2

Berichte aus dem
Karl-Steinbuch-
Forschungsprogramm

Karl Steinbuch Forschungsprogramm

MFG Stiftung Baden-Württemberg

Jens Kohler
Thomas Specht

# Sicheres Cloud Computing durch vertikal verteilte Datenstrukturen

Berichte aus dem
Karl-Steinbuch-
Forschungsprogramm
Nr. 2

Karl Steinbuch Forschungsprogramm

MFG Stiftung Baden-Württemberg

# Sicheres Cloud Computing durch vertikal verteilte Datenstrukturen

Jens Kohler
Thomas Specht

2

**Impressum**

Herausgeber:
MFG Stiftung Baden-Württemberg
Breitscheidstraße 4
70174 Stuttgart
Tel. +49 711.90 715 300
Fax +49 711.90 715 350

Ansprechpartnerin:
Dr. Andrea Buchholz
buchholz@mfg.de



Gestaltung: Ingo Juergens, Südgrafik

# Zusammenfassung

Die hier vorliegende Arbeit stellt die Konzeption und Implementierung der vertikalen Datenpartitionierung von Datenbanktabellen dar – ein vielversprechendes Konzept, um aktuellen Datenschutz- und Datensicherheitsbedenken im Cloud Computing zu begegnen. Dabei werden Daten nicht bei einem einzigen Cloud-Anbieter gespeichert, sondern auf mehrere öffentliche und private Cloud-Anbieter logisch aufgeteilt. Somit befindet sich in jeder Cloud nur ein kleiner Teil der Daten, der ohne die anderen Teile wertlos ist. Die Daten können von den Anbietern selbst nicht, z. B. zu Werbezwecken, missbraucht oder weiterverkauft werden. Dasselbe gilt für Hackerangriffe, bei denen Daten gestohlen werden.

Basierend auf dem abgeschlossenen Forschungsprojekt SeDiCo (A Secure and Distributed Cloud Datastore) wird in diesem Bericht beschrieben, wie der Ansatz der vertikalen Datenpartitionierung auf seine technische Machbarkeit geprüft, prototypisch implementiert und die Implementierung anschließend aus Machbarkeits- und Performancegesichtspunkten evaluiert wurde.

Zunächst wird der Begriff „Cloud Computing“ definiert, um ein einheitliches Verständnis zu schaffen. Außerdem wird die Problemstellung im Einführungskapitel anhand eines konkreten Szenarios mit Kundendaten dargestellt. Danach werden im zweiten Kapitel mit aktuellen Datenschutz- und Datensicherheitsproblematiken, dem „Vendor lock-in“ und der fehlenden Vergleichbarkeit von Cloud Diensten, drei zentrale Grundproblematiken des Cloud Computings dargestellt. Dementsprechend werden Lösungskonzepte für diese Herausforderungen im darauffolgenden dritten Kapitel erarbeitet. Die angesprochenen Lösungskonzepte münden in einen implementierten Prototypen, der im vierten Kapitel von der Architektur bis zu seinen Einsatzmöglichkeiten im Unternehmensumfeld beschrieben wird. Das Kapitel schließt mit einer Performanceevaluation des Prototyps, um dessen Integration in Unternehmenslandschaften bewerten zu können. Letztlich werden im fünften Kapitel Arbeiten und Projekte mit ähnlichen Konzepten und Architekturen vorgestellt, die diese Arbeit in einen wissenschaftlichen Gesamtkontext einordnen. Im darauffolgenden Fazit wird die Notwendigkeit weiterer wissenschaftlicher Arbeiten herausgestellt und im Ausblick werden dann die weiteren geplanten Forschungsarbeiten skizziert.

# Inhalt

# 1 Einführung

Das hier vorgestellt Konzept und seine Implementierung basieren auf der Idee, die Nutzung von „Cloud Computing“ sicherer zu machen als bisher.[1] Die erhöhte Sicherheit entsteht dabei durch die Verteilung der Daten auf verschiedene Cloud-Provider. Obwohl der Begriff des „Cloud Computings“ sehr weit gefasst werden kann, haben die wissenschaftliche Community und auch die Industrie mittlerweile ein recht konkretes Bild davon, was das Cloud Computing charakterisiert. Die Definition des National Institute of Standards and Technology (Nist 2013) hat sich durchgesetzt, da dies die umfassendste und zugleich prägnanteste ist.

> „Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models. […]“.

U. a. werden in dieser Definition die drei Servicemodelle „Infrastructure as a Service“ (IaaS), „Platform as a Service“ (PaaS) und „Software as a Service“ (SaaS) genannt. Da die Servicemodelle aufeinander aufbauen, wird in der Literatur meist eine pyramidenförmige Darstellung, wie in ABB. 1, dafür verwendet.

1 Hintergrund ist das im Rahmen des Karl-Steinbuch-Forschungsprogramms durchgeführte Projekt SeDiCo – A Secure and Distributed Cloud Datastore. Siehe http://stiftung.mfg.de/de/forschungsforderung/abgeschlossene-projekte/sedico.

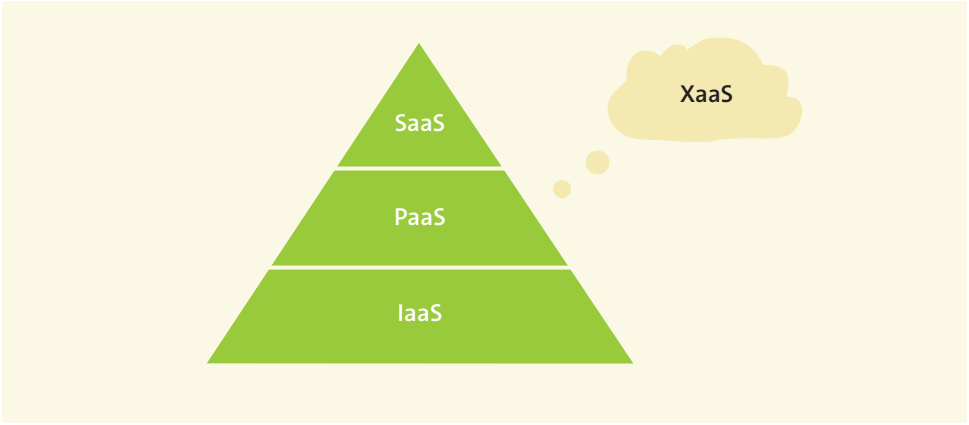


ABB. 1 Cloud Service-Modelle

So stellt das SaaS-Modell Anwendungen bereit, die direkt aus der Cloud ohne clientseitige Installation betrieben werden. Die Administration der darunterliegenden physischen Hardware, der Software sowie Betrieb, Wartung und Ressourcenanpassung sind Aufgaben des Cloud Anbieters. Die mittlere PaaS-Schicht hingegen, stellt Dienste für Entwicklungsumgebungen als Entwicklungsplattform bereit. Dabei

greift der Benutzer durch standardisierte Schnittstellen auf die jeweilige Programmierumgebung zu. Auch hier ist der Cloud-Betreiber für die grundlegende physische Hardware, das Betriebssystem, Softwareupdates, usw. zuständig. Der Anwender programmiert seine Anwendungen praktisch in der Cloud, mit allen Vorteilen, wie z. B. dynamische Anpassung der Ressourcen bei schwankenden Nutzerzahlen. Auf die unterste Schicht, die so genannte IaaS-Schicht, bauen alle anderen Servicemodelle auf. Der Anwender mietet Ressourcen in Form von physikalischer Hardware (Rechner, Massenspeicher, Netzwerk) und hat dabei völlige Entscheidungsfreiheit, wie er diese nutzen möchte. Die Basis dabei bilden virtuelle Maschinen (Betriebssysteme), die auf der gemieteten Hardware ausgeführt werden. Letztlich lassen sich dadurch PaaS- und SaaS-Modelle aufsetzen und betreiben (Hlipala 2014). In ABB. 1 als Wolke gekennzeichnet, ist der Begriff „XaaS", der für „Everything as a Service" steht. Das grundlegende Konzept dabei ist, alles was in der Cloud angeboten wird, als sog. „Service" zu begreifen. Das übergreifende Ziel ist somit einheitliche und definierte Angebote im Cloud Computing zu schaffen, die sich miteinander vergleichen lassen.

Unser Ansatz bewegt sich im „IaaS"-Bereich. Dies wurde aus Gründen der Anbieterunabhängigkeit und o. g. Entscheidungsfreiheit so gewählt, da auf dieser untersten Ebene die Kapselung der verschiedenen Anbieterschnittstellen in ein Abstraktionsframework am einfachsten zu realisieren ist[2]. In den oberen Schichten des Servicemodells („SaaS" und „PaaS") wäre dies aufgrund der Heterogenität der Dienste so nicht möglich gewesen. Durch Virtualisierungstechnologien lassen sich im „IaaS"-Modell Ressourcen dynamisch zu- und wieder abschalten (Meir-Huber 2010). Dies ermöglicht ein einfaches „scale-out"[3] aber auch ein einfaches „scale-up"[4] der Ressourcen (Maged et al. 2007). Im Gegensatz dazu bieten die anderen beiden Servicemodelle diese Möglichkeit nicht, da sie keinen direkten Zugriff auf die zugrundeliegenden Ressourcen ermöglichen.

Diese Arbeit hat die Erstellung eines Frameworks zur verteilten und sicheren Datenspeicherung in der Cloud zum Ziel. Die Unterscheidung in private, öffentliche und hybride Cloud wird dabei stets beachtet. Das Framework soll aber transparent in allen Cloud-Ausprägungsformen Anwendung finden. Den Anwendern wird eine Lösung angeboten, vorhandene Daten verteilt und damit sicher, auch verschlüsselt, über mehrere Cloud-Anbieter hinweg, abzulegen. Dadurch, dass es nicht mehr nötig ist, potenzielle Cloud-Provider im Vorhinein auszuwählen, wird durch SeDiCo ein Schritt in Richtung Datenschutz unternommen. Zudem wird die aktuelle Problematik des „Vendor lock-in" aktiv angegangen. Je leichter und transparenter es für die Konsumenten von Cloud-Diensten ist, den Provider zu wechseln, desto mehr wird diese Möglichkeit auch genutzt werden.

Ein beispielhaftes Anwendungsszenario stellt sich wie folgt dar: ein international tätiges Reiseunternehmen stellt sich die Frage, ob es nicht sinnvoll wäre, die ständig wachsende Kundendatenbank in eine Cloud auszulagern. Die skalierbare Rechenleistung würde komplexe Abfragen beschleunigen und wegen des dynamisch wachsenden Speichers, würden zukünftige Investitionen in neue physi-

2 So hätte die Arbeit auch im SaaS-Modell mit verschiedenen Anbietern und Datenbanken realisiert werden können. In diesem Modell hängt allerdings die dynamische Skalierung vom jeweiligen Cloud-Anbieter ab. Für eine unabhängige wissenschaftliche Evaluation war das IaaS-Modell daher vorzuziehen.

3 Bezeichnet die Erweiterung einer Infrastruktur um zusätzliche physische Maschinen, z. B. zur Lastverteilung über mehrere – im Idealfall – gleiche Maschinen

4 Bezeichnet die Erweiterung einer Infrastruktur um einzelne Hardwarekomponenten wie z. B. Arbeitsspeicher, Prozessoren, Festplatten, etc.

sche Festplatten entfallen. Ferner ließen sich in der Cloud Technologien wie RAID oder andere Techniken zur Datensicherung ohne direkten Zugang zur physischen Hardware realisieren. Da es sich im obigen Szenario um Kundendaten (Tabelle „Customer“, s. ABB. 2) handelt, muss die Wahl aus Gründen des Datenschutzes und aus gesetzlichen Gründen auf eine private Cloud-Lösung fallen. Die derzeit wohl am heißesten diskutierten Themen des Cloud Computings beziehen sich auf Datensicherheit- und Datenschutzaspekte. So beschränken rechtliche Bestimmungen wie z. B. die „OECD Guidelines“, die „European Community Directive 95/46/EC“ oder das BDSG § 4 den Einsatz von Clouds, da sie die Speicherung von Daten über Landesgrenzen hinweg gesetzlich verbieten (Accorsi et al. 2011).

Die hier dargestellte Lösung stellt nun ein Framework bereit, welches unabhängig von der zugrundeliegenden Datenbank die Daten auf Tabellenebene teilt. Der Ansatz der verteilen Datenbanken ist dabei nicht neu. Er wird gerade bei verteilten Architekturen und der Entwicklung von Komponentensoftware eingesetzt. Neu an diesem Ansatz ist die Verwendung verteilter Datenstrukturen, die Aufteilung einzelner Partitionen und deren Ablage auf verschiedenen Cloud-Lösungen. Denkbar wäre u. a. auch eine geteilte Ablage, bei der eine Partition in einer öffentlichen Cloud verschlüsselt liegt und die andere Partition in der privaten Cloud gehalten wird. Betrachtet man dabei die vertikale Partitionierung, so muss die Problematik des Datenschutzes neu bewertet werden. Gesetzt der Fall, dass die öffentlich abgelegte und ggf. verschlüsselte Partition in die falschen Hände gerät, sind die Daten ohne die andere, in der privaten Cloud abgelegte Partition wertlos. ABB. 2 verdeutlicht diesen Sachverhalt anhand der o. g. Kundendatenbank.

| Id | name | fore-name | birth | credit-Card | insur-anceNo | street | zip | reg-Cust | sol-Score |
|---|---|---|---|---|---|---|---|---|---|
| 1 | Cust1 | fname1 | 2001-01-01 | 12345 | 12345 | street1 | 11111 | 1 | 10 |
| 2 | Cust2 | fname2 | 2001-01-02 | 67890 | 11244 | street2 | 22222 | 0 | 20 |
| 3 | Cust3 | fname3 | 2001-01-03 | 12579 | 13468 | street3 | 33333 | 1 | 30 |
| 4 | Cust4 | fname4 | 2001-01-04 | 24689 | 17654 | street4 | 44444 | 0 | 40 |

public

| Id | name | fore-name | birth |
|---|---|---|---|
| 1 | Cust1 | fname1 | 2001-01-01 |
| 2 | Cust2 | fname2 | 2001-01-02 |
| 3 | Cust3 | fname3 | 2001-01-03 |
| 4 | Cust4 | fname4 | 2001-01-04 |

private

| Id | credit-Card | insur-anceNo | street | zip | reg-Cust | sol-Score |
|---|---|---|---|---|---|---|
| 1 | 12345 | 12345 | street1 | 11111 | 1 | 10 |
| 2 | 67890 | 11244 | street2 | 22222 | 0 | 20 |
| 3 | 12579 | 13468 | street3 | 33333 | 1 | 30 |
| 4 | 24689 | 17654 | street4 | 44444 | 0 | 40 |

ABB. 2 Anwendungsszenario der vertikalen Partitionierung

Die ursprüngliche Tabelle: „Customer“, (s. zuvor in ABB. 2) wurde vertikal partitioniert und die Daten an verschiedenen Orten abgelegt. Sollten die Daten, die in der öffentlichen Cloud abgelegt sind, gestohlen werden, so sind sie für den Dieb wertlos, denn es gibt keinen Bezug zum eigentlichen Kunden. Über die „Id“ kann kein Bezug zu den Daten der Kunden hergestellt werden, da diese ja immer noch in der privaten Cloud abgelegt sind.

Ferner wird der Aspekt der unterschiedlichen Anbieter-APIs im Rahmen dieser Arbeit konzeptionell und prototypisch adressiert. Der Prototyp stellt ein Framework bereit, das unterschiedliche Anbieterschnittstellen in einer weiteren Abstraktionsschicht vereint. Zentraler Fokus ist die Verteilung von Datenbankpartitionen auf unterschiedliche IaaS-Plattformen. Da jeder IaaS-Anbieter nur einen Teil der Daten besitzt, wird der Datenschutz und die Datensicherheit gerade in öffentlichen Clouds erhöht.

Kapitel 2 greift nun drei zentrale Fragestellungen dieser Vorgehensweise auf und stellt diese genauer dar, bevor in den darauffolgenden Kapiteln 3 – 4 näher auf die erarbeiteten Lösungen eingegangen wird.

# 2 Problemstellungen

Der Ansatz adressiert drei Themen, die in diesem Kapitel nun eingeführt werden. Dabei wird der Aspekt des Datenschutzes und der -sicherheit primär behandelt. An zweiter Stelle folgt die Problematik des Vendor lock-ins gefolgt von der Analyse der fehlenden Möglichkeit aktuelle Cloud-Anbieter schnell und effizient zu vergleichen.

## 2.1 Datenschutz und -sicherheit

Cloud Computing als eines der aktuellen Hype-Themen erfährt durch aktuelle Datenschutzskandale, Spionageversuche und Sicherheitslücken enormen Gegenwind. Es ist aktuell eines der meist beachteten Themen in der Weiterentwicklung der IT-Landschaft. Endverbraucher wie auch Firmen sehen eine schier unglaubliche Explosion an Cloud Computing-Anwendungen. Wie bereits dargestellt (s. Kapitel 1) sieht die Definition des National Institute of Standards and Technology (NIST) im Cloud Computing drei zentrale aufeinander aufbauende „Service Modelle". Als Beispiele seien hier exemplarisch für SaaS Salesforce [5], für PaaS Google AppEngine [6] und für IaaS Amazon EC2 [7] genannt. Allerdings sehen sich diese Angebote mit wachsenden Bedenken im Bereich Datenschutz und Datensicherheit aufgrund regelmäßig auftretender Sicherheitslücken und Spionageversuche staatlicher sowie privater Organisationen konfrontiert. Nicht nur unter Privatanwendern, sondern auch im Unternehmensbereich erhöht sich die Skepsis Dienste und Anwendungen aus der Cloud zu nutzen. Auf der anderen Seite verspricht die dynamische Skalierbarkeit von Rechnerressourcen, wie z. B. Speicherplatz, Arbeitsspeicher und Prozessorleistung und deren minuten- bzw. stunden- genaue Abrechnung enorme Kostenvorteile.

Das zentrale Element dieser Publikation ist nun die Konzeption und Implementierung eines Verteilungsalgorithmus zur vertikalen Partitionierung. Dieser wird benötigt, um Datensätze auf unterschiedliche Cloud-Anbieter zu verteilen (Müller et al. 2014).

## 2.2 Vendor lock-in

Das Problem des Vendor lock-ins, bei dem sich im Vorhinein auf einen Anbieter festgelegt werden muss, wird hier in zweierlei Hinsicht fokussiert. Zum einen mit Bezug zu möglichen Cloud-Anbietern und zum anderen hinsichtlich der Datenbankhersteller. Beide Aspekte werden nun erläutert.

5 https://www.salesforce.com, eine Customer Relationship Management Anwendung

6 https://appengine.google.com, eine kollaborative Entwicklungsumgebung für Programmentwickler

7 https://aws.amazon.com/de/ec2/, das IaaS-Angebot von Amazon

### 2.2.1 Cloud-Anbieter

Bei der Betrachtung von Cloud Computing aus einer übergeordneten Cloud-Management Perspektive fällt auf, das schon oft versucht wurde, providerübergreifende Standards zu schaffen, vgl. u. a. (MacVittie 2011; Kerschbaum 2011; Hyvönen 2010). Die Definition von einheitlichen Standards wird noch zusätzlich verschärft, da eine allgemeingültige Definition derzeit immer noch diskutiert wird (Vaquero et al. 2009). Das National Institute of Standards and Technology (NIST) ist dabei mit seiner aktuellen Definition zwar führend (Nist 2013), jedoch ist Cloud Computing ein immer noch sehr weit gefasster Begriff in der IT-Welt. Das wichtigste Standardisierungsgremium W3C [8] ist durch eine Vielzahl unterschiedlicher Standards (Berners Lee 2010) gekennzeichnet. Im Maschinenbau oder in der Elektrotechnik existieren gängige ISO-Normen. Dies ist ein gutes Beispiel, für gemeinsame und etablierte Standards. Leider ist dies im Cloud Computing nicht der Fall. Ferner liegt darin die große Anzahl unterschiedlicher Schnittstellen begründet. In der Literatur wird eine entsprechende Nachfrage nach einheitlichen Cloud-Standards identifiziert, da es nicht möglich ist verschiedene Cloud-Provider gleichzeitig zu nutzen, bzw. es mit erheblichem Aufwand verbunden ist, einen Provider im Nachhinein zu wechseln (Buyya et al. 2008) und (Vecchiola et al. 2010). Hierfür sind unterschiedliche Schnittstellen, Abrechnungsmodelle und verschieden strukturierte SLAs die Hauptgründe (Kohler 2012).

