# EIN ANSATZ ZUR VERTIKALEN DATENPARTITIONIERUNG IN DER CLOUD

Patrick Müller, Jens Kohler, Thomas Specht
Hochschule Mannheim
Paul-Wittsack-Straße 10, Mannheim, 68163, Deutschland
E-Mail: patrick@4-haen.de, j.kohler@hs-mannheim.de, t.specht@hs-mannheim.de

**STICHWORTE**

Cloud, vertikale Datenpartitionierung, verteilter JOIN

**KURZFASSUNG**

Cloud Computing als eines der aktuellen Hype-Themen erfährt durch aktuelle Datenschutzskandale, Spionageversuche und Sicherheitslücken enormen Gegenwind. 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. stundengenaue Abrechnung enorme Kostenvorteile. Diese Arbeit zeigt daher 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. Die Arbeit ist im Rahmen des SeDiCo-Projekts an der Hochschule Mannheim entstanden. Es wurde dabei nicht nur das Konzept erarbeitet, sondern auch die technische Machbarkeit mit einer prototypischen Implementierung nachgewiesen.

**EINLEITUNG**

Cloud Computing ist aktuell eines der am meisten beachteten Themen in der Weiterentwicklung der IT-Landschaft. Endverbraucher wie auch Firmen sehen eine schier unglaubliche Explosion an Cloud Computing-Anwendungen. Als Beispiele seien hier exemplarisch für „Software as a Service“ (SaaS) Salesforce, für „Platform as a Service“ (PaaS) Google AppEngine und für „Infrastructure as a Service“ IaaS Amazon EC2 genannt. Demgegenüber stehen wachsende Bedenken im Bereich Datenschutz und Datensicherheit aufgrund regelmäßig auftretender Sicherheitslücken und Spionageversuche staatlicher sowie privater Organisationen.

Um diesen Bedenken entgegenzutreten, entwickelt das von der MFG Stiftung geförderte Forschungsprojekt SeDiCo ein Framework zur sicheren und verteilten Datenspeicherung. SeDiCo steht dabei für „A **Se**cure and **Di**stributed **C**loud Datast**o**re“. Zur Erhöhung der Sicherheit wird im Rahmen dieses Projektes auf Datenbankebene ein Ansatz für die vertikale Partitionierung von Daten bei verschiedenen Cloud-Anbietern entwickelt. Sollte es einem Angreifer gelingen, eine der Partitionen zu kompromittieren, kann er damit nichts anfangen, da er nur einen Bruchteil der Daten hat.

Es ist wichtig die Verteilung der Daten nach Datenschutzaspekten vorzunehmen, damit beispielsweise Kreditkartennummer und Name eines Kunden in jedem Falle in zwei unterschiedlichen Partitionen liegen. Ideen zur sicheren Aufteilung der Daten werden ebenfalls von anderen Projekten verfolgt, vgl. (Ganapathy et al. 2011; Plattner 2013).

Durch eine Abstraktion verschiedener Cloud-Plattformen und Datenbankanbieter soll die Abhängigkeit von bestimmten Providern, der sogenannte „Vendor lock-in“, vermieden werden. Je leichter ein Wechsel zwischen verschiedenen Cloud-Diensten realisiert werden kann, desto häufiger wird diese Möglichkeit von den Benutzern der Dienste voraussichtlich auch genutzt.

Das zentrale Element des SeDiCo-Projektes ist ein Verteilungsalgorithmus zur vertikalen Partitionierung. Dieser wird benötigt, um Datensätze auf unterschiedliche Cloud-Anbieter zu verteilen. Ein solcher Verteilungsalgorithmus wird im Rahmen dieses Artikels von der Konzeption bis zur Implementierung aufgezeigt und schließlich bewertet.

