen virtuellen Container, in dem die Schlüssel mit ihren Daten abgelegt werden sollen. Um sinnvolle Container und Schlüssel für die zu speichernden Daten zu erstellen, müssen die auf den Key-Value-Store abgesetzten Anfragen bereits vorher bekannt sein. Der komplette Zugriffsschlüssel – bestehend aus dem Container und dem eigentlichen Schlüssel – wird in einem Hash-Index abgelegt, um zügige Abfragen nach diesem Schlüssel zu ermöglichen. Die Aufnahme weiterer Indizes ist nicht vorgesehen. Aufgrund des Designs werden die Daten in einem Key-Value-Store nicht normalisiert abgelegt. Redundanzen lassen sich somit nicht vermeiden und Beziehungen auf Datenmodell-Ebene sind nicht möglich. Dadurch müssen Daten auch beim Auslesen nicht wieder mittels Join Bedingungen zusammengeführt werden. Es liegt daher in der Verantwortung der Applikation, die Integrität der Daten innerhalb eines Key-Value-Stores sicherzustellen.[58]

3.4.2 Spaltenorientierte Datenbank

Im Gegensatz zu den relationalen Datenbanken, die ihre Informationen in Zeilen von zwei-dimensionalen Tabellen mit einem festem Schema – welches durch die Spaltendefinitionen einer jeden Tabelle vorgegeben wird – ablegen, werden Daten in einer spaltenorientierten Datenbank nach Spalten zusammengefasst und gespeichert. Was zunächst nach einem kleinen Unterschied klingt, hat jedoch für das Design und den Betrieb der Datenbank essentielle Auswirkungen. Die Daten müssen nicht mehr einem festen Schema folgen und jede Zeile, die ähnlich wie bei den Key-Value-Stores über einen eindeutigen, alphanumerischen Schlüssel identifiziert wird, kann eine unterschiedliche Anzahl von Spalten aufweisen. Sie sind daher konzeptionell zwischen den Key-Value-Stores und den relationalen Datenbanken anzusiedeln.[59]

Diese Architektur hat nicht nur den Vorteil das Schemaänderungen zur Laufzeit einfach möglich werden, sondern bedingt auch eine platzsparende und verbesserte Speicherung der Daten, da innerhalb der Zeilen nur diejenigen Spalten gesichert werden müssen, die auch Daten beinhalten.

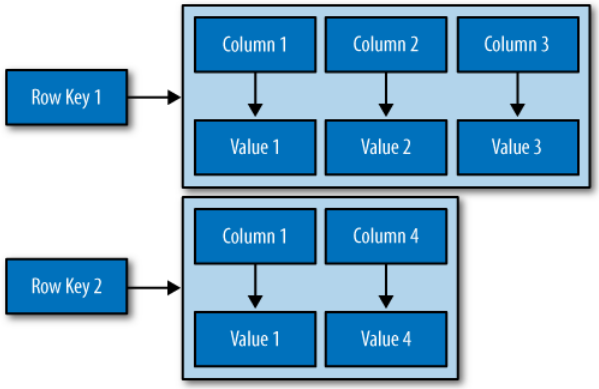
---

[58] vgl. Hewitt, E. (2010), S. 261f

[59] vgl. Redmond, E. / Wilson, J. (2011), S. 7

Der Inhalt einer Spalte wird hierbei i.d.R. durch ein Schlüssel-Wert-Paar repräsentiert. Mehrere logisch zusammenhängende Schlüssel-Wert-Paare können über eine übergeordnete Spaltenfamilie gruppiert werden, was am ehesten noch mit einer Tabelle in der relationalen Welt verglichen werden kann. Der Zugriff auf die Daten erfolgt wie bei den Key-Value-Stores über den eindeutigen Zugriffsschlüssel für die Zeile, der ggf. gesetzten Spaltenfamilie und dem Schlüssel für den Spaltenwert.[60]

Abbildung 19: Struktur einer spaltenorientierten Datenbank



Quelle: Hewitt, E. (2010), S. 44

Wie auch schon die Key-Value-Stores verzichten die spaltenorientierten Datenbanken auf die Nutzung von Join Bedingungen in Abfragen und der Erstellung weiterer Indizes zum allzeit bestehenden Zeilenschlüssel-Index. Für das Aufsetzen nützlicher Spaltenfamilien müssen die Abfragekriterien aus der Anwendung im Vorfeld bekannt sein. Um dabei die Zugriffsgeschwindigkeit auf die gespeicherten Daten je nach Kriterium zu erhöhen, sind die Daten in der spaltenorientierten Datenbank u.U. redundant in mehreren unterschiedlichen Spalten vorzuhalten, womit die Sicherstellung der Datenintegrität ebenfalls auf die Anwendungsseite verlagert wird.[61] Dies hat jedoch den entscheidenden Vorteil, dass eine mehrdimensionale Sicht auf die Daten ermöglicht werden kann. Dadurch sind spaltenorientierte Datenbanken gerade für Analyse- und Auswertungsaufgaben, wo Daten aus einer großen Menge an Informationen extrahiert und zusammengefasst werden müssen, prädestiniert.[62]

---

[60] vgl. Hewitt, E. (2010), S. 42f

[61] vgl. Hewitt, E. (2010), S. 56ff

[62] vgl. Hewitt, E. (2010), S. 264

3.4.3 Dokumentorientierte Datenbank

Der erste Schritt bei der Überführung eines Relationenmodells in die Tabellenform einer relationalen Datenbank ist die Normalisierung der Informationen. Dadurch werden verschachtelte Strukturen, die als Attributwerte im Modell auftauchen können, auf eigene Entitätstypen aufgebrochen. Um nachher im RDBMS wieder an diese zusammengehörenden Informationen zu gelangen, müssen die Abfragen alle benötigten Tabellen miteinander in Verbindung setzen. Dieses Vorgehen funktioniert jedoch nur für strukturierte Daten. Ist im Vorfeld das Schema der Daten nicht bekannt, wird eine Modellierung in der relationalen Datenbankwelt schwierig bis unmöglich. Hier bieten sich die dokumentorientierten Datenbanken an, die unter einem eindeutigen Zugriffsschlüssel ein beliebiges, semi-strukturiertes Dokument ablegen können.[63] Dokumente werden dabei logisch zu Kollektionen zusammengefasst, müssen aber innerhalb einer Kollektion nicht über dasselbe Schema respektive überhaupt über ein Schema verfügen. Jedes Dokument steht für sich allein und enthält alle Informationen, die zusammen mit dem Dokument erfasst bzw. in der Kollektion abgelegt wurden.

In den meisten Dokumentdatenbanken werden die Daten dabei als JSON-formatierter Text abgelegt. Dabei handelt es sich um die Darstellung eines kompletten Objektgraphen inkl. aller untergeordneten Objekte und Listen in semi-strukturierter Textform, wobei die Daten mit Hilfe von Schlüssel-Wert-Paaren repräsentiert werden. Dokumente können andere Dokumente entweder direkt beinhalten oder diese separat abgelegten Dokumente über deren eindeutigen Schlüssel referenzieren. Da jedes Dokument für sich steht, sind Redundanzen in den Informationen wahrscheinlich. Auch hier ist die Anwendungsseite in der Pflicht, die Integrität der Daten in der Datenbank sicherzustellen. Komplexe Abfragen werden von einigen dokumentorientierten Datenbanken wie bspw. MongoDB mit einer eigenen Abfragesprache unterstützt. Um ad hoc Abfragen auf die gespeicherten Dokumente zu ermöglichen, können abhängig vom Anbieter zusätzliche Indizes angelegt werden – in MongoDB bspw. bis zu 64 Indizes pro Kollektion.[64]

---

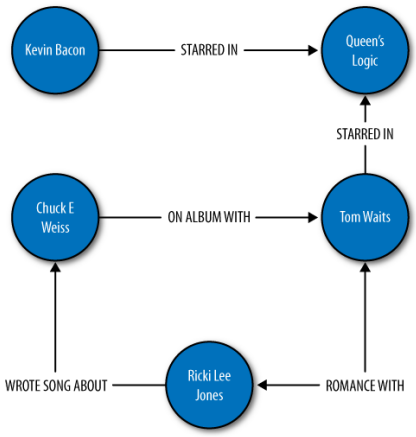
[63] vg. Hewitt, E. (2010), S. 250ff

[64] vgl. Banker, K. (2011), S. 3ff

3.4.4 Graph-Datenbank

Graph Datenbanken verfolgen einen komplett anderen Ansatz, indem sie mathematische Graphen bestehend aus Knoten und Kanten in ihrer Datenbank verwalten. Während die Beziehungen zwischen Relationen im relationalen Modell noch eine untergeordnete Rolle spielte, werden die Kanten in der Graph-Datenbank ebenso stark wie die Knoten gewichtet. Knoten und Kanten können dabei zusätzliche deskriptive Informationen in Form von Schlüssel-Wert-Paaren erhalten. Den Vorteil von Graph-Datenbanken erkennt man schon bei der Modellierung des Systems. Ist es möglich, die Informationen über Boxen und Verbindungslinien auf ein Whiteboard zu bringen, kann diese Darstellung so nahezu 1:1 in die Graph-Datenbank übernommen werden. Eine Normalisierung ist nicht notwendig. Durch die hervorgehobene Bedeutung der Verbindungen zwischen einzelnen Knoten haben Graph-Datenbanken einen enormen Vorteil bei der Erfassung, Analyse und Auswertung von stark miteinander verknüpften Daten wie sie bspw. in einem sozialen Netzwerk zu finden sind.[65]

Abbildung 20: Struktur eines Graphen in einer Graph-Datenbank



Quelle: Hewitt, E. (2010), S. 257

Die Daten in der Graph-Datenbank sind schemalos und ein bestehender Graph kann leicht mit neuen Knoten und Kanten erweitert werden. Redundanzen in den Knoten lassen sich durch die Wiederverwendung existierender Informationen vermeiden. Da die Speicherung von Zusatzinformationen in Form von Attributen an den einzelnen Knoten oder Kanten jedoch nur begrenzt sinnvoll ist, werden Graph-Datenbanken i.d.R. nicht

[65] vgl. Hewitt, E. (2010), S. 256ff

alleine sondern immer in Verbindung mit einem weiteren Datenbank-System betrieben. In den Attributen werden dann bspw. die Zugriffsschlüssel auf die Informationen in der parallel existierenden Datenbank abgespeichert.

Da der Graph in einer Graph-Datenbank immer zusammengehörend abgelegt und manipuliert wird, unterstützt dieser Typ von Datenbank als Einziger der hier vorgestellten NoSQL Datenbanken ebenfalls die ACID Eigenschaften mit Hilfe von Datenbank Transaktionen.[66]

4 Aufbau eines verteilten Systems

4.1 Komponenten eines verteilten Systems

Bei einem verteilten System handelt es sich um "eine Ansammlung unabhängiger Computer, die den Benutzern wie ein einzelnes kohärentes System erscheinen".[67] Die Systeme können aus verschiedenen, teilweise auch räumlich getrennten Komponenten bestehen, die jedoch insofern miteinander integriert sind, dass sie gemeinsam an der Lösung einer Aufgabe teilhaben. Diese Aufgabe kann in der Bereitstellung einer Ressource oder eines Dienstes bestehen (bspw. zentrale Datenhaltung, zentrales Buchhaltungssystem) respektive eine Erhöhung der Verarbeitungsleistung durch Parallelisieren der Prozesse zum Ziel haben.[68] Der Aufbau eines verteilten Systems lässt sich in einem Drei-Schichten-Modell darstellen, welches aus der Präsentationsschicht, der Anwendungsschicht und der Datenhaltungsschicht besteht.

4.1.1 Präsentationsschicht

"Auf der obersten Ebene befindet sich die Mensch-Maschine-Schnittstelle. Die Daten der darunter liegenden Ebene werden in der für den Anwender optimalen Art präsentiert und Benutzereingaben werden als Steuerungsbefehle weitergeleitet".[69] Diese Schicht ist nicht zwangsweise mit einer Benutzeroberfläche verbunden, kann sie doch im service-orientierten Architekturen durch die Dienstschnittstelle zum Aufrufen von Servicefunktionen ersetzt worden sein.

---

[66] vgl. Redmond, E. / Wilson, J. (2011), S. 225ff

[67] Tanenbaum, A. (2008), S. 19

[68] vgl. Schill, A. / Springer, T. (2007), S. 5f

[69] Schäfer, W. (2010), S. 204

Wichtig dabei ist, dass die Daten, die in der Präsentationsschicht verarbeitet werden, meist eine andere Repräsentation haben, als sie nachher auf den nachfolgenden Schichten vorliegen.[70] Im hier betrachteten Szenario besteht die Präsentationsschicht daher entweder aus einer auf HTML-basierenden Web Anwendung oder auf einem Web Dienst, der seine Funktionalität über SOAP-basierte Web Services oder als REST-basierte Dienste zur Verfügung stellt. Um die Anwendung für Clients verfügbar zu machen, muss diese Schicht auf einen oder mehreren Web Servern aufgespielt werden.[71]

4.1.2 Anwendungsschicht

Die Anwendungsschicht beherbergt die eigentliche Geschäftslogik der Anwendung. Sie besteht dazu aus domänenspezifischen Anwendungsobjekten, die losgelöst von der Repräsentation der Daten in der Präsentationsschicht und der Ablage der Daten in der Datenhaltungsschicht sind. Sie empfängt die Daten und Interaktionen von der vorgelagerten Schicht, verarbeitet diese und liefert die Ergebnisse der Verarbeitung an die Präsentationsschicht zurück. Sie steht damit als Vermittler zwischen Präsentations- und Datenhaltungsschicht, wobei sie zusätzlich zum stupiden Weiterleiten der Daten die Geschäftsregeln überprüft und die Datenkonsistenz sicherstellt.[72]

In unserem Szenario werden für die Bereitstellung der Geschäftslogik-Objekte i.d.R. Middleware-Komponenten wie Applikationsserver eingesetzt. Diese stellen eine Laufzeitumgebung für Anwendungsbausteine zur Verfügung und bieten den in ihnen laufenden Komponenten zusätzliche, hilfreiche Dienste in Form von bspw. Nachrichten-Warteschlangen, Transaktionsunterstützung, zur Berücksichtigung von Sicherheitsaspekten als auch zur Überwachung der Komponenten während der Laufzeit an. Alternativ kann die Geschäftslogik des Systems auch über service-orientierte Schnittstellen der auf SOAP- oder REST-basierende Dienste zur Verfügung gestellt werden. Die Server der Anwendungsschicht können ebenfalls in mehrfach redundanter Form bereitgestellt werden, um die Verarbeitungslast der zahlreichen Anfragen der vorgelagerten Schicht auf die verfügbaren Systeme aufzuteilen.[73]

---

[70] vgl. Schäfer, W. (2010), S. 204

[71] vgl. Tanenbaum, A. (2008), S. 589ff

[72] vgl. Schäfer, W. (2010), S. 205

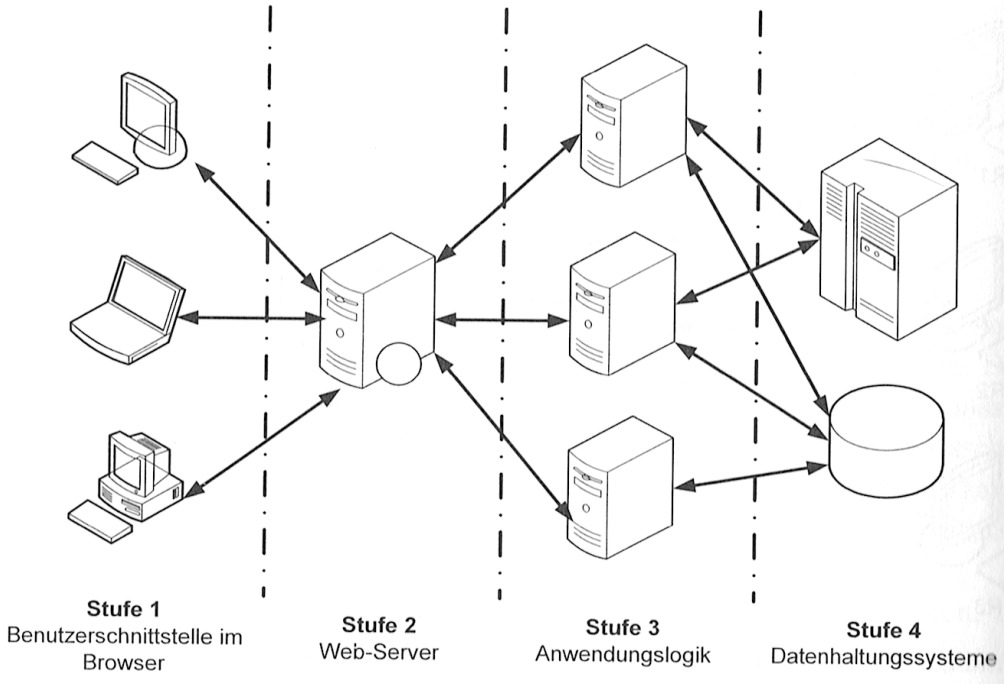
[73] vgl. Schill, A. / Springer, T. (2007), S. 208ff

4.1.3 Datenhaltungsschicht

"Die Datenhaltungsschicht als letztes Glied stellt Dienste und die notwendige Infrastruktur bereit, die Rohdaten zu speichern oder weiterzureichen".[74] Als wichtigste Aufgabe ist die dauerhafte Speicherung der Daten anzusehen. Die Daten können dabei entweder in einem Dateisystem oder einer Datenbank permanent abgelegt werden – die Entscheidung darüber ist abhängig von den Anforderungen der Anwendung.[75]