8 http://www.w3.org/

### 2.2.2 Datenbankhersteller

Ein ähnliches Problem wie bei den Cloud-Anbietern besteht bei den Datenbankanbietern. Bei der Betrachtung des SQL-Standards treten viele unterschiedliche datenbankherstellerspezifische Implementierungen hervor (Bargmeyer 2011). So erweitern Datenbankhersteller den Standard um eigene Funktionen, wie z. B. die „auto increment“-Funktion in MySQL, die z. B. zum automatischen Erhöhen eines Primärschlüsselwertes verwendet werden kann. Diese und andere Erweiterungen machen eine herstellerübergreifende Datenübernahme unmöglich, da beispielsweise die o. g. „auto increment“-Funktion bei anderen Datenbankherstellern nicht existiert. Ist also die Entscheidung für einen Datenbankanbieter gefallen, lassen sich die darin abgelegten Daten nur mit viel Aufwand zu einem anderen Anbieter übernehmen.

Hier setzt diese Arbeit an und implementiert im Prototyp mit Oracle Express und MySQL zwei unterschiedliche Datenbanken, die sich für den Anwender transparent verwenden lassen. Ferner werden in einer Datenbankabstraktionsschicht Schnittstellen vorbereitet, um weitere Datenbanken in das Framework zu integrieren.

## 2.3 XaaS-Bewertung

Das Beziehen von IT-Ressourcen nach aktuellem Bedarf und deren nutzungsbezogene Abrechnung machen Cloud Computing insbesondere für den Unternehmensbereich interessant. Allerdings ist der Angebotsvergleich verschiedener Cloud-Anbieter aufgrund unterschiedlicher Preismodelle extrem aufwändig. So werden Leistungen im IaaS/PaaS-Bereich nach Ressourcenverbrauch bzw. nach eingesetztem Betriebssystem (Lizenzkosten) providerabhängig abgerechnet. Im SaaS-Bereich hingegen werden gleiche oder ähnliche Dienste zu unterschiedlichen Preisen angeboten. Weiterhin ist es für die Nutzer nicht möglich, individuelle Anforderungen an die Anbieter zu stellen bzw. kundenspezifische Service Level Agreements (SLAs) auszuhandeln. Diese Problemstellung lässt sich somit auf den gesamten XaaS-Bereich übertragen und ist neben den unterschiedlichen Anbieterschnittstellen auch Ursache für den o. g. „Vendor lock-in“ (Kohler/Specht 2013).

Eben diese Arbeit (Kohler/Specht 2013) schlägt hierzu einen zentralen Marktplatz vor, auf dem sich Anwender und Anbieter treffen, um die aufwändigen Vertragsverhandlungen möglichst automatisiert durchzuführen.

Da nun die drei wesentlichen Problemstellungen des Projekts herausgearbeitet sind, werden im folgenden Abschnitt 3 die zugehörigen Lösungen dargestellt.

# 3 Lösungskonzepte

Analog zu Kapitel 2 werden hier nun die entsprechenden Lösungskonzepte skizziert.

## 3.1 Datenschutz und -sicherheit

Diese Publikation zeigt ein Konzept zur verteilten Datenhaltung, bei dem Daten logisch auf verschiedene Clouds aufgeteilt werden, sodass bei jedem Provider nur ein Teil liegt, der ohne die anderen Teile wertlos ist. Die Aufteilung bewegt sich auf Datenbankebene. Dabei wird der Ansatz der vertikalen Partitionierung verwendet, um die Daten logisch aufzuteilen. ABB. 3 illustriert den gesamten Ansatz.

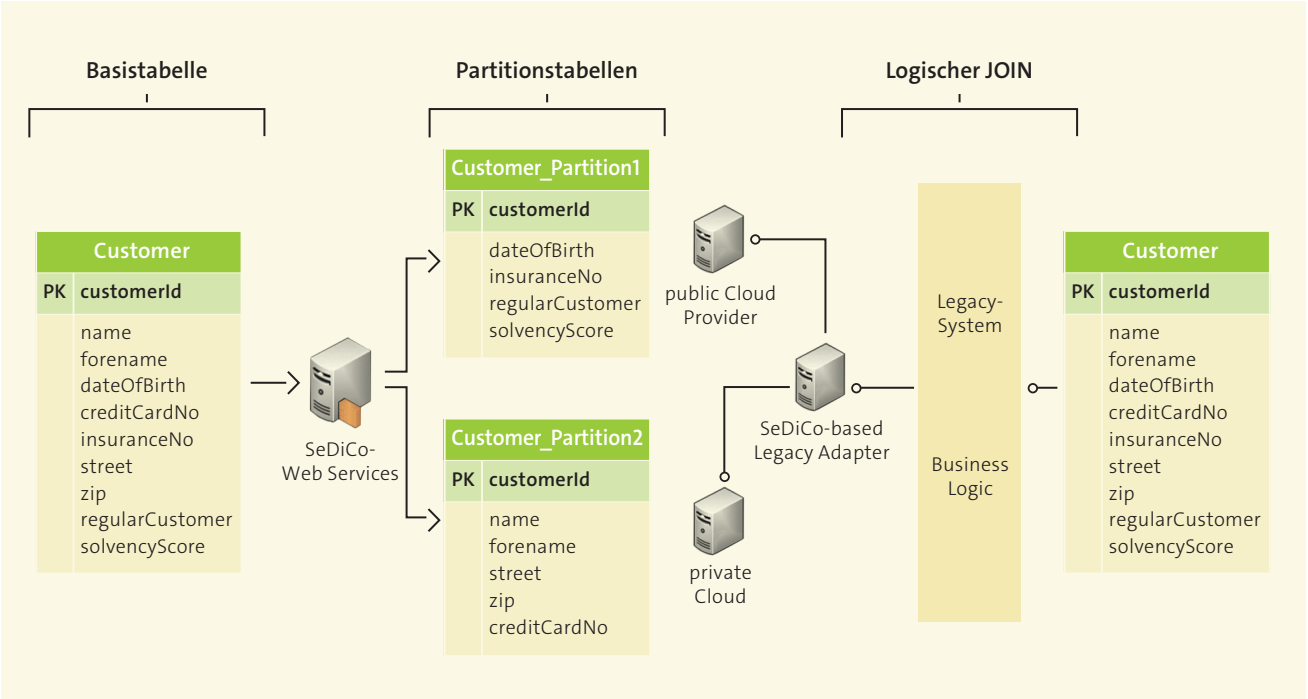


ABB. 3 Grundlegender Ansatz des SeDiCo-Projekts

Es ist wichtig, die Verteilung der Daten nach Datenschutzaspekten vorzunehmen, damit beispielsweise Kreditkartennummer und Name eines Kunden (man denke an unser Anwendungsbeispiel „Reiseunternehmen“) in jedem Falle in zwei unterschiedlichen Partitionen liegen. Ideen zur sicheren Aufteilung der Daten werden ebenfalls von anderen Projekten verfolgt (s. a. Kapitel 5 und vgl. Ganapathy et al. 2011; Plattner 2013).

Bisherige Ansätze implementieren vor allem die horizontale Verteilung wegen der erhöhten Skalierbarkeit und der besseren Performance bei der dynamischen Erweiterung der zugrundeliegenden Cloud-Ressourcen (scale-up und scale-out). Plattner zeigt hier verschiedene Möglichkeiten der horizontalen Partitionierung (Plattner 2013). U. a. werden dabei komplette Datensätze anhand ihrer Eigenschaften auf verschiedene Ressourcen verteilt (range partitioning). So wäre es denkbar, dass eine Tabelle „Kunden" nach dem Alter der Kunden partitioniert wird. Kunden zwischen 18 und 30 Jahren würden in eine Partition ausgelagert. Eine zweite Partition würde Kunden zwischen 31 und 50 Jahren beinhalten und die letzte Partition würde alle Kunden, die älter als 51 Jahre sind, beinhalten. Ein anderer Ansatz, losgelöst von den Eigenschaften der Daten, wäre die Daten nach einem bestimmten Algorithmus, z. B. round robin oder gar zufällig in verschiedene Partitionen zu verteilen. Für weitere horizontale Verteilungsstrategien sei an dieser Stelle auf die Literatur verwiesen (Plattner 2013). Unabhängig von der Partitionierungsstrategie haben die horizontale und die vertikale Partitionierung die Aufteilung der Daten in disjunkte Teilmengen als gemeinsames Ziel. Weiterhin müssen beim Lesen der Daten geeignete Join-Operationen implementiert sein, um die verteilten Daten wieder zusammenzuführen. Im Gegensatz zur horizontalen Partitionierung trennt die vertikale Partitionierung dazu einzelne Datensätze (Tupel) logisch auf und dupliziert deren Primärschlüssel in alle Partitionen. Anhand dieser Schlüssel lassen sich die Daten anschließend auch wieder zusammenfügen.

Mit Blick auf die Themen „Cloud Computing" und „Big Data" spielen zwei Faktoren die wichtigsten Entscheidungskriterien für oder gegen einen Ansatz, zum einen die Performance und zum anderen die Sicherheit. Gerade in der logischen Trennung der Tupel liegt die Motivation des hier dargestellten Verteilungsprinzips, mit der Konzeption und Implementierung eines sicheren Datenspeichers in der Cloud, begründet (Müller et al. 2014).

Nachdem die Betrachtung der vertikalen Partitionierung und deren Unterscheidung zur horizontalen Aufteilung abgeschlossen ist, soll nun ein detaillierter Blick auf den vertikalen Join geworfen werden. Die Aufteilung der Daten (ABB. 3) macht deutlich, dass die aufgeteilten Daten irgendwann wieder zusammengefügt werden müssen. Nur so ist eine transparente Verwendung des gesamten Frameworks möglich, ohne dass vorhandene Programme und Systeme, sog. „Legacy Systeme" angepasst werden müssen. Dieses Zusammenführen (Join) ist im „SeDiCo-based Legacy-Adapter" implementiert (ABB. 3). Dies wirft aus softwarearchitektonischer Sicht allerdings die Frage auf, auf welcher Ebene die Daten nun zusammengeführt werden sollen. ABB. 4 zeigt die fünf möglichen Architekturschichten, die für eine Zusammenführung der Daten in Frage kommen.

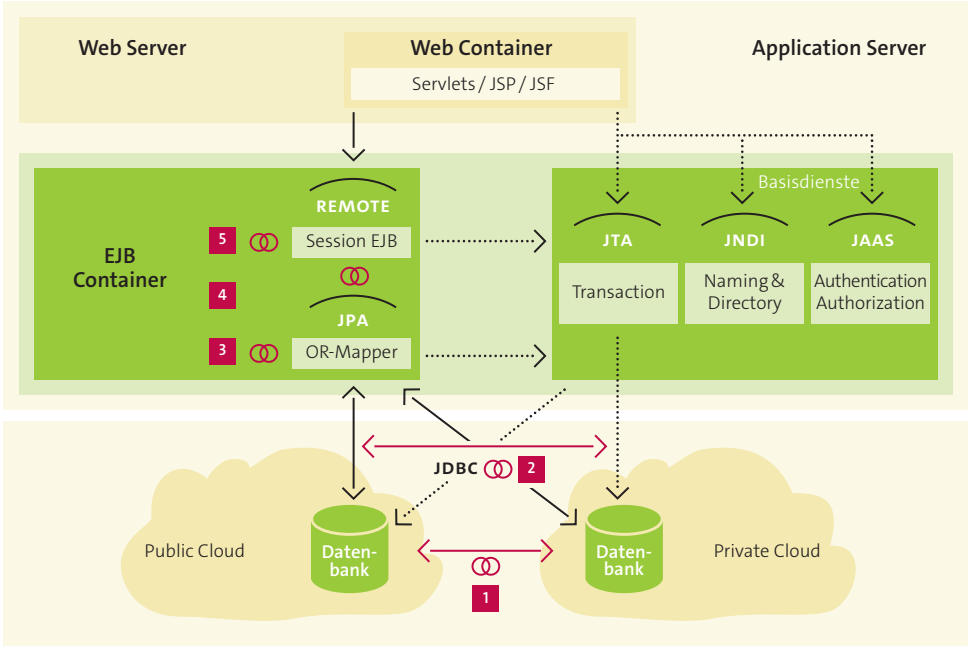


ABB. 4 Mögliche Architekturschichten für den Join der Daten (Müller et al. 2014)

Eine detaillierte Analyse der einzelnen Schichten mit ihren Vor- und Nachteilen würde den Rahmen dieser Arbeit sprengen. Deshalb sei an dieser Stelle auf Müller et al. 2014 verwiesen.

## 3.2 Vendor lock-in

Analog zur Unterscheidung in Abschnitt 2.2 ist auch der folgende Abschnitt in die zwei Teile „Cloud-Anbieter“ und „Datenbankanbieter“ aufgeteilt. Zunächst wird daher auf das Lösungskonzept zum Vendor lock-in Problem der Cloud-Anbieter eingegangen, bevor dann der Datenbank-Vendor lock-in behandelt wird.

### 3.2.1 Cloud-Anbieter

Um der o. g. Problematik des Vendor lock-ins zu begegnen, wurden verschiedene Frameworks evaluiert. Die Evaluation bezog sich dabei auf drei aktuelle state-of-the-art Frameworks: jclouds[9], deltacloud[10] und libcloud[11]. Als Ergebnis wurde herausgearbeitet, dass sich jclouds am besten in den Prototyp integrieren lässt. Hauptgründe waren dabei die Java-Unterstützung und die SSL-Unterstützung. Für eine detaillierte Analyse der genannten Frameworks sei an dieser Stelle auf die Literatur (Kaiser 2013) verwiesen.

ABB. 5 verdeutlicht die Integration des jclouds-Frameworks in die Architektur des Prototyps.

9 http://jclouds.apache.org/

10 https://deltacloud.apache.org/

11 https://libcloud.apache.org/

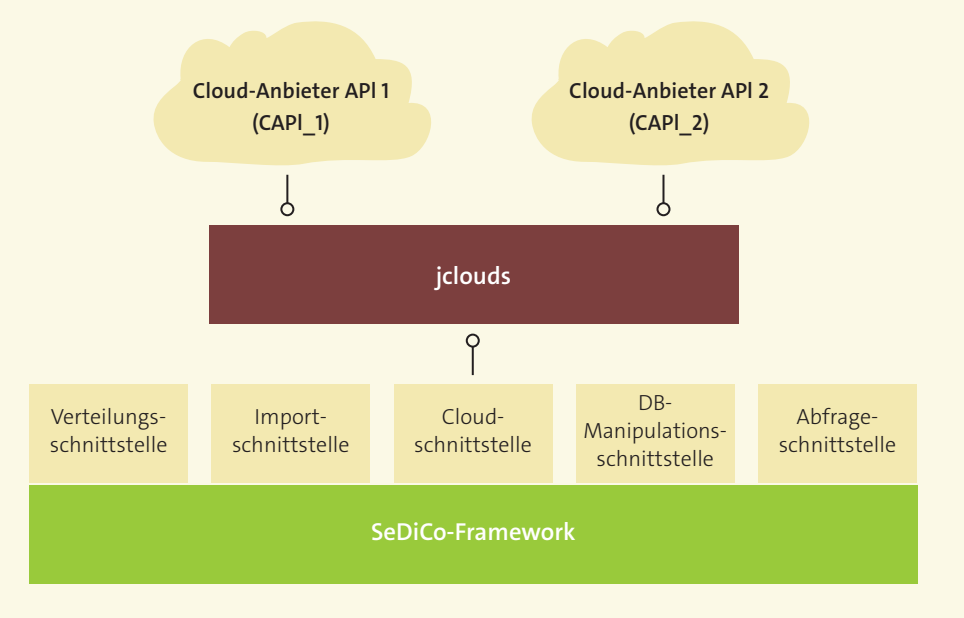


ABB. 5 SeDiCo Cloud-Abstraktionsschicht

LISTING 1 bis LISTING 3 stellen die Verwendung des jclouds-Frameworks exemplarisch für den gesamten Prototyp dar. Als Szenario sei hier der Anwendungsfall beschrieben, dass der Anwender seine Datenbanktabelle vertikal geteilt und sich für zwei unterschiedliche Cloud-Anbieter entschieden hat. Analog zu ABB. 3 entscheidet er eine Partition in der öffentlichen Amazon EC2 Cloud zu speichern und die andere in seiner privaten Eucalyptus Cloud abzulegen. Dazu fragt er nun die öffentliche Amazon EC2-Cloud nach verfügbaren virtuellen Maschinen (sog. Instanzen) ab. Die Weboberfläche des SeDiCo-Prototyps bietet dazu entsprechende Filterkriterien, wie ABB. 6 verdeutlicht.

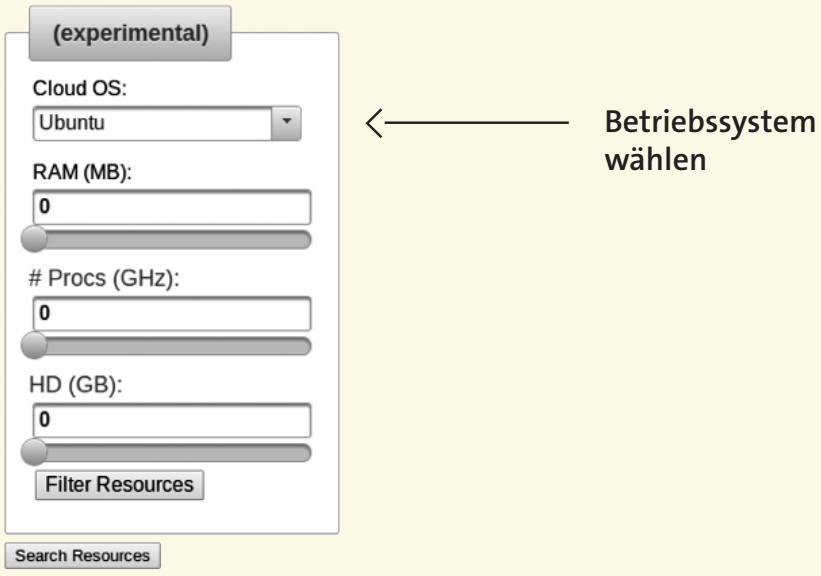


ABB. 6 Filterkriterien für virtuelle Cloud-Instanzen

Nach dem Festlegen der gewünschten Auswahlkriterien abstrahiert die Cloud-Schnittstelle die Aufrufe und Filterkriterien vom jeweiligen Cloud-Provider. LISTING 1 verdeutlicht dies auf Ebene des Programmcodes.

```
1.  Set<? extends Image> images = this.client.listImages();
2.  int i=1;
3.  for(Image image:images) {
4.          String imgOs = image.getOperatingSystem().getFamily().toString();
5.          String imgReg = image.getLocation().getId().toString();

7.          //... Code zur Ausgabe an den Anwender
9.          i++;
10. }
```

LISTING 1 SeDiCo Cloud-Abstraktionsschicht

So zeigt Zeile 1 die Anfrage an den Cloud-Provider, alle verfügbaren Instanzen aufzulisten. Das Resultat dieser Anfrage wird dann in den Zeilen 3 – 9 verarbeitet und an den Anwender ausgegeben. Aus Zeile 1 geht die Abstraktion besonders gut hervor, da sie zeigt, dass mit einer einzigen Zeile Programmcode alle von jclouds unterstützten Cloud-Anbieter abgefragt werden. Der Einfachheit halber wurden bei obigem Aufruf (Zeile 1) erst alle verfügbaren Instanzen geladen und danach (Zeile 3 – 9) die Filterkriterien angewendet. Aus Performancegründen und bei der Betrachtung der Netzlast ist diese Vorgehensweise durchaus vertretbar, wie die Evaluation des Prototyps ergeben hat[12]. Hat der Anwender nun eine passende virtuelle Maschine (Cloud-Instanz) gefunden und diese ausgewählt, wird die jeweilige Partition auf der gewählten Instanz abgelegt. Der Zugriff auf die ausgewählte Instanz erfolgt wiederum über das jclouds-Framework, s. LISTING 2.