**MOTIVATION**

Bisherige Ansätze implementieren vor allem die horizontale Verteilung (s. a. NoSQL) 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 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 Verteilungs-

strategien sei an dieser Stelle auf die Literatur verwiesen (Plattner 2013).
Unabhängig von der Partitionierungsstrategie haben die horizontale und die vertikale Partitionierung als gemeinsames Ziel die Aufteilung der Daten in disjunkte Teilmengen. Weiterhin müssen beim Lesen der Daten geeignete Join-Operationen implementiert sein, um die verteilten Daten wieder zusammenzuführen.

Im Gegensatz zur horizontalen 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“ sind die zwei Faktoren Performance und Sicherheit die wichtigsten Entscheidungskriterien für oder gegen einen Ansatz. Gerade in der logischen Trennung der Tupel liegt die Motivation des SeDiCo-Projekts mit der Konzeption und Implementierung eines sicheren Datenspeichers in der Cloud, begründet.
Im Projekt wird die technische Machbarkeit des Ansatzes nachgewiesen. Weitere Analysen bezüglich der Performance, insbesondere beim Join der Daten, würden allerdings den Rahmen des Projekts sprengen. Es ist aber geplant diese Fragestellung in einem Folgeprojekt anzugehen.

## PROBLEMSTELLUNG

Die Betrachtung der Problemstellung verläuft zweistufig. In der ersten Stufe werden verschiedene Ansätze zur Verteilung der Daten betrachtet. In der zweiten Stufe (Lösungsansatz) wird analysiert, wie sich die partitionierten Daten wieder zusammenfügen lassen. Dies ist insbesondere für bestehende Anwendungen (nachfolgend „Legacy Systeme“) ein zentraler Punkt, da SeDiCo den Anspruch erhebt, anwendbar zu sein, ohne Änderungen an bestehenden Systemen durchführen zu müssen. Die prototypische Implementierung fokussiert sich hierbei auf komponentenbasierte Java-Anwendungen.

### Vertikale Partitionierung

Für die Implementierung des Verteilungsalgorithmus kommen die fünf in Abbildung 1 gezeigten Schichten infrage.

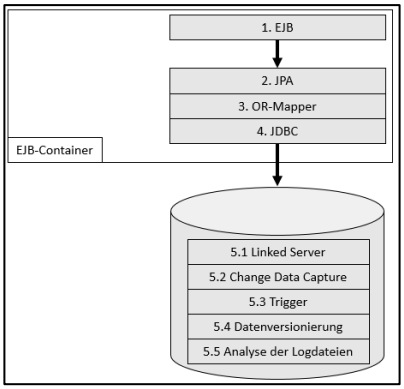


Abbildung 1: Schichten zur Verteilung

**Schicht 1: Enterprise JavaBeans (EJBs)**
In der ersten Schicht besteht die Möglichkeit EJBs so zu annotieren, dass die jeweiligen Properties in verschiedene Datenbanken geschrieben werden, um damit die Partitionierung zu realisieren. Dabei wäre jede Entitätsklasse in je eine Klasse pro Partition aufzuteilen. Der Vorteil dieses Ansatzes besteht vor allem in seiner Einfachheit. Damit einher geht jedoch der Nachteil, dass eine bestehende Anwendung stark angepasst werden müsste. Zusätzlich ist dieser Ansatz für einen Anwendungsprogrammierer nicht transparent ist, da die gesplitteten Klassen bei der Entwicklung immer sichtbar blieben.

**Schicht 2: Java Persistence API (JPA)**
Als zweite Variante wäre es möglich, die JPA so zu erweitern, dass die Daten vor dem Schreibvorgang in die Datenbank abgefangen und partitioniert werden. JPA-Implementierungen besitzen bereits einen SQL-Parser, daher ist davon auszugehen, dass eine Implementierung mit diesem Ansatz einen deutlich geringeren Aufwand bedeuten würde. Nachteilig ist jedoch, dass alle Zugriffe auf die Daten durch JPA erfolgen müssten. Zudem wäre die Nutzung von erweiterten Datenbankfunktionalitäten wie z. B. Stored Procedures oder Trigger mit dieser Lösung nicht möglich.

