Bulgarian Academy of Sciences
Institute of Information and Communication Technologies

Jens Kohler

Optimizing Query Strategies in Fixed Vertical Partitioned and Distributed Databases and their Application in Semantic Web Databases

Author's Summary of the Dissertation

Doctoral Program: Informatics

Professional Area: 4.6 Informatics and Computer Science

Supervisor: Prof. Dr. Kiril Simov

Sofia, 2017

Table of Contents

Introduction

Importance of the Topic

Storing data in relational databases has a long history since Codd defined the relational model and its normal forms in (Codd, 1970). Such relational databases still build the foundation for various applications throughout all application domains even with todays growing data volumes. It is assumed that, despite a rapid dissemination of In-Memory or NoSQL databases, relational databases will keep their important role. Hence, also relational databases are used as a foundation to store huge volumes of data and this is exactly where Cloud Computing offers dynamic and scalable capabilities. Renting such technological assets and capabilities from external cloud providers is an interesting approach. The pay-as-you-go character of these cloud offers, promise the usage of computing assets without large initial investments. In Cloud Computing environments, dedicated services are used for a certain time and are paid only for the respective usage. Moreover, as there are dedicated services, the complexity to integrate and use them is considered lower compared to paradigms like service-oriented architectures. As there are still open **data security** and **data protection** challenges, the usage of especially public Cloud Computing is far behind the expectations of e.g. Gartner (Carlton, 2013) and IDC (Gens & Shirer, 2013). Hence in this thesis, **data security** and **data protection** challenges for **relational databases** are addressed with the definition and an implementation of a framework for a ***SE*cure and *DI*stributed *C*loud Data St*O*re**, exploiting a *fixed vertical partitioning and distribution* (FVPD) scheme. ***The main contribution of this work is to show that the proposed framework provides comparable response times to non-partitioned relational databases using cloud infrastructures and contemporary hardware devices.***

An approach that contributes to the broad dissemination of using especially public clouds is *SeDiCo*, a framework for a *SE*cure and *DI*stributed *C*loud Data st*O*re. The key concept of this approach is to vertically partition relational database data and store the respective partitions in different databases operated in different clouds. The author of this work firstly proposed this so-called *Security-by-Distribution* concept in 2012 (Kohler & Specht, 2012) and developed and

implemented it prototypically from 2012 to 2014 (Kohler & Specht, 2014a)[1]. Although these works proved the technological feasibility, the approach still suffers from severe performance problems when the partitioned and distributed data are accessed. These performance issues are in the focus of this thesis, which aims at investigating, developing and evaluating new ways of accessing those data. In order to not exceed the limits of this work, this thesis focuses on the query response time. On the one hand, recent analyses of the author show that the insert, update, and delete operations are also affected (Kohler & Specht, 2014b) (Kohler & Specht, 2014c). On the other hand, (Krueger et al., 2010) showed that $\sim$ 90% of all operations in enterprise databases are *queries* (i.e. *selects*). Hence, the focus of this work is on the query response time and the insert, update and delete performance are considered as key questions of future work tasks. Finally, it can be stated that the usage of Cloud Computing capabilities still are a weighing between security and performance and this thesis aims at minimizing this gap with the definition and the evaluation of adequate query patterns.

A motivating example of the entire *SeDiCo* framework is drawn in Fig. 1 which illustrates the *fixed vertical partitioning and distribution* (FVPD) approach with a simple *CUSTOMER* relation.

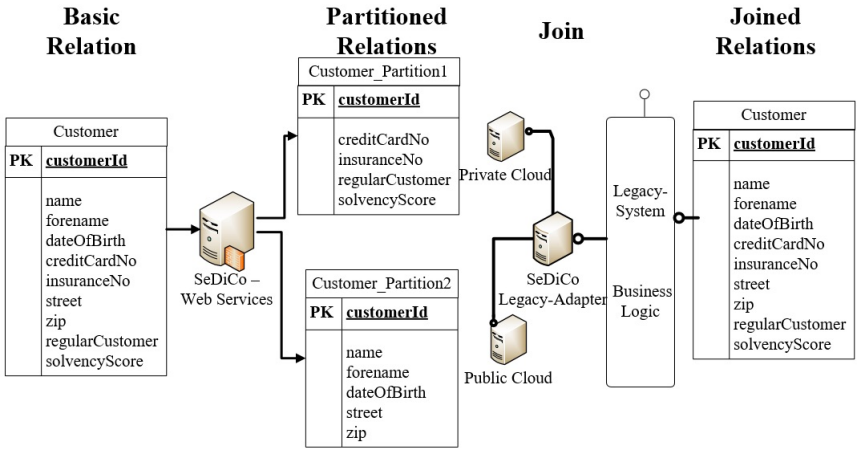

Figure 1: Motivating *SeDiCo* Example

In this example, there are 2 vertical partitions one containing more sensitive data (*Customer_Partition1*) and less sensitive data (*Customer_Partition2*). The basic idea is now that an intruder (e.g. to the public cloud partition) is not able to reconstruct entire *CUSTOMER* rows, since the partitioning and distribution

[1]In close cooperation with the theses supervised by the author listed at the end of the full version of the thesis. *SeDiCo* is available under GPL-License at: http://github.com/jenskohler

scheme is unknown to him. Therefore, it is of minor importance which data are stored in which cloud (public, private, community, hybrid) respectively.

Overview of the Main Results in the Area

The presented version of *SeDiCo* (cf. Chapter 2) was developed and implemented before the work on the thesis has started. The results of this preliminary work have shown that the ideas behind *SeDiCo* are feasible and work in practice. However, there is still an open question regarding the framework performance in practical use cases:

- **Performance optimization of the *SeDiCo* approach:** Although the feasibility of the original implementation is empirically shown and formally proved, the response time (especially for larger data sets, i.e. more than 10K rows) decreased tremendously. Thus, how can the response time for a FVPD query in practical use cases scenarios be improved, such that it is in the same order of magnitude as a non-FVPD query?

To the best of the author's knowledge, no one has followed a vertical database partitioning approach in the context of data security and privacy yet. Hence, this thesis conceptualizes, implements and evaluates advanced query mechanisms in order to improve the overall response time of SeDiCo.

Current figures of the initial implementation can be found in the author's previously published work, e.g. (Kohler, Simov, & Specht, 2015) (Kohler & Specht, 2014b) and in Section 6.3.

All in all, this thesis uses these figures as a basic performance metric and compares the investigated advanced query mechanisms to it.

Previous works (e.g. (Son & Kim, 2004) (Grund et al., 2011)) on vertical database partitioning have been conducted in the context of performance optimization tasks. These optimization approaches are workload driven, i.e. the approach depends on the queries issued against the database. *SeDiCo* is different as it follows a *fixed* vertical partitioning approach.

Another interesting field of research with respect to the vertical partitioning and distribution approach is *Cloud Computing*.

With respect to this, the thesis with its FVPD approach proposes the possibility to partition and distribute data, such that each of a certain amount of different

cloud providers only gets a logically independent data chunk, which is not usable without the others. Thus, the FVPD approach fosters the usage of (possibly untrustworthy public) Cloud Computing, which is a promising alternative to huge investments in IT infrastructures.

Goals and Tasks of the Thesis

The response time evaluation of the initial *SeDiCo* implementation (Kohler & Specht, 2014b) and (Kohler & Specht, 2014c) showed that there is a tremendous performance loss (factor ∼460 considering the average response time) with the vertical partitioning and distribution approach. However, with an advanced level of data security and privacy (Kohler & Specht, 2015a), this approach enables the usage of public cloud infrastructures. This shows that the *SeDiCo* approach is still a weighing between security and performance.