12 Effizienter wäre es natürlich, die Filterkriterien sofort beim ersten Aufruf anzuwenden. Aus Gründen der Wiederverwendbarkeit, der Erweiterbarkeit und der Wartbarkeit wurde diese Vorgehensweise aber verworfen.

```
1.  public void runCommandsonInstance(String instanceId, List<String> commands){
2.      Status state = this.getInstancestate(instanceId);
3.      if(state.equals(Status.RUNNING)){
4.              Session session = this.JSchStack();

5.              Channel channel = session.openChannel("shell");
6.              ((ChannelShell) channel).setAgentForwarding(true);
7.              ((ChannelShell) channel).setPtyType("vt100"); //oder vt102

8.              InputStream in = channel.getInputStream();
9.              PrintStream shellStream = new PrintStream(channel.getOutputStream());

10.             channel.connect();
11.             for (String command : commands) {

12.                     byte[] tmp = new byte[1024];
13.                     if (in.available() > 0) {
14.                     int inLength = in.read(tmp, 0, 1024);
                }

15.             while (true) {
16.                     if (in.available() == 0) {
17.                             shellStream.println(command);
```

LISTING 2 Verbindung zu einer virtuellen Cloud-Instanz aufbauen

```
18.                                     shellStream.flush();
19.                                     break;
                                }
                        }
20.                     if (channel.isClosed()) {
21.                             break;
                        }
22.                     channel.disconnect();
23.                     session.disconnect();
                        }
        }
```

Dieses Listing zeigt nun exemplarisch, nachdem eine Verbindung zu einer bestimmten Cloud-Instanz (über die „instanceId" – Zeile 2) hergestellt wurde, wie sich diese verwenden lässt. So wird in den Zeilen 2–3 vorab geprüft, ob die Instanz überhaupt gestartet ist und in der Cloud läuft. Im positiven Fall wird dann eine Verbindung zur Instanz aufgebaut. Über diese Verbindung lassen sich nun Programme installieren (z. B. ein DBMS) oder eben auf der Instanz installierte Programme benutzen (z. B. vorhandene Daten in eine Datenbank schreiben).

Nachdem die Verbindung nun erfolgreich hergestellt ist, zeigt LISTING 3 nun, wie auf die Instanz zugegriffen wird.

LISTING 3 Virtuelle Cloud-Instanzen verwenden

```
1. LoginCredentials credentials = node.getCredentials();

2. miscSSHpem SSHpem = new miscSSHpem();
3. SSHpem.writePEMfile(credentials.credential, credentials.getUser(), node.getGroup());

4. try {
5.      RunScriptOptions rso = new RunScriptOptions();
6.      rso.shouldRunAsRoot();

7.      Map<? extends NodeMetadata, ExecResponse> responses =
                this.client.runScriptOnNodesMatching(
                        runningInGroup(groupName),
8.                      "echo \"sudo apt-get -y install mysql mysql-server", rso);

9.      Collection<ExecResponse> execs = responses.values();

        for (ExecResponse entry : execs){
10.             //... falls Fehler auftreten, diese an den Benutzer ausgeben
11.     }
12.  } catch (RunNodesException e) {
13.     e.printStackTrace();
    }
```

Zunächst wird der Benutzer authentifiziert (Zeile 1–3). Nach erfolgreicher Authentifizierung lässt sich die Instanz verwenden. So zeigen die Zeilen 5–10 die Installation eines MySQL-Servers auf der Cloud-Instanz.

Um die Beschreibung der Cloud-Abstraktionsschicht nun abzuschließen, sei angemerkt, dass sich dieser Programmcode unverändert auch mit neu am Markt

auftretenden Cloud-Anbietern verwenden lässt, sofern diese durch das jclouds-Framework unterstützt werden.

Um obiges Szenario noch einmal aufzugreifen, werden also beide Partitionen über dieselbe Schnittstelle (exemplarisch dargestellt in LISTINGS 1 – 3) einmal in die private und einmal in die öffentliche Cloud publiziert (Kaiser 2013).

### 3.2.2 Datenbanksysteme

Analog zur Integration der unterschiedlichen Cloud-Schnittstellen wurde das ähnliche Problem des Datenbank-Vendor lock-ins angegangen, wie ABB. 7 verdeutlicht.

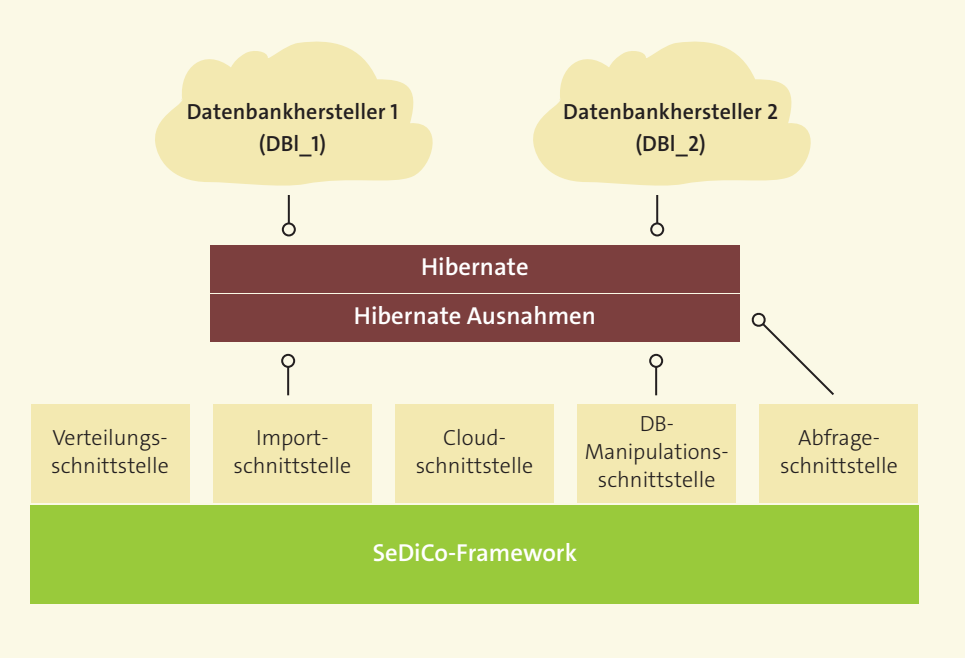


ABB. 7 SeDiCo Datenbank-Abstraktionsschicht

Basis für die Integration der verschiedenen Datenbank-APIs ist das Hibernate-Framework[13]. Dieses Framework deckt viele Unterschiede[14] der beiden verwendeten Datenbanken ab. Etwaige Sonderfälle[15] kamen bei der Evaluation des Projektes zum Vorschein und wurden manuell in den Prototyp integriert (s. a. „Hibernate Ausnahmen" in ABB. 7).

Letztlich wurde mit Hilfe der beiden Abstraktionsschichten und der in diesem Abschnitt beschriebenen Architektur das Problem des Vendor lock-ins angegangen. Für den Prototyp wurden exemplarisch zwei unterschiedliche Cloud-Provider und Datenbankschnittstellen implementiert und integriert. Dabei handelt es sich um Eucalyptus[16] als privater Cloud-Anbieter und Amazon EC2[17] als öffentlicher Cloud-Anbieter sowie um MySQL und Oracle Express als Datenbanksysteme. Die Integration weiterer Cloud-Provider und Datenbanken ist somit nun durch die o. g. definierten Schnittstellen ohne weiteres möglich.

13 http://hibernate.org/

14 So müssen beispielsweise die verschiedenen Datentypen (Oracle „varchar(255)" und MySQL „varchar") aufeinander gemappt werden. Beim Versuch einen MySQL „varchar" Datentyp in Oracle zu definieren, wird dieser Datentyp, der in MySQL standardmäßig 256 Zeichen lang ist, in Oracle mit der Länge 0 definiert.

15 Ein anderer Sonderfall, der standardmäßig nicht durch Hibernate abgedeckt wird, ist beispielsweise der Oracle-Datentyp „decimal(0,0)". Dieser Datentyp existiert in MySQL so nicht. Er wurde daher auf einen „integer" gemappt.

16 https://www.eucalyptus.com/

17 http://aws.amazon.com/

## 3.3 XaaS-Bewertung

Neben der Verteilung bleibt allerdings die aufwändige Auswahl geeigneter IaaS-Anbieter problematisch, da kein einheitlicher Standard existiert, um Dienste und Plattformen zu beschreiben. In Kohler/Specht 2013 wurde daher ein Marktplatz für den transparenten Vergleich verschiedener IaaS-Provider konzeptionell erarbeitet. Die grundlegende Idee ist das Zusammenbringen von Anbietern von SLAs und Nachfragern auf einer zentralen Plattform, wie es ABB. 8 verdeutlicht.

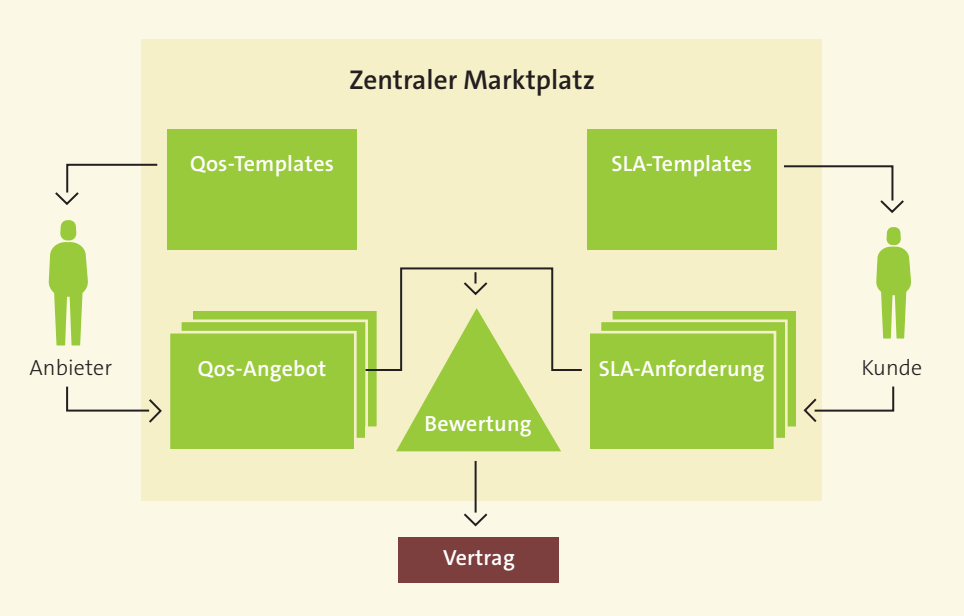


ABB. 8 Marktplatz zur XaaS-Bewertung und SLA-Aushandlung (Kohler/Specht 2013)

Ein solcher Marktplatz erfordert allerdings einheitliche Schablonen (sog. „Templates“), um SLAs zu definieren und später auszuhandeln. Mit eben dieser Kriteriendefinition beschäftigt sich die Arbeit Kohler/Specht 2013. Im Rahmen dieser Arbeit wurden wissenschaftliche Arbeiten mit konkretem Bezug zu SLAs, d. h. mit konkreten Kriterien und zugehörigen Messwerten über den Zeitraum von 2002, vom Aufkommen der service-orientierten Architekturen, bis in das Jahr 2013 zum aktuellen Cloud Computing Trend analysiert. Ergebnis dieser Analyse ist die Herausarbeitung von immer wiederkehrenden Kriterien und deren Messwerten, um eine grundlegende Basis für SLA-Schablonen zu schaffen.

Analog zu Kohler/Specht 2013 sind die herausgearbeiteten Kriterien hier noch einmal mit der Kategorisierung in Verfügbarkeit, Skalierbarkeit, Kosten, Sicherheit und Usability tabellarisch aufgelistet.

### Verfügbarkeit

| Kriterium | Messpunkt |
|---|---|
| Verfügbarkeit | Prozentwert des Verhältnisses aus Betriebs- und Ausfallzeit |
| MTBF | Zeitraum (Minuten) des fehlerfreien Betriebs der Server-Infrastruktur zwischen zwei Ausfällen |
| MTTR | Geschwindigkeit (Minuten) vom Ausfall der Server-Infrastruktur bis zu deren Wiederherstellung durch den Anbieter |
| Recovery | Zeit (Minuten) bis zur Wiederherstellung der Anwendungsdaten nach einem Ausfall |

TAB. 1 SLA Messkriterien zur Verfügbarkeit

### Skalierbarkeit

| Kriterium | Messpunkt |
|---|---|
| CPU | Anzahl |
| RAM | Größe in MB |
| Provisionierungszeit | Geschwindigkeit in Sekunden von der Bestellung einer VM bis zu deren Verwendung |
| Festplattenspeicher | Größe in MB |
| Durchsatz | Maximale Zeit in Millisekunden von der Anfrage an einen Dienst, bis dieser das Ergebnis liefert (bzw. im Fehlerfall eine Fehlermeldung) |
| Virtuelle Maschinen | Maximal mögliche Anzahl |
| Benutzeranzahl | Anzahl der Anwender, die den Dienst gleichzeitig nutzen können |
| Eingehender Datenverkehr | Datenvolumen in GB vom Anbieter zum Anwender |
| Ausgehender Datenverkehr | Datenvolumen in GB vom Anwender zum Anbieter |
| Antwortzeit | Ping-Zeit in Millisekunden |
| Bandbreite | Übertragungsrate der Netzwerkanbindung in Megabit |
| Durchsatz | Maximale Zeit in Millisekunden von der Anfrage an einen Dienst, bis dieser das Ergebnis liefert (bzw. im Fehlerfall eine Fehlermeldung) |

TAB. 2 SLA Messkriterien zur Skalierbarkeit

## Kosten

| Kriterium | Messpunkt |
|---|---|
| Lizenzkosten | Kosten in Euro pro Monat pro Lizenz |
| Laufzeit | Gültigkeit der SLA in Tagen/Wochen/Monaten |
| Preis | Preis in Euro des Komplettangebots für die Nutzung (Tage/Wochen/Monate) |

TAB. 3 SLA Messkriterien zu den Kosten

## Sicherheit

| Kriterium | Messpunkt |
|---|---|
| Datensicherheit | Zugangskontrolle zu Anbieter-Host-Systemen, Firewall und Virenschutz des Anbieter-Host-Systems, redundante Rechenzentren |
| Authentisierung | Authentisierungsmethode, z. B. 2-Faktor-Authentisierung |
| Compliance | Integration der Cloud in bestehende Infrastruktur und Einhaltung bestehender Compliance-Richtlinien |
| Data Retention and Deletion | Zuverlässige und nachprüfbare Vernichtung der Daten nach Kündigung des Cloud-Providers |
| Brandschutz | Brandmeldeanlage, Brandfrüherkennung, geeignete Löschtechnik, regelmäßige Brandschutzübungen |
| Redundante Rechenzentren | Entsprechende räumliche Trennung der Rechenzentren, dass ein Schadensereignis nicht gleichzeitig das ursprünglich genutzte Rechenzentrum und das, in dem die Ausweichkapazitäten genutzt werden, beeinträchtigen |
| Malware-Schutz | Virenschutz, Trojaner-Detektion, Spam-Schutz, etc. |
| Sicherheitsmaßnahmen gegen netzbasierte Angriffe | IPS/IDS-Systeme, Firewall, Application-Layer, Gateway, DDoS-Mitigation, etc.) |
| Verschlüsselte Kommunikation | Art der Verschlüsselung der Netzkommunikation zwischen Anbieter und Nutzer, z. B. TLS/SSL |
| Verschlüsselte Kommunikation Cloud Provider | Art der Verschlüsselung der Netzkommunikation zwischen Cloud Standorten |
| Verschlüsselte Kommunikation Drittanbieter | Art der Verschlüsselung der Netzkommunikation mit Drittanbietern, falls auf deren Angebote zurückgegriffen wird |
| Datensicherung | Regelmäßige Datensicherungen, (Umfang, Speicherintervalle, Speicherzeitpunkte und Speicherdauer) |
| Cloud Standorte | Offenlegung der Standorte des Cloud-Service-Anbieters (Land, Region) |
| Cloud Sub-Standorte | Offenlegung der Subunternehmer des Cloud-Service-Anbieters, die für die Erbringung der Cloud Services wesentlich sind |
| Host-Transparenz | Transparenz, welche Software durch den Cloud-Service-Anbieter installiert, wird sowie über die daraus resultierenden Sicherheitserfordernisse/-risiken |
| Grund-Konfiguration des Hosts | Einsatz gehärteter Betriebssysteme, Deaktivierung unnötiger Dienste |
| Anbieter-Zertifizierung | ITIL, CoBit, etc. |

TAB. 4 SLA Messkriterien zur Sicherheit

### Usability

TAB. 5 SLA Messkriterien zur Usability

| Kriterium | Messpunkt |
|---|---|
| Monitoring | Festlegung der Zuständigkeit und Definition der Methodik |
| Support | Verfügbarkeit des Supports, Benennung Ansprechpartner |
| Integration | Integration der Cloud-Anwendung mit bestehender Infrastruktur (z. B. Browser, SAP, E-Mail, etc.) |
| Cloud-Anbindung | Unterstützung von Synchronisationsmechanismen, um die Arbeit ohne Cloud-Anbindung zu ermöglichen |
| Portabilität | Definition von Schnittstellen um Daten bzw. Anwendungen in andere Clouds zu migrieren |
| Verwendung von Standards | Einhaltung und Verwendung entsprechender Web-Standards, z. B. OWASP |
| Anpassbarkeit | Realisierung von individuellen Kundenwünschen, Anpassbarkeit der Anwendung aus der Cloud |

Die Einteilung in die fünf o. g. Kategorien „Verfügbarkeit“, „Skalierbarkeit“, „Kosten“, „Sicherheit“ und „Usability“, verdeutlicht zudem die Problematik der Messbarkeit solcher Kriterien. So sind die ersten drei Kriterien als funktionale Kriterien sehr gut quantifizier- und damit messbar, wohingegen die letzten beiden Kriterien „Sicherheit“ und „Usability“ nur sehr schwer und eingeschränkt messbar sind. Da die fehlende Quantifizierbarkeit nicht zentraler Bestandteil dieser Arbeit ist, sondern der Fokus auf der Demonstration der technischen Machbarkeit des Ansatzes der verteilten Datenbanken liegt, können diese Fragestellungen hier nicht weiter verfolgt werden.

# 4 Prototyp

Dieser Abschnitt stellt nun den implementierten Prototyp anhand von Architekturschaubildern detailliert vor. Der Prototyp ist als Open Source Software veröffentlicht. D. h., die Software ist sog. "freie Software" und lizenziert unter den Bedingungen der GNU Affero General Public License (AGPL 2007). Der Prototyp kann unter folgender Adresse heruntergeladen werden:

https://github.com/jenskohler/SeDiCo

## 4.1 Architektur

Die grundlegende Idee des gesamten Frameworks ist in ABB. 9 dargestellt.

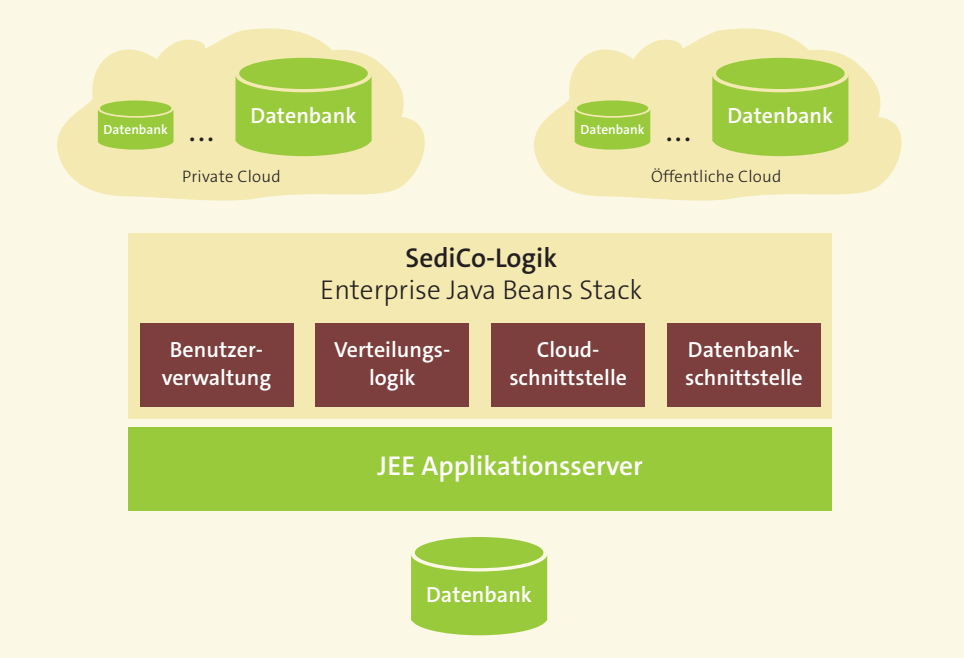


ABB. 9 Generische SeDiCo-Architektur

Zentraler Ausgangspunkt ist also eine Datenbank (unten in ABB. 9). Aus dieser Datenbank soll eine Tabelle ausgewählt und anschließend in zwei Teile partitioniert werden. Dabei wird der Anwender durch eine Webanwendung unterstützt, die auf einem JEE Applikationsserver läuft. Ferner besteht der Prototyp aus vier zentralen Elementen:

- einer Benutzerverwaltung, über die sich Anwender registrieren und einloggen. Diese Verwaltung ist Grundlage für ein dezidiertes Rechtemanagement, um die Anwendung in Unternehmensinfrastrukturen zu integrieren. Im Prototyp ist allerdings bisher nur eine rudimentäre Login-Funktion implementiert. Entsprechende Schnittstellen für z. B. eine LDAP-Integration sind vorbereitet, aber nicht ausprogrammiert.