**Schicht 3: Object-Relational Mapper (OR-Mapper)**
Anstatt einer JPA-Erweiterung wird in Schicht 3 eine OR-Mapper-Erweiterung in Betracht gezogen. Der Ansatz bliebe praktisch derselbe, es könnten jedoch die spezifischen Programmierschnittstellen des gewählten OR-Mappers genutzt werden. Da bei dieser Lösung der Ansatz mit dem Ansatz aus Schicht 2 vergleichbar ist, sind die Vor- und Nachteile ebenso dieselben.

**Schicht 4: JDBC-Treiber**
Eine weitere Möglichkeit besteht darin, vorhandene JDBC-Treiber zu kapseln. Aufrufe an JDBC müssten dazu partitioniert und dann an die eigentlichen Treiber durchgereicht werden. Da die JDBC-Treiber-Implementierungen und die genutzten SQL-Dialekte verschiedener Hersteller unterschiedlich sind, müsste eine solche Implementierung jedoch für jeden SQL-Hersteller speziell angepasst werden, was auf einen sehr hohen Implementierungsaufwand hinauslaufen würde. Zudem müsste diese zusätzliche Schicht laufend Änderungen an den JDBC-Treibern berücksichtigen, was auf lange Sicht nicht praktikabel zu gewährleisen wäre.

**Schicht 5.1: Linked Server**
Ein Datenbankfeature ist das sog. „Linked Server“. Damit ist die Zusammenschaltung mehrerer Datenbanken gemeint, die dann in einem einzigen Query angesprochen werden können. Dieser Ansatz funktioniert allerdings nur innerhalb der jeweiligen Datenbanken und nicht datenbankübergreifend. Aus diesem Grund wird auch dieser Ansatz nicht weiter verfolgt.
Des Weiteren gibt es bei der Betrachtung der Datenbankebene vier weitere Ansätze, die sich, im Gegensatz

zum „Linked Server“-Feature, auf die Nachverfolgung von Änderungszugriffen beziehen. Diese werden nun nachfolgend kurz angerissen.

**Schicht 5.2**: **Change Data Capture (CDC)**
Einige Datenbanken besitzen eine eingebaute Funktionalität namens „CDC“. Dies ist eine Methode, um Änderungen an Tabellen festzustellen. Allerdings unterstützen nicht alle Datenbanken, insbesondere nicht die freien Datenbanken, diese Funktionalität. Daher wird diese Lösung im Folgenden nicht weiter betrachtet.

**Schicht 5.3: Trigger**
Eine weitere Möglichkeit ist es, die zu überwachende Tabelle mit je einem Trigger für Inserts, Updates und Deletes zu versehen. Im Trigger müsste dann der Primärschlüssel der geänderten Zeile in eine spezielle Trackingtabelle übernommen werden. Dazu kommt ein Timestamp, um die korrekte Reihenfolge der Änderungen sicherzustellen. Ein externer Prozess könnte in einem beliebigen Intervall die Tracking-Tabelle abfragen, die entsprechenden Zeilen aus der Quelltabelle laden und auf die Zielserver verteilen. Nachteilig ist bei diesem Ansatz, dass nicht jede (freie) Datenbank Trigger unterstützt und das die Änderungen nur zeitverzögert übernommen werden, was zu einer „eventuell konsistenten“ Datenbank führt, die nur noch die BASE-Bedingungen erfüllt (Pritchett 2008).

**Schicht 5.4: Datenversionierung**
Das Hinzufügen einer Spalte an die Quelltabelle wäre eine weitere mögliche Herangehensweise. Diese Spalte müsste entweder eine Ganzzahl enthalten, die mit jeder Änderung hochgezählt wird, oder einen Zeitstempel (Plattner 2013). Dieser Wert aus der Quelltabelle könnte dann ebenfalls regelmäßig mit der Zieltabelle abgeglichen werden. Der Nachteil ist in diesem Fall erneut der Verlust der ACID-Kriterien.