Hence, the objectives of this thesis are finding strategies, concepts and corresponding implementations to improve the response time to a level that it is in the same order of magnitude as a non-partitioned and non-distributed approach. This results in a minimization problem of the required time to retrieve the result set of a certain query that is issued against **fixed vertically partitioned and distributed** ***(FVPD) data.***

With respect to this, the hypotheses that are investigated can be formulated as follows:

Hypothesis 0: The definition of a Fixed Vertically Partitioned Schema (FVPD) for relational databases improves the level of data security and data protection by separating (i.e. partitioning) and distributing logically coherent data to different storage locations.

Hypothesis 1: Query Rewriting improves the response time to a level that is in the same order of magnitude as a non-partitioned and non-distributed scenario due to partitioned and parallelized query and join implementations.

Hypothesis 2: Caching data improves the response time to a level that is in the same order of magnitude as a non-partitioned and non-distributed scenario due to the usage of In-Memory caches.

Hypothesis 3: Using Solid State Disks (SSDs) as distributed secondary storage devices for the FVPD data improves the response time to a level that is

in the same order of magnitude as a non-partitioned and non-distributed scenario due to faster access times of the memory.

Based on the hypotheses, the following tasks are conducted:

Task 1: the definition of a methodology for creating an FVPD schema for relational data and a proof of the correctness of the methodology;

Task 2: the conceptualization of adequate query mechanisms for relational FVPD data sets;

Task 3: the implementation of these relational query mechanisms in Java;

Task 4: the evaluation of these relational query mechanisms in terms of their response time;

Task 5: the comparison of all developed relational query mechanisms against each other and against the initial *SeDiCo* implementation;

Task 6: the application of the FVPD methodology in the Semantic Web with Resource Description Framework-based (RDF-based) data.

The expected results can be subsumed as follows:

Result 1: a formal correctness proof of the FVPD methodology;

Result 2: ready-to-use FVPD query execution methods;

Result 3: an evaluation of the query mechanisms that acts as a guideline for their concrete application in different scenarios;

Result 4: a classification of the query mechanisms, which ones are applicable in which scenarios;

Result 5: a conceptual transfer of the relational FVPD approach to other application domains (i.e. the Semantic Web with RDF-based data) to emphasize the generic character of the approach;

Result 6: a demonstration of how the entire *SeDiCo* approach can be applied in the Semantic Web on RDF-based data.

Contributions of the Thesis

With the successful implementation and evaluation of the before-mentioned tasks, the thesis contributes to the current state-of-the-art with the following aspects.

Contribution 1: Definition of a *Security-by-Distribution* Principle for Relational Databases

In this thesis, there is a *Security-by-Distribution* principle introduced that uses vertical relational database partitioning to logically separate database tables into chunks that are worthless without the others, but can be joined based on the containing primary key. This principle is used in the so-called *SeDiCo* framework. The respective chunks are distributed (ideally) across different clouds and only the user who partitioned and distributed the rows knows the partitioning distribution scheme of the partitions (chunks). This increases the level of security and privacy and enables the storage of data in especially public cloud infrastructures.

Contribution 2: Development of FVPD Query Strategies

The previously mentioned *Security-by-Distribution* approach requires new ways of accessing the partitioned and distributed rows, as they have to be joined, i.e. entirely reconstructed before they are actually accessible. All approaches are conceptualized, implemented and evaluated in the presented thesis.

Contribution 3: FVPD Query Strategy Integration into the *SeDiCo* Framework

This thesis is created in the context of the *SeDiCo* framework development. As a further result, the approaches conceptualized and illustrated in this thesis are implemented and positively evaluated ones are integrated into the framework. This will develop the entire framework to a feasible opportunity in practical usage scenarios, which will allow further performance analyses in various application domains, where relational databases build the foundation for applications.

Contribution 4: FVPD Performance Evaluation and Classification

The developed query mechanisms are evaluated with respect to their response time and compared to each other to provide a short but precise

overview about all investigated approaches and their respective response time.

Contribution 5: Transfer of the FVPD Methodology to other Databases

Here, the entire *SeDiCo* approach is transferred to a Semantic Web scenario, based on the *resource description framework* (RDF). Firstly, this demonstrates the universal application character of the basic approach[2] and secondly, it proves that the approach can be transferred and applied to other application domains with a clearly stated and demonstrated integration effort.

Methodology Used for the Research

In order to answer the research question, the *Design Science Research* (DSR) methodology described by Hevner et al. is used (Hevner & Chatterjee, 2010). The aim is to *extend boundaries of human and organizatorial capabilities by creating new and innovative artifacts* (Hevner et al., 2004).

The entire *SeDiCo* framework development and its associated research work are aligned to this DSR Cycles. To illustrate this in more detail, Figure 2 maps the DSR Cycles to the presented thesis.

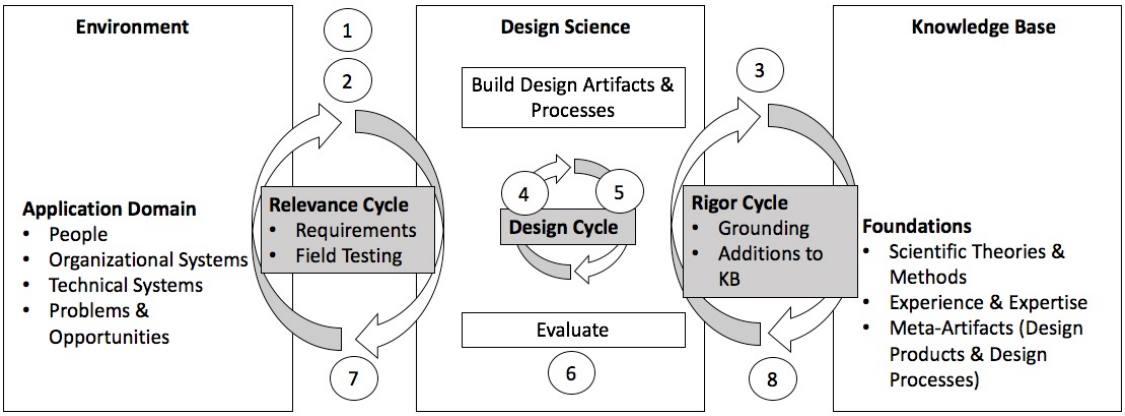

Figure 2: Design Science Research Cycle Mapped to Thesis Chapters

[2]other thinkable application scenarios could involve NoSQL datastores with its four fundamental architectures (column, document, key-value stores and graph databases)

Chapter 1

Problem Definition

This chapter states the formal definitions of central notions and general concepts and their adaptions to the context of the presented work. A more detailed description can be found in the full version of the thesis.

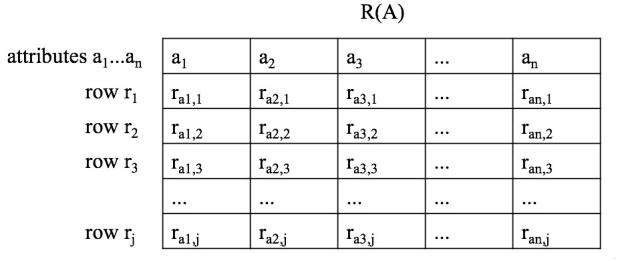

Figure 1.1: Relational Model

In Figure 1.1 the first *row* is the *header of the table* containing the attribute names. The degree of the table is n. The *cardinality* of the relation is j. Each cell r_{kl} has an *attribute value* for the attribute k in row l with $1 \leq k \leq n$ and $1 \leq l \leq j$.

In order to uniquely identify a certain row r_l, there is the concept of a *primary key*. A *primary key* A_k is a set of one or more attributes ($A_k \subseteq A$), such that the attribute values for the attributes in A_k are unique for every row r in $R(A)$. For the sake of better readability, this thesis focuses on relations with a primary key containing just one attribute[1].

In order to access data in a relational database, different operators over the attributes (i.e. *projection*) and rows (i.e. *selection*) are performed.