- der konkreten Verteilungslogik, die die Partitionierung von Tabellen realisiert. Im Prototyp ist eine Tabelle auszuwählen, die über einen einfachen Primärschlüssel alle Tupel eindeutig identifiziert. Aktuell werden Tabellen aus zusammengesetzten Primärschlüsseln nicht unterstützt. Dies wird allerdings bei der weiteren Entwicklung des Projekts berücksichtigt. Ebenso ist dabei eine Aufteilung von Tabellen in beliebige Partitionen angedacht. Um die technische Machbarkeit des Ansatzes der vertikalen Partitionierung aufzuzeigen und die Performance zu evaluieren, wurde die Verteilungslogik im SeDiCo-Projekt allerdings nur mit zwei Partitionen durchgeführt.

- der Kapselung verschiedener Cloud-APIs in einer einheitlichen Cloud-Schnittstelle. Die Integration unterschiedlicher Cloud-Schnittstellen wurde bereits in Abschnitt 3.2 beschrieben. Daher sein an dieser Stelle darauf verwiesen.

- die Integration verschiedener Datenbankschnittstellen. Ähnlich wie bei der Kapselung verschiedener Cloud-Schnittstellen war die Integration unterschiedlicher Datenbankdialekte ein zentrales Thema im Projekt. Basis für die Integration der verschiedenen Datenbank-APIs war das Hibernate-Framework. Dieses Framework deckt viele Unterschiede der beiden Datenbanken ab. Allerdings hat der Test des Prototyps auf auf einige Sonderfälle hingewiesen, die nicht durch das Hibernate-Framework abgedeckt sind (vgl. Abschnitt 3.2).

ABB. 10 greift nun die oben skizzierte Architektur auf (ABB. 9) und instanziiert die einzelnen Komponenten beispielhaft mit konkreten Systemen.

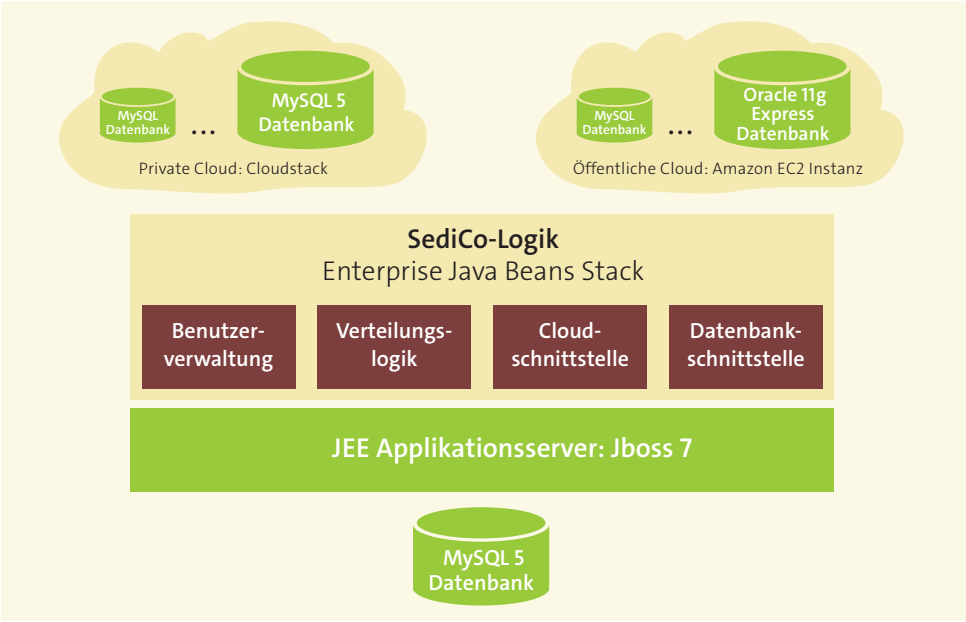


ABB. 10 SeDiCo-Architektur

Ausgangsbasis für den Prototyp ist nun eine MySQL-Datenbank, die in 2 Teile partitioniert werden soll. Alternativ ist es mit dem implementierten Prototyp möglich, als Ausgangsbasis eine Oracle-Datenbank zu teilen. Der Nutzer kann dann entscheiden, welche Zieldatenbank er für die Teile nutzen möchte. Als Zieldatenbanken sind wieder MySQL- und Oracle-Datenbanken möglich. Ferner ist eine Mischung beider Datenbanken und somit deren gleichzeitige Verwendung möglich. Überdies ist durch die Verwendung des JEE-kompatiblen JBoss Applikationsservers, die Integration in heterogene Unternehmensinfrastrukturen erheblich erleichtert, da dieser durch sämtliche JEE-kompatiblen Applikationsserver transparent ausgetauscht werden kann.

Letztlich ist damit die gesamte Funktionalität des Prototyps beschrieben. Der nun folgende Abschnitt 4.2 konzentriert sich auf die Analyse der Performance und stellt einen Vergleich zwischen partitionierten und nicht-partitionierten Daten beim Abrufen, Einfügen und Ändern her.

## 4.2 Performanceevaluation

ABB. 11 bis ABB. 38 stellen die Performancemessungen des Prototyps dar. Zunächst wird die Performance beim Abrufen von vorhandenen Datensätzen (Tupel) betrachtet. Anschließend folgen Performanceanalysen zum Ändern und zum Einfügen von neuen Datensätzen. Dieser Abschnitt stellt die wesentlichen Kennzahlen der beiden Publikationen von Kohler und Specht dar. Für detailliertere Informationen sei an dieser Stelle darauf verwiesen (Kohler/Specht 2014; Kohler/Specht 2014a; Kohler/Specht 2014b).

Zunächst wird die verwendete Infrastruktur skizziert.

### 4.2.1 Infrastruktur

Sämtliche Messungen wurden auf folgender Hardwareinfrastruktur durchgeführt:

- jede verwendete Maschine mit je 2 Prozessoren (Intel Core i7 mit je 2,9 GHz) mit eingeschaltetem Hyper-Threading und 8 GB RAM
- der SeDiCo-Client wurde unter der Oracle Java Platform (JDK) 7u45 kompiliert und ausgeführt
- die physischen Maschinen sind über ein 100-Mbit-Netzwerk verbunden

Die folgenden Abschnitte präsentieren nun die Messwerte für die Abfrageperformance, bevor dann näher auf die Änderungs- und zuletzt auf die Einfügeperformance eingegangen wird. Letztlich schließt das Kapitel mit einem Fazit.

### 4.2.2 Abfrageperformance

Um eine grundlegende Basis zu schaffen, wurde bei der Messung der Performance zunächst die Performance der Datenbanken ohne Partitionierung gemessen. Außerdem wurden die Datenbanken und der SeDiCo-Client auf derselben Maschine installiert. Somit sind in den nun folgenden Messwerten ABB. 11 bis ABB. 15 keine Netzwerklatenzen enthalten.

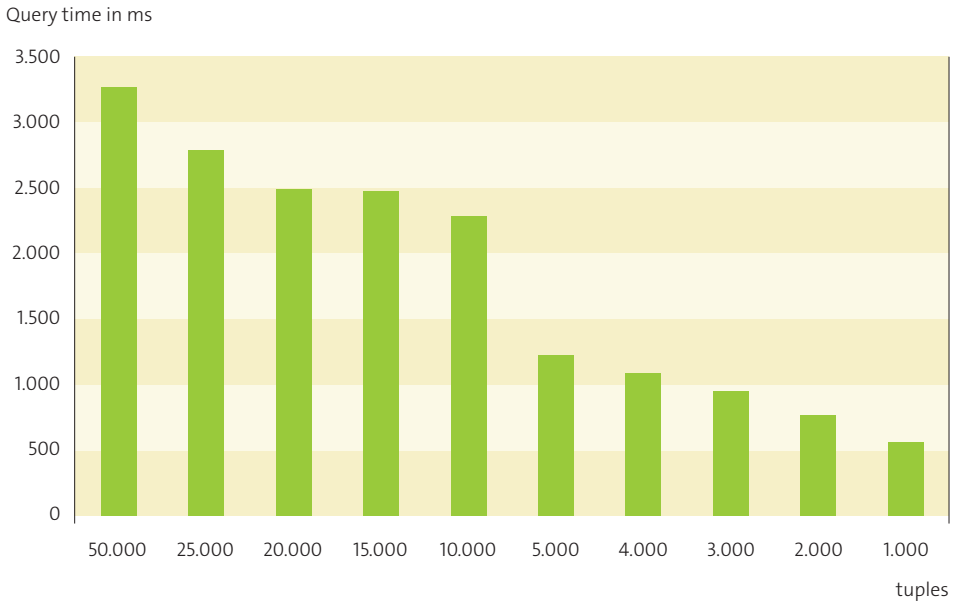


ABB. 11 MySQL-Basisperformance bei Abfragen ohne Partitionierung (ohne Netzwerkverbindung)

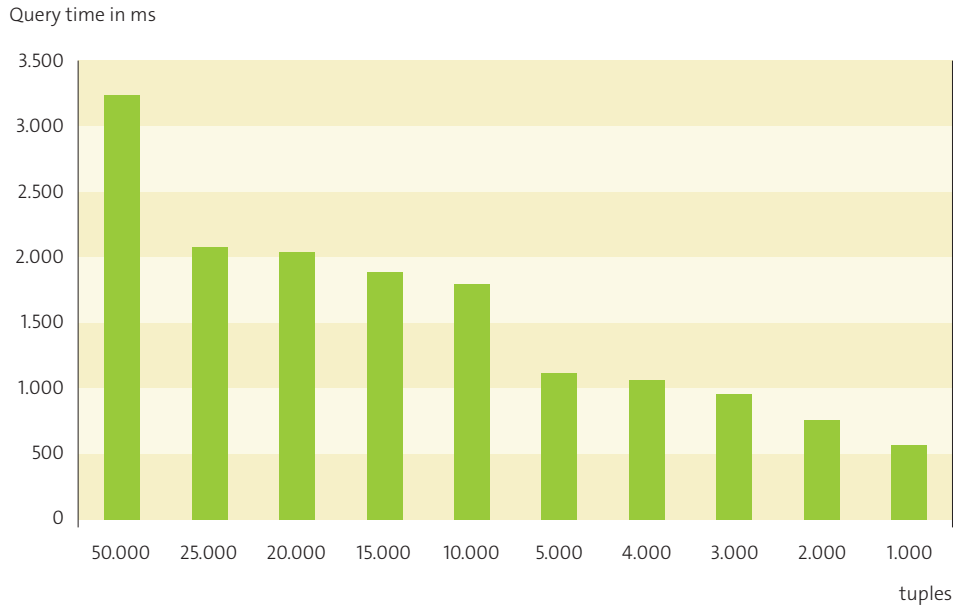


ABB. 12 Oracle-Basisperformance bei Abfragen ohne Partitionierung (ohne Netzwerkverbindung)

Als Nächstes ist analog zu ABB. 11 und ABB. 12 die Performance mit Partitionierung, aber ohne Netzwerklatenzen dargestellt.

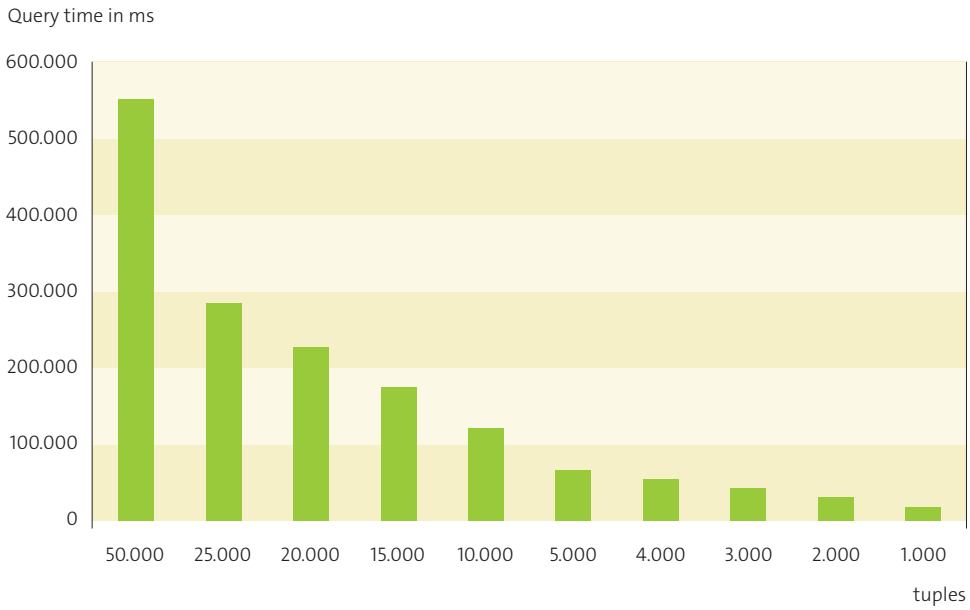


ABB. 13 MySQL-Performance bei Abfragen mit 2 Partitionen (ohne Netzwerkverbindung)

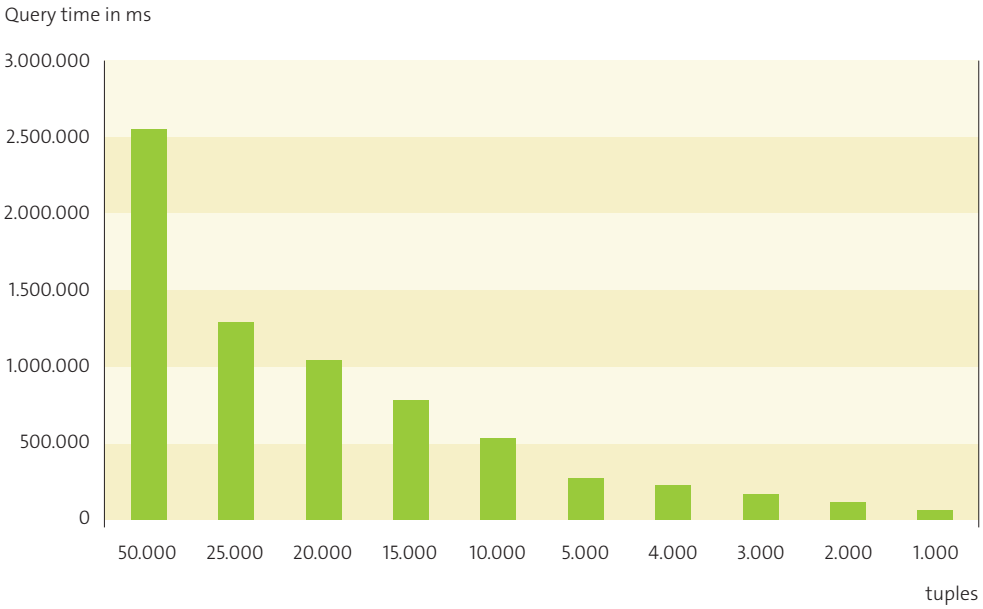


ABB. 14 Oracle-Performance bei Abfragen mit 2 Partitionen (ohne Netzwerkverbindung)

ABB. 15 zeigt nun die Performance bei gleichzeitiger Verwendung beider Datenbanksysteme, wobei eine Partition auf einer MySQL-Datenbank und die andere auf einer Oracle-Datenbank abgelegt ist. Da alle Datenbanken und der SeDiCo-Client auf derselben physischen Maschine installiert sind, sind ebenfalls keine Netzwerklatenzen enthalten.

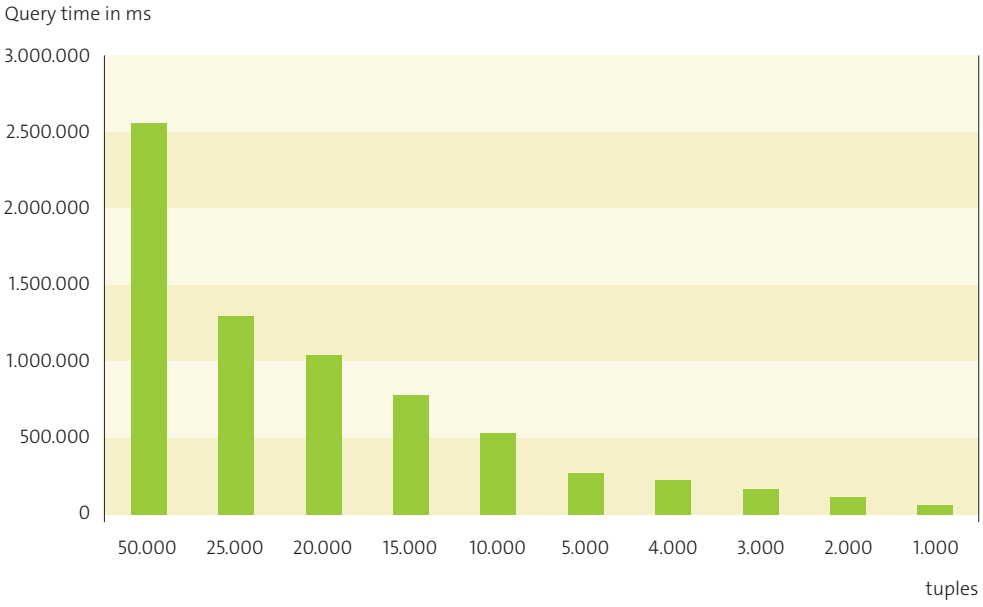


ABB. 15 Kombinierte Oracle- und MySQL-Performance bei Abfragen mit 2 Partitionen (ohne Netzwerkverbindung)

ABB. 16 bis ABB. 20 illustrieren nun die Performance der einzelnen Datenbanksysteme. Diese sind jetzt allerdings auf unterschiedlichen physischen Maschinen installiert, die über ein 100-Mbit-Netzwerk miteinander verbunden sind. So sind insgesamt 3 Maschinen in die Messung involviert: ein dediziertes MySQL-, ein dediziertes Oracle-Datenbanksystem und ein System für den SeDiCo-Client.

Analog zum obigen Vorgehen wird zuerst die Basismessung ohne Partitionierung dargestellt.

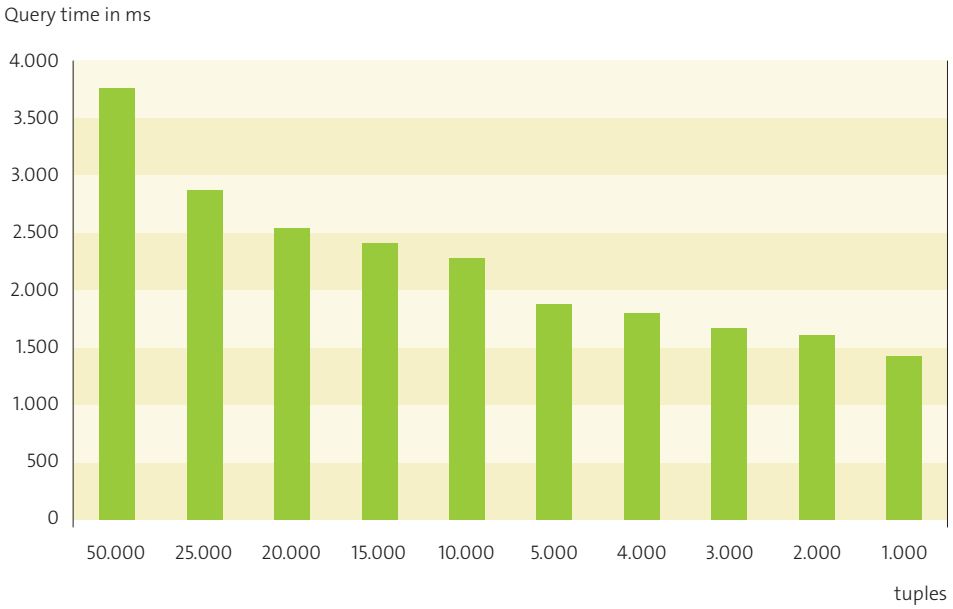


ABB. 16 MySQL-Basisperformance bei Abfragen ohne Partitionierung (mit Netzwerkverbindung)

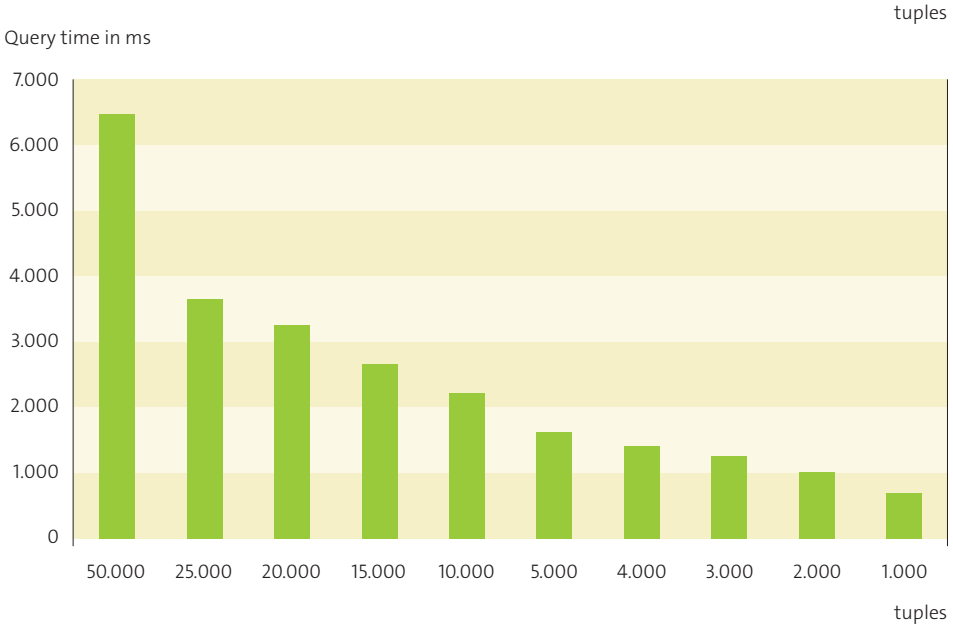


ABB. 17 Oracle-Basisperformance bei Abfragen ohne Partitionierung (mit Netzwerkverbindung)

Als Nächstes folgt die Messung mit zwei Partitionen über das Netzwerk.