Die Datenhaltungsschicht ist dabei nicht limitiert auf einen einzigen Mechanismus, wie die Daten gespeichert werden. In umfangreichen Systemen kommen durchaus unterschiedliche Datenquellen mit u.U. verschiedenen Zugriffsmethoden und -strategien zum Einsatz.[76]

Abbildung 21: Das Schichten-Modell eines verteilten Systems



Quelle: Schill, A. / Springer, T. (2007), S. 330

[74] Schäfer, W. (2010), S. 205

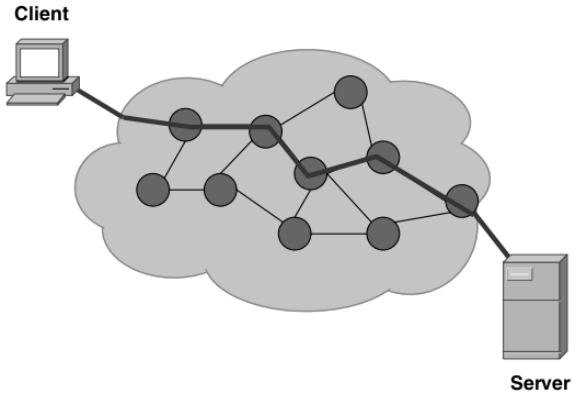
[75] vgl. Tanenbaum, A. (2008), S. 59

[76] vgl. Redmond, E. / Wilson, J. (2011), S. 222

4.2 Kommunikation in einem verteilten System

Ein verteiltes System kennzeichnet sich dadurch, dass Nutzer und Anbieter einer Anwendung oder eines Dienstes nicht direkt miteinander verbunden sind. Stattdessen tauschen beide Parteien gemeinsam über ein Computernetzwerk Nachrichten aus. Während aus Sicht der Anwendung die beiden Rechner direkt miteinander zu kommunizieren scheinen, müssen je nach Größe des angebundenen Netzwerkes mehrere Zwischenstellen beim Transport der Nachricht passiert werden.[77] Je nach Größe und Reichweite des Netzwerkes unterscheidet man hierbei das private, lokale Netzwerk (LAN) an einem dedizierten Standort – auch als Intranet bezeichnet – und das öffentliche Weitverkehrsnetzwerk (WAN), dessen bekannteste Ausprägung das Internet darstellt.[78]

Abbildung 22: Aufbau eines Computernetzwerkes



Quelle: Scherff, J. (2007), S. 7

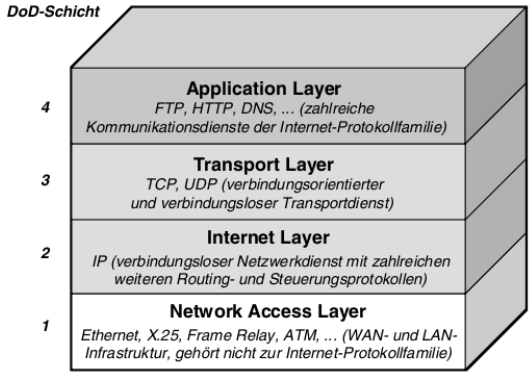
Die Kommunikation in diesen genannten Netzwerken findet i.d.R. über die TCP/IP Protokoll-Familie statt. Diese beschreibt die Regeln für den Aufbau und den Ablauf einer Verbindung zwischen zwei miteinander interagierenden Systemen. Sie ist in vier Schichten unterteilt: Network Access Layer, Internet Layer, Transport Layer und Application Layer, die jeweils unterschiedliche Aufgaben bei der Kommunikation übernehmen, wobei der Network Layer genau genommen nicht zur Protokoll Familie gehört, da er die physikalischen Voraussetzungen für die Verbindung zweier Endpunkte

[77] vgl. Scherff, J. (2007), S. 6ff

[78] vgl. Scherff, J. (2007), S. 27ff

umfasst.[79]

Abbildung 23: TCP/IP Protokoll Familie



Quelle: Scherff, J. (2007), S. 87

Nachfolgend werden die wichtigsten Protokolle und deren Eigenschaften für ein gemeinsames Verständnis grob skizziert.

4.2.1 Die Basisprotokolle TCP, IP und UDP

Das IP Protokoll, als Teil des Internet Layers der Protokoll Familie, bietet eine "verbindungslose, unzuverlässige Übertragung ohne Garantie (Best Effort Service), um einzelne Datenpakete (Datagramme) unabhängig voneinander von einem sendenden Endsystem (Source) über mehrere Netze hinweg (Internetworking) zu einem empfangenden Endsystem (Destination) weiterzuleiten (Forwarding). Während der Übertragung können Datenpakete verloren gehen oder außerhalb der gesendeten Reihenfolge ankommen, da sie u.U. über verschiedene Wege geleitet werden".[80] Aktuell im Einsatz ist weiterhin die IP Version 4, während bereits seit geraumer Zeit aufgrund der sich erschöpfenden Adressräume im IPv4 der Einsatz von IP Version 6 proklamiert wird. Ein Paket im IPv4 Format besteht aus einem mindestens 20 Byte langem Kopf und den eigentlichen Rohdaten. Im Kopf finden sich neben den obligatorischen Versions-, Steuerungs- und Größeninformationen vor allem die weltweit eindeutigen IP Adressen von Sender und Empfänger wieder.[81]

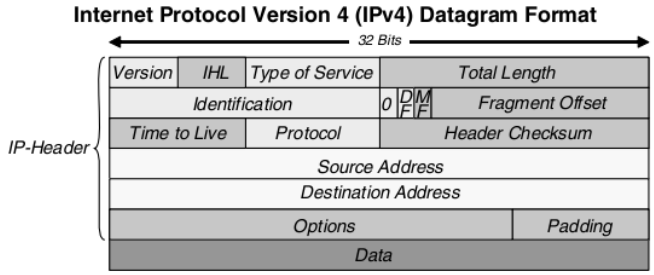
---

[79] vgl. Scherff, J. (2007), S. 85ff

[80] Scherff, J. (2007), S. 321

[81] vgl. Scherff, J. (2007), S. 321ff

Abbildung 24: Aufbau eines IPv4 Paketes

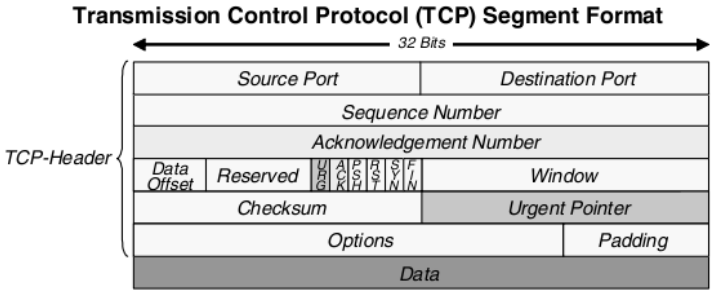


Quelle: Scherff, J. (2007), S. 322

Zu der über dem IP Protokoll liegenden Transportschicht gehört das TCP Protokoll, welches auf dessen Basis "eine verbindungsorientierte, zuverlässige Prozess-zu-Prozess-Kommunikation" [82] bietet. Es nimmt die Daten der Anwendungsschicht entgegen, zerteilt sie in Stücke, die dann fortlaufend nummeriert über die Vermittlungsschicht abgeschickt werden.[83]

Da auf einem System mehrere Anwendungen Nachrichten verschicken können, wird über die im Kopf eines TCP Paketes vermerkten Port-Nummern die Zuordnung zu den Anwendungen auf dem Quell- und Zielsystem vollzogen. Die Zuverlässigkeit der Kommunikation wird über ein Handshake Verfahren sichergestellt, bei dem der Empfänger jedes erhaltene Paket dem Sender über Einstellungen im Kopf des TCP Paketes bestätigen muss.[84]

Abbildung 25: Aufbau eines TCP Paketes



Quelle: Scherff, J. (2007), S. 326

---

[82] Scherff, J. (2007), S. 324

[83] vgl. Tanenbaum, A. (2008), S. 145

[84] vgl. Scherff, J. (2007), S. 324ff

Alternativ zum TCP Protokoll kann in der Transportschicht auch auf das UDP Protokoll zurückgegriffen werden, welches auf Basis von IP Paketen eine “verbindungslose, unzuverlässige Prozess-zu-Prozess-Kommunikation ohne Garantie ermöglicht”.[85] Entsprechend gering ist auch der Kommunikationsaufwand. Der Kopf einer UDP Nachricht führt lediglich die Port-Nummern der Anwendungen auf Quell- und Zielrechner sowie Längen und Prüfsummen-Informationen auf. Es wird häufig für die Versendung von Daten verwendet, wo lediglich wenige Pakete zu übertragen sind – etwa bei der DNS oder DHCP Abfrage.[86]

4.2.2 Das HTTP Protokoll

Die Kommunikation im Web wird über das HTTP Protokoll, welches auf den Austausch von HTTP Request und HTTP Response Nachrichten basiert, abgewickelt. Es ist Teil der Anwendungsschicht der TCP/IP Protokoll Familie und setzt auf dem TCP Protokoll auf.[87] Während die bisherigen Schichten für den Aufbau einer Verbindung zwischen Sender und Empfänger eine jeweils eindeutige IP Adresse verwendet haben, werden im HTTP Protokoll Ressourcen über eine URL spezifiziert. Diese URL spiegelt den Ort einer Ressource im Netzwerk wider. Sie hat den folgenden Aufbau:

Abbildung 26: Aufbau einer URL

Quelle: Tanenbaum, A. (2008), S. 610

Der Hostname ist eine sprechende Bezeichnung für einen im Netzwerk registrierten Server. Dieser für Menschen eher einprägsame Name wird vor dem Absenden einer HTTP Anfrage an den Server mit Hilfe eines Verzeichnisdienstes (in diesem Fall das DNS) in die zugehörige IP Adresse des Systems übersetzt. Ist der Port in der URL nicht angegeben, so wird der Standard-HTTP Port 80 verwendet. Der Pfadname bestimmt die auf dem Server befindliche Ressource.[88]

---

[85] Scherff, J. (2007), S. 327

[86] vgl. Scherff, J. (2007), S. 327

[87] vgl. Scherff, J. (2007), S. 337

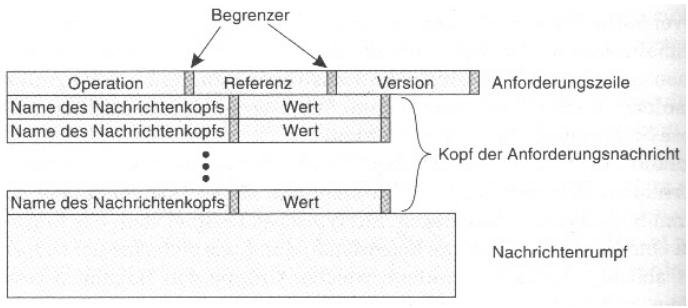
[88] vgl. Tanenbaum, A. (2008), S. 610

Um den Server mitzuteilen, welche Aktivität auf der Ressource ausgeführt werden soll, unterstützt das HTTP Protokoll folgende Operationen:

- **HEAD**: Anfrage nach dem Kopf einer Ressource (den Meta-Informationen),
- **GET**: Anforderung zur Auslieferung einer Ressource,
- **PUT**: Anforderung zur Speicherung einer Ressource,
- **POST**: Anforderung zur Neuanlage einer Ressource,
- **DELETE**: Anforderung zum Löschen einer Ressource.[89]

Die Operation und der Pfadname einer Ressource werden vom Client in seiner Anfrage an den Server in der obligatorischen Anforderungszeile der HTTP Request Nachricht spezifiziert. Zusätzlich kann er weitere Informationen in den HTTP Kopf aufnehmen, um bspw. zu signalisieren, in welchem Format er die Daten in der Antwort des Servers erwartet. Im Falle einer PUT oder POST Operation liefert der Client die an den Server zu übertragenden Informationen als Rumpf der HTTP Request Nachricht mit.

Abbildung 27: Aufbau einer HTTP Request Nachricht



Quelle: Tanenbaum, A. (2008), S. 606

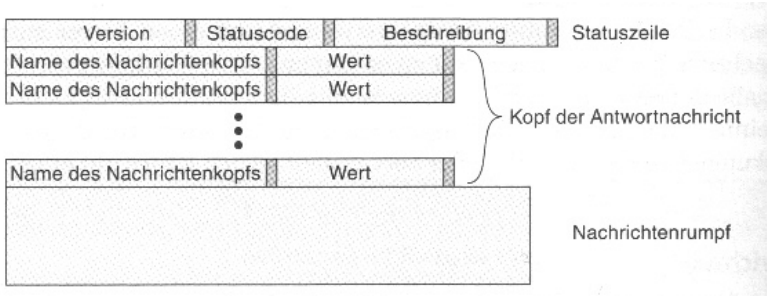
Die Antwort des Servers beginnt zwingend mit einer Statuszeile, in der der Server über den Statuscode signalisiert, ob die Anfrage erfolgreich verarbeitet werden konnte. Sollte bei der Behandlung der Anfrage ein Fehler aufgetreten sein – bspw. die Ressource ist unter diesem Pfad nicht verfügbar – so existieren eine Menge an standardisierten Fehlercodes, die der Server an den Client zur weiteren Behandlung zurückgeben kann. In der Antwort vom Server können weitere Informationen im Kopf der HTTP Response Nachricht abgelegt sein – bspw. wie lang die Antwort gültig ist oder auch die neue Adresse der Ressource, wenn diese zwischenzeitlich verschoben wurde.

---

[89] vgl. Tanenbaum, A. (2008), S. 604f

Wenn der Client mittels der GET Operation den Zugriff auf die Ressource verlangt hat, liefert der Server bei erfolgreicher Verarbeitung der Anfrage den Inhalt der Ressource als Bestandteil des Rumpfes der HTTP Response Nachricht an den Client.[90]

Abbildung 28: Aufbau einer HTTP Response Nachricht



Quelle: Tanenbaum, A. (2008), S. 606

Im Falle einer Web Anwendung tauschen Client und Server über dieses Verfahren Daten in Form von Dokumenten – d.h. HTML-formatierte Web Seiten, Bilder als auch text-formatierte JavaScript und Layoutanweisungen – aus, die auf dem Client durch den Web Browser dargestellt und ausgeführt werden.[91]

4.2.3 SOAP-basierte Web Services

Das Simple Object Access Protocol (SOAP) wird in der Fachwelt oft gleichgesetzt mit dem Begriff der Web Services. Es ist in diesem Kontext eine auf dem HTTP Protokoll basierende Kommunikationsmethode, die Dienste eines System funktional betrachtet und über ein standardisiertes, auf den Austausch von XML-formatierten Nachrichten basierendes Protokoll nutzbar macht. Ein SOAP Request wird dabei in eine SOAP Nachricht bestehend aus einem Kopf und einem Rumpf verpackt und über den Rumpf einer HTTP Nachricht an die Service-Gegenstelle übermittelt.

Der SOAP Kopf kann dabei verschiedene Steuerungsinformationen wie bspw. Verschlüsselungs- und Authentifizierungsdaten beinhalten. Die Nutzdaten der Anwendung werden als XML-formatierter Datenstrom in den Rumpf der SOAP Nachricht aufgenommen.

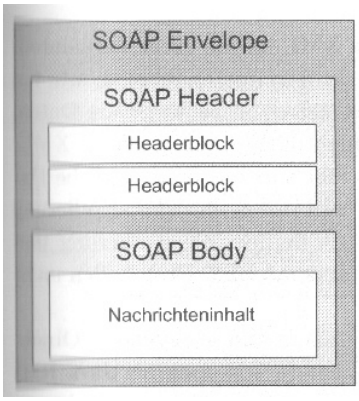
---

[90] vgl. Tanenbaum, A. (2008), S. 605f

[91] vgl. Tanenbaum, A. (2008), S. 590ff

Die möglichen Operationen und Aufrufkonventionen eines SOAP-basierten Web Service sind über eine ebenfalls standardisierte Web Service Description Language (WSDL) von einem Client jederzeit einsehbar. Nachteilig beim Einsatz von XML-kodierten Nachrichten ist jedoch die damit verbundene Größe der zu übertragenden Informationen. Sie gilt daher als aufwendige und u.U. ineffiziente Kodierung für bestimmte Datentypen – bspw. für unstrukturierte Daten.[92]

Abbildung 29: Aufbau einer SOAP Nachricht



Quelle: Schill, A. / Springer, T. (2007), S. 73

4.2.4 REST-basierte Web Services

Einen anderen Ansatz verfolgt die auf dem HTTP Standard aufbauende REST Architektur. Hierbei stellt ein Server seinen Clients Ressourcen über eine eindeutige URL zur Verfügung, auf welche der Client die HTTP Operationen PUT, GET, POST und DELETE ausführen kann. Diese können von ihrer Bedeutung her mit den Standardfunktionen einer Datenbank – namentlich und analog zur obigen Reihenfolge CREATE, READ, UPDATE und DELETE – verglichen werden.[93]

REST-basierte Dienste beruhen damit auf den Standardstrukturen des HTTP Protokolls – siehe Kapitel 4.2.2. Die Daten der Ressource werden im Rumpf der HTTP Nachrichten verpackt und an den Empfänger geschickt. Dabei kann REST abhängig von den Anforderungen des Clients verschiedene Darstellungsformate für die Daten liefern – u.a. als XML-formatierter oder JSON-formatierter Objektgraph.[94]

---

[92] vgl. Schill, A. / Springer, T. (2007), S. 68ff

[93] vgl. Schäfer, W. (2010), S. 262f

[94] vgl. Weber, J. et.al. (2010), S. 7ff

4.3 Abgrenzung global verteilte Systeme

Während die bisherige Betrachtung den Aufbau von verteilten Systemen im Fokus hatte, die in dieser Form sowohl in einem lokalen Netzwerk eines Unternehmens als auch im globalen Internet betrieben werden können, soll sich in der anschließenden Analyse auf die Anwendungsarchitektur einer global verteilten, über das Internet zur Verfügung gestellten Web Anwendung bzw. Web Dienst konzentriert werden. Durch die zunehmende Popularität von webbasierten Anwendungen und der fortschreitenden Mobilisierung der Anwender wird davon ausgegangen, dass die Web Applikation raschen Zuspruch findet, der sich in fortwährend steigenden Nutzerzahlen äußert. Damit einhergehend wachsen sowohl die Anzahl der Anfragen an das System als auch die Daten, die in dem System gespeichert werden müssen. Es entsteht ein sogenannter Netzwerk-Effekt: mit steigender Nutzeranzahl und Informationen im System wächst auch der Nutzen für und die Begeisterung bei dessen Anwender.