1.1 Selection

Let $A = \{a_1, a_2, \ldots, a_n\}$ be a set of attributes and $R(A)$ be a relation. A *selection* operator determines which *rows* meet the criteria φ and which are therefore collected into a *result set* (depicted as $\leftarrow$). *Rows* that do not meet

[1]This is not a loss of generality because the algorithms presented in the thesis can be extended to relations with primary keys that consist of more than one attribute.

these criteria are omitted. The following *selection* collects all *rows* in a *result set* RS which meet the selection criteria ω formulated over the relation attributes $\varphi := (a_i = \omega_i, ..., a_j = \omega_j)$ which is issued against a relation $R(A)$:

$$RS \leftarrow \sigma_{(a_i=\omega_i,...,a_j=\omega_j)} R(A) \tag{1.1}$$

with a_i as the ith attribute of relation $R(A)$, and $1 \leq i \leq j \leq n$.

1.2 Projection

The next operator relevant for the thesis is a *projection* Π over a relation $R(A)$. Let $A = \{a_1, a_2, \ldots, a_n\}$ be a set of attributes and $R(A)$ be a relation. A *projection* is essential for accessing *rows* in a *relation*, as it specifies which *attributes* of the *relation* are collected in the *result set*. Thus, it can be noted that in contrast to the above-mentioned *selection*, a *projection* results in a *vertical* subset of a *relation* (Elmasri & Navathe, 2015).

The following *projection* Π collects all *rows* in a *result set* RS that meet the attribute list $(a_i, ...a_j)$ which is issued against a relation $R(A)$:

$$RS \leftarrow \Pi_{(a_i,...,a_j)} R(A) \tag{1.2}$$

with a_i as the ith attribute of relation $R(A)$, with $1 \leq i \leq j \leq n$. Thus, since not all attributes are included in this *projection*, only attributes $a_i, ..., a_j$ are collected in the *result set* RS and by definition (Codd, 1970) duplicate rows are removed from the result set.

1.3 Join

The *join* operator $\bowtie$ (with Θ as the join condition[2] allows the combination of *relations* in a sense that each row from a relation R is joined with a corresponding row in relation S. Hence, a *join* $\bowtie$ is defined as[3]

[2] e.g. depending on which condition is used for Θ (possible are: $=, \neq, <, >, \leq, \geq$)
[3] a more detailed derivation can be found in the full version of the thesis

$$R \bowtie_{a_1=b_1} S := \{(r_{a_1,k}, ..., r_{a_i,k}) \oplus (s_{b_1,l}, ..., s_{b_m,l}) \mid R(r_{a_1,k}, ..., r_{a_i,k}) \wedge S(s_{b_1,l}, ..., s_{b_m,l}) \wedge (r_{a_1,k}, ..., r_{a_i,k}) \; and \; (s_{b_1,l}, ..., s_{b_m,l}) \; meet \; (r_{a_1,k} = s_{b_1,l})\} \quad (1.3)$$

Here, the $\oplus$ denotes a special case of a concatenation of the rows in R and S: based on the equality of the *primary key attributes* a_1 and b_1 respectively, the rows in the relations R and S are merged together.

Compared to Equation (1.3) a compact notation for the FVPD (natural) join can be reached if all attributes (that are not important for the join condition) are omitted and its results are given in the following definition:

$$R \bowtie_{a_1} S := \{(R) \oplus (S)\} \quad (1.4)$$

1.4 Problem Formulation

The key approach of this thesis is to create vertical partitions of a relation $R(A)$ and to distribute them across different clouds. For reasons of clarity, the FVPD approach described in this thesis focuses on two vertical partitions $S_v(B)$ and $T_v(C)$. Based on this, the *response time* of this vertical partitioning approach (FVPD) is evaluated.

Hence, the research problem of this thesis can be summed up with finding adequate query strategies that improve the overall response time to a level that is in the same order of magnitude as a query against a non-partitioned and non-distributed database setup. A formal definition of this is a *minimization problem* of the time t required to generate the joint result set based on FVPD relations $S_v(B)$ and $T_v(C)$. This can be stated as follows:

$$min_t(RS_{v\_query_1}S_v(B) \bowtie_{a_1} RS_{v\_query_2}T_v(C))$$

With respect to this minimization problem, a lower bound[4] is the time t_{lower}, required to collect the same result set with a non-partitioned relation $R(A)$:

$$t_{lower} = RS_{query}(R(A))$$

An upper bound for the *response time* is determined by the FVPD *query* itself:

$$t_{upper} \geq (RS_{v\_query_1} S_v(B) \bowtie_{a_1} RS_{v\_query_2} T_v(C))$$

The lower and the upper bounds are determined by the time complexity of the FVPD approach. The dominant factor here is the *join* of the FVPD relations as defined in Section 1.3. Therefore, in a naïve approach, this *join* results in the Cartesian Product of the FVPD relations[5], which yields to an upper bound of $\mathcal{O}(n^2)$ with n as the number of rows in the relations and the exponent indicating the number of relations. An analogous consideration can also be made for the lower bound. Again, with the *join* as the predominant factor for the performance of the entire FVPD approach, more sophisticated *join* algorithms are worth considering. A theoretical lower bound with approaches like the *Hash Join* is $\mathcal{O}(n + m)$ with n as building a hash table of all n rows in relation R and m as hashing the corresponding rows of relation S against the hash table[6]. The lower bound for the *Sorted-Merge Join* can be determined as $\mathcal{O}(n * log(m))$ with n as sorting all n rows in relation R and merging m rows of relation S against this sorted list[7]. Above all, it is assumed that a query against an FVPD data set cannot be faster than the same query against a non-FVPD data set (i.e. a single database relation). This results in an absolute lower bound of $\mathcal{O}(n)$, which can be stated as a database query against a relation containing n rows and all these n rows are collected in the result set.

[4]note that the complexity is (in the best case) $\mathcal{O}(n)$ with n as the number of rows in R, e.g. if relation R is stored completely in an In-Memory cache,

[5]except that the primary key a_1 is not replicated in the result

[6]note that n and m denote the cardinality of relations R and S and therefore it follows that $n = m$

[7]note that n and m denote the cardinality of relations R and S and therefore it follows that $n = m$

Both, the lower and the upper bounds could also be determined in concrete figures in experimental setups in (Kohler & Specht, 2014b) for different numbers of rows, ranging from 0 to 288K rows per relation.

Chapter 2

Definition of the FVPD Methodology and its Original Implementation in the *SeDiCo* Framework

The main idea is that a *relation* is divided in several *partitions* in a way, such that each individual *partition* contains logically independent (i.e. irrelevant) *tuples*. Thus, in order to use FVPD data, they have to be *joined* first and this requires mechanisms to separate a *relation* in several parts and strategies to query the respective *partitions*, such that the *join* produces an equal result set compared to the original query over the original *relation*. In the rest of the thesis it is assumed (without the loss of generality) that the original *relation* contains one single *primary key attribute* and that its FVPD *partitions* as *two relations* satisfy the necessary and sufficient conditions to represent the original *relation*. The presented results of the thesis are also correct for *relations* with more than one *primary key attribute* and more than *two FVPD partitions*.