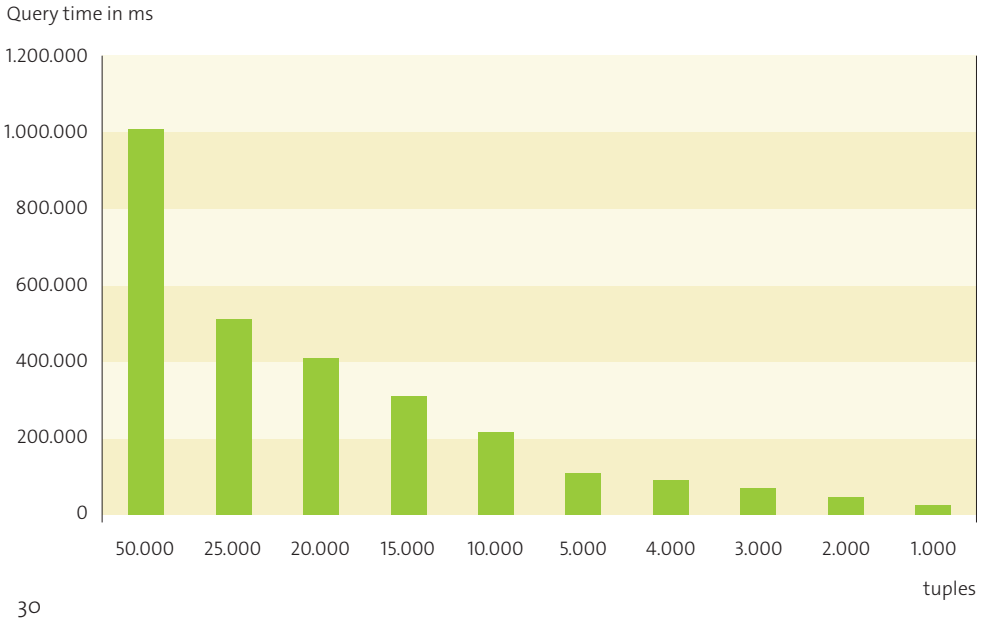


ABB. 18 MySQL-Performance bei Abfragen mit Partitionierung (mit Netzwerkverbindung)

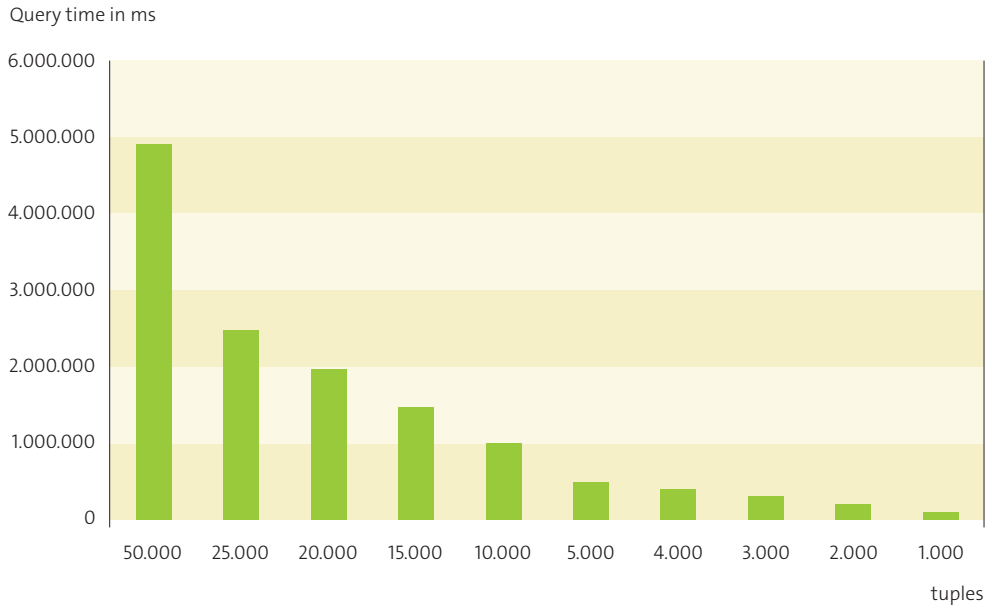


ABB. 19 Oracle-Performance bei Abfragen mit Partitionierung (mit Netzwerkverbindung)

Zuletzt ist die kombinierte Performance beider Datenbanksysteme angeführt, wobei jedes System auf einer eigenen physischen Maschine läuft, die durch ein 100-Mbit-Netzwerk miteinander verbunden sind.

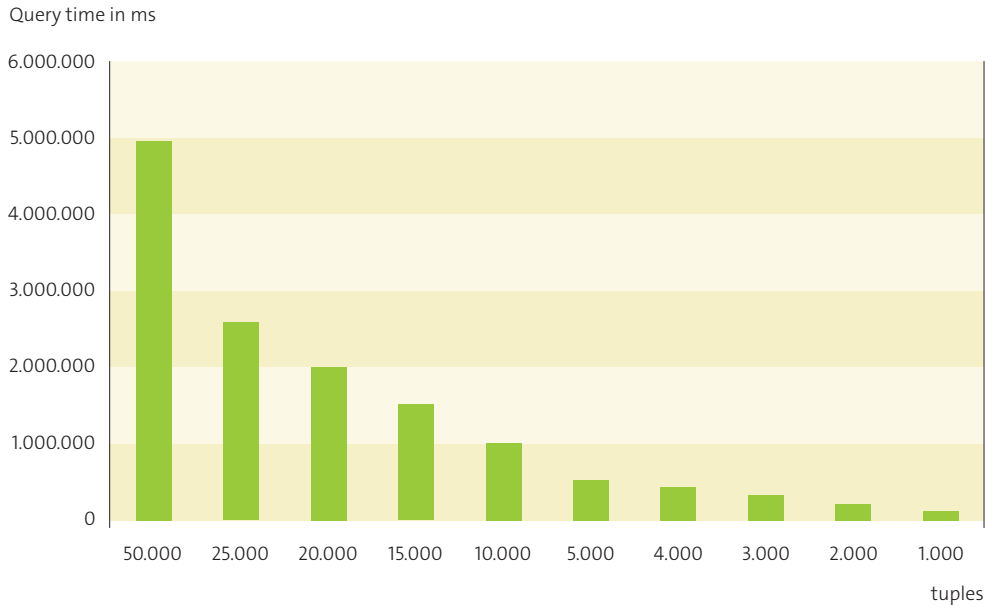


ABB. 20 Kombinierte MySQL- und Oracle-Performance bei Abfragen (mit Netzwerkverbindung)

### 4.2.3 Änderungsperformance

Analog zum Vorgehen bei der Analyse der Abfrageperformance ist auch hier zunächst als Grundlage die Basisperformance der jeweiligen Datenbanksysteme ohne Partitionierung und ohne Netzwerklatenzen für das Ändern von existierenden Datensätzen aufgeführt.

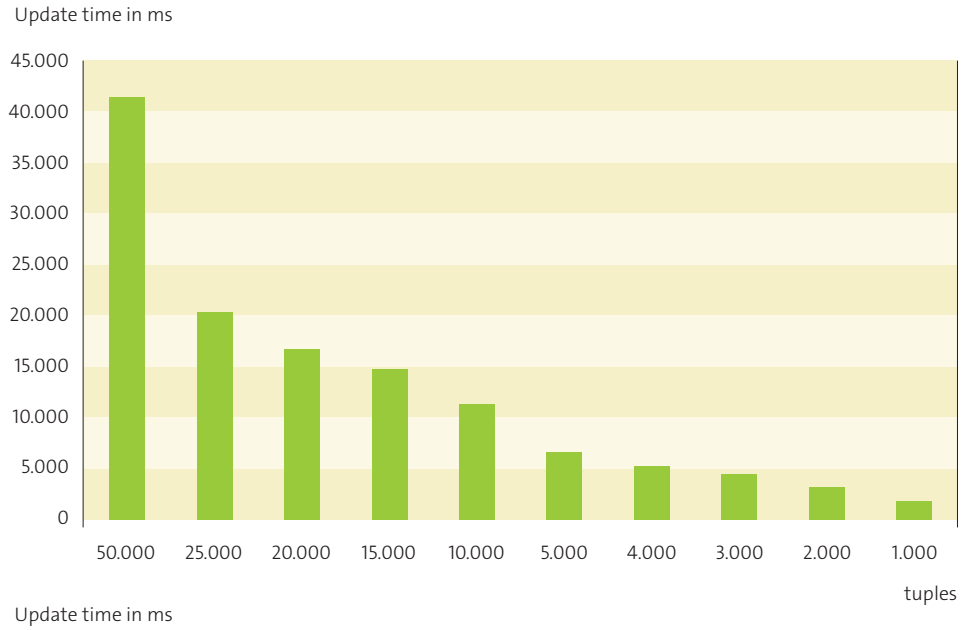


ABB. 21 MySQL-Basisperformance bei Änderungen ohne Partitionierung (ohne Netzwerkverbindung)

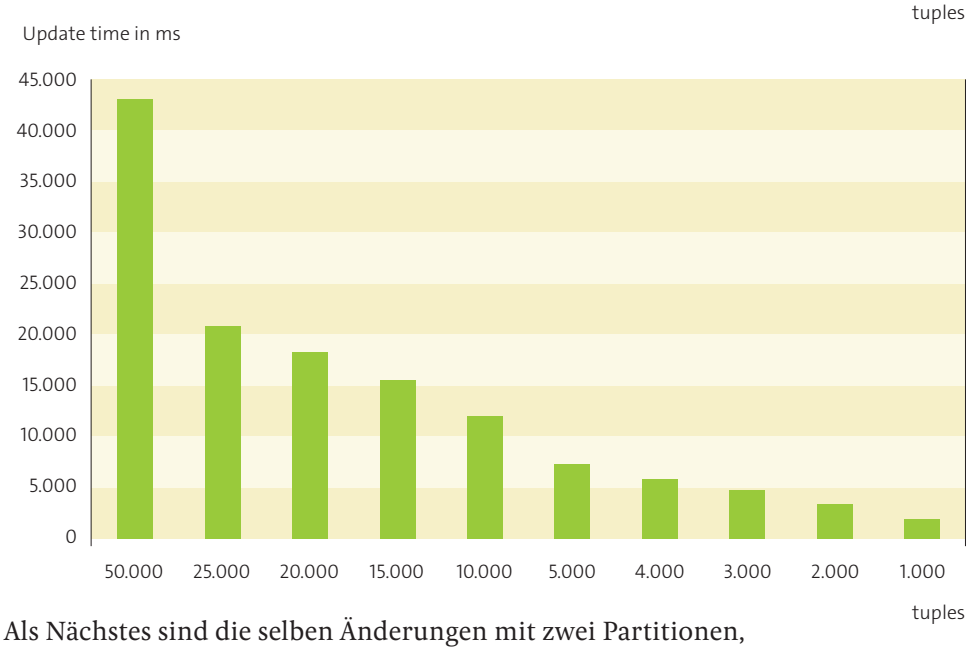


ABB. 22 Oracle-Basisperformance bei Änderungen ohne Partitionierung (ohne Netzwerkverbindung)

Als Nächstes sind die selben Änderungen mit zwei Partitionen, aber ohne Netzwerklatenzen angegeben.

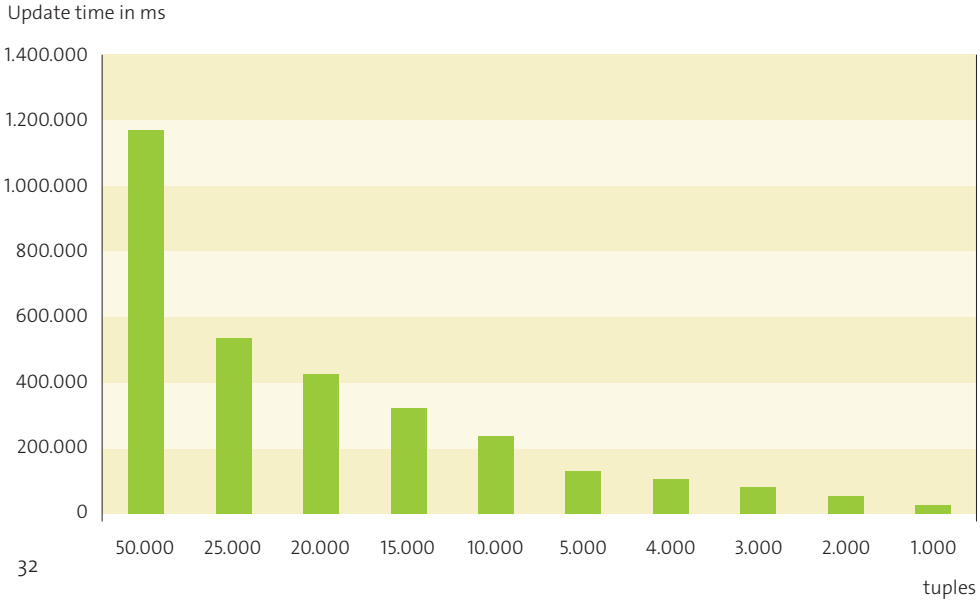


ABB. 23 MySQL-Performance bei Änderungen (ohne Netzwerkverbindung)

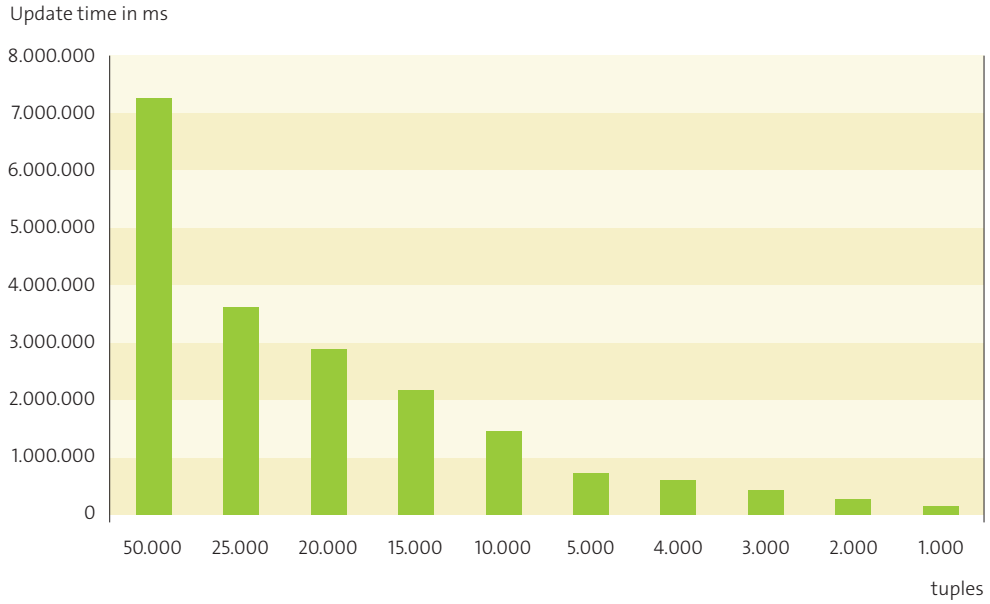


ABB. 24 Oracle-Performance bei Änderungen (ohne Netzwerkverbindung)

Zuletzt ist die Performance beider Datenbanksysteme bei gleichzeitiger Verwendung angegeben. Hier ist eine Partition auf einem MySQL- und die andere auf einem Oracle-System abgelegt. Da alle Datenbanksysteme und auch der Client auf derselben physischen Maschine installiert sind, enthalten die Angaben keine Netzwerklatenzen.

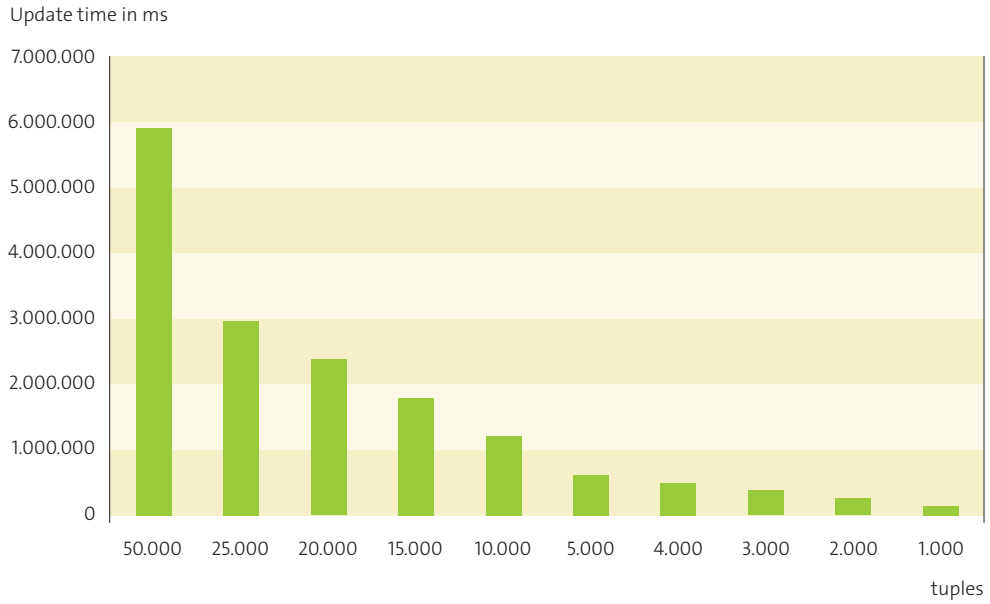


ABB. 25 Kombinierte Oracle- und MySQL-Performance bei Änderungen (ohne Netzwerkverbindung)

Für die nun folgenden Messungen wurden jeweils unterschiedliche physische Maschinen für die Datenbanksysteme und auch für den SeDiCo-Client verwendet. Alle Maschinen wurden dabei über ein 100-Mbit-Netzwerk verbunden.

Zunächst ist, analog zur Messung der Abfrageperformance, wieder die Basisperformance der Datenbanksysteme ohne Partitionierung aber mit Netzwerklatenzen angegeben.