**Schicht 5.5: Analyse der Logdateien**
Als letzte Möglichkeit könnten die binären Logdateien eines SQL-Servers ausgelesen werden. Alle Änderungen müssten dazu aus der Logdatei extrahiert und auf den Zielservern erneut ausgeführt werden. Diesen Mechanismus verwenden einige Datenbanken zur Master-Slave-Replikation. Selbst Informationen wie z.B. Auto-Increment IDs finden sich in diesem Log wieder und könnten auf einem zweiten Server ausgeführt werden. Eine solche Implementierung müsste auf jeden Fall für jede Datenbank neu angepasst werden, da die exakte Form des Logging nicht standardisiert ist und potenziell mit jeder neuen Datenbankversion wechseln kann.

**Entscheidung für ein Konzept**
Tabelle 1 präsentiert eine Übersicht der Vor- und Nachteile der unterschiedlichen Konzepte. Die Schichten 5.1 sowie 5.2, die aufgrund ihrer fehlenden Praktikabilität bereits vorher ausgeschlossen wurden, werden nicht weiter betrachtet.

Die Konzepte sind in vier Kategorien bewertet worden. Die Kategorien sind:

1. Implementierungsaufwand: Wie groß ist der Implementierungsaufwand für diese Arbeit?
2. Eingriff in bestehende Anwendungen: Wie groß sind die Änderungen, die an einem Legacy System vorgenommen werden müssten?
3. Erweiterbarkeit: Kann auf eine Schemaänderung der Tabelle automatisch reagiert werden?
4. Verzögerung der Partitionen: Werden die Partition sofort (synchron) oder mit einer Verzögerung (asynchron) aktualisiert?

Tabelle 1: Vergleich der Partitionierungskonzepte

| | EJB | ORM | JDBC | Trig-ger | Ver-sion | Logs |
|---|---|---|---|---|---|---|
| 1. | 2 | 1 | 0 | 1 | 1 | 0 |
| 2. | 0 | 2 | 2 | 2 | 1 | 2 |
| 3. | 1 | 1 | 1 | 0 | 1 | 1 |
| 4. | 1 | 1 | 1 | 1 | 0 | 0 |
| **Σ** | **4** | **5** | **4** | **4** | **3** | **3** |

Der Bewertungsmaßstab ist grundsätzlich so gewählt worden, dass eine größere Zahl eine bessere Bewertung bedeutet. Ein kleiner Implementierungsaufwand ist demnach eine größere Zahl. Die Bewertungsskala für die ersten beiden Kategorien bewegt sich zwischen 0 und 2, um Abstufungen zu ermöglichen. Zusätzlich sind diese beiden Kategorien wichtiger einzustufen wie die beiden letzten Kategorien, da beispielsweise die Erweiterbarkeit eine nicht so wichtige Rolle spielt wie der Implementierungsaufwand. Um diese geringere Gewichtung auszudrücken, wurden die letzten beiden Kategorien entweder mit 0 oder 1 bewertet.

Aufgrund der Ergebnisse, wird die Variante ORM-Implementierung ausgewählt. Der Hauptgrund hierfür ist die zu erwartende geringste Komplexität ohne größere Einschränkungen in der Benutzbarkeit der Partitionierungsstrategie. Für die Implementierung wird eine Lösung auf Basis von JPA (Schicht 2) angestrebt. In Fällen, wo JPA Funktionalität fehlt wird der OR-Mapper direkt angesprochen. Als OR-Mapper und Implementierung von JPA wird Hibernate eingesetzt.

## LÖSUNGSANSATZ

Der hier präsentierte Lösungsansatz gliedert sich nun in zwei Schritte. Im ersten Schritt gilt es die vertikale Partitionierung zu implementieren. Danach wird mit dem vertikalen JOIN gezeigt, wie sich die Daten transparent auch wieder zusammenfügen lassen.