2.1 Fixed Vertical Partitioning and Distribution (FVPD) Definition

Definition 1. *Non-FVPD relation*

Let $A = \{a_1, a_2, \ldots, a_n\}$ be a set of attributes. Let $R(A)$ be a relation R with attributes A such that a_1 is the only key attribute for $R(A)$. Then the relation R is called a **Non-FVPD relation**. □

Definition 2. *FVPD relations for the non-FVPD relation $R(A)$*

Let $A = \{a_1, a_2, \ldots, a_n\}$ be a set of attributes and let R be a non-FVPD relation with attributes A. Let B and C be two sets of attributes such that:

- $B \cup C = A$,

  and

- $B \cap C = \{a_1\}$

Then, the two relations $S_v(B)$ and $T_v(C)$ are **FVPD relations** *for the non-FVPD relation $R(A)$, if and only if*

- $|S_v(B)| = |T_v(C)| = |R(A)|$

  and

- $R(A) = S_v(B) \bowtie_{a_1} T_v(C)$. □

The condition $B \cap C = \{a_1\}$ is called **disjointness** criterion, because the sets of attributes in the partitions B and C are disjoint expect for the primary key attribute a_1. The condition $|S_v(B)| = |T_v(C)| = |R(A)|$ is called **completeness** criterion, because there are one-to-one correspondences from sets of tuples in $S_v(B)$ to the set of tuples in $T_v(C)$ on the basis of the value of the primary key attribute a_1.

Definition 3. *Reconstruction queries*

Let $A = \{a_1, a_2, \ldots, a_n\}$ be a set of attributes and let R be a non-FVPD relation with attributes A. Let $S_v(B)$ and $T_v(C)$ be FVPD relations for relation $R(A)$.

Let $\Pi_{(a_i,\ldots,a_j)}, (1 \leq i < j \leq n)$ be a projection query for $R(A)$, such that

$RS \leftarrow \Pi_{(a_i,\ldots,a_j)} R(A)$.

Let $\Pi_{v_1(a_1,a_k,\ldots,a_l)}$ be a projection query for $S_v(B)$ and let $\Pi_{v_2(a_1,a_m,\ldots,a_o)}$ be a projection query for $T_v(C)$ with $1 \leq i \leq k, l, m, o \leq j \leq n$, such that

$RS_{v1} \leftarrow \Pi_{v_1(a_1,a_k,\ldots,a_l)} S_v(B)$ *and* $RS_{v2} \leftarrow \Pi_{v_2(a_1,a_m,\ldots,a_o)} T_v(C)$.

The projections queries $\Pi_{v_1(a_1,a_k,\ldots,a_l)}$ and $\Pi_{v_2(a_1,a_m,\ldots,a_o)}$ are called **reconstruction queries** *for the projection query $\Pi_{(a_1,\ldots,a_j)}$, if and only if*

$RS = \Pi_{(a_i,\ldots,a_j)}(RS_{v1} \bowtie_{a_1} RS_{v2})$. □

Definition 4. *FVPD methodology*

Let $A = \{a_1, a_2, \ldots, a_n\}$ be a set of attributes and let R be a non-FVPD relation with attributes A. Let $B = \{a_1, a_2, \ldots, a_k\}$ and $C = \{a_1, a_{k+1}, \ldots, a_n\}$ for $2 \leq k \leq n-1$ be two sets of attributes such that:

- $B \cup C = A$,

  and

- $B \cap C = \{a_1\}$,

  and

- *the result sets for the projections on* B *and* C*:* $RS_{v1} \leftarrow \Pi_{(a_1,a_2,\ldots,a_k)}R(A)$ *and* $RS_{v2} \leftarrow \Pi_{(a_1,a_{k+1},\ldots,a_n)}R(A)$ *contain no sensitive or relevant information respectively.*

Then, the two relations $S_v(B) = RS_{v1}$ *and* $T_v(C) = RS_{v2}$ *are* **FVPD relations** *for the non-FVPD relation* $R(A)$*.*

□

2.1.1 Correctness of FVPD methodology

Theorem 1 states the correctness of the original *SeDiCo* approach. The proof of the Theorem consists in two steps: (1) presentation of the algorithm for the query rewriting into the reconstruction queries, and (2) the proof that the two rewritten queries are in fact *reconstruction queries* for the original one.

Theorem 1. *Let* $A = \{a_1, a_2, \ldots, a_n\}$ *be a set of attributes and let* R *be a non-FVPD relation with attributes* A*. Let* $S_v(B)$ *and* $T_v(C)$ *be FVPD relations for relation* $R(A)$*.*

For each Π_ω*,* $(\omega = (a_i, \ldots, a_j) : 1 \leq i < j \leq n)$*, projection query for* $R(A)$*, such that*

$RS \leftarrow \Pi_\omega R(A)$*,*

there exist two projection queries $\Pi_{v_1(a_1,a_k,\ldots,a_l)}$ *for* $S_v(B)$ *and* $\Pi_{v_2(a_1,a_m,\ldots,a_o)}$ *for* $T_v(C)$*, that are reconstruction queries for the original projection query* Π_ω*.*

The proof of Theorem 1 can be found in the full version of the thesis.

This proof verifies hypothesis 0, stating that the FVPD methodology improves the level of security and privacy in the context of relational databases. As stated in Definition 4, the approach can be extended to more than *two FVPD partitions* and to more than *one primary key attribute*. Here *SeDiCo*, as an implementation of this methodology, not only provides a framework but also showed the technological feasibility.

2.2 Data Distribution: The *SeDiCo* Approach

The basic approach of *SeDiCo* (A *SE*cure and *DI*stributed *C*loud Data st*O*re) is to divide data into several partitions and distribute them across various clouds. Thus, every cloud provider only gets a chunk of the data that is worthless without the other parts. Based on this logical and physical data distribution, the level of security and privacy in the cloud is enhanced.

Figure 2.1 illustrates the *Security-by-Distribution* approach with a simplified example based on the TPC-W *CUSTOMER* relation.

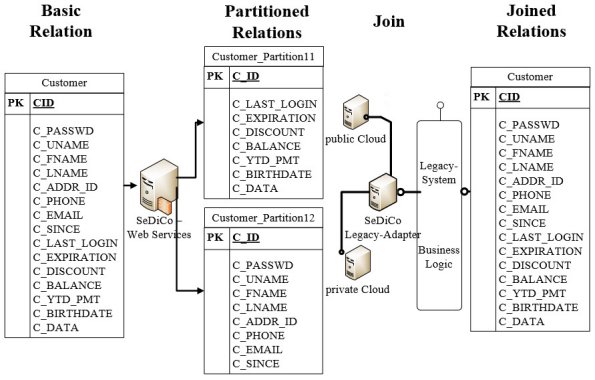

Figure 2.1: *SeDiCo* Architecture with TPC-W CUSTOMER Data Scheme

Since it is possible to distribute data across various clouds and various database systems, the entire setup can be regarded as a so-called *distributed database system* (Elmasri & Navathe, 2015).

A concluding architectural overview about all these concepts can be found in Figure 2.2.

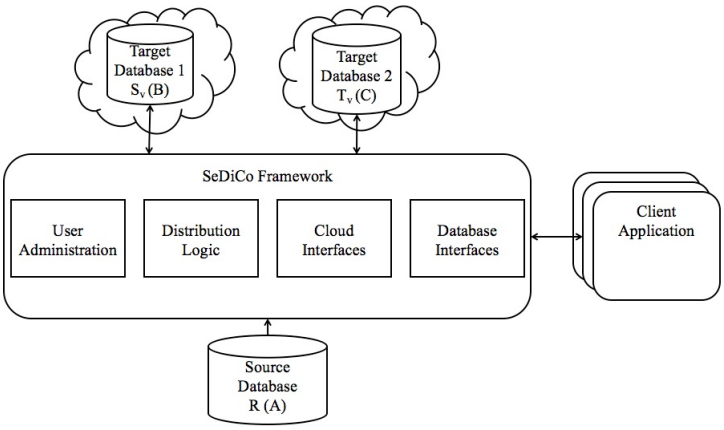

Figure 2.2: *SeDiCo*'s Architectural Overview

The *SeDiCo* framework targets on database administrators, developers and architects, who aim at transferring database data into a dynamically scalable cloud infrastructure. Also, system administrators and architects who intend to use a cloud-based infrastructure for creating redundant or high availability database systems are addressed. For these target groups, *SeDiCo* offers a solution to use all kinds of cloud deployment models, i.e. public, private, hybrid and community clouds, for the storage of database data, which is transparently usable for new but also legacy applications.