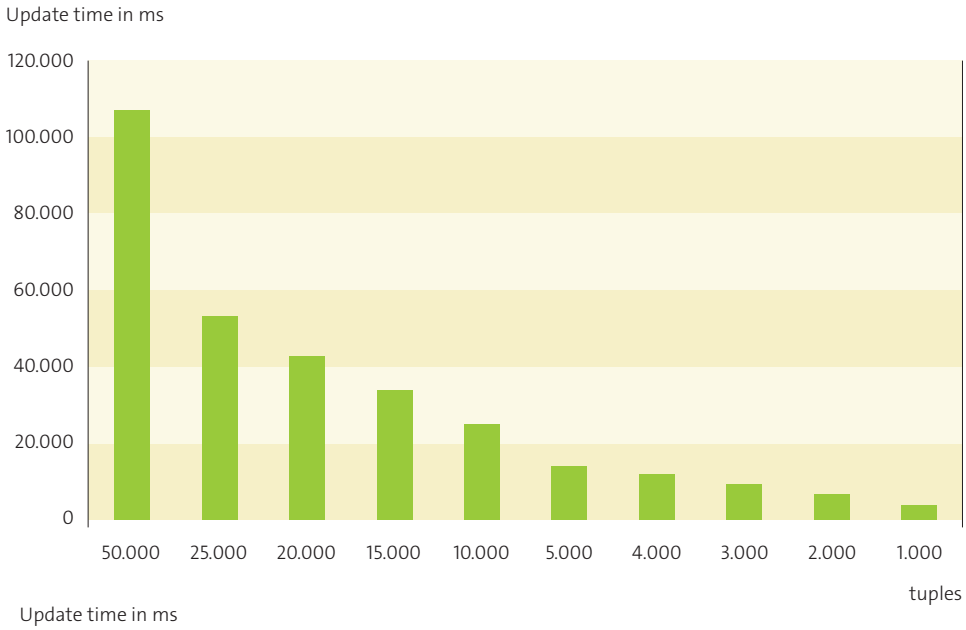


ABB. 26 MySQL-Basisperformance bei Änderungen ohne Partitionierung (mit Netzwerkverbindung)

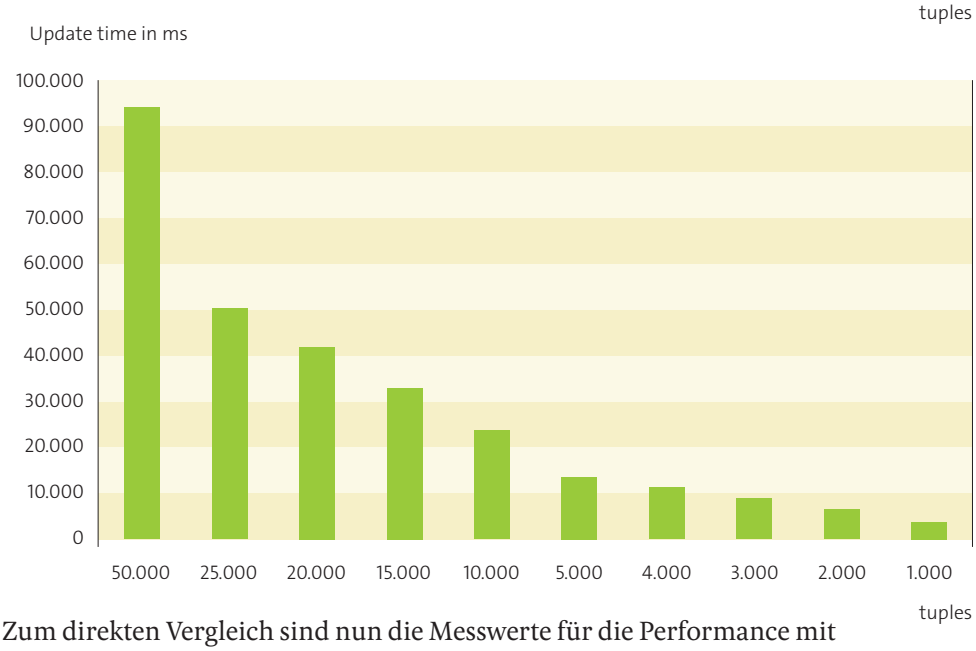


ABB. 27 Oracle-Basisperformance bei Änderungen ohne Partitionierung (mit Netzwerkverbindung)

Zum direkten Vergleich sind nun die Messwerte für die Performance mit zwei Partitionen und einer 100-Mbit-Netzwerkverbindung aufgeführt.

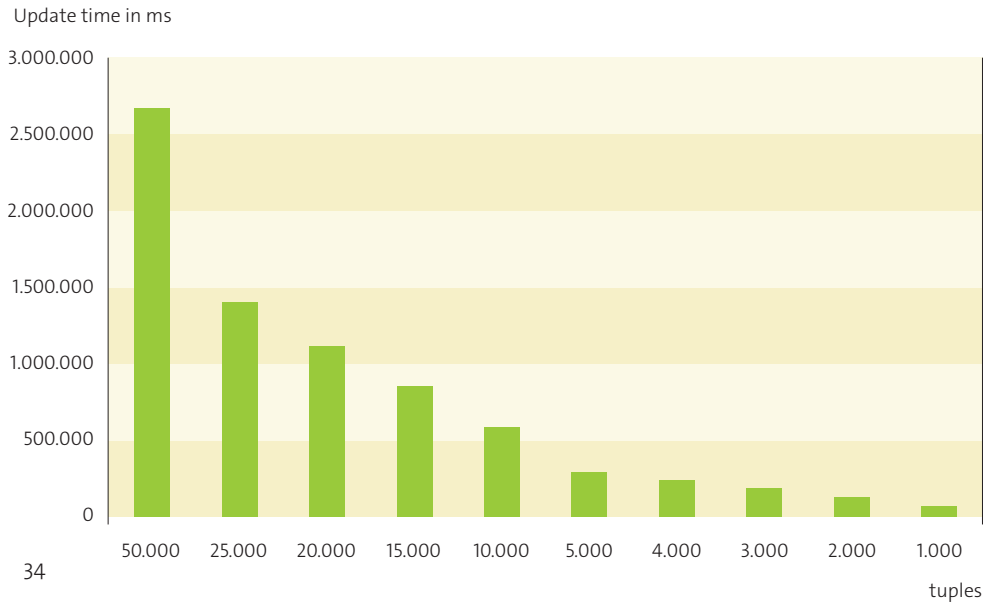


ABB. 28 MySQL-Performance bei Änderungen mit Partitionierung (mit Netzwerkverbindung)

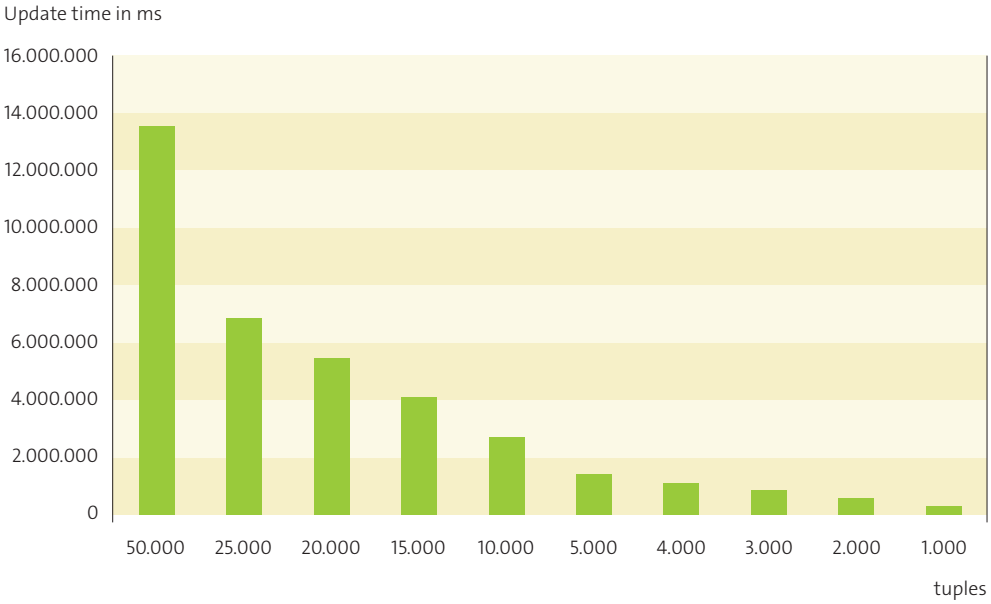


ABB. 29 Oracle-Performance bei Änderungen mit Partitionierung (mit Netzwerkverbindung)

Zuletzt ist nun die Performance bei gleichzeitigem Verwenden beider Datenbanksysteme angegeben, dabei liegt eine Partition auf einem MySQL- und die andere Partition auf einem Oracle-System. Die beiden Datenbanksysteme und der SeDiCo-Client, der die Daten abruft, sind durch ein 100-Mbit-Netzwerk miteinander verbunden.

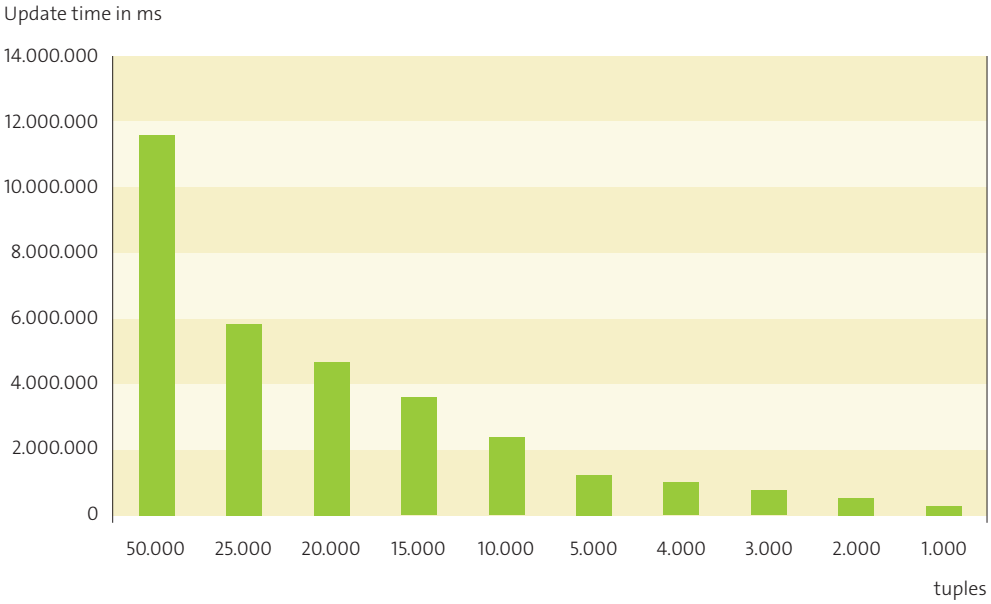


ABB. 30 Kombinierte MySQL- und Oracle-Performance bei Änderungen (mit Netzwerkverbindung)

### 4.2.4 Einfügeperformance

Analog zum Vorgehen bei der Analyse der Abfrage- und Änderungsperformance ist auch hier zunächst als Grundlage die Basisperformance der jeweiligen Datenbanksysteme ohne Partitionierung und ohne Netzwerklatenzen für das Einfügen von neuen Datensätzen (Tupel) in die Datenbank aufgeführt.

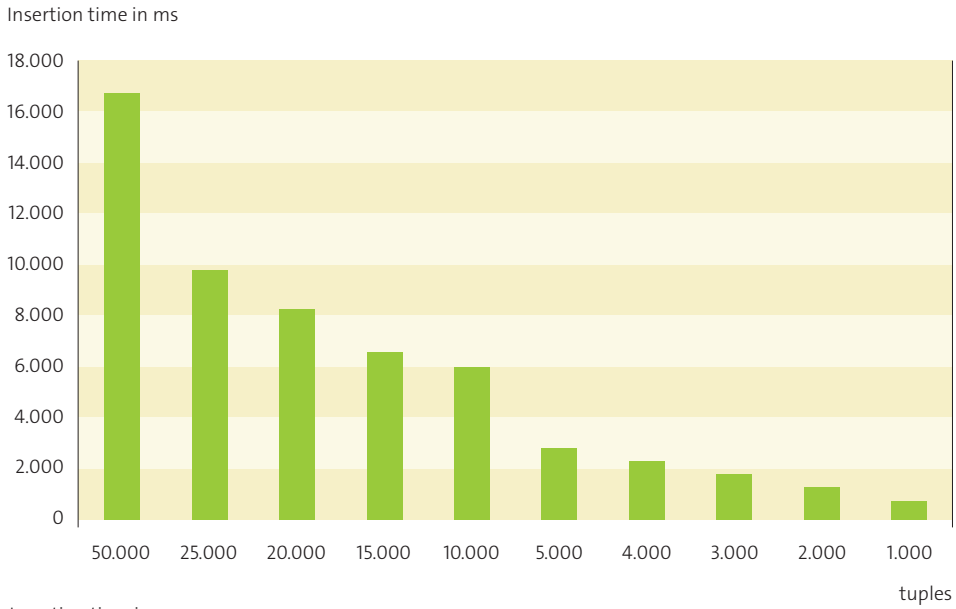


ABB. 31 MySQL-Basisperformance beim Einfügen ohne Partitionierung (ohne Netzwerkverbindung)

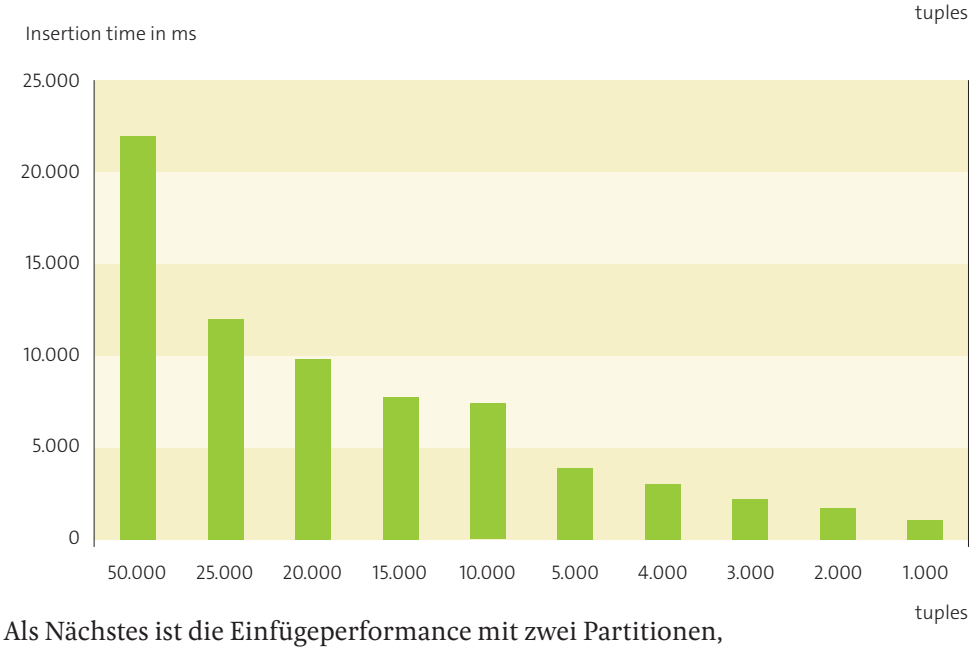


ABB. 32 Oracle-Basisperformance beim Einfügen ohne Partitionierung (ohne Netzwerkverbindung)

Als Nächstes ist die Einfügeperformance mit zwei Partitionen, aber ohne Netzwerklatenzen angegeben.

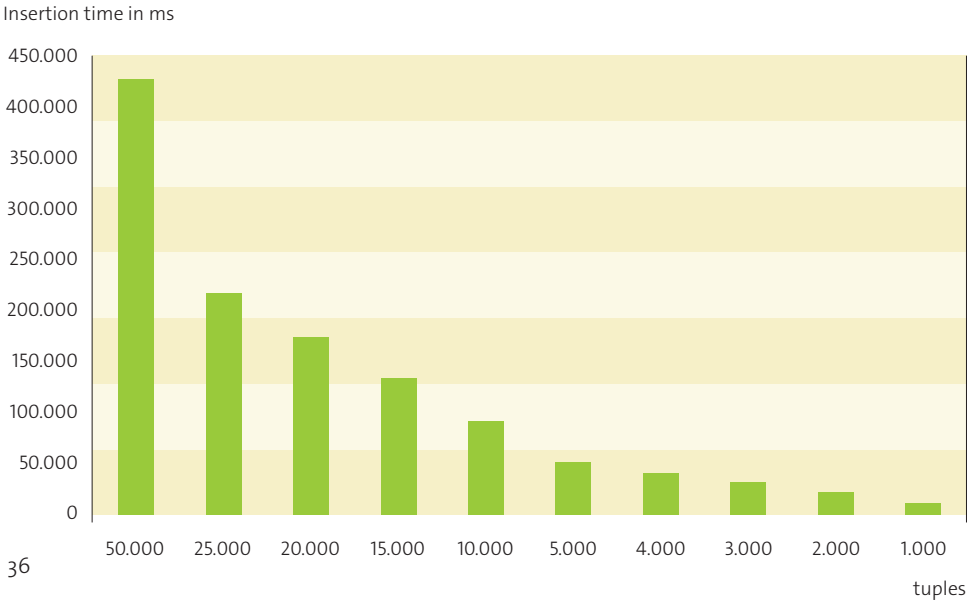


ABB. 33 MySQL-Performance beim Einfügen mit Partitionierung (ohne Netzwerkverbindung)

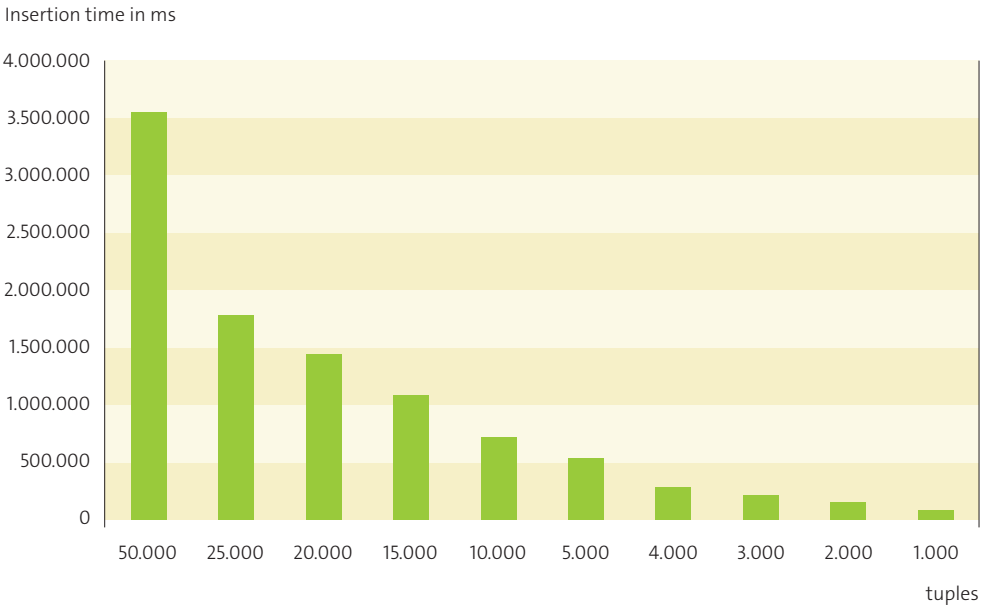


ABB. 34 Oracle-Performance beim Einfügen mit Partitionierung (ohne Netzwerkverbindung)

Zuletzt ist die Performance bei der Datenbanksysteme bei gleichzeitiger Verwendung angegeben. Hier ist eine Partition auf einem MySQL- und die andere auf einem Oracle-System abgelegt. Da alle Datenbanksysteme und auch der Client auf derselben physischen Maschine installiert sind, enthalten die Angaben keine Netzwerklatenzen.

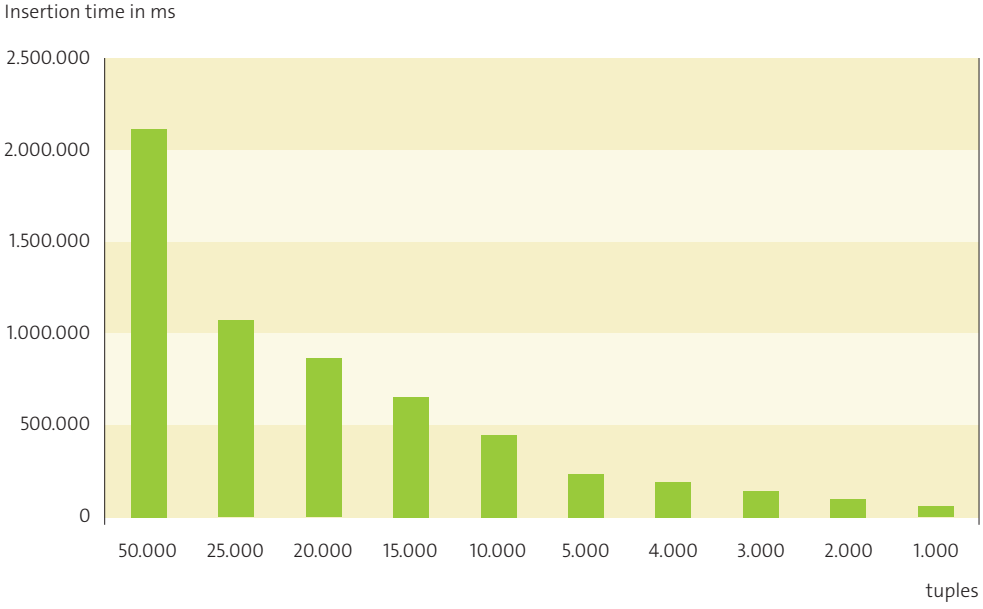


ABB. 35 Kombinierte MySQL- und Oracle-Performance beim Einfügen (ohne Netzwerkverbindung)

Für die nun folgenden Messungen (ABB. 36 bis ABB. 38) wurden jeweils unterschiedliche physische Maschinen für die Datenbanksysteme und auch für den SeDiCo-Client verwendet. Alle Maschinen wurden dabei über ein 100-Mbit-Netzwerk verbunden.