**Vertikale Partitionierung**
Die folgende Abbildung 2 zeigt den Ablauf eines Partitionierungsvorganges.

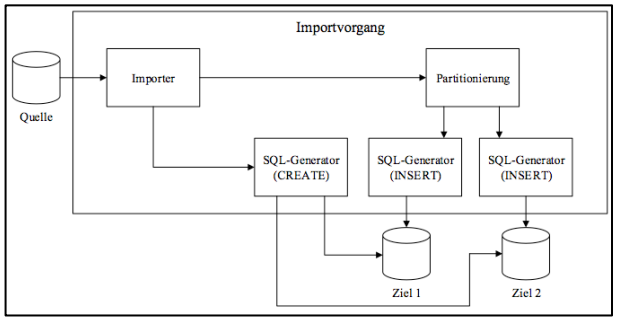


Abbildung 2: Partitionierung von Bestandsdaten

Zu Beginn werden alle bestehenden Datensätze aus der Quelldatenbank importiert. Danach werden die ausgelesenen Datensätze sowie deren Metadaten weitergereicht. Aus diesen wird dann ein Skript zur Erstellung der SQL-Tabellen auf den Zielservern generiert und anschließend ausgeführt. Der Partitionierungsalgorithmus generiert hieraus pro Partition eine neue SQL-Tabelle, die jeweils nur die Daten für die entsprechende Partition enthält. Im letzten Schritt werden aus den partitionierten Tabellen SQL-Skripte zum Einfügen (Insert) der Datensätze auf den Zielservern generiert. Mit der Ausführung der Skripte ist die initiale Übernahme der Daten in die Partitionen abgeschlossen.

Die Überwachung von neuen Datensätzen, die nach erfolgreicher initialer Datenpartitionierung in die entsprechenden Partitionen geschrieben (Insert) werden, ist durch sog. Hibernate Life-Cycle-Events realisiert. Diese Daten werden im Folgenden „Echtzeitdaten“ genannt. Der gleiche Mechanismus wird ebenso für das Ändern (Update) und das Löschen (Delete) der partitionierten Daten verwendet. Eine Änderung an einer Entität löst ein Ereignis aus, woraufhin die zu ändernden Daten eingelesen werden. Die darauffolgenden Schritte sind analog zur initialen Partitionierung, wobei je nach Ereignis neben INSERT-Statements auch DELETE- oder UPDATE-Statements (s. o.) generiert werden müssen. Abbildung 3 zeigt den Ablauf einer solchen Partitionierung.

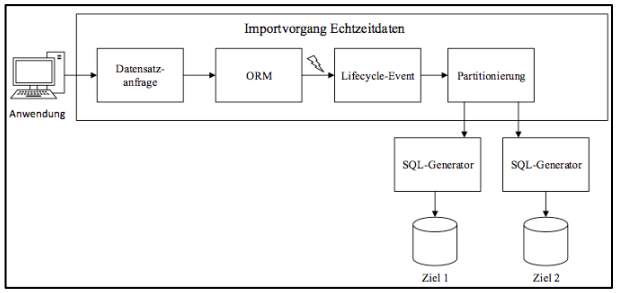


Abbildung 3: Partitionierung von Echtzeitdaten

**Vertikaler JOIN**

Das Zusammenfügen der Daten ist nun der zweite Schritt bei der Betrachtung der Problemstellung. Die aufgeteilten Daten lassen sich anhand ihres Primärschlüssels wieder zusammenfügen (JOIN). Dies ist dem MapReduce-Ansatz (Dean and Ghemawat 2004) sehr ähnlich, bei dem die disjunkten Teilmengen jeder Partition berechnet werden (Map) und anschließend auf die Lösungsmenge vereinigt (reduziert) werden (Reduce).