Dieser Effekt lässt sich derzeit vor allem bei den Web Anwendungen beobachten, die auf zwischenmenschliche Beziehungen und der individuellen Beteiligung der Nutzer beim Erstellen der Inhalte abzielen – bspw. Facebook oder Twitter. So veröffentlichte Twitter im März 2011 eine Statistik nach der durchschnittlich 150 Millionen Tweets (Nachrichten mit 140 Zeichen Länge) pro Monat über die Plattform verschickt werden. Der damals aktuelle Rekord der Plattform lag bei 6.939 Tweets pro Sekunde.[95]

Doch nicht nur die Anzahl der Nutzer führt bei einer webbasierten Anwendung in dieser Größenordnung zu enormen Herausforderungen im täglichen Betrieb, auch mit der Menge der von den Nutzern erstellten Daten müssen diese Anwendungen umgehen können. In ihrem Antrag zum Börsengang nennt die Firma Facebook einen Datenbestand an Fotos, Videos und Musik in Höhe von über 100 Petabytes in ihren Rechenzentren.[96] Es handelt sich hierbei zwar um extreme Beispiele, doch darf nicht vergessen werden, dass sowohl Facebook als auch Twitter noch recht junge Firmen sind, die beide vor wenigen Jahren mit sehr einfachen Mitteln angefangen haben und aufgrund ihres raschen Erfolges mehrmals gezwungen waren, ihre Angebote im Internet

---

[95] vgl. Twitter (2011)

[96] vgl. Perez, Sarah (2012)

komplett zu überarbeiten, um mit den Anforderungen Schritt halten zu können.[97]

Es wird daher bei den folgenden Betrachtungen davon ausgegangen, dass das System jeweils mit optimaler Auslastung betrieben werden soll. Eine optimale Auslastung ist gegeben, wenn das verteilte System jederzeit innerhalb der im Vorfeld definierten Antwortzeit auf Anfragen reagiert und unter Berücksichtigung eines Puffers zum Abfangen von Lastspitzen wenige bis gar keine ungenutzten Rechenkapazitäten existieren. Diese Annahmen bedeuten jedoch auch, dass das verteilte System mit der Anzahl der Nutzer, den von ihnen abgesetzten Anfragen und den zu verwaltenden Daten skalieren muss.

Unter der Skalierbarkeit eines Systems "bezeichnet [man] die Fähigkeit [...] wachsende quantitative Anforderungen durch Hinzufügen von Ressourcen auszugleichen, ohne dass eine Änderung von Systemkomponenten notwendig wird".[98] Dabei sind jedoch Speicher, Rechenleistung oder auch die Ressource selbst in einem zentralen System nicht beliebig erweiterbar. Es wird daher zwischen vertikaler und horizontaler Skalierbarkeit unterschieden.

Vertikale Skalierbarkeit (Scale-Up) dient der Erhöhung der Rechenleistung eines einzelnen Systems. Dies kann bspw. durch den Einsatz neuerer Prozessoren und der Nutzung von Mehrprozessor- oder Mehrkernsystemen erreicht werden. Die horizontale Skalierbarkeit (Scale-Out) wird im Gegensatz dadurch erzielt, dass mehrere einzelne Systeme über ein Computernetzwerk aneinander geschlossen werden. Im Unterschied zum Einsatz von Mehrkernsystemen verfügen diese Anwendungssysteme über eine verteilte Architektur und müssen Informationen über Nachrichten austauschen.[99] Dem Vorteil der theoretisch unbegrenzten Erweiterbarkeit stehen im Falle der horizontalen Skalierung der Systeme jedoch erhöhte Aufwände bei der Kommunikation und Probleme bzgl. der Synchronisation der einzelnen Knoten gegenüber.[100] Diese Umstände sind bei der Entscheidung für eine Anwendungsarchitektur und deren Umsetzung entsprechend zu berücksichtigen.

---

[97] siehe hierzu die Online Artikel highscalability (2009) und highscalability (2011)

[98] Schill, A. / Springer, T. (2007), S. 6

[99] vgl. Bengel, G. / Baun, C. (2008), S. 33f

[100] vgl. Tanenbaum, A. (2008), S. 34f

5 Datenhaltung in global verteilten Systemen

5.1 Szenario und Nutzbarkeitsbetrachtung

Auf Basis der Beschreibung der Komponenten eines verteilten Systems in Kapitel 4.1 werden für die Verarbeitung der Anfragen von den Clients auf Anwendungsebene mehrere redundante Web- und Anwendungsserver eingesetzt. Da die für die Ausführung der Anwendung benötigten Dateien i.d.R. semi-strukturierte oder unstrukturierte, sich selten bis nie ändernde Daten beinhalten, die zum Betrieb der Anwendung auch nur auf den einzelnen Servern lokal vorhanden sein müssen und zusätzlich für alle Anwender gleichermaßen Gültigkeit besitzen, lassen sich diese Informationen leicht duplizieren. Es bietet sich demnach an, mit vorgefertigten System-Images zu arbeiten, wobei jedes Image auf eine bestimmte Rolle im verteilten System zugeschnitten ist. Diese Images können dann dazu verwendet werden, neue Ressourcen für diese Rolle kurzfristig aufzusetzen und in das vorhandene System zu integrieren. Hierzu dienen i.d.R. virtuelle Systeme; wobei mehrere virtuelle Systeminstanzen auf einem physikalischen Rechner zeitgleich und zur effizienten Ausnutzung seiner Ressourcen betrieben werden können.

Um die gleichmäßige Verteilung der Anfragen der Endanwender auf den im Rechenzentrum laufenden Web Servern zu gewährleisten, müssen diese über einen zentralen Eingangspunkt – in diesem Beispiel ein Reverse Proxy[101] – einlaufen. Dieser Load Balancer kann anhand der URL im Kopf des HTTP Requests eine spezifische Weiterleitung der Nachricht an einen für diesen Kontext zuständigen Web Server veranlassen. Dazu sind spezielle Routing Konfigurationen im Reverse Proxy notwendig. Ebenfalls befindet sich ein Load Balancer zwischen den Web Servern und den Anwendungsservern. Dieser kann die Anfragen entweder nach dem einfachen Round-Robin-Verfahren oder aber anhand der Auslastung der einzelnen Systeme verteilen. Für die letztgenannte Funktionalität müssen die Anwendungsserver ihre aktuelle Auslastung und ggf. weitere Performance Indikatoren an den Load Balancer zurückmelden.[102]

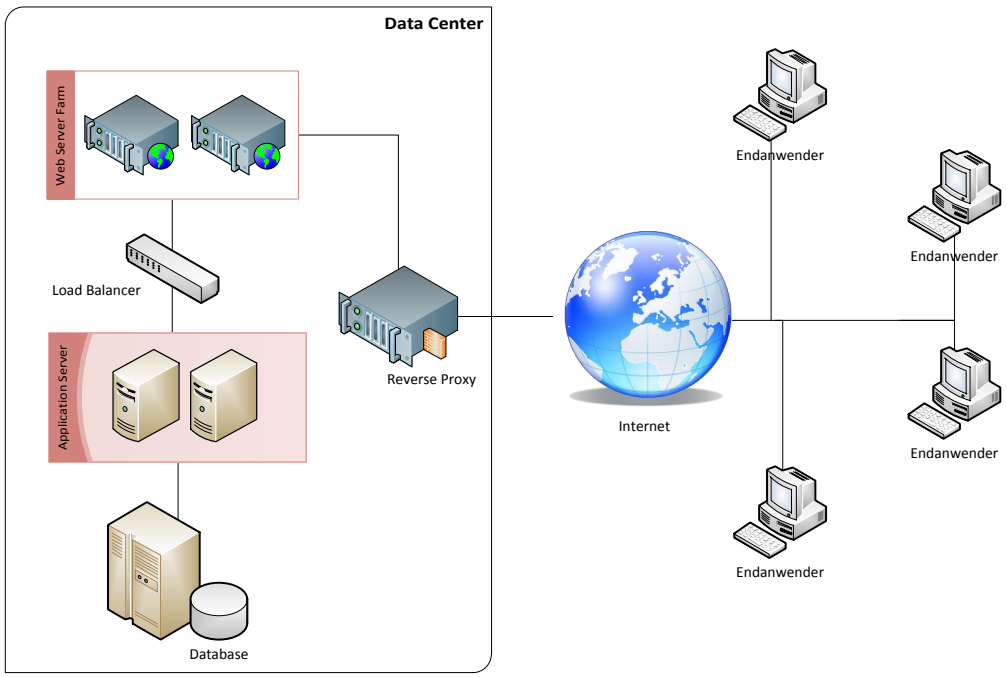
Alle Anwendungsserver greifen in diesem Szenario auf einen zentralen Datenbank-server zu, welcher die Datenbanken der Anwendung beinhaltet.

---

[101] als Reverse Proxy kann bspw. der Apache Web Server dienen, indem dieser dafür entsprechend konfiguriert wird (siehe http://www.apachetutor.org/admin/reverseproxies)

[102] vgl. Tanenbaum, A. (2008), S. 600ff

Abbildung 30: Systemarchitektur mit zentraler Datenhaltung



Quelle: eigener Entwurf

Um die horizontale Skalierbarkeit auf Anwendungsebene sicherzustellen, sind die Komponenten der Web Server und Anwendungsserver lose miteinander verbunden (loosly coupled). Die Verknüpfung zwischen Web- und Anwendungsmodulen erfolgt somit dynamisch zur Laufzeit. Weiterhin ist dafür zu sorgen, dass die einzelnen Komponenten über den jeweiligen Zeitrahmen zur Verarbeitung einer Anfrage keine Zustandsinformationen behalten (stateless). Vorteilhaft bei dieser Art der Programmierung ist, dass nachfolgende Nachrichten eines Clients nicht an einen spezifischen Server gebunden sind, sondern flexibel über die vorhandenen Ressourcen verteilt werden können. Nachteilig dabei ist jedoch, dass die Zustände der Anwendung nun nicht mehr in den Komponenten der Anwendungsschicht selbst gespeichert werden können und daher als einzige zentrale Komponente zur Verwaltung von Anwendungsdaten (bspw. die Session Daten eines Nutzers) die Datenbank in diesem Szenario genutzt werden kann. [103] Dadurch wird die Last auf den Datenbankserver noch zusätzlich erhöht.

[103] vgl. Schill, A. / Springer, T. (2007), S. 22ff

Er wird daher kurz- oder mittelfristig zum Flaschenhals in dieser Systemarchitektur und gibt als empfindlichstes Glied in der Verarbeitungskette den maximalen Rahmen für die Skalierbarkeit des Gesamtsystems vor.[104]

Es ist evtl. noch möglich, vom Datenbankserver über die Methoden der vertikalen Skalierung mehr Performance und Datendurchsatz zu erhalten – bspw. indem mehr RAM Speicher, eine bessere CPU oder schnellere Festplatten im Server verbaut werden. Dennoch sind diesem Verfahren schnell wirtschaftliche als auch physikalische Grenzen gesetzt, da sie sowohl die Gesamtkosten des Systems inkl. dessen Wartungsaufwand in die Höhe treiben als sich auch an den derzeit technisch Machbaren orientieren müssen. Daher ist zu befürchten, dass allein über diese Mittel der Datenbankserver nicht mit den Anforderungen der Anwendung mit wachsen kann.[105]

Gleichfalls stellt der zentrale Datenbankserver einen Single-Point-of-Failure (SPOF) dar. Dies bedeutet, dass die Anwendung in Summe nur dann funktionieren kann, wenn auch der Datenbankserver verfügbar und erreichbar ist. Sollte es jedoch aufgrund eines Defektes des Servers bzw. eines Netzwerkproblems nicht möglich sein, mit dem Datenbankserver in Kontakt zu treten, so ist das gesamte System unbenutzbar.[106]

5.2 Einsatz von Data Cache Layer

Eine einfache Möglichkeit, die Verarbeitungslast innerhalb des Anwendungssystems zu senken, besteht darin, auf unterschiedlichen Ebenen des Systems temporäre Zwischenspeicher für Daten einzuführen. Diese Data Cache Layer dienen dazu, häufig genutzte und sich selten ändernde aber auch schwierig zu rekonstruierende und rechenintensive Daten oder Ergebnisse der nachgelagerten Stufe im schnellen Arbeitsspeicher einer Systemebene vorzuhalten, um damit sowohl den Aufwand zum erneuten Übermitteln der Anfragen und Ergebnisse über das Netzwerk als auch den Aufruf der Funktion im nachgelagerten System generell einsparen zu können.

Besonderes Augenmerk kommt hierbei den Puffern auf Seite der Endanwender und der Anwendungsserver zu. Die Cache Server auf der Anwendungsseite halten Daten aus der Datenbankebene in ihrem Arbeitsspeicher und entlasten damit den Datenbankserver von

---

[104] vgl. Pritchett, D. (2008), S. 48

[105] vgl. Abbott, M. / Fisher, M. (2011), S. 36ff

[106] vgl. Abbott, M. / Fisher, M. (2011), S. 155ff

sich häufig wiederholenden Routineanfragen – bspw. für Stamm- oder Konfigurationsdaten. Der Pufferspeicher beim Endanwender wird bei Web Anwendungen i.d.R. durch den Web Browser zur Verfügung gestellt. Ist eine eigene Client Applikation lokal auf dem Computer des Nutzers installiert, die mit dem System über Web Dienste kommuniziert, so müssen bei deren Entwicklung mögliche Cache Szenarien für die Daten des Servers berücksichtigt werden.

5.2.1 Serverseitige Cache Layer

Auf der Serverseite können Daten und Zwischenergebnisse auf unterschiedlichen Stufen in Zwischenspeichern abgelegt werden. Diese Stufen erweitern das dreischichtige Anwendungsmodell um weitere Ebenen, in denen die Cache Server angesiedelt sind. Bei diesen Rechnern handelt es sich um Systeme mit wenig CPU Last aber hohem RAM Speicherbedarf, da die Cache Server die Daten für einen schnellen Zugriff im flüchtigen Arbeitsspeicher des Systems vorhalten müssen. Der externe Zugriff auf die Daten im Cache erfolgt hierbei über einen Mechanismus, der ähnlich den Key-Value-Stores funktioniert: Daten jedweder Beschaffenheit (strukturiert, semi-strukturiert oder unstrukturiert) werden über einen eindeutigen Schlüssel im Cache abgelegt und können im Nachhinein über diesen Schlüssel auch wieder ausgelesen werden. Dieser Schlüssel kann entweder auf Basis einer bereits bestehenden, eindeutigen Objekt-ID oder unter Anwendung eines Hashing-Verfahrens auf dem Datenobjekt erzeugt werden.[107]

Da die Datenbank-Ebene in der bisherigen Betrachtung den Flaschenhals des Systems ausgemacht hat; da sie viele verschiedene Anfragen aus der Anwendungsebene verarbeiten und u.U. komplizierte Datenabfragen bewältigen muss, ist ein Einsatz eines Cache Layers oberhalb der Datenbank-Ebene zwingend vorgeschrieben. Gerade im relationalen Datenbank-Umfeld, wo aus den normalisiert abgelegten Daten der Datenbank mittels Join Verbindungen erneut Geschäftsobjekte rekonstruiert werden müssen, lohnt sich die Zwischenspeicherung der Ergebnisse. Die Komponenten der Anwendungsebene erfragen hierbei zunächst beim Cache Layer, ob ein bestimmtes Objekt bereits vorhanden ist. Wurde das Objekt im Cache gefunden, so kann die Anwendung direkt mit den zwischengespeicherten Informationen weiterarbeiten.

---

[107] vgl. Abbott, M. / Fisher, M. (2011), S. 107ff

Andernfalls muss die Anwendung eine Anfrage an die Datenbankschicht schicken. Das zurückgelieferte Ergebnis wird dann von der Anwendungsseite unter dessen charakteristischem Zugriffsschlüssel in den Data Cache abgelegt. Hierbei kann die Anwendungsseite auch eine Gültigkeitsdauer für das Objekt hinterlegen – die meisten Cache Systeme entfernen ungültig gewordene Informationen selbständig aus ihren Speichern.