Zunächst ist, analog zur Messung der Abfrage- und Änderungsperformance, wieder die Basisperformance der Datenbanksysteme ohne Partitionierung aber mit Netzwerklatenzen angegeben.

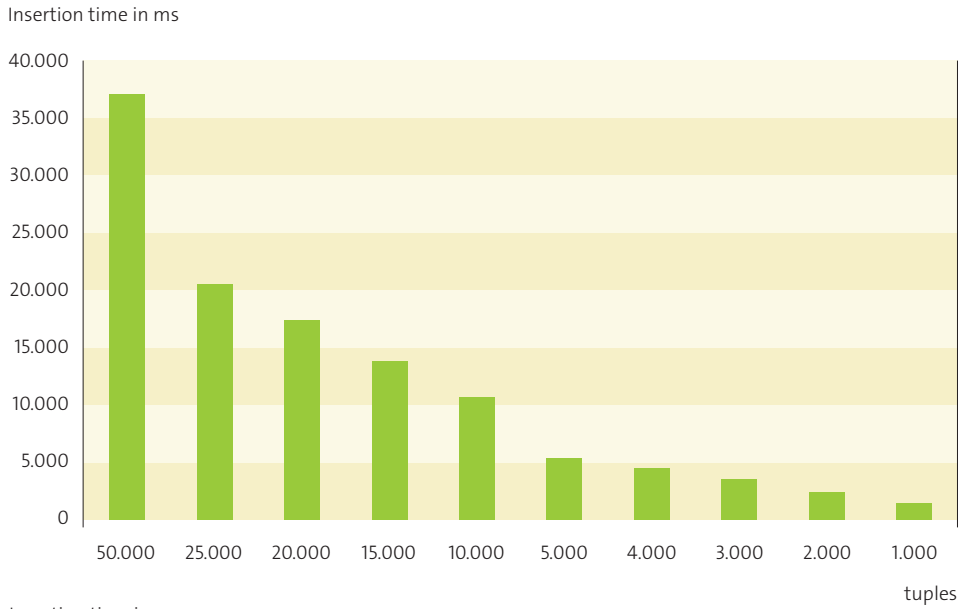


ABB. 36 MySQL-Basisperformance beim Einfügen mit Partitionierung (mit Netzwerkverbindung)

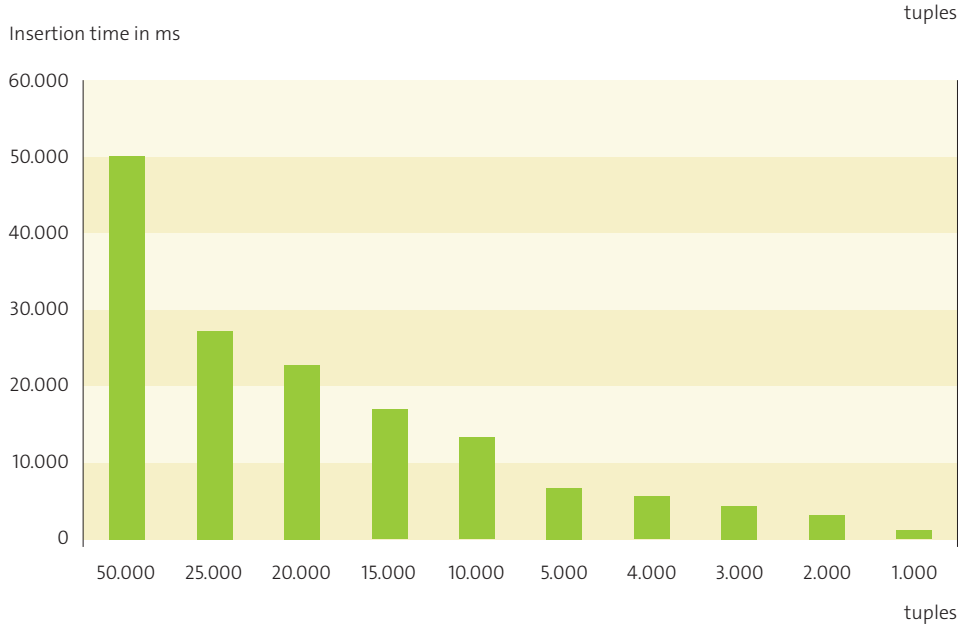


ABB. 37 Oracle-Basisperformance beim Einfügen mit Partitionierung (mit Netzwerkverbindung)

Zuletzt ist wieder die kombinierte Performance beider Datenbanksysteme angeführt, wobei jedes System auf einer eigenen physischen Maschine läuft, die durch ein 100-Mbit-Netzwerk miteinander verbunden sind.

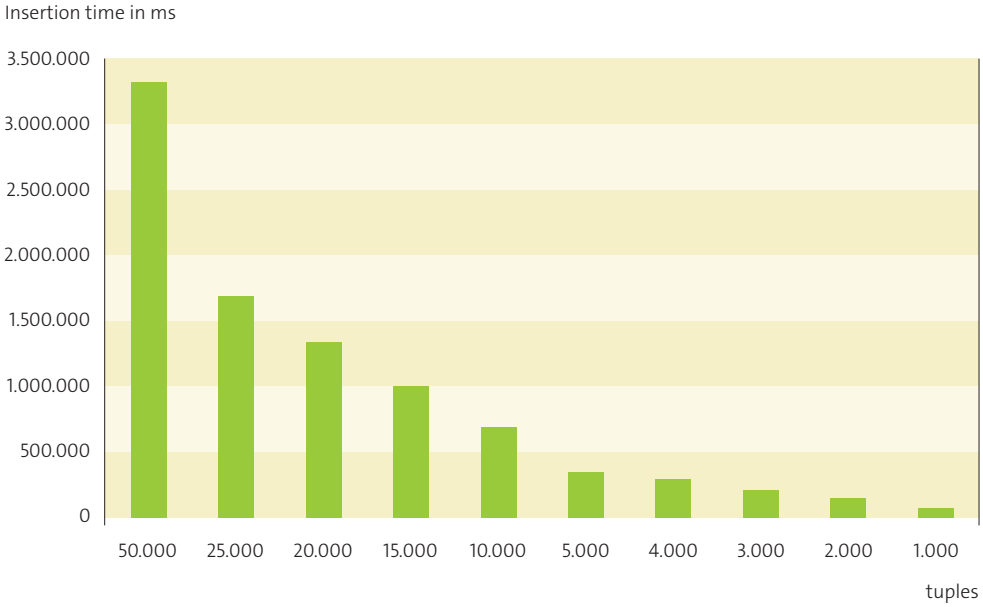


ABB. 38 Kombinierte MySQL- und Oracle-Performance beim Einfügen (mit Netzwerkverbindung)

### 4.2.5 Fazit

Das verwendete Datenvolumen erzeugt in seiner Standardkonfiguration 288.000 Tupel und basiert auf einer Arbeit von Bezenek et al. (Bezenek et al. 2000). Dies stellte die Obergrenze für die hier präsentierten Resultate dar. Aus Gründen der Übersichtlichkeit wurden nur die Resultate zwischen 1.000 und 50.000 Datensätze in ABB. 38 übernommen. Alle Messungen wurden insgesamt 3-mal durchgeführt und anschließend wurde die durchschnittliche Zeit gebildet. Ferner wurden keine Optimierungen an den Datenbanksystemen (z. B. die Erhöhung der Puffergröße, die Erhöhung der maximal zulässigen, gleichzeitigen Verbindungen, etc.) vorgenommen[18]. Letztlich bleibt anzumerken, dass alle Benchmarks mit Hilfe der frei verfügbaren DBMS MySQL und Oracle durchgeführt wurden. Die enormen Unterschiede (teilweise 50 % Performanceverlust von Oracle im Vergleich zu MySQL) resultieren aus der Tatsache, dass das Oracle DBMS in der freien Version nur einen Prozessor und nur 1 GB RAM verwendet, wohingegen MySQL die volle zugrundeliegende Hardware nutzt.

Krueger et al. zeigen in ihrer Studie, dass das verwendete Datenvolumen durchaus repräsentativ für den Einsatz im Unternehmensumfeld ist (Krueger et al. 2011). TAB. 6 stellt die Ergebnisse der Studie zu den Datenvolumen im Unternehmensumfeld dar[19].

18 mit den genannten Optimierungen war es möglich, den Performanceverlust auf 40% auf einem MySQL System mit 4 Kernen und 8 GB RAM zu senken

19 befragt wurden 12 SAP-Kunden mit ca. 74.000 Tabellen pro Kunde

| Anzahl Tabellen | Anzahl Datensätze pro Tabelle |
|---|---|
| 46.418 | 0 |
| 15.553 | 1–100 |
| 6.290 | 100–1.000 |
| 2.685 | 1.000–10.000 |
| 1.385 | 10.000–100.000 |
| 925 | 100.000–1 Mio. |
| 579 | 1 Mio.–10 Mio. |
| 144 | < 10 Mio. |

TAB. 6 Datenvolumen im Unternehmensumfeld

Generell lässt sich festhalten, dass sich der Ansatz der vertikalen Verteilung aufgrund der gezeigten technischen Machbarkeit insbesondere im Unternehmensumfeld nutzen lässt. Der gezeigte Performanceverlust wirkt sich primär bei Datenvolumen ab 1.000 Tupel negativ auf die Performance aus, was für die meisten Datenbanken im Unternehmensumfeld keine Rolle spielt. Die Performanceverluste bei größeren Datenvolumen werden in Kapitel 7 aufgegriffen. Die zukünftigen Arbeiten beschäftigen sich dort mit der Einführung eines Caches, der Daten im schnellen Hauptspeicher vorhält, um nicht immer direkt auf die verteilen Datenstrukturen in der Cloud zugreifen zu müssen.

# 5 Verwandte Arbeiten

Im Gegensatz zu Kapitel 7 wird hier auf tangierende Projekte eingegangen, die ähnliche Vorgehensweisen verfolgen und die nützliche Hinweise auf weitere mögliche Entwicklungen geben. Im Fokus der weiteren Arbeit stehen aber die im Ausblick erwähnten Frage- und Problemstellungen.

Wie bereits in Abschnitt 3.2 dargestellt, ist die Abstraktion verschiedener Cloud-Anbieterschnittstellen ein zentrales Thema dieser Arbeit. Mit der Kapselung der unterschiedlichen Schnittstellen beschäftigen sich außer den genannten Abstraktionsframeworks jclouds, libcloud und deltacloud tangierende Arbeiten wie Withana/Pale 2011, Binz et al. 2013, Loutas et al. 2010 oder Nguyen et al. 2011. Allerdings ist anzumerken, dass sich die genannten Arbeiten ausschließlich mit der technischen Abstraktion der Schnittstellen beschäftigen. Sicherheitsaspekte wie in dieser Arbeit bleiben dabei außen vor (Kaiser 2013). Darüber hinaus ist mittlerweile eine Sammlung verschiedener Austauschformate und Vorgehensweisen entstanden [20], die eine einfache Datenübernahme von und zu unterschiedlichen Cloud-Anbietern ermöglichen. Hier beteiligen sich verschiedene Standardisierungsgremien wie z. B. die OASIS [21], OCCI [22] oder die ETSI [23]. Die konkrete Implementierung der Datenübernahme bleibt allerdings nach wie vor dem Nutzer überlassen.

Mit dem Konzept der Datenverteilung beschäftigt sich auch das MimoSecco-Projekt [24], das allerdings die Entwicklung einer Middleware zum Ziel hat. Damit bewegt sich dieses Projekt architektonisch eine Stufe höher als SeDiCo, da es sich mit der hardwarebasierten Absicherung (mittels sog. „Tokens“) von beliebigen Clients befasst. Daten lassen sich damit verteilt, über wechselnde Standorte sicher bearbeiten und abrufen. Im Fokus steht dabei nicht die Datenbankebene sondern der Zugriff auf Dienste und Dateien im Allgemeinen (Schiefer 2012). In eine ähnliche Richtung gehen die Arbeiten im DepSky-Projekt [25]. Hier wird, ebenfalls auf Dateiebene, eine Verschlüsselung und Verteilung, im Sinne einer Datenreplikation vorgeschlagen (Bessani 2011). Durch die vorgeschlagene redundante Datenhaltung wird die Ausfallsicherheit erhöht. Eine logische Verteilung der Daten wie in dieser Arbeit wird dabei nicht beachtet.

Ein weiterer Ansatz mit Bezug zur Datenpartitionierung, der sich mit der Performancesteigerung von Datenbanksystemen beschäftigt, ist der Ansatz der horizontalen Partitionierung (Plattner 2013). Dieser Ansatz kommt ursprünglich aus dem In-Memory Computing, bei dem Datensätze auf viele verschiedene physische Rechner (sog. Knoten) verteilt werden. So verteilen sich der Berechnungsaufwand und der Netzwerkverkehr auf viele unterschiedliche Rechner. Dabei ist dann nicht der gesamte Datenbestand zu durchsuchen, sondern nur der Teil-Datenbestand

20 http://cloud-standards.org/

21 https://www.oasis-open.org/

22 http://occi-wg.org/

23 http://csc.etsi.org/

24 http://mimosecco.de/

25 https://code.google.com/p/depsky/

eines spezifischen Knotens, der die Daten auch wirklich vorhält. Als konkretes Szenario könnte so z. B. eine Tabelle „Kunden“ nach dem Alter der Kunden horizontal partitioniert werden. Kunden zwischen 18 und 30 Jahren würden in eine Partition auf einen Knoten ausgelagert. Eine zweite Partition auf einem anderen Knoten würde Kunden zwischen 31 und 50 Jahren und die letzte Partition auf einem dritten Knoten würde alle Kunden, die älter als 51 Jahre sind, beinhalten (Müller et al. 2014).

Eine andere Vorgehensweise wäre, die Daten parallel mit mehreren gleichzeitigen Verbindungen eines Clients aus den Cloud-Datenbanken zu holen. So könnte der SeDiCo-Client mit mehreren parallel laufenden Threads die Daten nicht einzeln sondern gesammelt aus den jeweiligen Cloud-Datenbanken lesen und schreiben. Diese Vorgehensweise wäre aktuell mit entsprechendem Implementierungsaufwand im SeDiCo-Client möglich. Darüber hinausgehende Ansätze wie CUDA oder MapReduce (Dean/Ghemawat 2004) verfolgen hier ähnliche Vorgehensweisen, sind aber architektonisch gesehen aufwändiger in den bestehenden SeDiCo-Client zu integrieren (Kohler/Specht 2014b).

Weiterhin ist ein zentrales Element dieser Arbeit die Datenhaltung auf Datenbankebene. Bei der Betrachtung des SQL-Standards treten viele unterschiedliche datenbankherstellerspezifische Implementierungen hervor (Bargmeyer 2011). So erweitern Datenbankhersteller den Standard um eigene Funktionen, wie z. B. die „auto increment“-Funktion in MySQL, die z. B. zum automatischen Erhöhen eines Primärschlüsselwertes verwendet werden kann. Diese Erweiterungen machen eine herstellerübergreifende Datenübernahme unmöglich, da beispielsweise die o. g. „auto increment“-Funktion bei anderen Datenbankherstellern nicht existiert.

Außerdem finden sog. „NoSQL“-Datenbanken[26] mit ihren teilweise aufgeweichten ACID-Kriterien[27] gerade im In-Memory Computing eine weite Verbreitung (Plattner 2013). Ebenso wie objektrelationale Datenbanken sind diese Datenbanken durch eine andere Art der Datenspeicherung (z. B. key-value Paare, Colum Stores, Document Stores, etc.) gekennzeichnet. Eine Unterstützung dieser Datenbanken durch den SeDiCo-Prototyp wäre wünschenswert, jedoch ist derzeit aufgrund der hohen Anzahl an unterschiedlichen NoSQL-Implementierungen[28] nicht ersichtlich, welche Anbieter bzw. welche konkrete Art der Datenspeicherung sich letztlich durchsetzen wird. Daher ist an dieser Stelle mit der Integration noch abzuwarten.

Andere Arbeiten in diesem Kontext beschäftigen sich mit der Skalierbarkeit von Datenbanken. So wird im Leads-Projekt[29] an der Hochschule in Karlsruhe untersucht, bis zu welchem Datenvolumen SQL-Datenbanken gut skalieren und ab welchem Volumen NoSQL Datenbanken die bessere Wahl sind. Mit Blick auf neue Herausforderungen im Kontext von „Big Data“ (s. a. Kapitel 7) gibt dieses Projekt sehr nützliche Hinweise und Impulse für die weitere Arbeit.

26 http://nosql-databases.org/

27 die letztlich die Transaktionssicherheit bei SQL-Datenbanken sicherstellen

28 http://nosql-databases.org/

29 http://stiftung.mfg.de/de/forschungsforderung/projekte/leads

# 6 Fazit

Als grundlegendes Fazit bleibt anzumerken, dass die technische Machbarkeit des vertikalen Verteilungsansatzes mit dieser Arbeit erfolgreich demonstriert wurde. Zu den größten Herausforderungen zählte die Kapselung der verschiedenen Cloud-Anbieterschnittstellen in eine einheitliche Schnittstelle. Glücklicherweise konnte hier auf das jclouds-Framework zurückgegriffen werden (s. a. Abschnitt 3.2). Ähnlich verhielt es sich mit der Integration der unterschiedlichen Datenbanksysteme. Der Prototyp unterstützt nun mit MySQL und Oracle Express zwei unterschiedliche Datenbanksysteme. Dies wurde mit einigen speziellen Ausnahmen durch Hibernate erfolgreich realisiert (s. a. Abschnitt 4.1). Damit ist letztlich die technische Machbarkeit des Ansatzes gegeben.

Allerdings machen neue Fragestellungen auf konzeptioneller Seite weitere wissenschaftliche Arbeiten notwendig. Diese sind in Abschnitt 5 und in Abschnitt 7 näher erläutert. Insbesondere die in Abschnitt 7 dargestellten Punkte werden für die zukünftige Arbeit Berücksichtigung finden.

Letztlich zeigen die oben dargestellten Benchmark-Resultate (s. Abschnitt 4.2), dass der Ansatz der vertikalen Partitionierung einen Performanceverlust mit sich bringt. Dies gilt nicht nur für das Einfügen von neuen Daten, sondern auch für das Ändern und das Abfragen von Daten. Wie eine Analyse der Datenvolumen im Unternehmensumfeld zeigt, enthalten die meisten Tabellen dort weniger als 1.000 Tupel (s. a. TAB. 6). Dies relativiert den Performanceverlust fast vollständig, was den Einsatz des Prototyps im Unternehmensbereich sehr interessant macht. Des Weiteren erhöht der vertikale Ansatz die Sicherheit bei der Verwendung von öffentlichen Cloud-Infrastrukturen. Die Arbeit im Projekt zeigt also, dass die Nutzung von öffentlichen Clouds nur bei größeren Datenvolumen ein Abwägen zwischen Sicherheit und Performance darstellt.

# 7 Ausblick

Im Gegensatz zu Kapitel 5 beschreibt dieses Kapitel nun abschließend konkrete Frage- und Problemstellungen, die in weiteren wissenschaftlichen Arbeiten angegangen werden sollen.

Die Verwendung der Hibernate-Events[30] bringt das bekannte SELECT n+1 Problem mit sich (Bauer/King 2007). Durch die Erzeugung einer einzigen Query, die n Entitäten zurückgibt, werden n Queries erzeugt und an die zweite Partition gesendet. Bei einer hinreichend großen Anzahl an Datensätzen leidet die Performance stark unter dieser Herangehensweise. Für einen praktikablen Einsatz mit einer hohen Anzahl an Datensätzen wäre es daher erforderlich, für diesen Bereich eine verbesserte Strategie, analog zu z. B. Batch-Updates (Cordts et al. 2011; Bauer/King 2007) zu entwickeln.