Die Zusammenführung der partitionierten Daten wird grundsätzlich durch den Primärschlüssel eines Datensatzes ermöglicht. Dieser wird in beiden Partitionen abgelegt und kann damit verwendet werden, um Daten aus unterschiedlichen Partitionen zu integrieren. Ähnlich wie bei der Partitionierung von Echtzeitdaten, wird das PreLoad-Event genutzt, um Ladeanforderung abzufangen und an die entsprechende Partition weiterzureichen. Das PreLoad-Event ist in der aktuellen Fassung von JPA nicht vorhanden, weshalb hier auf ein reines Hibernate Ereignis zurückgegriffen werden musste. Abbildung 4 stellt diesen Ansatz bildlich dar.

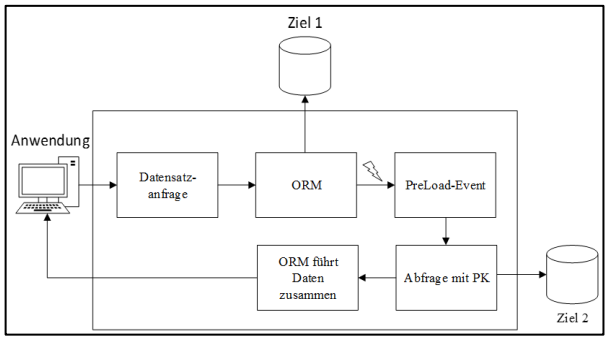


Abbildung 4: Abruf von Daten

## WEITERFÜHRENDE FRAGESTELLUNGEN

Der Einsatz des PreLoad-Events erfordert momentan ein bekanntes SELECT n+1 Problem (Bauer und King 2007). Durch die Erzeugung einer einzigen Query, die n Entitäten zurückgibt, werden erneut 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-Upates (Cordts et al. 2011; Bauer und 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 Unternehmens-

netzwerks 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 des Projekts beherrschbar zu machen, wurde die technische Machbarkeit hier für zwei Partitionen gezeigt. Die Erweiterung um 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 Performanceverlust dieses Ansatzes für große Datenmengen zu verkraften ist.
Gerade bei der Betrachtung der Performance wäre ein naheliegender Schritt 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.
Weitere Fragestellungen, die weitere erhebliche Forschungsarbeit benötigen werden zudem im Ausblick dieser Arbeit aufgegriffen.

## VERWANDTE ARBEITEN

Das hier vorgestellt Konzept und seine Implementierung basieren auf der Idee, die Nutzung von „Cloud Computing“ sicherer zu machen als bisher. 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) hat sich durchgesetzt, da dies die umfassendste und zugleich prägnanteste ist. U. a. werden in dieser Definition die drei Servicemodelle „SaaS“, „PaaS“ und „IaaS“ genannt. Das SeDiCo-Projekt und damit auch diese Arbeit bewegen sich dabei im „IaaS“-Bereich. Dies wurde aus Gründen der Anbieterunabhängigkeit so gewählt, da auf dieser untersten Ebene die Kapselung der verschiedenen Anbieterschnittstellen in ein Abstraktionsframework am einfachsten zu realisieren ist. Dies liegt in der Heterogenität der Dienste in den oberen Schichten des Servicemodells („SaaS“ und „PaaS“) begründet. Durch Virtualisierung lassen sich im „IaaS“-Modell Ressourcen dynamisch zu- und wieder abschalten (Meir-Huber 2010). Dies ermöglicht ein einfaches scale-out aber auch ein einfaches scale-up der Ressourcen (Maget et al. 2007). Im Gegensatz dazu bieten die anderen beiden Servicemodelle diese Möglichkeit nicht, da sie keinen direkten Zugriff auf die zugrundeliegenden Ressourcen ermöglichen.
Mit dem Konzept der Datenverteilung beschäftigt sich auch das MimoSecco Projekt, das allerdings die Entwicklung einer Middleware zum Ziel hat. Damit bewegt sich dieses Projekt 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 2013). Andere Arbeiten in diesem Kontext beschäftigen sich mit der Skalierbarkeit von Datenbanken. So wird im Leads-Projekt 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. Ausblick) gibt dieses Projekt sehr nützliche Hinweise und Impulse für die weitere Arbeit im SeDiCo-Projekt.