Figure 2.2 illustrates the entire *SeDiCo* framework. *SeDiCo* is implemented in Java as the most widely used programming language in nowadays enterprises (TIOBE, 2016). Basically, there are four central aspects: the *user administration*, the *distribution logic*, the *cloud interfaces* and the *database interfaces*. The key components for this thesis are the *distribution logic* and the *database interfaces*. It is possible to use the *SeDiCo* framework with different database implementations (e.g. MySQL, Oracle, MariaDB, etc.). However, although these database systems implement the SQL standard, the concrete implementation differs from database system to database system. This requires an additional layer that abstracts from the concrete database system implementations and this is done in the *database interfaces* component with Hibernate (RedHat, 2016) as an ORM (Object-Relational Mapper)[1]. Here, Hibernate introduces a high-level query language (JPQL, Java Persistence Query Language) upon SQL, which is independent from the concrete underlying database system. Hibernate, its implications for *SeDiCo* and the distribution logic component are introduced in more detail in the full version of the thesis.

2.2.1 FVPD Join

A key element in *SeDiCo* is the *join* of rows that match a query. Transferred to the presented FVPD approach, a *join* corresponds to *joining* query matching rows in order to reconstruct them. Thus, all join algorithms that are described in this section implement the *natural join* (cf. Section 1.3), and the replicated *primary key attribute* a_1 appears only once in the respective result set. Thus, for the join both partitions R and S has to be iterated to find query-matching rows.

[1] An ORM has several advantages: firstly, it bridges the gap between the object-oriented programming and the relational database paradigms, secondly, it abstracts from a concrete database implementation, and thirdly, it ensures transaction safety with the usage of a so-called *session*

This results in a run time complexity (with respect to the response time) of $\mathcal{O}(n^2)$, with n indicating the cardinality of R and S.

The complexity of this initial FVPD approach (without any optimization) is summarized in Table 2.1.

Table 2.1: Query Mechanism Complexity

Query Mechanism	**Join Algorithm**	**Complexity**
Initial FVPD approach	Nested-Loops Join	$\mathcal{O}(n^2)$

The presented version of *SeDiCo* was developed and implemented before the work on the thesis has started. The results of this preliminary work have shown that the ideas behind *SeDiCo* are feasible and work in practice. However, there is still an open question regarding the framework performance in practical use cases:

- **Performance optimization of *SeDiCo* approach:** Although the feasibility of the original implementation is empirically shown and formally proved, the response time (especially for larger data sets, i.e. more than 10K rows) decreased tremendously. Thus, how can the response time for a FVPD query in practical use cases scenarios be improved, such that it is in the same order of magnitude as a non-FVPD query?

Chapter 3

Background and Related Work

This chapter covers the architectural background for the key concepts of the entire *SeDiCo* framework and relates them to current research topics. The structure of this chapter is aligned to Figure 3.1[1].

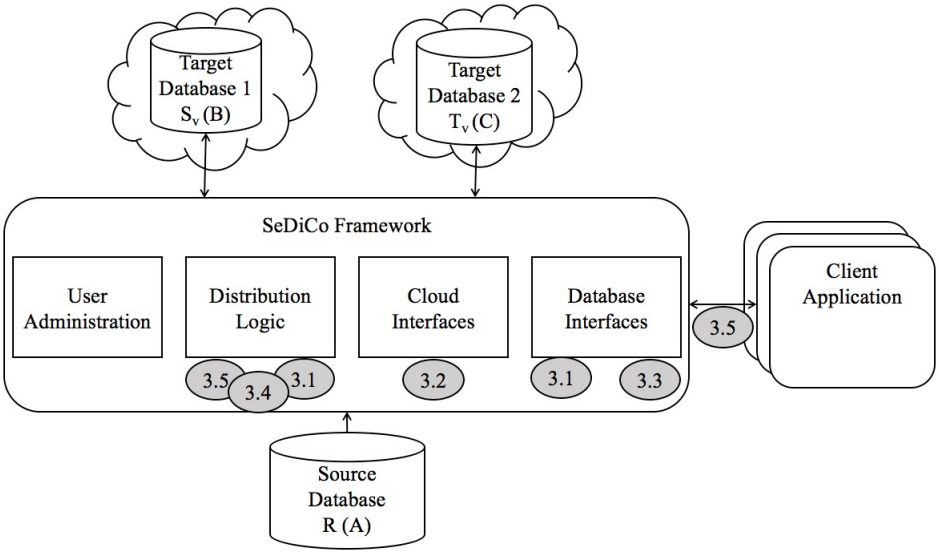

Figure 3.1: *SeDiCo* Architecture Mapped to Chapter Content

First, security and privacy challenges are addressed in Section 3.1 with the *Security-by-Distribution* approach. This is motivated by the usage of Cloud Computing architectures (cf. Section 3.2). The *Security-by-Distribution* principle with different database systems demands an abstraction layer that encapsulates different vendor-specific SQL implementations into a centralized interface. Therefore, object-relational mappers (ORMs) are in the focus of Section 3.3. Another key element is the investigation of the related work for the caching approach in 3.4. Last not least, Section 3.5 presents several alternative benchmarks with a strong focus on databases to evaluate the before-mentioned approaches. The complete presentation of the background and the related work can be found in the full version of the thesis.

[1]note that the user administration component is out of the scope of this work and is not described in more detail here. Section 4.1 is a central aspect in the distribution logic and in the database interfaces component and is therefore illustrated twice

Chapter 4

Conceptualization

Generally, there are 3 approaches investigated in this thesis in order to minimize the response time of FVPD data: a *query rewriting*, a *caching*, and an *SSD-based* one. Previously published works of the author can be found in (Kohler, Simov, Fiech, & Specht, 2015)[1] for the *query rewriting* in (Kohler & Specht, 2015c) and in (Kohler & Specht, 2015a) concerning the *caching* approach. The *SSD-based* one has not been published or evaluated so far. These approaches are conceptualized here to optimize the original *SeDiCo* framework implementation outlined in Section 2.

4.1 Query Rewriting Approach

The fundamental idea behind this approach is to not only partition and distribute relations and their rows, but also to partition queries accordingly. This section formalizes the entire *query rewriting* approach based on a *projection* issued against two partitions $S_v(B)$ and $T_v(C)$.

Since the partitions are restricted to be *disjoint* and *complete*, it is ensured that all attributes are matched only once, except for the primary key (a_1) (*disjointness*) and none of the attributes is omitted (*completeness*). After the query parsing, the query is partitioned and issued against the respective partitions. Finally, the result sets with the matching rows are collected and the result sets are joined into a final result set.

A nice advantage of this *query rewriting approach* is that both selections can be run in parallel so that the corresponding result sets can be produced simultaneously.

[1]This work also demonstrates how additional query filter, join, etc. criteria (previously denoted as ω) are implemented in the *SeDiCo* framework. However, they are ommited here as they are out of the scope and for the sake of better readability.

4.2 Caching Approach

The three *caching* mechanisms presented in this work can be distinguished as follows:

- Server-Based Caching

  These caches are server-based caches (i.e. a cache for every partition) that are operated on different servers between the vertical database partitions and the clients. Every cache only stores tuples from its respective cloud partition and clients access these caches rather than the actual database partitions. Performance improvements are expected from faster access of the cache memory but the actual join of the tuples have to be performed in the clients.

- Local Caching

  This is a cache for each client, as there is a 1:1 connection between client and cache. Here, tuples are already joined (reconstructed) in the cache, which promises performance improvements.