Doch nicht nur zum Zwischenspeichern von Datenbank-Ergebnissen lassen sich Cache Systeme einsetzen. Wie bereits oben erwähnt, können sie auch für das temporäre Puffern von rechenintensiven bzw. aufwendig zu rekonstruierenden Ergebnissen der Anwendungsschicht eingesetzt werden. Dieser Cache Layer ist demzufolge zwischen den Web Servern und den Anwendungsservern angesiedelt und wird von den Web Servern verwaltet. Hierin finden sich auch die evtl. Session Daten der Anwender wieder.[108]

Je nach Aufbau und Struktur der Web Seiten, die von dem Anwendungssystem generiert werden, kann ein Cachen ganzer Seiten oder einzelner Teile davon sinnvoll erscheinen, um die Last auf den Web Servern zu reduzieren. Der in Abbildung 32 eingezeichnete Data Cache zwischen Reverse Proxy und Web Server Farm ist jedoch eher virtuell anzusehen, da in vielen Fällen der Reverse Proxy mit einer entsprechenden Konfiguration installiert wird und diese Aufgabe übernimmt. Damit dieser entscheiden kann, welche Web Seiten er in seinem Cache aufheben soll, müssen die von den Web Servern generierten HTTP Response Nachrichten spezielle Attribute im HTTP Kopf aufweisen. Diese werden in der anschließenden Betrachtung der clientseitigen Cache Möglichkeiten noch näher erläutert.[109]

Damit der Einsatz von Cache Layern in der Systemarchitektur auf der Serverseite zum gewünschten Ergebnis – der Reduzierung der Last auf den einzelnen Stufen – führt, muss das Cache System Indikatoren bereitstellen, wie häufig ein Objekt im Cache aufgefunden werden konnte (cache-hit) und wie oft das nicht der Fall gewesen ist (cache-miss). Das daraus ermittelte Verhältnis (cache-ratio) ermöglicht eine Einschätzung über die Wirksamkeit der Cache Layer und gibt erste Hinweise auf evtl.

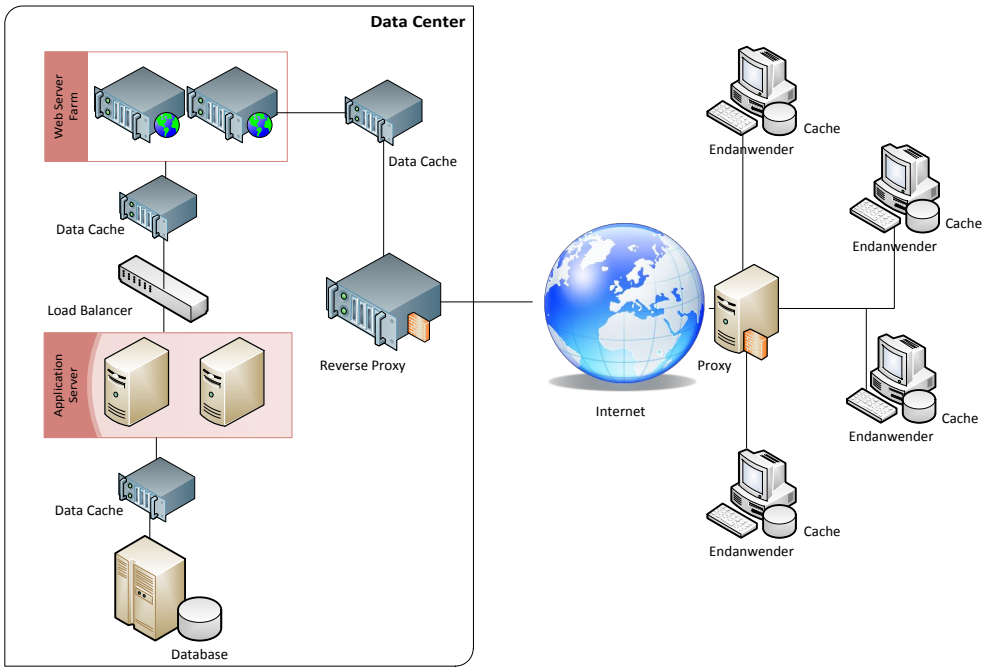
---

[108] vgl. Abbott, M. / Fisher, M. (2010), S. 381ff

[109] vgl. Abbott, M. / Fisher, M. (2010), S. 384ff

Maßnahmen, die zur besseren Ausnutzung der Cache Systeme in der Anwendungsarchitektur ergriffen werden sollten.[110]

Abbildung 31: Systemarchitektur mit zentraler Datenhaltung und Cache Layern



Quelle: eigener Entwurf nach Abbott, M. / Fisher, M. (2011), S. 101ff

5.2.2 Clientseitige Cache Layer

Während mit Hilfe der Cache Layer die Last im Rechenzentrum des Anbieters einer Web Anwendung reduziert werden kann, müssen die Daten, die für die Darstellung der Web Anwendung auf der Clientseite notwendig sind, weiterhin durch das globale Netzwerk (Internet) übertragen werden. Letzte Studien haben ergeben, dass sich die Übermittlung der Daten durch das Netzwerk hierbei als Flaschenhals erweist. Die sogenannte “middle mile” zwischen Rechenzentrum des Anbieters und Web Browser des Nutzers ist durch hohe Netzwerk-Latenzzeiten, Einschränkungen hinsichtlich des möglichen Datendurchsatzes sowie allgemeine Verfügbarkeitsbeeinträchtigungen gekennzeichnet. Um hier eine spürbare Geschwindigkeitssteigerung aus Anwendersicht zu erhalten, müssen sowohl die Transportwege kurz als auch die Anzahl der zu übermittelnden Daten zwischen Anbieter und Endanwender klein gehalten werden.[111]

---

[110] vgl. Abbott, M. / Fisher, M. (2010), S. 378ff

[111] vgl. Leighton, T. (2008), S. 22ff

Beim Aufrufen einer Web Seite im Browser des Anwenders wird zunächst ein HTTP GET Request an die eingegebene URL geschickt. Dadurch wird als Erstes das HTML Dokument, welches unter dieser Adresse zu finden bzw. aufgrund dieser Anfrage dynamisch vom Web Server generiert worden ist, auf den Client übertragen. Dieses wird vom Web Browser in dessen lokalen Cache abgelegt. Im HTML Dokument selbst können Verweise auf zusätzlich benötigte Dokumente existieren, die der Web Browser für die Anzeige der Web Seite von diesem oder auch von einem anderen Web Servern abholen muss. Dafür werden wiederum separate HTTP GET Requests an die jeweiligen Adressen verschickt. Die Resultate dieser Anfragen speichert der Web Browser ebenfalls im lokalen Zwischenspeicher ab und die letztendliche Darstellung der Web Seite erfolgt abschließend anhand der lokal gepufferten Informationen.

Bei den Daten, die für die Anzeige der Web Seite benötigt werden, handelt es sich meist um statische Inhalte wie Bilder, Video- oder Audiodaten, Layoutanweisungen in Form von CSS Dateien und clientseitigem JavaScript Programmcode. Diese Bestandteile einer Web Seite stellen das unveränderliche Grundgerüst der eigentlichen Web Anwendung dar und werden dynamisch zur Laufzeit durch die benutzerspezifische Abfrage von Daten und über die eigentliche Interaktion mit der Web Anwendung mit zusätzlichen Informationen angereichert. Die dafür erforderlichen Anfragen werden von der Web Anwendung über eine asynchrone HTTP Kommunikation im Hintergrund an den Server übertragen bzw. die Ergebnisse einer Anfrage nach diesem Schema auf dem Client empfangen und an die Web Anwendung weitergereicht (AJAX).[112]

Um die Bereitstellung der statischen Inhalte zu beschleunigen, bieten sich mehrere Vorgehensweisen an. Zunächst können sich seltene oder nie ändernde Daten einer Web Anwendung über einen längeren Zeitraum im Cache des Web Browsers verweilen. Dazu muss der Web Server beim Versand der HTTP Response Nachricht in deren Kopf einen speziellen Schlüssel mit dem Namen Expires aufnehmen. Dieser Schlüssel erhält als Wert das Ablaufdatum der Daten in dieser HTTP Response. Der Web Browser prüft anhand dieser Information, ob die lokal zwischengespeicherten Daten noch gültig sind und verzichtet in diesem Fall auf eine erneute Anfrage beim Web Server.

---

[112] vgl. Tanenbaum, A. (2008), S. 589ff

Des Weiteren sind die für den Start der Web Anwendung notwendigen Dokumente zu minimieren, indem bspw. einzelne JavaScript Dateien oder Layoutanweisungen zu einer Datei zusammengefasst werden. Dadurch lassen sich die Anzahl der notwendigen Round Trips zum Server beim Aufruf der Web Seite senken. Umfangreiche HTML Seiten, CSS oder JavaScript Dateien, welche konzeptionell im Textformat vorliegen, lassen sich zudem leicht mittels GZip Kompression vor der Übertragung auf der Serverseite verkleinern.[113]

Um die Transportwege für diese statischen Inhalte möglichst kurz zu halten, können sowohl Proxy Server beim Endanwender oder dessen Internet Service Provider (ISP) benutzt als auch die Dienstleistungen eines Content-Delivery Netzwerkes (CDN) in Anspruch genommen werden. Im ersten Fall bündelt ein Proxy Server die Anfragen von wenigen Clients an viele verschiedene Web Anbieter. Die Clients kommunizieren mit den einzelnen Anbietern immer über diesen Vermittler. Der Proxy Server speichert dabei alle erhaltenen Antworten zwischen und bedient Anfragen von Clients zuerst aus seinem Cache. Er ist das Pendant zum Reverse Proxy Server auf der Anbieterseite. Wie auch der Cache im Web Browser selbst, benötigt er die entsprechenden Informationen im Kopf der HTTP Response Nachrichten des Web Servers, welche Daten er für welchen Zeitraum gefahrlos zwischenspeichern kann. Dafür dienen die Schlüssel Expires, Last-Modified und Cache-Control im Kopf einer HTTP Nachricht; über den Schlüssel Cache-Control kann das Puffern der Daten auf den Zwischenstufen auch komplett ausgeschaltet werden – dafür dient die Einstellung “no-cache, no-store”[114].

Ein Content-Delivery-Netzwerk ist ein weltweit verteiltes Netzwerk eines Dienstleistungsunternehmens, welches sich darum kümmert, die statischen Inhalte einer Web Anwendung – insbesondere Bilder, Audio- und Videodateien aber zunehmend auch die HTML, CSS und JavaScript Dokumente – geographisch nah an die Endanwender zu bringen, um somit die zur Übertragung notwendigen Kommunikationswege zu verkürzen. Diese Edge-Server genannten Systeme stehen bestenfalls im Rechenzentrum eines Internet Service Providers und bedienen über eine spezielle DNS Konfiguration die Anfragen von Rechnern aus der näheren Umgebung.

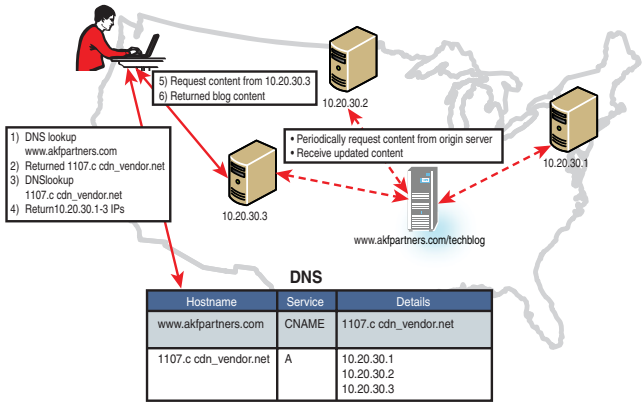
---

[113] vgl. Souders, S. (2008), S. 32f

[114] die möglichen Einstellungen und deren Bedeutung sind im HTTP 1.1 Standard (RFC 2616) zu finden

Werden diese Systeme gut positioniert und global eingesetzt, so entlasten sie den Hauptserver im Rechenzentrum des Anbieters von einem Großteil der Anfragen nach statischen Inhalten. Der Abgleich zwischen dem Web Server des Anbieters und den Edge Servern erfolgt hierbei periodisch und asynchron, so dass ein zeitlicher Versatz zwischen dem Einspielen von neuen statischen Inhalten und der nachgelagerten Verteilung über das CDN auftritt.[115] Da das CDN jedoch hauptsächlich für die Verteilung von sich selten oder nie ändernden Anwendungs- oder Konfigurationsdaten bzw. unstrukturierten Daten wie bspw. Bilder und Videos verwendet wird, kann während des Betriebs einer Web Anwendung problemlos Rücksicht auf den zu erwartenden Zeitversatz beim Aufspielen von neuen Inhalten genommen werden.

Abbildung 32: Funktionsweise von Content-Delivery Netzwerken



Quelle: Abbott, M. / Fisher, M. (2011), S. 89

Das Zwischenspeichern von Daten auf der Clientseite hat jedoch nicht nur positive Auswirkungen auf die Geschwindigkeit der Anwendung, sondern sorgt auch dafür, dass der Client unabhängiger von einer allzeit existierenden Netzwerkverbindung zum Server wird. Durch die Weiterentwicklungen im HTML Standard wird es Entwicklern mittlerweile ermöglicht, auch strukturierte Laufzeitdaten und dynamisch erzeugte Informationen im Browser Cache abzulegen.[116] Um eine Synchronisation zwischen Server und Web Browser zu ermöglichen, können die HTTP Response Nachrichten mit einem speziellen Schlüssel im HTTP Kopf versehen werden – das sogenannte ETag.

---

[115] vgl. Abbott, M. / Fisher, M. (2011), S. 88ff

[116] diese WebStorage genannte Funktion im HTML5 Standard bietet einen lokalen Cache Zugriff per Key-Value-Store Semantik

Dieses stellt eine Art Versionsinformation für die Daten dar. Sobald die Daten auf dem Server geändert werden, wird ein neuer Wert für das ETag vergeben und der Client erhält die neuen Daten auf Anfrage vom Web Server zurückgeliefert. Wurden die Informationen zwischenzeitlich nicht angepasst, so sendet der Web Server einen passenden Statuscode (304 - Not Modified) an den Client, womit dieser weiß, dass er mit dem lokal gepufferten Informationen weiterarbeiten kann.[117]

Die erneute Abfrage von Daten kann auf der Clientseite entweder über einen Pull-Mechanismus – bspw. durch Einsatz eines in der Anwendung eingebauten Timers, der periodisch im Hintergrund die lokalen Daten mit denen auf dem Server abgleicht, oder mittels eines Push-Verfahrens vom Server in Richtung der Clients erfolgen. Dieses Push-Verfahren nutzt in HTTP-basierten Architekturen eine langlebige TCP Verbindung, über die der Server dem Client Statusänderungen mitteilen kann.[118] Der Client ist in der Verantwortung, diese separate TCP Verbindung mit dem Server immer wieder zu erneuern, um weitere Benachrichtigungen erhalten zu können.[119]

Der Einsatz von Data Cache Layern lohnt sich vor allem in Bereichen, wo sehr oft lesend auf die Informationen zugegriffen wird. Der schreibende Zugriff auf die Daten in der Web Anwendung führt in dieser Architektur zwangsläufig dazu, dass die Anfragen bis auf den zentralen Datenbankserver durchgestellt werden müssen. Gleichzeitig sind evtl. zwischengespeicherte und durch die Änderung ungültig gewordene Informationen aus den diversen Cache Layern auf der Serverseite programmatisch zu entfernen und der Client mit Hilfe eines der o.g. Mechanismen über die Änderungen zu informieren. Das Einbeziehen von Data Cache Layern führt daher zu Mehraufwendungen in der Entwicklung der Applikation und im Betrieb der Systeme. In Anwendungen, wo häufig Daten geändert oder erzeugt werden, sorgen allerdings auch die Data Caches nicht für eine deutliche Entlastung des zentralen Datenbankservers.[120]

---

[117] vgl. Abbott, M. / Fisher, M. (2011), S. 95ff

[118] die Unterstützung von langlaufenden TCP Verbindungen für den Aufbau einer Zwei-Wege-Kommunikation zwischen Client und Server ist als Funktionalität unter dem Begriff WebSockets in den HTML5 Standard aufgenommen worden

[119] vgl. Keppler, K. (2007), S. 122ff

[120] vgl. Abbott, M. / Fisher, M. (2010), S. 390ff

5.3 Horizontale Verteilung der Daten

Um eine Lastverteilung analog zur Anwendungsebene auch für die Datenbank zu erhalten, muss der zentrale Datenbankserver ebenfalls in horizontaler Richtung skaliert werden. Dabei werden die zentralen Datenbanken auf mehrere preisgünstige Rechner verteilt – dieser Vorgang wird auch als Sharding bezeichnet. Diese Aufteilung kann nach funktionalen Kriterien oder anhand eines Schlüsselattributs innerhalb der Daten erfolgen; um eine flexible Skalierung der Daten über eine große Anzahl von Systemen zu ermöglichen, müssen jedoch beide Aspekte gleichzeitig berücksichtigt werden.

5.3.1 Verteilung nach Funktionalität

Durch den weiterhin ansteigenden Funktionsumfang von IT Anwendungssystemen werden aktuelle Applikationen in der Softwareentwicklung häufig nach dem Konzept der service-orientierten Architekturen entworfen. Hierbei wird der komplette Funktionsumfang einer Anwendung in separate, in sich abgeschlossene Dienstleistungen unterteilt, die getrennt voneinander definiert, entwickelt und betrieben werden können. Vorteile dieser service-orientierten Sichtweise sind u.a. die Wiederverwendbarkeit von Funktionen und die Sicherstellung der losen Kopplung zwischen den einzelnen Sub-Systemen einer Anwendung.[121] Als technische Umsetzung werden für SOA-basierte Programme i.d.R. Web Services verwendet. Diese können – wie im Kapitel 4.2.3 und 4.2.4 beschrieben – als SOAP- oder REST-basierte Dienste realisiert werden und verwenden somit standardisierte Kommunikationsmethoden. Als Beispiel sei hier ein e-Commerce-Shop genannt, der in die Dienste Kundenverwaltung, Auftragseingang, Auftragsbearbeitung, Rechnungsstellung, Versand und Bestandsverwaltung unterteilt ist (siehe Abbildung 34).