## ZUSAMMENFASSUNG UND AUSBLICK

Mit dieser Arbeit wurde die technische Machbarkeit der vertikalen Datenverteilung nachgewiesen. Damit die Komplexität der Verteilung im Rahmen des SeDiCo-Projekts beherrschbar bleibt, wurde dieses prototypisch anhand einer Tabelle mit fiktiven Kundendaten und eines Primärschlüssels gezeigt. Verschiedene Ansätze für die Verteilung wurden in die Überlegungen miteinbezogen. Des Weiteren wurden die Vor- und Nachteile der entsprechenden Ansätze aufgezeigt. Diese Analyse brachte zum Vorschein, dass die Verteilung mit Hibernate und den zugehörigen Listenern am besten zu realisieren ist. Dies bezieht sich zum einen auf die einfache Integration in bestehende Infrastrukturen und zum anderen auf das nachfolgende Zusammenführen (vertikaler JOIN) beim Abruf der Daten. Im Folgenden werden nun weitere Herausforderungen genannt, die den Rahmen des SeDiCo-Projekts sprengen würden. Sie geben jedoch wichtige Anhaltspunkte und damit die Richtung für künftige nachfolgende Forschungsprojekte vor.

Diese Arbeit konzentrierte sich auf das Kernthema des SeDiCo-Projekts: die vertikale Datenverteilung. Neben diesem zentralen Aspekt steht die Abstraktion verschiedener Cloud-Schnittstellen im Vordergrund. Um einen einheitlichen Zugriff zu unterschiedlichen Cloud-APIs zu gewährleisten, wurde das jclouds Framework benutzt. Nach voriger Evaluation anderer Abstraktionsframeworks wie Libcloud und Deltacloud stellt sich jclouds als das am besten für das Projekt geeignete Framework dar. Dabei ist wichtig, die Daten nicht nur zu verteilen, sondern sie auch verteilt, bei unterschiedlichen Anbietern zu speichern.

Über diese Arbeit hinausgehende Fragestellungen beschäftigen sich nun mit der vertikalen Verteilung von mehreren Tabellen, die über Fremdschlüsselbeziehungen verknüpft sind. Weiterhin werfen Tabellen mit zusammengesetzten Schlüsseln (sog. Compound Keys) weitere Fragestellungen auf. Zuletzt müssen auf Datenbankebene die ACID-Kriterien (Haerder and Reuter

1983) betrachtet werden. Gerade in verteilten Datenbanken ist die Transaktionssicherheit nicht ohne weiteres zu gewährleisten, da Techniken wie z. B. das 2-Phasen-Freigabeprotokoll mit dem zugehörigen Locking-Mechanismus immer ein Abwägen zwischen Performance und Konsistenz bedeutet. Überdies ist die künftige Entwicklung von NoSQL-Datenbanken nicht zu vernachlässigen. Gerade NoSQL-Datenbanken sind prädestiniert für die Verteilung der Daten über mehrere Knoten. Allerdings wird dabei aus Performancegründen die horizontale Verteilung (s. a. MapReduce) wegen dem besseren scale-out genutzt. Es wird also weiterhin zu untersuchen sein, ob sich die hier vorgeschlagene vertikale mit der horizontalen Partitionierung in einer Art und Weise verknüpfen lässt, um die Vorteile beider Partitionierungsarten, also eine hohe Performance und eine erhöhte Sicherheit, zu erreichen. Mit Blick auf den momentanen Trend zu „Big Data“ und dem Versuch diesen mit In-Memory Datenbanken beherrschbar zu machen, eröffnen sich hier weitere Forschungsthemen, die den Rahmen eines einzelnen Projekts bei Weitem sprengen würden.