- Remote Caching

  Firstly, it has to be noted that this approach violates *SeDiCo*'s *Security-by-Distribution* approach, because a single central server that stores already joined tuples is used. However, in order to develop a basic performance metric, this approach is considered useful in the context of this work for the sake of comparability.

4.3 SSD-based Approach

The SSD-based approach is similar to the original *SeDiCo* approach.

The fundamental idea is that a major performance gain concerning the collection of the result sets and the join performance can be achieved with the usage of new hardware technologies (i.e. Solid State Drives, SSDs) that store the respective partitions.

Chapter 5

Implementation

With respect to the formal description of the *query rewriting*, the *caching*, and the *SSD-based* query mechanisms, this chapter now outlines their concrete implementation. Figure 5.1 gives an overview about the location of the respective mechanisms and their integration into the *SeDiCo* framework. Hence, Figure 5.1 is used as an overview about the structure of this chapter which firstly outlines the concrete *query rewriting* implementation, secondly, the *caching* and lastly the *SSD-based* approach. The complete outline of the implementation can be found in the full version of the thesis.

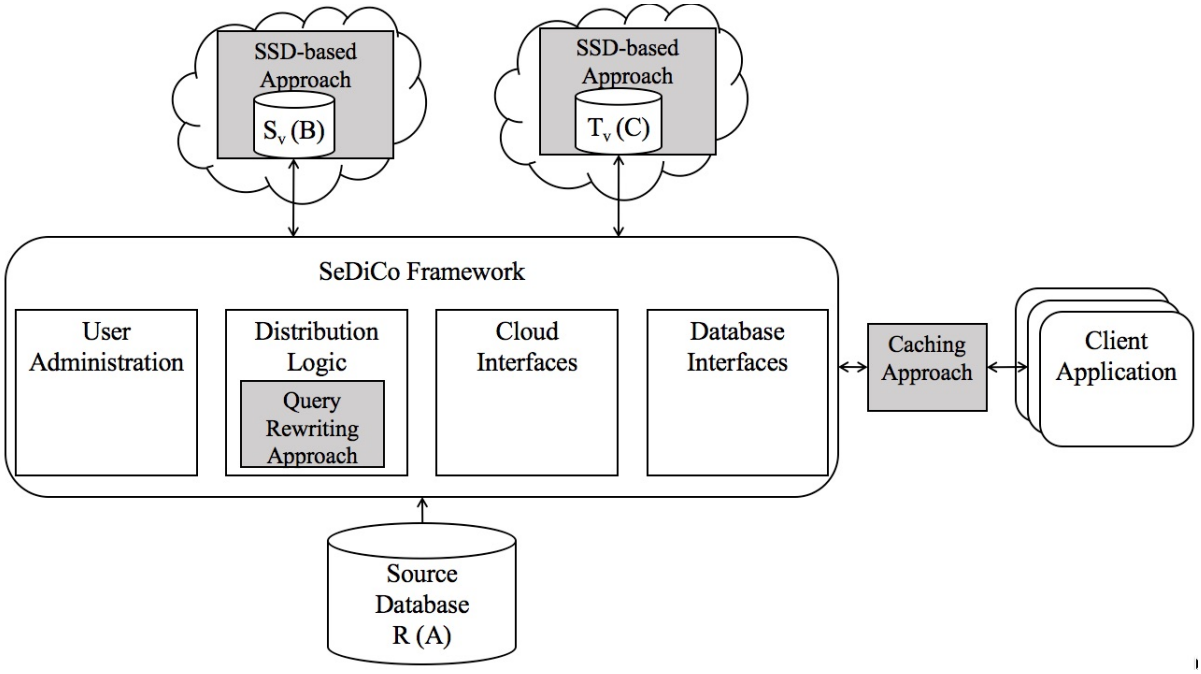

Figure 5.1: *SeDiCo* Query Mechanism Integration Overview

Chapter 6

Evaluation

6.1 Evaluation Environment

In order to achieve a better comparability throughout all previous works of the author and this thesis, all evaluations are performed with a data set that ranges from 0 to 288K randomly generated rows, which result in an overall database size of:

Table	Size in MB
CUSTOMER ($R(A)$)	147

Table 6.1: Data Set Size of Relation $R(A)$

Tables	Size in MB
CUSTOMER_p1 ($S_v(B)$)	56
CUSTOMER_p2 ($T_v(C)$)	113

Table 6.2: Data Set Size of Vertical Partitions $S_v(B)$ and $T_v(C)$

Furthermore, it has to be mentioned that the figures in this section are only excerpts of the evaluation because of the great variation in the query times from 1 to 288K tuples. Hence, the figures only illustrate the query times from 1K to 88K tuples, which is considered the best trade-off between informational value and readability.

Evaluation Environment In the *local* evaluation all components are installed on one single machine (with a dual core 2.4 GHz processors, 8 GB RAM, 500 GB HDD, CentOS 6, Java 1.7_79 64Bit, MySQL 5.6, Oracle Express 11g). The main reason for this is to avoid typical side-effects in Cloud Computing environments (i.e. changing (Internet) network bandwidths, unknown utilization of physical hosts and virtual machines, etc.). In contrast to the *local* evaluation, the *remote* evaluation uses a (private) cloud infrastructure and a client connected via local area network (LAN) to reduce the above-mentioned side-effects to a

minimum. Based on this these capabilities, two private cloud infrastructures (Eucalyptus 3 (Hewlett Packard, 2016) and CloudStack 3.2.2 (Apache, 2016)) with à 5 computing nodes (1 cloud management server and 4 cloud nodes) (with the hardware dimensions stated above) were set up.

It further has to be noted that the following performance evaluations also include the measurements of a simultaneous usage of both database systems. This is the so-called *combined response time*. As the *SeDiCo* framework offers the possibility to use both database systems for partitions at the same time, it is possible to store one FVPD partition in an Oracle and the other in a MySQL database (or vice versa).

6.2 Basic Database Performance Evaluation

This initial performance measurement serves as a basic performance metric for all following evaluations. All performance measurements were conducted three times and the average times of all measurements are listed in the figures of this chapter. With this approach, unreproducible side-effects like Java's Garbage Collector or changing host utilization could be reduced to a minimum.

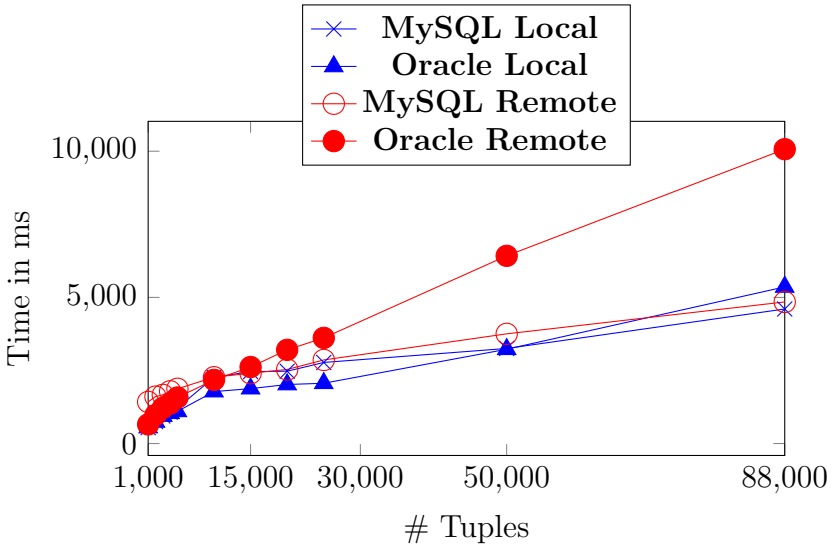

Figure 6.1: Initial Response Times

6.2.1 Conclusion

Considering the average response time of Hibernate, based on a non-distributed and non-partitioned data set, these results show that querying a MySQL database requires ∼1,6 seconds and an Oracle database which is nearly similar requires

~1,7 seconds. These values stem from a *local* environment where all components are installed on one single physical machine. Although this is not applicable in real world scenarios, these figures provide a basic performance metric. In a *remote* setup (i.e. a client-server environment), querying a MySQL database requires ~2 seconds and an Oracle database needs ~3 seconds to answer the query. Thus, it can be concluded that the network overhead is ~0.3 seconds (MySQL) and ~1,4 seconds (Oracle) and this is insofar interesting as the following performance measurements will show, how this network overhead affects the *Security-by-Distribution* approach of the *SeDiCo* framework.