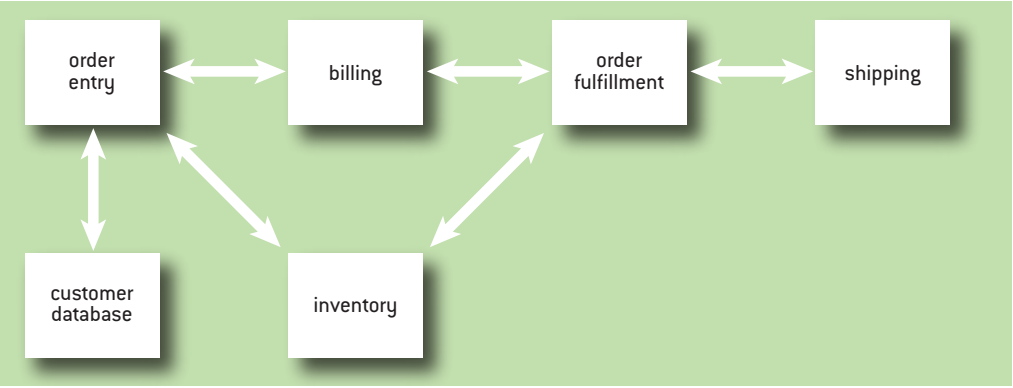
Da jedes Sub-System für die Erfüllung seiner Aufgaben auf bestimmte Daten zugreifen muss bzw. ebenso spezifische Daten erzeugt, können diese Daten meist direkt dem jeweiligen Dienst zugeordnet werden. Durch diesen Schritt kann auf eine zentrale Datenbankinstanz verzichtet und die Informationen anhand der Zugehörigkeit zu den Sub-Systemen der Anwendung aufgeteilt werden.[122]

---

[121] vgl. Josuttis, N. (2007), S. 8f

[122] vgl. Rys, M. (2011), S. 2

Abbildung 33: Überblick zur Datenverteilung nach Funktionalität



Quelle: Rys, M. (2011), S. 2

5.3.2 Verteilung nach Schlüsselattribut

Durch die funktionale Aufteilung des Systems erhält jeder Service seine eigene Datenbankinstanz, in der die Informationen für und aus dessen Betrieb festgehalten werden. Da jedoch selbst in einem hoch komplexen Anwendungssystem die Anzahl der Sub-Systeme überschaubar ist, reicht die alleinige Verteilung der Daten nach Diensten zumeist nicht aus. Die Daten pro Service müssen also noch weiter zerlegt werden, um die Skalierbarkeit des Gesamtsystems zu gewährleisten.

Hierzu wird in jedem Sub-System nach einem Schlüsselattribut in den Daten gesucht, welches eine relativ homogene Verteilung der gesamten Informationen erlaubt. So ist bspw. eine Verteilung von Kundendaten nach deren Anfangsbuchstaben des Nachnamens nicht sinnvoll, da die Daten hierbei asymmetrisch über die vorhandenen Partitionen verstreut wären.[123]

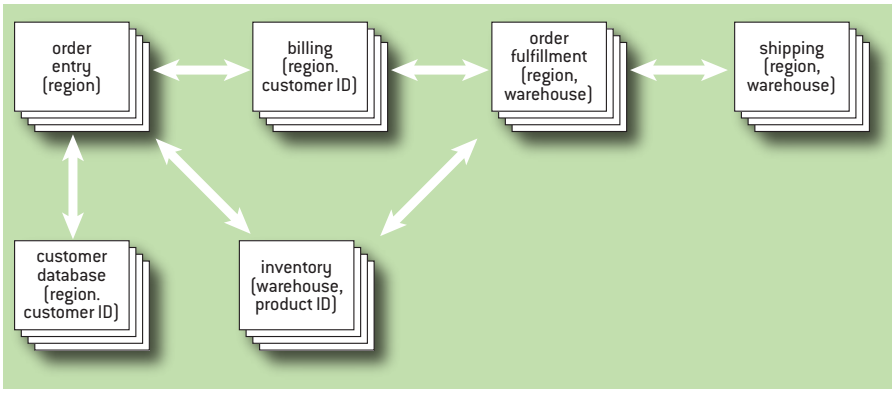
Mögliche Merkmale für eine sinnvolle Distribution der Daten sind daher u.a. zeitliche und örtliche Kriterien – bspw. nach Jahr bzw. nach Region. Alternativ kann eine Verteilung der Daten auch anhand von technischen Schlüsselwerten bzw. von vorher festgelegten Wertebereichen für numerische Ziffern erfolgen – bspw. alle Kundennummern zwischen 100000-499999, 500000-999999, ...[124]

---

[123] einen hohen Anteil haben hier Nachnamen mit dem Anfangsbuchstaben S, gefolgt von K und B während Nachnamen mit U, I, oder X sehr selten vorkommen (siehe Datendieter (2011))

[124] vgl. Rys, M. (2011), S. 2f

Abbildung 34: Überblick zur Datenverteilung inkl. Schlüsselattribut

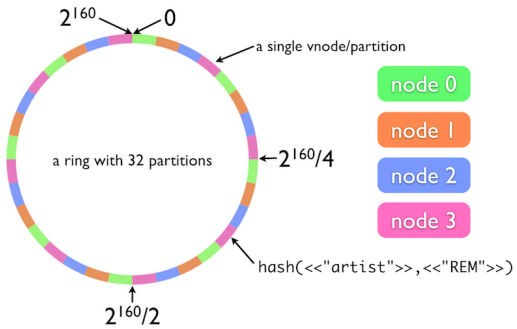


Quelle: Rys, M. (2011), S. 2

Bei den im Voraus festgelegten Kriterien für die Aufteilung der Daten muss besondere Sorgfalt gelten, da nachträgliche Änderungen des Verteilungsschemas nur durch massiven Eingriff in das bestehende System durchgeführt werden können.

Eine automatische Verteilung der Daten bieten bisher nur wenige Datenbanken in ihrem System an – z.B. Riak. Dafür wird im Server ein 160-Bit großer, aus Ganzzahlen bestehender Raum benutzt (Riak Ring). Dieser wird über die Konfiguration des Datenbanksystems in gleichwertige Teile untergliedert (hier bspw. in 32 Elemente). Diesen Elementen werden zur Laufzeit die vorhandenen Datenbankserver gleichmäßig zugeordnet. Durch ein Hash-Verfahren auf den Schlüsselattributen der Daten entscheidet das System, in welchem Bereich des Rings die Informationen gehören und findet auf diese Weise selbständig den dafür zuständigen Datenbankserver heraus.[125]

Abbildung 35: Hash-Ring zur automatischen Verteilung der Daten



Quelle: Riak (2012)

[125] vgl. Riak (2012)

Ohne diesen Automatismus muss die Logik für den korrekten Zugriff auf den Datenbestand entweder auf der Anwendungsebene liegen oder vom Datenbanktreiber respektive dem benutzten Framework für den Datenzugriff zur Verfügung gestellt werden.[126] Hierbei können je nach Abfragekriterium in der Anwendungsschicht u.U. auch mehrere Datenpartitionen betroffen sein. Diese müssen dann von der Applikation separat angefragt und die einzelnen Ergebnisse im Anwendungsserver zu einem Gesamtresultat zusammengeführt werden – näheres in Kapitel 5.5 über MapReduce.

Die horizontale Verteilung der Daten nach den beschriebenen Verfahren hat zusätzlich Einfluss auf das Datenmodell. Da die Daten nun nicht mehr in einer zentralen Datenbank abgelegt werden, sind Fremdschlüssel-Einschränkungen, die im relationalen Modell für die Konsistenzprüfung der Informationen auf Datenbankebene vorgesehen sind, zwischen den verschiedenen Datenpartitionen nicht mehr möglich. Dadurch verlagern sich diese Prüfungen in die vorgelagerte Anwendungsschicht und müssen von der Geschäftslogik bereitgestellt werden.

Durch die Distribution der Informationen über mehrere Datenbankserver – sowohl nach funktionalen Gesichtspunkten als auch über ein gezieltes Schlüsselattribut innerhalb einer Entität – kann die Verarbeitungslast auf Datenbankebene auch für die anfallenden Schreibzugriffe pari verteilt werden. Dadurch wird eine horizontale Skalierung der Datenbankserver ermöglicht, die somit mit den wachsenden Datenanforderungen in der Anwendung mithalten können. Nach der derzeitigen Vorgehensweise wird jedoch ein Datum immer nur in einer Serverinstanz abgelegt, wodurch es weiterhin zu Ausfällen kommen kann, wenn das einzelne System außer Betrieb oder über das Netzwerk nicht erreichbar ist. Im Gegensatz zur zentralen Datenbankinstanz betreffen diese einzelnen Ausfälle jetzt nicht mehr alle Funktionen der Anwendung bzw. alle Anwender gleichermaßen sondern nur noch einen Teilbereich davon, dennoch stellen die einzelnen Datenbankserver somit weiterhin einen Single-Point-Of-Failure für ihren Bereich dar. Um dieses Problem zu vermeiden und die Verfügbarkeit der Anwendung insgesamt zu erhöhen, müssen die Daten der einzelnen Partitionen redundant vorgehalten werden.[127]

---

[126] bspw. bietet das in der Java Entwicklung verbreitete Hibernate Framework eine Unterstützung für Shards an (siehe http://www.hibernate.org/subprojects/shards.html)

[127] vgl. Abbott, M. / Fisher, M. (2010), S. 362ff

5.4 Replikation der Daten

Im laufenden Betrieb eines IT Systems, welches den bisherigen Ausführungen folgend auf einer nennenswerten Anzahl von Standard PC Servern basiert, die über ein Netzwerk miteinander verbunden sind, kommt es in regelmäßigen Abständen zum Ausfall von einzelnen Komponenten oder Systemen. Daher muss das IT System so konzipiert sein, dass es auf diese Störfälle vorbereitet ist und dadurch die Gesamtfunktionalität der Anwendung nicht beeinträchtigt wird. Zur Sicherstellung der Anwendungsverfügbarkeit müssen sowohl die betrieblichen Funktionen der Software in den oberen Schichten (Web- und Applikationsserver) als auch die Informationen in den Datenbanken mehrfach dupliziert werden. In den höheren Schichten der Anwendung lassen sich die statischen Daten zur Ausführung und Konfiguration der Applikation leicht vervielfältigen – bspw. über den anfangs bereits betrachteten Mechanismus der virtuellen System-Images für spezifische Anwendungsrollen. Der Hauptgrund für diese einfache Handhabung liegt in der geringen Häufigkeit der Änderungen auf dieser Ebene begründet. Eine neue Anwendungsversion bzw. neue Konfigurationseinstellungen werden nur bei Bedarf vorgenommen. Das Einspielen der Änderungen erfolgt lediglich durch geschultes IT Personal des Anbieters und unter Berücksichtigung eines vorher definierten Schemas für das Ausrollen neuer Varianten.

Die Replikation der Daten in der Datenbankebene hat allerdings größere Auswirkungen auf das Design und die Funktion der Anwendung. Abhängig von der eingesetzten Datenbank kann hierbei zwischen einer Master-Slave Replikation und einer Replikation zwischen gleichberechtigten Partnern (alias Master-Master) unterschieden werden.[128] In beiden Fällen müssen die auf einem Master Server erfolgten Schreibvorgänge an die übrigen Systeme im Verbund weitergeleitet werden. Dadurch kann es je nach Distanz, Netzwerkauslastung und Verfügbarkeit der Systeme zwischen den einzelnen Servern zu temporären Diskrepanzen in den Datenbeständen kommen. Diesen Sachverhalt kann die Anwendung aus verschiedenen Perspektiven begegnen; ihre möglichen Ausprägungen spiegeln sich im CAP Theorem von Eric Brewer[129] wider.

---

[128] vgl. Vogels, W. (2008), S. 15f

[129] das Theorem wurde von Dr. Brewer im Jahr 2000 während eines Vortrages mit dem Thema “Towards Robust Distributed Systems” vorgestellt (siehe auch Brewer, E. (2000))

5.4.1 Folgen des CAP Theorems

Das CAP Theorem besagt, dass eine komplexe verteilte Anwendung von den drei wünschenswerten Eigenschaften: Consistency, Availability und Partition Tolerance immer höchstens zwei Merkmale unterstützen kann.

Ein System fällt unter die Kategorie Consistency, wenn es eine strenge Konsistenz der Daten zu jedem Zeitpunkt sicherstellt. D.h. jeder Client erhält stets den momentan aktuellen Zustand bei einer Anfrage zurück. Auf der Datenbankebene wird diese Anforderung meist durch das Nutzen von Transaktionen nach dem ACID Schema sichergestellt. Das System erfüllt zusätzlich die Anforderungen nach der Availability, wenn es jederzeit auf Anfragen reagiert und diese positiv beantworten kann, auch wenn einzelne Systembestandteile ggf. nicht mehr verfügbar sind. Es ist zudem in der Rubrik Partition Tolerance angesiedelt, wenn es eine flexible Verteilung der Daten auf separate Server erlaubt, die über ein Netzwerk miteinander kommunizieren. Lt. dem Theorem gibt es demnach nur die folgenden Kombinationsmöglichkeiten: CA, AP oder CP.[130]

Die Auswirkungen des Theorems auf eine Anwendung lassen sich dabei wie folgt ablesen: benötigt ein System strenge Konsistenzregeln (C), so muss es entweder auf die Verfügbarkeit (A) oder auf die Verteilung (P) verzichten. Beide Aspekte sind in der bisherigen Betrachtung bereits sichtbar geworden. Solange das System auf die Verteilung der Daten über mehrere Server verzichtet, sind über den zentralen Datenbankserver alle Daten des Systems zugänglich. Es kann daher unter der Annahme, dass der Datenbankserver selbst verfügbar ist, alle möglichen Anfragen von Clients erfolgreich bedienen. Dieses Schema entspricht der klassischen Behandlung von Daten über ein zentrales, relationales Datenbank-Management-System (bei Brewer: CA). Sind die Daten jedoch über mehrere Rechner verteilt, so muss das System zur Sicherstellung der Konsistenz der Daten, auf die Gesamtverfügbarkeit verzichten. Dieser Sachverhalt wird dadurch deutlich, dass zur Sicherung der Konsistenz über mehrere Server hinweg die Daten synchron über alle beteiligten Systeme geändert werden müssen. Unter Beibehaltung der ACID Transaktionseigenschaften werden die Änderungen an den Daten hierzu über ein Two- Phase- Commit Protokoll zwischen den Datenbankservern

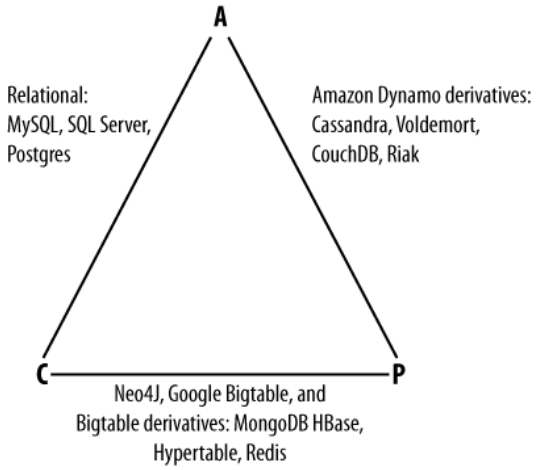
---

[130] vgl. Redmond, E. / Wilson, J. (2011), S. 230ff

abgestimmt. Ist jedoch einer der dafür notwendigen Server nicht mehr erreichbar, so kann die Transaktion nicht erfolgreich abgeschlossen werden – das Gesamtsystem ist in diesem Fall aus Sicht des Clients nicht mehr verfügbar (bei Brewer: CP).[131] Dabei ist es aus der Perspektive von Brewer unabhängig davon, ob die Daten über mehrere Server funktional verteilt oder mittels der Replikationsmechanismen dupliziert worden sind. Der kritische Aspekt der Betrachtung liegt in der synchronen Änderung der Informationen in den verteilten Datenbanken. Dieser sorgt dafür, dass durch die zunehmende Verteilung der einzelnen Sub-Systeme die Verfügbarkeit des Gesamtsystems sinkt.