## LITERATUR


Bauer, C., King, G. 2007. “Java Persistence mit Hibernate”. Carl Hanser Verlag. München, Wien. S. 473 ff.

Cordts, S. et al. 2011. “Datenbanken für Wirtschaftsinformatiker”. Vieweg+Teubner Verlag. 1. Auflage. S. 242.

Dean, J., Ghemawat, S. 2004. “MapReduce: Simplified Data Processing on Large Clusters”. OSDI'04: Sixth Symposium on Operating System Design and Implementation. San Francisco, CA. December.

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. New York, USA.

Haerder, T. and Reuter, A. 1983. “Principles of Transaction-Oriented Database Recovery”. In: ACM Computing Surveys. Volume 15, Issue 4. S. 287-317.

Maged M. et al. 2007. “Scale-up x Scale-out: A Case Study using Nutch/Lucene”. In: Parallel and Distributed Processing Symposium. Long Beach, CA.

Meir-Huber, M. 2010. „Cloud Computing: Praxisratgeber und Einstiegsstrategien”. 1. Auflage. entwickler.press. S. 10 ff.

Plattner, H. 2013. “A Course in In-Memory Data Manage ment: The Inner Mechanics of In-Memory Databases”. Springer. S. 63 ff.

Pritchett, D. 2008. “BASE: An Acid Alternative”. In: ACM Magazine Queue - Object-Relational Mapping. Volume 6, Issue 3, May/June. S. 48-55.

Schiefer, G. 2012. „Wer liest alle meine Daten in der Wolke?” OBJEKTspektrum Online-Themenspecial Cloud Computing. http://www.aifb.kit.edu/web/Article3082 [Zugriff: 28.10.2013]


## AUTHOR BIOGRAPHIES


**PATRICK MÜLLER** studierte Informatik an der Hochschule Mannheim. Er hat seine Bachelorarbeit zum Thema vertikal-verteilte Datenbanken am Institut für Unternehmensinformatik an der Hochschule in Mannheim sehr erfolgreich abgeschlossen. Diese Arbeit ist maßgeblich in das SeDiCo-Projekt von Prof. Dr. Thomas Specht eingeflossen. Derzeit arbeitet er bei einem mittelständischen Unternehmen als Softwareentwickler und Berater. Seine E-Mail Adresse lautet: patrick@4-haen.de.

**JENS KOHLER** arbeitet derzeit am Institut für Unternehmensinformatik als wissenschaftlicher Mitarbeiter. Neben seiner Tätigkeit als Mitarbeiter im SeDiCo-Projekt unterstützt er die Lehre an der Fakultät für Informatik mit einem Seminar zum wissenschaftlichen Arbeiten und diversen Veranstaltungen in den Grundlagenfächern „Grundlagen der Informatik“, „Algorithmen und Datenstrukturen“ sowie „Techniken der Programmentwicklung“. Seine E-Mail Adresse lautet: j.kohler@hs-mannheim.de. Seine Webseite finden Sie unter: http://www.informatik.hs-mannheim.de/~j.kohler.

**THOMAS SPECHT** leitet das Institut für Unternehmensinformatik an der Fakultät für Informatik der Hochschule Mannheim. Er promovierte mit einer Arbeit zum „frameworkbasierten Engineering multimedialer Online-Dienste“. Stationen seiner wissenschaftlichen Karriere waren das KIT in Karlsruhe (damals noch Forschungszentrum Karlsruhe), das Fraunhofer Institut für Arbeitswissenschaft und Organisation (IAO), die Duale Hochschule in Stuttgart und letztlich die Hochschule Mannheim. Seine derzeitigen Lehrgebiete umfassen die objektorientierte Modellierung und Programmierung, Webarchitekturen, -technologien und -frameworks, verteilte Softwaresysteme und Komponentensoftware. Seine E-mail Adresse lautet: t.specht@hs-mannheim.de.