6.3 *SeDiCo* Framework Performance Evaluation

Similar to the previous section, this evaluation is also divided into a *local* and a *remote* measurement and the following sections present the response times of the initial *SeDiCo* security-by-distribution approach without any optimizations.

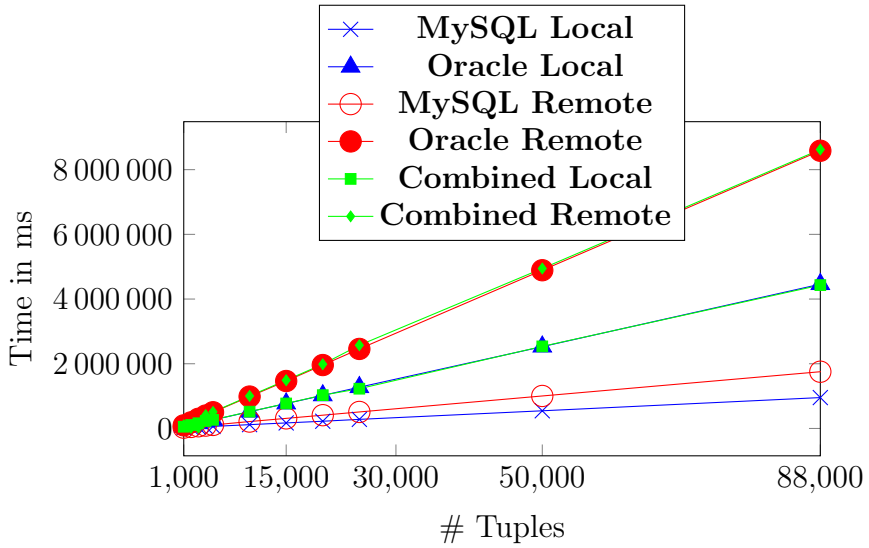

Figure 6.2: Initial *SeDiCo* Response Times

The figures from the *local* and from the remote evaluation in Fig. 6.2 show that the average response time of the FVPD data is considerably slower compared to the initial implementation in Fig. 6.1. The average response time for non-partitioned and non-distributed data compared to the average FVPD response time is ~1,6 seconds for MySQL and ~1,7 seconds for Oracle in the *local* setup and ~2 seconds for MySQL and ~3 seconds for Oracle in the *remote* setup. Compared to the FVPD setup in the initial *SeDiCo* approach, the response times are ~257 seconds for a MySQL and ~1,100 seconds for an Oracle database (*local*) and ~465 seconds for MySQL and ~2,200 seconds for Oracle in a *remote* setup.

6.3.1 Conclusion

Based on these figures, it has to be noted that the *SeDiCo* approach as is, unfortunately is not usable in practical usage scenarios. Above that, the combined response times in Fig. 6.2 show interesting results, as both, the *local* and the *remote* values, are similar to the results of the Oracle database. Thus, it can be concluded that the bottleneck in combined scenarios is always the slower database.

Comparing the *local* against the *remote* setup of this section shows that with a 1 Gbit Ethernet broadband LAN connection between the components, the network causes another performance degrade by factor ∼2 averagely.

6.4 Query Rewriting Evaluation

In this section, the response time of the *query rewriting approach* with its 3 FVPD join algorithms (*nested-loops*, *hash* and *sorted-merge join*) is evaluated.

The following figures (Fig. 6.3-Fig. 6.5) illustrate the respective join implementation for the FVPD partitions in the *local* as well as in the *remote* environment.