Um die Verfügbarkeit eines verteilten Anwendungssystems daher zu erhöhen, müssen die kritischen Teile des Systems mehrfach redundant ausgelegt werden und die Synchronisation von Änderungen asynchron im Hintergrund erfolgen. Diese Vorgehensweise hat jedoch – wie bereits eingangs erwähnt – Auswirkungen auf die Konsistenz der Daten, da es zu temporären Diskrepanzen zwischen den Datenbeständen der einzelnen Server kommen kann (Brewer: AP). Man spricht in diesem Zusammenhang auch von “Eventually Consistent”.[132]

Abbildung 36: Einstufung einzelner Datenbanksysteme mittels CAP Theorem



Quelle: Hewitt, E. (2010), S. 21

[131] vgl. Pritchett, D. (2008), S. 50f

[132] vgl. Vogels, W. (2008), S. 15f

Dies klingt zunächst nach einer dramatischen Einschränkung, hat jedoch in der praktischen Ausführung lediglich zur Folge, dass es global betrachtet zu kurzfristigen Unterschieden zwischen den Ergebnissen von Anfragen kommen kann, da die letzten Datenänderungen noch nicht auf allen Duplikaten durchgeführt worden sind. Daher kann es passieren, dass nachfolgende Lesezugriffe auf Informationen noch nicht den aktuellen Zustand widerspiegeln. Das Anwendungssystem stellt jedoch intern sicher, dass die Datenänderungen auf den gespiegelten Servern nachvollzogen werden und somit letztendlich wieder ein konsistenter Stand im System vorherrscht. Dieses Verhaltensmuster wird auch als BASE bezeichnet. Es steht für basically available, soft state, eventually consistent und ist damit als das Gegenstück zum ACID Schema in der relationalen Datenbankwelt anzusehen.[133]

Für Fälle, in denen “eventually consistent” nicht ausreichend ist, bedient man sich in der Praxis eines Quorum Consensus Verfahrens: ist N die Anzahl der Replikate in einem Sub-System, R die Menge der Server, die für einen lesenden und W die Zahl der Server, die für einen schreibenden Zugriff erfolgreich kontaktiert werden müssen, so wird die Konsistenz der Daten sichergestellt, indem gilt: W+R > N. Hierdurch wird erreicht, dass sich das Set an Servern, die zum synchronen Schreiben und Lesen der Daten benutzt werden, überlappt und somit das System in jedem Fall den aktuell gültigen Datensatz erhält. Nachteilig dabei ist, dass die Verfügbarkeit des Gesamtsystems jedoch auch dadurch in Mitleidenschaft gezogen wird; bspw. für den Fall wenn die für eine erfolgreiche Schreiboperation notwendige Anzahl von Servern nicht mehr verfügbar ist. Wird die Anzahl W > 1 gesetzt, bedeutet dies immer, dass die Datenänderungen synchron auf mehrere Server zu spiegeln sind – mit den bereits betrachteten Konsequenzen für die Anwendung (bei Brewer: CP). Ist jedoch die Anzahl W = 1 und N > 1, so entspricht diese Konfiguration dem Modell der asynchronen Verteilung der Daten über mehrere Systeme, wodurch die Verfügbarkeit der Anwendung aus Sicht des Clients sichergestellt ist, jedoch die stetige Datenkonsistenz im Gesamtsystem eingeschränkt wird (bei Brewer: AP).[134]

---

[133] vgl. Pritchett, D. (2008), S. 51ff

[134] vgl. Vogels, W. (2008), S. 18

5.4.2 Master-Slave Replikation

In einer Master-Slave Konfiguration fungiert ein Datenbankserver als Master, von dessen Datenbestand mehrere Slave-Datenbankserver abgeleitet sind. Die Slave Server stehen hierbei allerdings nur für einen Lesezugriff aus der Anwendung heraus zur Verfügung. Alle Schreiboperationen in diesem Verbund müssen über das Master System erfolgen, welches die Änderungen synchron oder asynchron an alle konfigurierten Slave Systeme weiterleitet. Die Anzahl der Slave Systeme ist daher nicht begrenzt; diese können auch geografisch weit auseinander liegen. Aufgrund der für die Schreibvorgänge bestehenden, starken Abhängigkeit zum Master Server bietet sich die Master-Slave Replikation jedoch eher für Anwendungen oder deren Teilbereiche an, deren Lese-/Schreibverhältnis eindeutig mehr in Richtung lesendem Zugriff tendiert. Da diese Form der Replikation leicht umzusetzen ist, wird sie auch von vielen gängigen Datenbanksystemen direkt unterstützt.[135]

Problematisch wird der Einsatz der Master-Slave Replikation dann, wenn der Master wegen eines technischen Defekts oder einer fehlenden Netzwerkverbindung nicht mehr ansprechbar ist. Neuere Datenbanksysteme – wie bspw. MongoDB – implementieren hier einen Abstimmungsprozess, bei dem die noch vorhandenen Slave Server auf Basis ihres jeweiligen Replikationsfortschritts eigenständig ein System als neuen Master auswählen und somit den weiteren Betrieb der Anwendung sicherstellen.[136]

In einem traditionellen Cluster mit relationalen Datenbank-Management-Systemen kommt hierzu i.d.R. ein eigenes Überwachungssystem zum Einsatz. Dieses kontrolliert stetig die Verfügbarkeit des Master-Servers mittels eines Heartbeat Signals und schaltet bei dessen Ausbleiben auf ein separates Standby-System um, welches ab diesen Zeitpunkt die Aufgaben des Master Systems übernimmt (Hot-Standby-Cluster). Damit beide Systeme auf dieselbe Datenbank aufsetzen können, liegen deren Datenbank-Dateien auf einem von beiden Servern erreichbarem NAS oder SAN.[137]

---

[135] vgl. Hewitt, E. (2010), S. 88

[136] vgl. Chodorow, K. / Dirolf, M. (2010), S. 135ff

[137] vgl. Bengel, G. / Baun, C. et al. (2008), S. 423f

5.4.3 Master-Master Replikation

Im Gegensatz zur Master-Slave Replikation, welche ein dediziertes System im Verbund zum Master deklariert, auf dem dann alle Schreibzugriffe erfolgen müssen, verfolgt die Master-Master Replikation einen Peer-to-Peer Ansatz, bei dem alle Systeme gleichgestellt und für Änderungsanfragen verfügbar sind. Dadurch wird das Installieren und Betreiben des Clusters wesentlich vereinfacht. Zudem existiert in diesem Verbund kein Single-Point-of-Failure, so dass auch zusätzliche Koordinationsmaßnahmen im Falle des Ausfalls eines Masters nicht mehr notwendig sind.[138] Dem gegenüber stehen jedoch erhöhte Aufwände, um die Daten im System zu synchronisieren. Da in diesem Falle Änderungen an einem Datensatz auf mehreren Servern gleichzeitig vorkommen können, muss das System diesen Konflikt erkennen und auflösen. Dazu muss zu jeder Information eine eindeutige Versionskennung im System hinterlegt werden. Hierfür werden in den meisten Datenbank-Management-Systemen, die über eine Master-Master Replikation verfügen, sogenannte Vektoruhren eingesetzt. Durch deren Nutzung wird für jede Änderung an einem Datum ein eindeutiges Token erzeugt, das zwar unabhängig von der Zeiteinstellung und Zeitzone des jeweiligen Systems ist, aber dennoch die Eigenschaft besitzt, die Reihenfolge von Änderungen nachvollziehbar zu machen. Dieses Token dient anschließend dazu, den gerade aktuellen Datensatz herauszufinden und damit alle Systeme im Verbund auf den neuesten Stand zu bringen. Die Aktualisierung der Systeme erfolgt dabei asynchron im Hintergrund, wodurch die Verfügbarkeit des Gesamtsystems gewährleistet wird (Modell AP).[139]

Die Zuteilung der Daten auf die vorhandenen Systeme im Cluster erfolgt i.d.R. nach vorgegebenen Kriterien oder alternativ durch den Einsatz eines Hash-basierten Ring Verfahrens (wie bspw. bei Riak, siehe auch Kapitel 5.3.2).[140]

Wegen der Gleichberechtigung der Systeme hat eine Master-Master Replikation den Vorteil, dass in Anwendungsbereichen mit vielen gleichzeitig eintreffenden Änderungsanfragen nicht auf einen einzigen Master Server zurückgegriffen werden muss und somit die Gesamtleistung des Systems aufrecht erhalten werden kann.

---

[138] vgl. Hewitt, E. (2010), S. 14f

[139] vgl. Redmond, E. / Wilson, J. (2011), S. 79ff

[140] vgl. Hewitt, E. (2010), S. 110ff

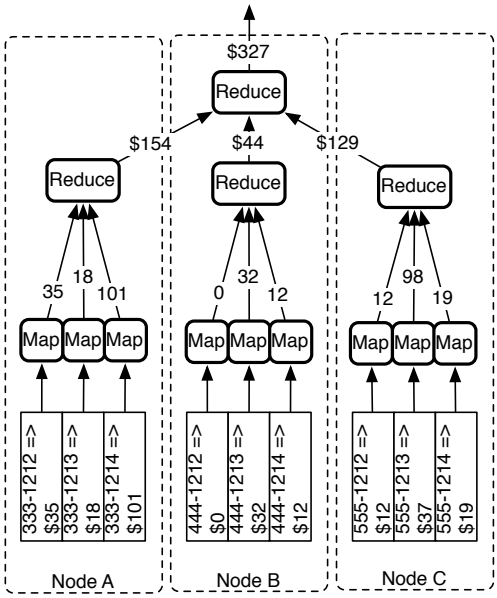
5.5 verteilte Datenanalyse per MapReduce

Mit der zunehmenden Aufteilung des Datenbestandes eines Anwendungssystems wird auch die Analyse der Daten mehr und mehr erschwert. Sind die Daten nicht nur funktional sondern auch mittels eines Verteilungsschlüssels auf mehrere Server im Verbund zerstreut, so muss die Anwendung bei der klassischen Auswertung der Daten zunächst mehrere verschiedene Datenbankserver kontaktieren und die passenden Daten von dort in ihren Arbeitsbereich transportieren. Sobald alle Daten im Arbeitsspeicher des Anwendungssystems verfügbar sind, kann es mit der Analyse der Informationen beginnen und ein Resultat für den Client erzeugen. Was für die Untersuchung eines kleinen Datenbestandes noch ein gangbarer Weg ist, wird mit zunehmendem Umfang der Datenbasis nicht mehr praktikabel durchführbar sein, da die Menge der von den einzelnen Datenbanksystemen an den Anwendungsserver zu übertragenden Informationen einfach zu umfangreich werden. Hier bietet sich der Einsatz des von Google entwickelten MapReduce Ansatzes an, welcher im Umkehrschluss nicht die Daten transportiert, sondern die Algorithmen zur Auswertung der Daten an die Datenbankserver schickt. Der Vorteil dabei ist: es ist i.d.R. schneller, den Algorithmus zu den Daten als umgekehrt die Daten zum Algorithmus zu schicken. Hierfür wird jedoch eine flexible und einfache Programmiersprache benötigt, die auf allen gängigen Betriebssystemen verfügbar sowie einsetzbar ist. Bei Google hat man sich daher für JavaScript entschieden. Der Anwendungsserver sendet zusätzlich mit der Datenbankabfrage zwei JavaScript Funktionen an die Server: eine Map-Funktion, welche die Ergebnisse der Abfrage Zeile für Zeile auf eine zum Resultat passende Datenstruktur abbildet und eine Reduce-Funktion, die daraus das Endresultat für jeden einzelnen Server ermittelt. Diese Aggregation kann dabei auch auf mehreren übereinander liegenden Stufen durchgeführt werden, wobei die Informationsquelle einer Ebene immer die Ergebnisse des vorgelagerten Systems darstellen. Abbildung 38 zeigt beispielhaft den Einsatz von MapReduce bei der Ermittlung einer Telefonrechnung unter der Annahme, dass die gesammelten Gesprächsdaten im Anwendungssystem nach einem Nummernpräfix auf verschiedene Server verteilt sind.[141]

---

[141] vgl. Redmond, E. / Wilson, J. (2011), S. 63ff

Abbildung 37: Einsatz des MapReduce Ansatzes



Quelle: Redmond, E. / Wilson, J. (2011), S. 65

6 Auswirkungen auf die Anwendungsarchitektur

6.1 Berücksichtigung der Skalierbarkeit

Die Skalierbarkeit eines Systems "bezeichnet die Fähigkeit [...] wachsende quantitative Anforderungen durch Hinzufügen von Ressourcen auszugleichen, ohne dass eine Änderung von Systemkomponenten notwendig wird".[142] Im Falle von global verteilten Anwendungssystemen handelt es sich hierbei um die Zunahme von Nutzern – und somit Anfragen an das System – als auch das Wachstum der von dem System zu verwaltenden Informationen. Dabei kann das System bzw. seine Sub-Systeme entweder auf vertikaler oder auf horizontaler Ebene skaliert werden. Bei der vertikalen Skalierung kann ein einzelnes System durch den Einbau von neuen, leistungsstärkeren Komponenten wie bspw. schnellere Mehrkern-Prozessoren, eine Erhöhung des RAM Speichers oder bessere Festplatten für die Steigerung des I/O Zugriffs erweitert werden. Diese Form stößt jedoch alsbald an ökonomische und physikalische Grenzen, da das System und dessen Wartung immer kostenintensiver bzw. aufgrund des derzeit technisch Machbaren ein Limit für die Erweiterbarkeit des Systems erreicht wird.

---

[142] Schill, A. / Springer, T. (2007), S. 6

Es ist daher sinnvoller, ein global verfügbares Anwendungssystem auf Basis der horizontalen Skalierung der einzelnen Sub-Systeme aufzubauen. Dieses System besteht aus mehreren preisgünstigen Standard PC Servern, die über ein Netzwerk miteinander kommunizieren und Informationen darüber austauschen.

Die Anwendungsarchitektur ist hierbei in drei Schichten zu unterteilen: die Präsentationsschicht in Form von Web Servern, die Anwendungsschicht mit der Geschäftslogik, welche innerhalb eines Applikationsservers läuft, und die Datenbankschicht mit den persistenten Daten der Anwendung. Während die höheren Schichten (Web- und Applikationsserver) i.d.R. bereits über einen Load Balancing Mechanismus verfügen, wird in vielen Systemen die Datenbankschicht durch ein zentrales, relationales Datenbank-Management-System zur Verfügung gestellt. Dieser Aufbau hat zwar den Vorteil, dass durch die Nutzung von lose gekoppelten sowie zustandslosen Komponenten in den Web- und Anwendungsschichten, die zunehmende Anzahl der Nutzeranfragen durch das Hinzufügen von neuen Ressourcen recht einfach begegnet werden kann, jedoch stellt sich rasch die zentrale Datenbankschicht als Flaschenhals heraus. Hierdurch wird die Anwendung nicht nur bzgl. des Umfangs der gespeicherten Daten sondern letztendlich auch im Hinblick auf die Anzahl der gleichzeitig eintreffenden Anfragen beschränkt, da die Verarbeitung der Anfragen meist einen Zugriff auf die Datenbank zur Folge hat. Ein erster möglicher Schritt ist daher, die Anzahl der lesenden Zugriffe auf den verschiedenen Schichten des Anwendungssystems durch das Einführen von zusätzlichen Data Cache Ebenen zu reduzieren. In diesen Cache Systemen können statische oder sich selten ändernde Daten wie bspw. Anwendungs- und Konfigurationsdateien als auch die Ergebnisse von häufig wiederkehrenden Routineanfragen temporär im schnellen Arbeitsspeicher vorgehalten und bei Bedarf der darüber liegenden Anwendungsschicht verfügbar gemacht werden – ohne die Dienste der nachgelagerten Schicht erneut bemühen zu müssen.

Jedoch stellt diese Form der Optimierung lediglich die Reaktionszeit des Servers sicher. Bei einer verteilten Web Anwendung spielen allerdings auch die Menge der vom Server zum Client zu übertragenden Daten sowie die dabei zurückzulegende Strecke eine wesentliche Rolle.

Im Falle der statischen Anwendungsdaten in Form von HTML und CSS Dokumenten sowie Bildern, Video- und Audiodateien bietet sich daher der Einsatz eines Content-Delivery-Netzwerkes an, um diese Informationen global zu verteilen und somit möglichst in der Nähe der Endanwender zu positionieren. Dadurch wird die Wegstrecke, welche die Daten der Anwendung bis zum Nutzer zurücklegen müssen, u.U. enorm verkürzt, wodurch eine konstante Datenrate sichergestellt werden kann, die besonders bei sequentiell zu verarbeitenden Daten (wie bspw. Multimedia Dateien) notwendig ist.

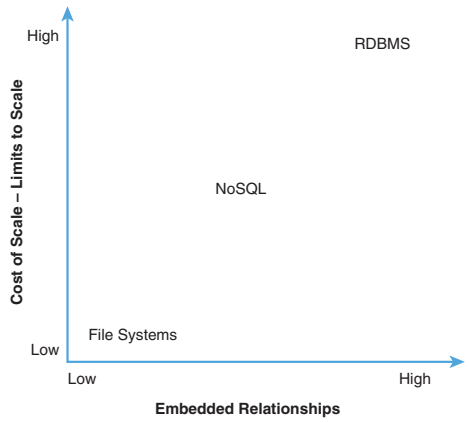
Es muss allerdings beachtet werden, dass der Einsatz von Cache Systemen aller Art nur für den lesenden Zugriff auf die Daten der Anwendung eine deutliche Entlastung des Gesamtsystems verspricht. Datenänderungen müssen hierbei dennoch durch alle Anwendungsebenen durchgereicht werden. Demnach ist im Vorfeld zu Prüfen, wie hoch das Verhältnis zwischen Lese- und Schreibzugriffen für den jeweiligen Bereich der Anwendung ausfällt.

Je nach Umfang der Daten und Anzahl von Schreibzugriffen muss daher auch die zentrale Datenbank weiter unterteilt werden, um eine horizontale Skalierung der Datenbankschicht des Systems zu erlauben. Diese als Sharding bezeichnete Technik kann den Datenbestand einer Anwendung nach funktionalen Kriterien oder mittels eines Schlüsselattributs innerhalb einer Entität, welches eine homogene Verteilung der Informationen innerhalb dieser Entität ermöglichen sollte, zerlegen. Für das Erzielen einer hohen Skalierbarkeit müssen allerdings beide Verfahren gleichzeitig berücksichtigt werden.

Da die meisten Bewegungsdaten innerhalb eines Anwendungssystems entweder eine örtliche oder zeitliche Komponente besitzen, bietet sich eine Aufteilung der Daten nach diesen Kriterien an. Für die Stammdaten einer Anwendung muss jedoch eine genauere Betrachtung des Datenbestandes erfolgen, da eine nachträgliche Änderung des Verteilungsschemas mit hohen Aufwänden und Risiken behaftet ist. Neuere Datenbank-Management-Systeme setzen in diesem Bereich auf eine automatische Verteilung der Stammdaten mit Hilfe eines Hash-basierten Ring Verfahrens, um eine homogene Distribution der Daten sicherzustellen.