Eine weitere Fragestellung ist der Zugriff auf die zweite Partition. Die momentane Implementierung erlaubt nur die Erstellung von Abfragen für die Primärpartition. Die Daten einer anderen Partition können nur geladen werden, wenn die erste Partition Tupel zurückliefert. Für den praktischen Einsatz wäre es erforderlich, eine Zugriffsmöglichkeit für alle Partitionen zu entwickeln. Als Stolperstein präsentiert sich dabei vor allem die Kombination von Abfragebedingungen, bei denen die Bedingung nur durch Anfragen an mehrere Server aufzulösen ist. Hierzu wird an dieser Stelle als Konzept eine Anlehnung an In-Memory Datenbanken vorgeschlagen. So sollte vor jeder Abfrage die komplette Datenbank in den Hauptspeicher des Servers geladen werden, der die Abfrage an die verschiedenen Partitionen stellt. Da sich dieser sinnvollerweise innerhalb des Unternehmensnetzwerks befindet und die Kommunikation SSL-verschlüsselt abläuft, wird diese Vorgehensweise als sicher angesehen. Die Partitionen werden also auf diese Weise wieder zusammengefügt und somit ist die Ausgangstabelle für sämtliche Abfragen wiederhergestellt.

Zuletzt stellt sich die Frage nach der Anzahl der jeweiligen Partitionen. Exemplarisch und um die Komplexität im Rahmen dieser Arbeit beherrschbar zu machen, wurde die technische Machbarkeit für zwei Partitionen gezeigt. Die Erweiterung auf 3 oder n Partitionen erfolgt bei einfachem Primärschlüssel allerdings analog, indem die Primärschlüssel dann eben in jede Partition repliziert werden. Performanceanalysen müssen allerdings erst noch zeigen, ob der dadurch zusätzlich entstehende Performanceverlust dieses Ansatzes für große Datenmengen zu verkraften ist.

Mit Blick auf den eben genannten Performanceverlust durch die vertikale Verteilung stellt sich die Frage, ob die einzelnen Partitionen zusätzlich zu ihrer Verteilung noch verschlüsselt werden sollten. Huber et al. stellen hier einen sehr interessanten Ansatz zur Ver- und Entschlüsselung von Datenbanktabellen dar

30 onPreDelete, onPreInsert, onPreUpdate, onPreLoad

(Huber et al. 2013). Dieser Ansatz konzentriert sich zwar ausschließlich auf das Ver- und Entschlüsseln von Datenbankdaten, könnte aber in den hier dargestellten Ansatz der vertikalen Verteilung integriert werden. Letztlich müsste allerdings dann die Performance neu analysiert werden, da die zusätzliche Ver- und Entschlüsselung mehr Aufwand mit sich bringt.

Gerade bei der Betrachtung der Performance wäre eine weitere Möglichkeit, die Daten zu komprimieren, um den Kommunikationsoverhead so gering wie möglich zu halten. Auf der anderen Seite stehen dabei wieder die Punkte Datensicherheit und -schutz, die eine zusätzliche Verschlüsselung der einzelnen Partitionen nahelegen. Hier kommt wieder einmal die zentrale Herausforderung dieser Thematik zum Vorschein, bei der es zwischen Performance und Sicherheit abzuwägen gilt (Müller et al. 2014).

Mit Bezug zu Steigerungen der Performance des hier dargestellten Ansatzes beschäftigt sich eine konzeptionelle Arbeit von Kohler und Specht (Kohler/Specht 2014a). Hier werden drei Caching- und Synchronisationsansätze diskutiert, die entsprechende Performancesteigerungen versprechen. Einmal das Vorhalten von häufig benötigten Daten im Arbeitsspeicher des Clients, dann das clientseitige Zwischenspeichern in einer temporären Datenbanktabelle und letztlich das Zwischenspeichern von häufig frequentierten Daten in XML- bzw. JSON-Dateien beim Client. Die konzeptionelle Arbeit ist damit abgeschlossen, allerdings konnte bisher keiner der Ansätze implementiert werden. So könnte die Einführung einer zusätzlichen Caching-Schicht (s. ABB. 39) entscheidende Performanceverbesserungen bringen. ABB. 39 stellt die Idee eines Caches für häufig frequentierte Daten als Gesamtkonzept dar.

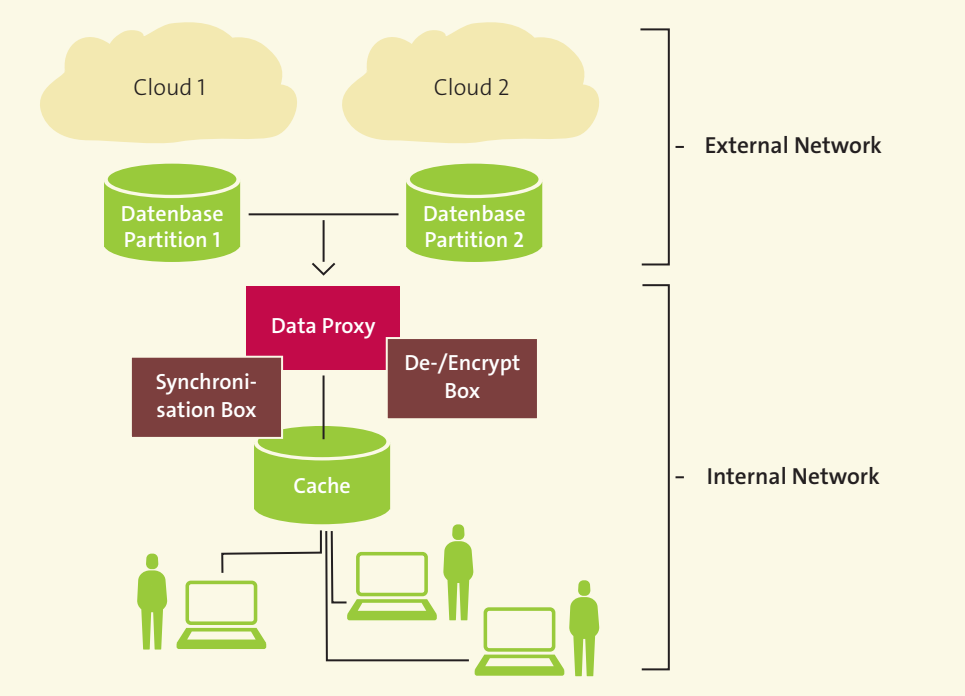


ABB. 39 Data Proxy Ansatz

Das Bild zeigt einen sog. „Data Proxy“, der das Zusammenführen der Daten implementiert. Bei diesem Data Proxy handelt es sich um den hier entwickelten SeDiCo-Client. Weiterhin werden in einem Cache häufig verwendete Daten schon zusammengeführt vorgehalten. Da dieser Cache im internen Unternehmensnetzwerk liegt, stellt dies aus sicherheitskritischer Sicht keine Einschränkung dar. Letztlich muss nur eine entsprechende Synchronisationslogik zwischen Cache und Data Proxy, die im Cache zwischengespeicherten Daten mit den Daten in den Cloud-Datenbanken aktuell halten (s. „Synchronisation Box“ in ABB. 39). Mit dieser Vorgehensweise müssten häufig benötigte Daten nicht immer aufwändig aus den Clouds geladen und zusammengeführt werden, sondern könnten direkt und schnell aus dem Cache bedient werden. Mit Bezug zu Synchronisations- und Cachingstrategien geben beispielsweise Segev/Park 1989 nützliche Hinweise auf eine entsprechende Realisierung. Anhand der Quelle ist ersichtlich, dass es sich dabei bereits um längerfristige Forschungsarbeiten handelt, die nun durch den Cloud Computing-Hype in Kombination mit mobilen Endgeräten einen neuen Aufschwung erleben.

Ein weiteres interessantes tangierendes Forschungsfeld ist das Problem der Klassifikation von Daten in „kritische Daten“, die das interne Unternehmensnetzwerk auf keinen Fall verlassen dürfen und in „weniger kritische Daten“, die in der öffentlichen Cloud abgelegt werden dürfen (Kohler/Specht 2014). So müssten die Daten z. B. mit Java Annotationen versehen werden, um sie entsprechend einzuordnen. Außerdem stellt sich die Frage nach dem manuellen Aufwand für den Benutzer. Hier könnte eine automatische Analyse mit Hilfe semantischer Technologien ein erfolgsversprechendes Konzept sein.

Diese Themen hätten den Rahmen der vorliegenden Arbeit bei Weitem gesprengt. Die hier genannten weiteren Fragestellungen bleiben aber im Fokus und bilden eine ausgezeichnete Grundlage für weitere Forschungstätigkeiten.

# Quellen


AGPL. 2007: GNU Affero General Public License. Version 3, 19. November 2007. Online abrufbar unter: http://www.gnu.org/licenses/agpl-3.0-standalone.html

ACCORSI, R./LOWIS, L./SATO, Y. 2011: Automated Certification for Compliant Cloud-based Business Processes, in: Buhl, H. (Hrsg.): Business & Information Systems Engineering. Volume 3, Issue 3. Heidelberg: Springer Verlag, S. 145–154.

BARGMEYER, B. 2011: ISO-Standard JTC 1/SC 32 – Data management and interchange Published Standards and Standards under development. Online abrufbar unter: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_tc_browse.htm?commid=45342

BAUER, C./KING, G. 2007: Java Persistence mit Hibernate. München, Wien: Carl Hanser Verlag.

BERNERS LEE, T. 2010: W3C-Standards. Online abrufbar unter: http://www.w3.org/TR/

BESSANI A. ET AL. 2011: DepSky: dependable and secure storage in a cloud-of-clouds. In: Proceedings Of EuroSys '11 The Sixth Conference On Computer Systems. Salzburg, Österreich, 10–13. April 2011, S. 31–46.

BEZENEK, T. ET AL. 2000: Characterizing a Java Implementation of TPC- W. In: Proceedings Of 3rd Workshop On Computer Architecture Evaluation Using Commercial Workloads (CAECW), 9. Januar 2000, Toulouse, Frankreich.

BINZ T. ET AL. 2013: OpenTOSCA – A Runtime for TOSCA-based Cloud Applications. In: Proceedings Of The 11th International Conference On Service Oriented Computing (ICSOC), Berlin, Deutschland, 2.–5. Dezember 2013, S. 692–695.

BUYYA, R./CHEE S. Y./VENUGOPAL, S. 2008: Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities. In: Proceedings Of The 10th IEEE International Conference on High Performance Computing and Communications (HPCC 2008), 25.–27. September 2008, S. 5–13.

CORDTS, S/BLAKOWSKI G./BROSIUS G.: 2011: Datenbanken für Wirtschaftsinformatiker. Wiesbaden: Vieweg+Teubner Verlag, 1. Auflage.

DEAN, J./GHEMAWAT, S. 2004. MapReduce: Simplified Data Processing on Large Clusters. In: Proceedings Of The Sixth Symposium On Operating System Design And Implementation, San Francisco, USA, 5. Dezember, S. 1–13.

GANAPATHY, V. ET AL. 2011: Distributing Data for Secure Database Services. In: Proceedings Of The 4th International Workshop On Privacy and Anonymity In The Information Society, Uppsala, Schweden, 21.–24. März 2011.

HLIPALA, C. 2014. Konzeption und die Implementierung einer dynamischen Skalierung für die CloudStack Cloud. Bachelorarbeit am Institut für Unternehmensinformatik, Hochschule Mannheim. September 2014.

HUBER, M. ET AL. 2013: Cumulus4j: A Provably Secure Database Abstraction Layer. In: Proceedings Of The CD-ARES 2013 Workshops: MoCrySEn and SeCIHD, Regensburg, Deutschland, 2.–6. September, S. 180–193.

HYVÖNEN, E. 2010: Preventing ontology interoperability problems instead of solving them, in: Hitzler, P./Janowicz, K. (Hrsg.): Journal of Semantic Web. Volume 1, Issue 1/2, April 2010. Amsterdam, Niederlande: IOS Press, S. 33–37.

KAISER, L. 2013: Framework-Evaluation von Cloud Abstraktionsschichten zur verteilten Datenintegration. Bachelorarbeit am Institut für Unternehmensinformatik, Hochschule Mannheim. August 2013.

KERSCHBAUM F. 2011: Secure and Sustainable Benchmarking in Clouds in: Buhl, H. (Hrsg.): Business & Information Systems Engineering. Volume 3, Issue 3. Heidelberg: Springer Verlag, S. 135–143.

KOHLER, J. 2012: SeDiCo – Towards a Framework for a Secure and Distributed Cloud Datastore. In: Proceedings Of The Chip-To-Cloud Security Forum, Nizza, Frankreich, 18.–20. September 2012.

KOHLER, J./SPECHT, T. 2013: Marktplatz für eine transparente und providerübergreifende XaaS-Bewertung. In: Proceedings Of The AKWI – Arbeitskreis Wirtschaftsinformatik an Fachhochschulen, Friedberg, Deutschland, 16.–18. September 2013.

KOHLER, J./SPECHT, T. 2014: Vertical Update-Join Benchmark in a Cloud Database Environment. In: Proceedings Of WiWiTa 2014 Wismarer Wirtschaftsinformatiktage, Wismar, Deutschland, 12.–13. Juni 2014.

KOHLER, J./SPECHT, T. 2014A: Analyse der JOIN-Problematik in vertikal verteilten Datenbanken. In: Proceedings Of The AKWI – Arbeitskreis Wirtschaftsinformatik an Fachhochschulen, Regensburg, Deutschland, 16.–19. September 2014.

KOHLER, J./SPECHT, T. 2014B: Vertical Query-Join Benchmark in a Cloud Database Environment. In: Proceedings Of The 2nd World Conference On Complex Systems, Agadir, Marokko, 18.–21. November 2014, S. 143–150.

KRUEGER, J. ET AL. 2011: Fast updates on read- optimized databases using multicore CPUs. In: Proceesings Of The 38th International Conference On Very Large Databases, Istanbul, Türkei, 27.–31. August 2011, S. 61–72.

LOUTAS N. ET AL. 2010: Towards a Reference Architecture for Semantically Interoperable Clouds. In: Proceedings Of The IEEE 2nd International Conference On Cloud Computing Technology and Science (CloudCom). Indianapolis, USA, 30. Nov. – 3. Dez. 2010, S. 143–150.

MACVITTIE, L. ET AL. 2010: F5 Whitepaper – Controlling the Cloud:Requirements for Cloud Computing. Online abrufbar unter: http://www.f5.com/pdf/white-papers/controlling-the-cloud-wp.pdf

MAGED M. ET AL. 2007: Scale-up x Scale-out: A Case Study using Nutch/Lucene. In: Proceedings Of The Parallel And Distributed Processing Symposium. Long Beach, USA, 26.–30. März, S. 1–8.

MEIR-HUBER, M. 2010: Cloud Computing: Praxisratgeber und Einstiegsstrategien. Frankfurt a. M: entwickler.press, 1. Auflage.

MÜLLER, P. 2013: Vertikale Datenpartitionierung in einer verteilten Cloud-Architektur. Bachelorarbeit am Institut für Unternehmensinformatik. August 2013.

MÜLLER P./KOHLER J./SPECHT T. 2014: Ein Ansatz zur vertikalen Datenpartitionierung in der Cloud, in: Müller, C./Marfurt, K. (Hrsg.): eJournal of AKWI – Arbeitskreis Wirtschaftsinformatik. Februar 2014, Luzern, Schweiz. ISSN: 2296–4592. Online abrufbar unter: http://akwi.hswlu.ch

NGUYEN, B./TRAN V./HLUCHY L.: 2011: Abstraction Layer for Cloud Computing, in: Petcu D./Vazquez-Poletti J. (Hrsg.): Scalable Computing: Practice and Experience. Volume 12. Issue 3, S. 371–374. Online abrufbar unter: http://www.scpe.org/

NIST (NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY) 2013: Definition Cloud Computing. Online abrufbar unter: http://www.nist.gov/itl/cloud/

PLATTNER, H. 2013: A Course in In-Memory Data Management: The Inner Mechanics of In-Memory Databases. Berlin-Heidelberg: Springer-Verlag.

SCHIEFER, G. 2012: Wer liest alle meine Daten in der Wolke? OBJEKTspektrum Online-Themenspecial Cloud Computing. Online abrufbar unter: http://www.aifb.kit.edu/web/Article3082

SEGEV A.; PARK J. 1989: Maintaining materialized views in distributed databases. In: Proceedings Of The 5th International Conference On Data Engineering, Los Angeles, USA, 6.–10. Februar 1989, S. 262–270.

VAQUERO, L. M. ET AL. 2009: A break in the clouds: towards a cloud definition, in: Keshav S. (Hrsg.): ACM SIGCOMM Computer Communication Review. Volume 39, Issue 1. January 2009. ACM: New York, S. 50–56.

VECCHIOLA, C./DUNCAN D./BUYYA, R. 2010: The Structure of the New IT Frontier: Market Oriented Computing, in: Strategic Facilities Magazin 10:10, S. 59–66.

WITHANA, E./PALE B. 2011: Sigri: Uniform Abstraction for Large-Scale Compute Resource Interactions. Technical Report TR693, Indiana University Bloomington. 2011. Online abrufbar unter: http://www.cs.indiana.edu/cgi-bin/techreports/TRNNN.cgi?trnum=TR693

## Abkürzungen

| | |
|---|---|
| ACID | Atomicity, Consistency, Isolation und Durability |
| API | Application Programming Interface |
| BDSG | Bundesdatenschutzgesetz |
| CUDA | Compute Unified Device Architecture |
| DBMS | Datenbankmanagementsystem |
| ETSI | European Telecommunications Standards Institute |
| IaaS | Infrastructure as a Service |
| IT | Informationstechnologie |
| LDAP | Leightweight Directory Access Protocol |
| NIST | National Institute of Standards and Technology |
| NoSQL | Not only SQL |
| OASIS | Advancing Open Standards For The Information Society |
| OCCI | Open Cloud Computing Interface |
| PaaS | Platform as a Service |
| QoS | Quality of Service |
| RAID | Redundant Array of Independent Disks |
| SaaS | Software as a Service |
| SLA | Service-Level-Agreement |
| SQL | Structured Query Language |
| XaaS | Everything as a Service |

## Über die Autoren

JENS KOHLER arbeitet derzeit am Institut für Unternehmensinformatik als wissenschaftlicher Mitarbeiter. Er unterstützt die Lehre mit einem Seminar und diversen Veranstaltungen zu Algorithmen und Techniken der Programmentwicklung.

THOMAS SPECHT leitet das Institut für Unternehmensinformatik. Seine Lehrgebiete umfassen die objektorientierte Modellierung und Programmierung, Webarchitekturen, verteilte Softwaresysteme und Komponentensoftware.

## Über die MFG Stiftung Baden-Württemberg

MFG Stiftung Baden-Württemberg

Die gemeinnützige MFG Stiftung wurde 2003 als Geschäftsbereich der MFG Medien- und Filmgesellschaft Baden-Württemberg ins Leben gerufen. Stifter ist die Wirtschafts- und Clusterinitiative bwcon (Baden-Württemberg: Connected). Das Ziel der MFG Stiftung ist die Aus- und Weiterbildung sowie die Förderung von Kunst, Kreativität und Kultur. Dabei sind ihre Schwerpunkte die Forschung und Entwicklung in den Bereichen Medien, IT und Film. Die MFG Stiftung vergibt Stipendien, leitet Forschungsprogramme und beauftragt wissenschaftliche Studien.

Mehr Informationen finden Sie im Internet unter
www.stiftung.mfg.de

## Über die Reihe

Die Berichte aus dem Karl-Steinbuch-Forschungsprogramm präsentieren aktuelle Erkenntnisse und Lösungen an der Schnittstelle von Kreativwirtschaft und Informations- und Kommunikationstechnologie. Ziel der Reihe ist es, Forschungsergebnisse praxisorientiert aufzubereiten und damit neben Wissenschaftlern auch Anwenderbranchen anzusprechen.

Mit dem Karl-Steinbuch-Forschungsprogramm unterstützt die MFG Stiftung Baden-Württemberg seit 2011 besonders innovative Forschungsarbeiten an baden-württembergischen Hochschulen für Angewandte Wissenschaften. In fünf Ausschreibungsrunden (2011 – 2015) werden Projekte mit einer Laufzeit von bis zu zwei Jahren realisiert. Das Programm wird aus Mitteln der Zukunftsoffensive III des Landes Baden-Württemberg finanziert. Weitere Informationen und aktuelle Neuigkeiten finden sich im Internet unter www.stiftung.mfg.de.