Generell lässt sich konstatieren, dass sowohl die relationalen Datenbank-Management-Systeme als auch die multidimensionalen Datenbanksysteme auf eine vertikale Skalierung der Server ausgerichtet sind. Die Informationen in diesen Systemen horizontal über mehrere Server zu verteilen, ist mit den üblicherweise angebotenen Funktionen nicht realisierbar und benötigt entsprechende Mehraufwendungen im Datenbankdesign und der Entwicklung der vorgelagerten Anwendungskomponenten. Zusätzlich verlieren diese Systeme im Zuge der horizontalen Skalierung viele ihrer besonderen Eigenheiten, da mit der Anpassung der Anwendungsarchitektur vor allem eine Denormalisierung des relationalen Modells sowie ein Verzicht auf Fremdschlüssel-Einschränkungen und verteilten, ACID kompatiblen Transaktionen einhergeht. Daher muss sich bei der Konzeption der Datenbankschicht überlegt werden, ob der Einsatz eines für die verteilte Datenhaltung spezialisierten Datenbank-Management-Systems nicht sinnvoller erscheint – für die Ablage von strukturierten Daten bietet sich bspw. ein spaltenorientiertes Datenbank-System an.

Abbildung 38: Einfluss der Normalisierung auf die Verteilung von Daten



Quelle: Abbott, M. / Fisher, M. (2011), S. 58

6.2 Sicherstellung der Verfügbarkeit

Die Verfügbarkeit gibt an, wie oft ein Dienst oder System in einer angegebenen Zeitspanne zur Verfügung steht. Sie misst sich anhand der Ausfallrate und der Ausfalldauer der Einzelkomponenten des Systems. Um die Verfügbarkeit von Systemen zu erhöhen, können einzelne Komponenten des Systems mehrfach redundant ausgelegt werden.

Dadurch wird es möglich, selbst bei einem Totalausfall einer Komponente mit dem System weiterzuarbeiten, da dessen Dienste von anderen Teilen des Systems übernommen werden.

Diese mehrfach redundante Auslegung der Schichten einer Anwendung erfordert jedoch auch, dass die Komponenten lose miteinander verbunden und zustandslos sind. Der Aufbau der Kommunikation zwischen den einzelnen Schichten darf dabei erst zur Laufzeit erfolgen und muss die aktuelle Topologie des Systems berücksichtigen. Nicht mehr funktionsfähige oder über das Netzwerk nicht mehr ansprechbare Kommunikationspartner können anhand eines Heartbeat Signals erkannt und isoliert werden. Diese Vorgehensweise benötigt jedoch eine permanente interne Kommunikation und Abstimmung zwischen den Anwendungsschichten und funktioniert über geographisch separierte Standorte hinweg nur unzuverlässig.

Daher bietet sich für die globale Verteilung des Anwendungssystems eine Replikation der Daten über verschiedene Rechenzentren an. Für die Daten auf den Web Servern und Anwendungsservern kann hierbei auch auf die bereits genannten System-Images zurückgegriffen werden, die über eine Master-Slave Replikation per Dateitransfer in die einzelnen Rechenzentren übertragen werden. Diese Form der Replikation bietet sich außerdem für Daten an, die über einen hohen Anteil an Lesezugriffen innerhalb der Anwendung verfügen – bspw. Konfigurations- oder Stammdaten. Änderungen müssen bei diesem Szenario immer am Master-System erfolgen, welches die Informationen im Anschluss an alle Slave Systeme weiterleitet.

Ist der Anteil der schreibenden Zugriffe auf die Informationen hoch, so ist eine Master-Slave Konfiguration daher eine schlechte Wahl, weil hierbei das Master-System zum Engpass wird. Es ist daher auf eine kooperierende Variante der Replikation zu setzen. Diese basiert auf einem Peer-to-Peer Netzwerk, in dem alle angeschlossenen Server gleichberechtigt und für Änderungsanfragen verfügbar sind. Die daraus resultierenden Probleme hinsichtlich der Synchronisation der Datenänderungen, die nun evtl. gleichzeitig auf mehreren Servern denselben Datensatz betreffen können, lassen sich unter Einbeziehung einer Versionierung der Datensätze lösen.

Wichtig dabei ist, dass die Versionsinformationen (i.d.R. per Vektoruhren realisiert) einen Aufschluss über die Reihenfolge der Datenänderungen ermöglichen, damit das System intern seinen Datenbestand eigenständig abgleichen kann. Zur Sicherstellung der Reaktionsgeschwindigkeit des Gesamtsystems sind hierbei alle Synchronisationsvorgänge asynchron im Hintergrund durchzuführen. Dafür werden in den meisten Fällen Warteschlangen benutzt, mit deren Hilfe die Nachrichten zwischen den einzelnen Servern ausgetauscht werden. Der Vorteil bei deren Anwendung liegt darin, dass sie eine nicht-flüchtige Speicherung der Nachricht übernehmen, wenn eines der Zielsysteme gerade nicht verfügbar sein sollte. Gleichzeitig stellen sie die chronologische Reihenfolge der Nachrichten durch eine sequentielle Verarbeitung auf der Seite des Empfängers sicher.

Es darf nicht unerwähnt bleiben, dass jede Form der asynchronen Replikation zu einem zeitlichen Versatz der Ausführung von Änderungsanweisungen führt, wodurch eine strikte Konsistenz der Daten innerhalb des Gesamtsystems nicht mehr zwingend gegeben ist.

6.3 Aufrechterhaltung der Wartbarkeit

Ein hoch komplexes, verteiltes Anwendungssystem ist nach dem service-orientierten Architekturansatz zu entwickeln. Dadurch wird die komplette Funktionalität der Anwendung in einzelne, funktional getrennte und autark agierende Sub-Systeme zerlegt, die sich selbständig entwickeln, betreiben und anpassen lassen. Um im laufenden Betrieb die Funktionalität der Gesamtanwendung sicherzustellen, müssen alle Teilbereiche über ausgereifte Protokollierungs- und Überwachungsmethoden verfügen. Diese laufen zentral in einem Steuerungssystem zusammen und können bei Überschreitung von vorher definierten Kennzahlen – bspw. für maximale Antwortzeiten des Systems – eine Alarmmeldung auslösen. Ebenfalls kann hierüber der Zustand und die Auslastung einzelner Sub-Systeme akribisch nachverfolgt werden und u.U. neue Ressourcen dem System hinzugefügt respektive nicht mehr benötigte oder defekte Ressourcen dem System entzogen werden.

Für die Datenhaltungsschicht ist diese Vorgehensweise jedoch nicht so einfach umzusetzen, da aufgrund der Schlüsseldefinitionen für das Aufteilen des Datenbestandes u.U. bereits ein festes Schema vorgegeben ist. Dieses im Nachhinein zu verändern, bedeutet im Worst-Case-Szenario, die kompletten Daten des Systems neu über die Datenbankserver zu verteilen. Es muss daher enorme Vorsicht bei der Definition der Verteilungskriterien für den Datenbestand des Systems gelten.

Des Weiteren treten beim Einsatz eines relationalen Datenbank-Management-Systems Probleme mit nachträglichen Schemaanpassungen auf, wenn der Datenbestand in den Tabellen eine entsprechende Größe erreicht und / oder die Daten zusätzlich über mehrere Systeme verteilt worden sind. Die nachträglichen Anpassungen an den Tabellen sind dann nicht mehr so einfach möglich, da evtl. Abhängigkeiten zu anderen Systemmodulen berücksichtigt werden müssen. Aufgrund des einheitlichen Schemas für alle Daten innerhalb einer Tabelle dauert das Ändern des Schemas umso länger, je mehr Daten in dieser Tabelle enthalten sind. Hier bieten alternative Datenbank-Management-Systeme, wie bspw. die schemalosen spalten- oder dokumentorientierten Datenbanken, enorme Vorteile an.

Mit zunehmender Größe der Daten in einem Anwendungssystem ist auch das Thema Datensicherung und -wiederherstellung nicht mehr so leicht zu lösen. Hier hat sich gezeigt, dass eine Verteilung und Replikation der Daten über mehrere geographisch unabhängige Rechenzentren einen Schutz vor Datenverlust bieten kann. Auch der Einsatz eines Datenbank-Management-Systems, welches im Falle des Ersatzes eines ausgefallenen Servers die Daten selbständig wieder auf den neuen Server spiegelt, sorgt für eine enorme Vereinfachung im täglichen Betrieb des Anwendungssystems.

Sämtliche Daten des Anwendungssystems können dabei entweder auf den per RAID-5 konfigurierten Festplatten der einzelnen Server oder auf einen oder mehreren SAN Laufwerken im Netzwerk liegen. Damit wird ein Zusammenschluss der einzelnen Speichermedien zu einem großen, verteilten Datenspeicher ermöglicht, den die Anwendung für die Ablage ihrer Daten und den Zugriff auf die Informationen benutzen kann.

6.4 Sicherstellung der Datenkonsistenz

Zur Wahrung der strikten Konsistenz der Daten müssen zunächst weiterhin die ACID Regeln der relationalen Datenbankwelt in Betracht gezogen werden. Diese lassen sich mit der zunehmenden Verteilung und Replikation der Daten über mehrere per Netzwerk verbundene Einzelsysteme nur gewährleisten, wenn auf die Verfügbarkeit des Gesamtsystems verzichtet wird. Daher muss für die Funktionalität jedes einzelnen Sub-Systems der Anwendung erwogen werden, inwieweit dessen angebotene Dienstleistung für den Betrieb der Gesamtanwendung zwingend notwendig ist. In diesem Fall sind Abstriche bei der stetigen Konsistenz der Daten des Sub-Systems zu machen, um dessen Verfügbarkeit sicherstellen zu können.

Handelt es sich jedoch um kritische Daten, bei denen eine konsistente Datenhaltung unumgänglich ist, so kann mit Hilfe eines Quorum Consensus Verfahrens beim Lese- und Schreibzugriff auf die Informationen in den verteilten Datenbanken sichergestellt werden, dass immer der aktuell gültige Datensatz zurückgeliefert wird. Dieses Vorgehen hat jedoch negativen Einfluss auf die Verarbeitungsgeschwindigkeit beim schreibenden Zugriff auf die Daten als auch auf die Verfügbarkeit des jeweiligen Sub-Systems, da die für die synchrone Verteilung der Informationen benötigten Ressourcen stets verfügbar sein müssen.

Der Anteil der Daten, die diesen strengen Restriktionen unterliegen, ist jedoch in fast allen Anwendungssystemen gering. Ein Großteil der Szenarien lässt sich daher auf die durch das BASE Verfahren bekannten Einschränkungen für die stetige Konsistenz der Daten abbilden. BASE steht dabei für basically available, soft state, eventually consistent. Darunter wird verstanden, dass eine Anwendung gemäß des CAP Theorems Abstriche bei der Konsistenz der Daten machen muss, wenn sie besonderen Wert auf die Verfügbarkeit legt und die Daten über ein Netzwerk auf verschiedene Server verteilt sind. Praktisch bedeutet der Verzicht der Konsistenz jedoch nicht, dass das System dauerhaft in einem inkonsistenten Zustand verbleibt – es geht vielmehr um den zeitlichen Versatz, der bei der Übertragung der Änderungen innerhalb des Systems auftreten kann.

Dieser ist naturgemäß abhängig von der Verfügbarkeit der Replikate, der aktuellen Netzwerkauslastung, dem Abstand zwischen den einzelnen Systemen sowie dem Umfang der zu übermittelnden Daten. Hier sollte beim Design der Anwendung versucht werden, die Größe der Daten im Falle eines Änderungsvorganges zu minimieren – bspw. durch den Einsatz eines Delta-Verfahrens, welches nur die Differenzen mittels eines optimierten, binären Protokolls überträgt.

Etwaige Problemstellungen, die eine Berichtigung von bereits partiell durchgeführten Datenänderungen nach sich ziehen, werden in global verteilten Systemen mittels Korrekturbuchungen – auch als Compensation bezeichnet – im Nachhinein von der Geschäftslogik auf der Anwendungsebene durchgeführt. Damit kann auf die synchrone Änderung der Daten und somit auf verteilte, ACID kompatible Transaktionen über ein aufwendiges Two-Phase-Commit Protokoll komplett verzichtet werden.

6.5 Blueprint für die Anwendungsarchitektur

Für die Architektur einer global verteilten Anwendung müssen die einzelnen Anwendungsschichten getrennt voneinander betrachtet werden. Dabei ist im Vorfeld die Gesamtfunktionalität des Anwendungssystems mittels des service-orientierten Architekturschemas (SOA) in funktional unabhängige Sub-Systeme aufzuteilen. Jedes Sub-System setzt sich anschließend aus den Schichten: Präsentationsschicht, Anwendungsschicht und Datenhaltungsschicht zusammen.

Die Präsentationsschicht einer web-basierten Anwendung besteht aus statischen, sich selten ändernden Daten, die entweder im Textformat oder als binäre Informationen vorliegen. Aufgrund der geringen Änderungshäufigkeit dieser Anwendungsdaten, die zudem i.d.R. eine globale Gültigkeit besitzen, lassen sich diese Informationen sehr leicht weltweit verteilen. Um diese Daten zusätzlich geographisch nah an die Endnutzer zu bekommen, bietet sich der Einsatz eines Content-Delivery-Netzwerkes an. Dadurch wird zum einen die Wegstrecke der Daten durch das globale Netzwerk verkürzt, als auch der Web Server im Rechenzentrum des Anbieters von den Anfragen nach statischen Inhalten entlastet. Das dafür notwendige, regionale Weiterleiten der Anfragen an die Server des Content-Delivery-Netzwerk Anbieters wird über entsprechende Einträge im globalen Verzeichnisdienst (DNS) ermöglicht.

Wird die Anwendung auf den Client des Endanwenders ausgeführt, so tauscht diese über eine HTTP-basierte Kommunikation mit den Servern des Anbieters Informationen im Hintergrund aus (AJAX). Dieser Nachrichtenaustausch findet dann direkt zwischen den Servern des Anbieters und dessen Nutzern statt. Auch hierbei wird für die Auflösung des verständlicheren Servernamens auf dessen IP Adresse eine DNS-Abfrage getätigt. Dadurch ist es möglich, die Anfragen je nach geographischer Herkunft auf verschiedene, global verteilte Rechenzentren weiterzuleiten. Damit diese Anfragen von den dortigen Servern beantwortet werden können, müssen die dafür benötigten Anwendungsdaten in allen Rechenzentren gleichermaßen vorhanden sein. Hier kann mit Hilfe von vorgefertigten System-Images der Aufwand für die globale Bereitstellung der Dienste minimiert werden. Diese lassen sich auf Dateisystem-Ebene leicht zwischen mehreren Standorten austauschen.

Die Anfragen sind zusätzlich von einem Load Balancer – bspw. einen Reverse Proxy Server – auf alle im Rechenzentrum verfügbaren Anwendungsressourcen zu verstreuen. Hierzu müssen die Komponenten und Module der Anwendungsebene frei von Zuständen und lose miteinander verbunden sein. Erst dadurch wird eine horizontale Skalierung der Anwendungsebene sichergestellt. Durch die zusätzliche Bereitstellung von Ressourcen in Form weiterer Anwendungsserver in einem Rechenzentrum wird außerdem die Verfügbarkeit dieses Teilbereiches der Anwendung erhöht. Zur schnellen Behebung von Störungen im Anwendungsbetrieb müssen die Systeme über ausreichende Protokollierungs- und Überwachungsfunktionen verfügen, damit etwaige Server- oder Netzwerkausfälle als auch die mögliche Überlastung des Gesamtsystems oder Teilen davon leicht diagnostiziert werden können.

Besonderes Augenmerk muss auf die Aufbewahrung der Laufzeitdaten der Anwendung gelegt werden. Diese lassen sich in Stammdaten, Bestandsdaten und Bewegungsdaten untergliedern. Die Stammdaten werden dabei recht selten verändert. Sie sind jedoch essentiell für die Erzeugung der Bewegungsdaten, da sie in diesen referenziert werden. Die Verarbeitung der Bewegungsdaten führt u.U. zu einer Anpassung der Bestandsdaten in einem Anwendungssystem.

Aufgrund des guten Lese-/Schreibverhältnisses bei den Stammdaten können diese sowohl in Data Cache Schichten innerhalb eines Rechenzentrums zwischengespeichert als auch mittels der Master-Slave Replikation zwischen verschiedenen Rechenzentren synchronisiert werden. Dabei bietet sich eine mehrfach redundante Ablage der Stammdaten in geographisch dispersen Rechenzentren zur Erhöhung des Ausfallschutzes und als zusätzliche Datensicherung an. Wird entweder der Umfang der Stammdaten oder die Anzahl der Anfragen so groß, dass sie nicht mehr von einem einzelnen Datenbanksystem verarbeitet werden können, sind diese über den als Sharding bezeichneten Mechanismus auf verschiedene Datenbankserver zu verteilen. Diese Aufteilung des Datenbestandes muss sowohl nach funktionalen Gesichtspunkten als auch mit Hilfe eines Schlüsselattributes in den Daten erfolgen.

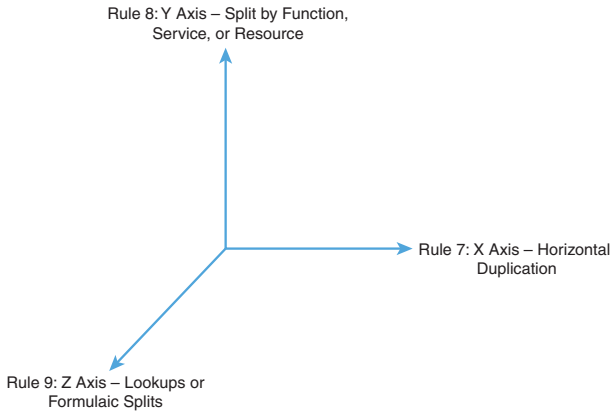
Da sowohl die Bestands- als auch die Bewegungsdaten eines Anwendungssystems eine örtliche und zeitliche Komponente haben, lassen sie sich sehr gut auf verschiedene Rechenzentren und Server aufteilen. Diese sollten dabei ebenfalls die Anfragen von Nutzern aus der näheren Region annehmen und verarbeiten. Je nach Anwendungsfall können die Daten aus den einzelnen Transaktionen über ein ETL-Verfahren in ein übergeordnetes, auf die nachträgliche Analyse der Informationen spezialisiertes Datenbank-Management-System übertragen werden. Hierbei ist es jedoch sehr wahrscheinlich, dass die global verstreuten Transaktionsdaten nicht mehr auf einem einzelnen Datenbankserver konsolidiert werden können. Eine horizontale Skalierung dieser Datenbankebene ist somit ein Muss.

Zur Sicherstellung der Verfügbarkeit der Informationen sind die einzelnen Systeme mehrfach zu replizieren. Hierbei muss jedoch das Verhältnis der Lese-/Schreibzugriffe untersucht werden, um sich zwischen einer Master-Slave (bei wenigen Schreib-zugriffen) oder einer Master-Master Replikation (bei vielen konkurrierenden Schreib-zugriffen) entscheiden zu können.

Mit den steigenden Anforderungen an das Anwendungssystem müssen dessen Komponenten durchgängig eine horizontale Skalierung ermöglichen. Dafür kann ein Anwendungssystem nach drei Kriterien aufgeteilt werden: die Aufspaltung nach Funktionalität, Dienstleistung oder Ressource (wird i.d.R. durch den SOA Ansatz

abgedeckt), die Verteilung der Daten nach einem Schlüsselattribut und das mehrfach redundante Vorhalten der Systeme. Lediglich wenn alle drei Maßnahmen durch das Anwendungssystem im Kleinen wie im Großen berücksichtigt werden, ist eine flexible Lastverteilung über alle Schichten des Systems hinweg umsetzbar.

Abbildung 39: Die drei Achsen der Skalierung von Systemen



Quelle: Abbott, M. / Fisher, M. (2011), S. 25

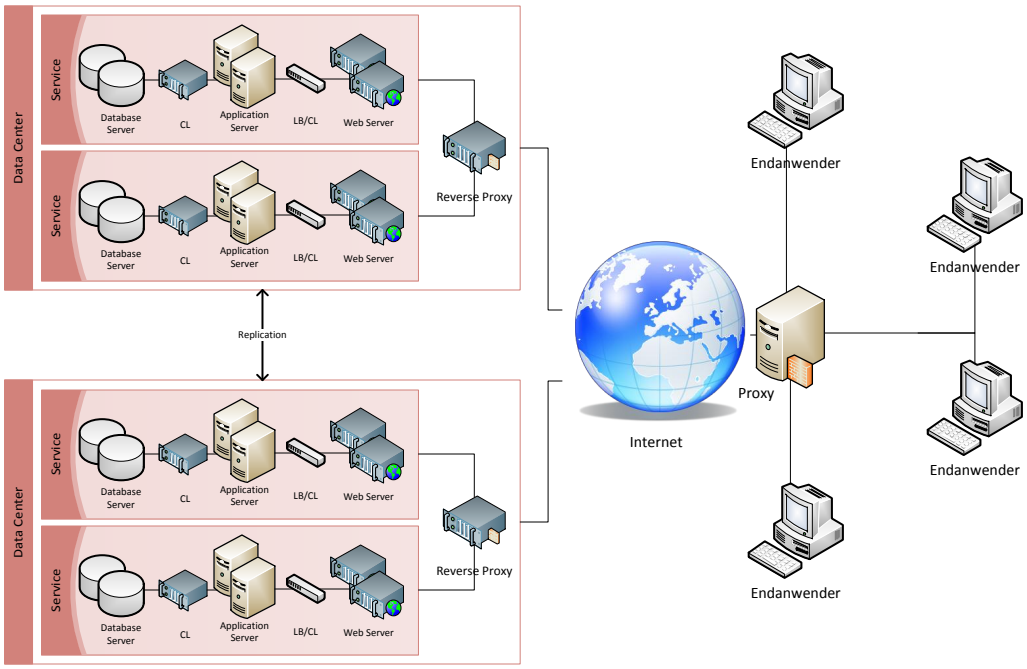
Es muss dabei beachtet werden, dass mit zunehmender Verteilung der Informationen in einem System die strikte Konsistenz der Daten nicht mehr garantiert werden kann. Hier muss für jedes einzelne Sub-System der Anwendung hinterfragt werden, ob eher eine hohe Verfügbarkeit des Dienstes oder eine strikte Datenkonsistenz innerhalb des Sub-Systems Vorrang hat.

Für die Ablage der Daten kommen dabei verschiedene Speichermöglichkeiten in Betracht. In jüngster Zeit hat sich allerdings herauskristallisiert, dass der generelle Einsatz von relationalen Datenbank-Management-Systemen für die persistente Speicherung jeglicher Informationen in einem Anwendungssystem ungünstig ist. Stattdessen muss abhängig vom Format und der Beschaffenheit der Daten nach einem optimierten Ablagesystem Ausschau gehalten werden. Durch die Zunahme von unstrukturierten und semi-strukturierten Informationen in den Datenbeständen eines Anwendungssystems haben sich außerdem neue Datenbank-Management-Systeme etabliert, die unter dem Oberbegriff NoSQL Datenbanken zusammengefasst werden.

Es ist bspw. für unstrukturierte, im Binärformat vorliegende Informationen wie Bilder, Video- oder Audiodateien sinnvoller, die Daten innerhalb eines verteilten Key-Value-Stores zu speichern. Sind die Informationen semi-strukturiert oder ist für die im Textformat vorliegenden Daten überhaupt kein wiederkehrendes Schema erkennbar, so können diese Informationen viel einfacher in einem dokumentorientiertem oder spaltenorientiertem Datenbanksystem abgelegt werden. Diese schemalosen Datenbanksysteme haben auch den Vorteil, dass eine Änderung des Datenbankschemas im Nachhinein wesentlich flexibler und problemloser als bei den relationalen Modellen möglich ist.

Für stark vernetzte Informationen, wo die eingegangenen Verbindungen eine hohe Relevanz für deren Auswertung besitzen, bieten sich die darauf spezialisierten Graph-Datenbanken als Ablagesystem an. Diese kommen jedoch meist nicht als alleinige Datenbank in Frage, da der Umfang der Meta-Informationen zu den einzelnen Knoten und Kanten recht begrenzt ist. Hierin lassen sich i.d.R. lediglich Zugriffsschlüssel zu Informationen aus einem weiteren Datenbanksystem hinterlegen.

Abbildung 40: Architektur eines global verteilten Anwendungssystem



Quelle: eigener Entwurf

7 Ausblick

Basierend auf den Vorarbeiten von Firmen wie Google, Amazon und Facebook haben sich verschiedene Open Source Datenbank-Systeme am Markt etabliert, die sich auf die verteilte Ablage von Informationen spezialisiert haben. Diese machen den bewährten relationalen Datenbank-Management-Systemen zunehmend Konkurrenz, da sie viele Unzulänglichkeiten der RDBMS, die mit der horizontalen Skalierung der Datenbankserver einhergehen, beseitigen.

Allerdings sind auch diese neuen Datenbank-Systeme – wie die Analyse in dieser Arbeit zeigt – nicht für jedes Szenario gleichermaßen geeignet, so dass in der Konzeption des Anwendungssystems eine Auswahl des optimalen Ablagesystems basierend auf der Beschaffenheit, Struktur und Änderungshäufigkeit der Daten erfolgen muss. Da die immer komplexer werdenden Anwendungssysteme nach dem SOA-Ansatz in funktional getrennte Sub-Systeme unterteilt werden, kann für jeden Dienst ein eigenes Ablagesystem ausgewählt werden, welches dessen Datenstrukturen und Zugriffsmethoden optimal unterstützt. Durch diese Vorgehensweise werden jedoch die bisher auf Datenbankebene vollzogenen Aufgaben zur Sicherstellung der Datenintegrität und -konsistenz mehr und mehr in die vorgelagerte Anwendungsschicht verschoben. Die Aufwände für deren Entwicklung und Qualitätssicherung steigen somit enorm an.

Der MapReduce Ansatz zeigt außerdem, dass es nicht nur allein um eine effiziente Speicherung der Daten innerhalb eines massiv verteilten Systems ankommt. Die wirklichen Herausforderungen stellen sich bei der Auswertung der immensen Daten ein, die in solchen Systemen über den ganzen Globus verteilt gesammelt worden sind. Da die aktuellen NoSQL Datenbanksysteme i.d.R. weder ad hoc Abfragen noch eine universelle Abfragesprache anbieten, muss daher jede neue Datenanalyse als Funktion im Anwendungsprogramm umgesetzt werden. Inwieweit dieses Vorgehen langfristig tragbar ist, kann derzeit nicht abgeschätzt werden. Es lässt sich jedoch konstatieren, dass sich beide Datenbank-Welten mit der Zeit funktional aneinander angleichen müssen, um somit die Vorzüge aus beiden Systemen zu vereinen.

Anhangsverzeichnis

Anhang **Seite**

Quellenverzeichnis

a) Monographien

Abbott, M. / Fisher, M. (2010),
The Art of Scalability, Addision-Wesley Pearson Education, Boston MA, 2010

Abbott, M. / Fisher, M. (2011),
Scalability Rules, Addison-Wesley Pearson Education, Boston MA, 2011

Banker, K. (2011),
MongoDB in Action, Manning Publications, Greenwich CT, 2011

Bengel, G. / Baun, C. et al. (2008),
Masterkurs Parallele und Verteilte Systeme, Vieweg+Teubner, GWV Fachverlage GmbH Wiesbaden, 2008

Bodendorf, F. (2006),
Daten- und Wissensmanagement, 2. Auflage, Springer Verlag Berlin, Heidelberg, 2006

Chodorow, K. / Dirolf, M. (2010),
MongoDB: The Definitive Guide, O'Reilly Media Inc, Sebastopol CA, 2010

Engels, C. (2009),
Basiswissen Business Intelligence, W3L-Verlag, Herdecke, 2009

Gillenson, M. (2005),
Fundamentals of Database Management Systems, John Wiley & Sons Inc., Hoboken NJ, 2005

Hewitt, E. (2010),
Cassandra: The Definitive Guide, O'Reilly Media Inc, Sebastopol CA, 2010

Horn, C. / Kerner, I. / Forbrig, P. (2003),
Lehr-und Übungsbuch Informatik, 3. Auflage, Fachbuchverlag Leipzig, München, 2003

Hunt, A. / Thomas, D. (2000),
The Pragmatic Programmer, Addison Wesley Longman Inc., Reading Massachusetts, 2000

Josuttis, N. (2007),

SOA in Practice, O'Reilly Media Inc, Sebastopol CA, 2007

Kemper, A. / Eickler, A. (2011),

Datenbanksysteme - Eine Einführung, 8. Auflage, Oldenbourg Verlag, München, 2011

Mandl, P. (2010),

Grundkurs Betriebssysteme, 2. Auflage, Vieweg+Teubner Fachverlage GmbH, Wiesbaden, 2010

Redmond, E. / Wilson, J. (2011),

Seven Databases in Seven Weeks, B2.0 vom 19.12.2011,

Pragmatic Programmers LLC, Dallas, 2011

Schäfer, W. (2010),

Softwareentwicklung – Einstieg für Anspruchsvolle, Pearson Studium, München, 2010

Scherff, J. (2007),

Grundkurs Computernetzwerke, Vieweg & Sohn Verlag, Wiesbaden, 2007

Schill, A. / Springer, T. (2007),

Verteilte Systeme, Springer Verlag Berlin, Heidelberg, 2007

Stahlknecht, P. / Hasenkamp, U. (2005),

Einführung in die Wirtschaftsinformatik, 11. Auflage, Springer Verlag Berlin, Heidelberg, 2005

Tanenbaum, A. (2008),

Verteilte Systeme - Prinzipien und Paradigmen, 2. Auflage, Pearson Education Deutschland GmbH München, 2008

Weber, J. et.al. (2010),

REST in Practice, O'Reilly Media Inc, Sebastopol CA, 2010

b) Zeitungsartikel

Bayer, M. (2011a),

Master-Data-Management sorgt für den Durchblick, in: Computerwoche Nr. 27-28, 04.07.2011, S.12ff

Bayer, M. (2011b),

Big Data – die Datenflut steigt, in: Computerwoche Nr. 46, 14.11.2011, S.14ff

Bayer, M. (2011c),

Baustelle Daten-Management, in: Computerwoche Nr. 46, 14.11.2011, S.18ff

Florescu, D. (2005),

Managing Semi-Structured Data, in: acm Queue Vol. 3 No. 8, October 2005, S. 18ff

Jacobs, A. (2009),

The Pathologies of Big Data, in: acm Queue Vol. 7 No. 6, July 2006, S. 1ff

Keppler, K. (2007),

Am Drücker, in: iX 6/2007, Juni 2007, S. 122ff

Leighton, T. (2008),

Improving Performance on the Internet, in: acm Queue Vol. 6 No. 6, October 2008, S. 20ff

Pritchett, D. (2008),

BASE - An ACID Alternative, in: acm Queue Vol. 6 No. 3, May/June 2008, S. 48ff

Rys, M. (2011),

Scalable SQL, in: acm Queue Vol. 9 No. 4, April 2011, S. 1ff

Souders, S. (2008),

High Performance Web Sites, in: acm Queue Vol. 6 No. 6, October 2008, S. 30ff

Vogels, W. (2008),

Eventually Consistent, in: acm Queue Vol. 6 No. 6, October 2008, S. 15ff

c) Internet Quellen

Amazon (2007),

Dynamo: Amazon's Highly Available Key-value Store,

http://www.allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf,

Stand: 04.01.2012

Brewer, E. (2000),

Towards Robust Distributed Systems,

http://ww.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf,

Stand: 20.01.2012

Cole, Jeremy (2011),

Big and Small Data at @Twitter,

http://www.youtube.com/watch?v=5cKTP36HVgI,

Stand: 20.01.2012

Datendieter (2011),

Buchstabenhäufigkeit von deutschen Nachnamen,

http://www.datendieter.de/item/Buchstabenhaeufigkeit_von_deutschen_Nachnamen,

Stand: 06.02.2012

Einstein, A.,

http://www.zeit-und-wahrheit.de/einstein-zitat-wir-leben-in-einer-zeit-vollkommener-13400/,

Stand: 12.02.2012

Facebook (2009),

Cassandra - A Decentralized Structured Storage System,

http://www.cs.cornell.edu/projects/ladis2009/papers/lakshman-ladis2009.pdf,

Stand: 04.01.2012

Google (2006),

Bigtable: A Distributed Storage System for Structured Data,

http://research.google.com/archive/bigtable-osdi06.pdf,

Stand: 04.01.2012

HighScalability (2009),

Scaling Twitter: Making Twitter 1000 Percent faster,

http://highscalability.com/blog/2009/6/27/scaling-twitter-making-twitter-10000-percent-faster.html,

Stand: 06.02.2012

HighScalability (2011),

The Real News is Not that Facebook serves up 1 Trillion Pages A Month,

http://highscalability.com/blog/2011/9/23/the-real-news-is-not-that-facebook-serves-up-1-trillion-page.html,

Stand: 06.02.2012

Intel (2011),

Intel Solid-State Drives: an introduction,

http://www.youtube.com/watch?v=UpYOlPyo0b0,

Stand: 20.01.2012

Manhard, K. (2008a),

Business Intelligence (Teil 3): Datenmodellierung – Relationale und Multidimensionale Modelle,

http://www.tecchannel.de/server/sql/1744994/business_intelligence_teil_3_datenmodellierung_relationale_und_multidimensionale_modelle/index4.html,

Stand: 04.01.2012

Manhard, K. (2008b),

Business Intelligence (Teil 3): Datenmodellierung – Relationale und Multidimensionale Modelle,

http://www.tecchannel.de/server/sql/1744994/business_intelligence_teil_3_datenmodellierung_relationale_und_multidimensionale_modelle/index10.html,

Stand: 09.01.2012

Manhard, K. (2008c),

Business Intelligence (Teil 3): BI Analyzemethoden OLAP & Data Mining,

http://www.tecchannel.de/server/sql/1739309/business_intelligence_teil_4_bi_analysemethoden_olap_data_mining/index4.html,

Stand: 09.01.2012

MSDN (2012),

JOIN Operations,

http://msdn.microsoft.com/en-us/library/bb397908(d=printer).aspx,

Stand: 09.01.2012

Perez, Sarah (2012),

Visualizing Facebook's Media Storage,

http://techcrunch.com/2012/02/02/visualizing-facebooks-media-store-how-big-is-100-petabytes/,

Stand: 04.02.2012

Riak (2012),

Concepts,

http://wiki.basho.com/Concepts.html,

Stand: 06.02.2012

Twitter (2011),

#numbers,

http://blog.twitter.com/2011/03/numbers.html,

Stand: 20.01.2012

Ehrenwörtliche Erklärung

Ich versichere, dass ich die beiliegende Projektarbeit ohne Hilfe Dritter und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt und die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Essen, 16. Februar 2012

_____

Unterschrift