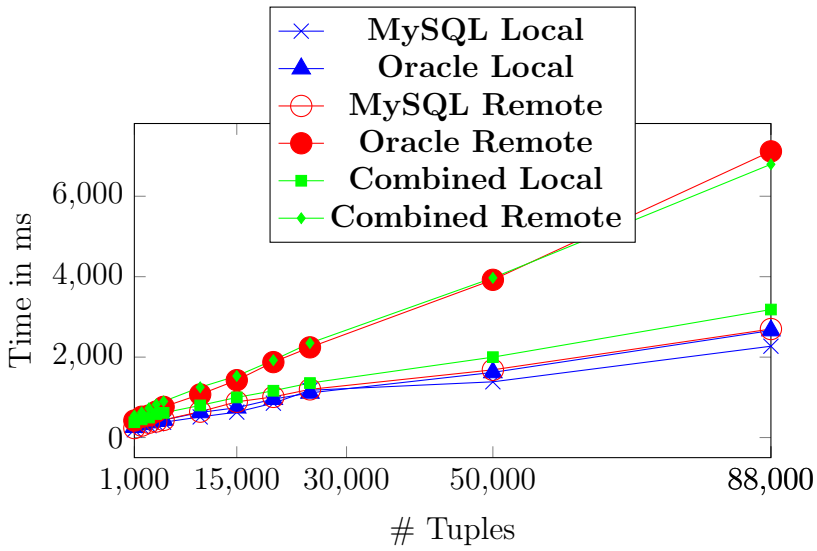

Figure 6.3: FVPD Query Rewriting Nested-Loops Response Time

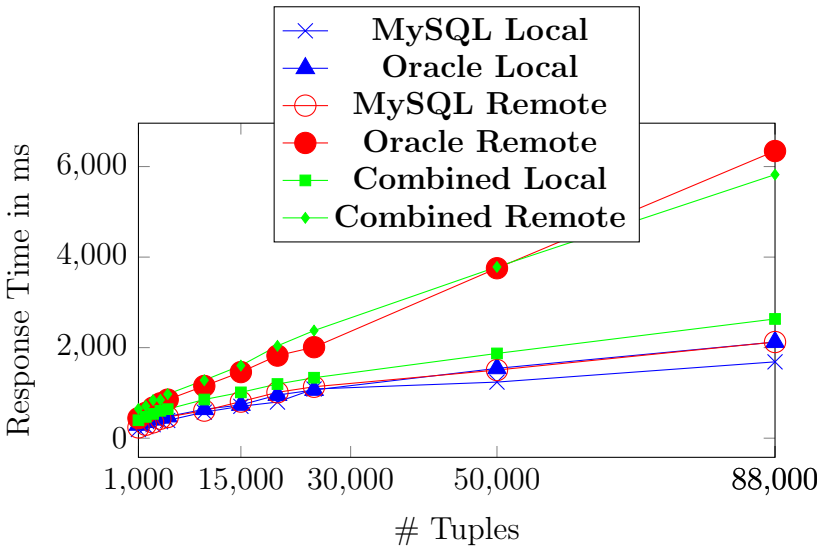

Figure 6.4: FVPD Query Rewriting Hash Join Response Times

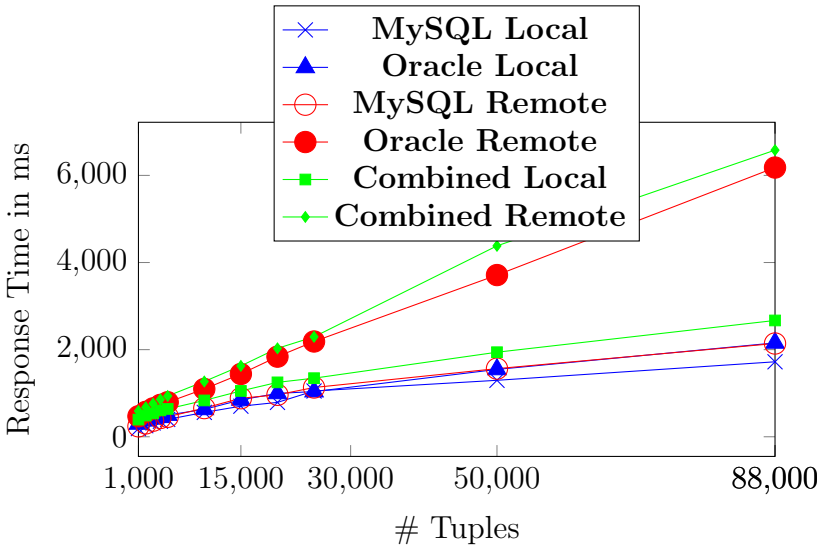

Figure 6.5: FVPD Query Rewriting Sorted-Merge Join Response Times

6.4.1 Conclusion

In contrast to (Kohler & Specht, 2015b), the *SSD-based query rewriting approach* evaluation showed that the *hash join* ($\mathcal{O}(n+m)$) and the *sorted-merge join* ($\mathcal{O}(n * log(n))$) produced almost similar results. As expected, both outperformed the *nested-loops join* ($\mathcal{O}(n^2)$) (Fig. 6.3 - Fig. 6.5) if the average performance is considered.

All join algorithms benefit from the fact that the rows are collected in sorted order based on their primary key values from the underlying FVPD database

partitions. This reduces the join phase (join, probe or merge phase), as e.g. the inner loop of the *nested-loops join* can stop as soon as the first matching row is found. The same holds for the probe phase in the *hash join* and for the merge phase in the *sorted-merge join*.

Above that, taking a closer look into the hash and the sorted-merge join, which both produced almost similar query response times, it can be noted that collecting query matching rows from the FVPD partitions (i.e. the build and the sort phase) are similar. Both algorithms only differ in their join (i.e. their probe and merge) phases. The total response time of both algorithms depicted above show that the collection phase heavily predominates the join phase and that the join phase is an exceptionally small part of the total response time. Thus, even if the probe phase (*hash join*) outperforms the merge phase (*sorted-merge join*) by factor $\sim$2, the overall response times of both algorithms are almost similar.

Furthermore, regarding the *nested-loops join* performance in Fig. 6.3, the results show a remarkable performance gain compared to the initial *SeDiCo* implementation depicted in Fig. 6.2. Moreover, the *hash* and the *sorted-merge join* achieved even greater performance improvements as Fig. 6.4 and Fig. 6.5 show.

6.5 Caching Evaluation

Server-Based Caching As in this implementation every partition has its own cache, none of these reside at the client. Thus, there is only a *server-based* evaluation for this implementation. Yet, this evaluation distinguishes between a *lazy* and a *parallel* row fetching strategy, which fetches rows from the caches either partition-wise or simultaneously in different threads.

Local and Remote Caching Finally, this section evaluates the *local* and the *remote* caching approach and the respective results are illustrated in Fig. 6.7

6.5.1 Conclusion

Considering the pure cache performance without the cache warming phase, both, the *local* and the *remote* implementations outperform the *server-based* one. This is not surprising, as in the *local* and *remote* caches, the rows are already joined and thus this is comparable to traditional non-partitioned and non-distributed database caching approaches.

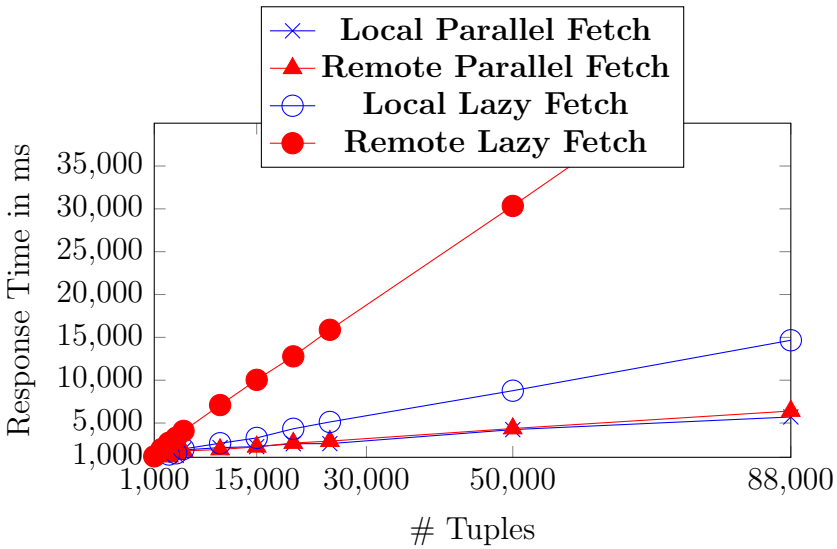

Figure 6.6: FVPD Server-Based Parallel and Local Response Times

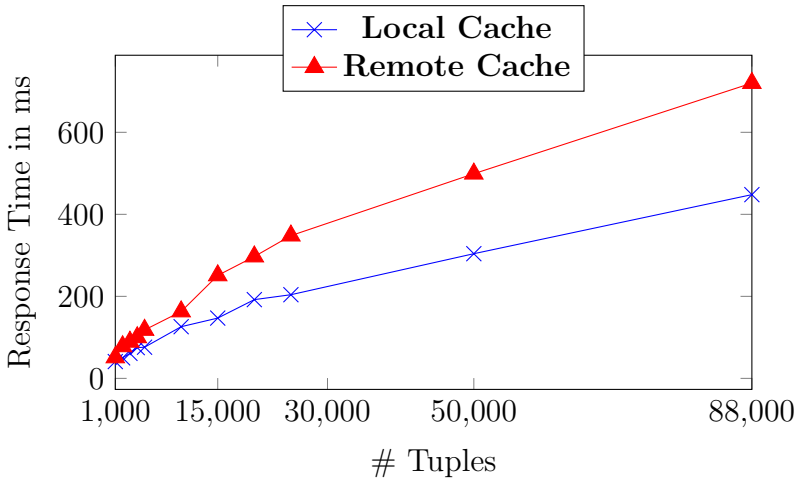

Figure 6.7: FVPD Local and Remote Caching Response Times

Above that, this evaluation showed that the *local* and the *remote* implementations also outperform the *query rewriting* and the *SSD-based* approaches. However, taking the *cache warming* phase into consideration, the figures above get relativized and then the *query rewriting* approach outperforms the *caching approach* again.

The cache warming phase is neglected in this evaluation because it only has to be performed once, e.g. at the start of the *SeDiCo* client. Once the cache is warmed, all queries can be run against the cache. In addition to this however, updating the cache (e.g. regularly time-based, user-triggered, via cache invalidation, etc.) is another requirement, which is not considered in this evaluation. This is also too dependent on the specific database workload and thus regarded as a future

work task in concrete application domains, where the *SeDiCo* framework will be integrated.

Moreover, considering the decentralized implementation it can be concluded that the parallel fetch outperforms the lazy fetch by factor ∼2 (local fetch) and by factor ∼7 (remote fetch). However, the concrete fetch strategy is heavily dependent on the database workload and if the partitioning scheme is defined such that most queries can be answered with only the values of a single partition, the lazy fetch outperforms the parallel fetch, even if the partitions are accessed simultaneously there. Finally, the results confirmed the assumption that collecting rows form the cache memory is faster that directly from the FVPD partitions and this is the case for the entire caching approach.

6.6 SSD-based Evaluation

Basic SSD-based Performance Evaluation Analogous to the previous evaluations in this work, firstly, a basic SSD performance metric with a non-distributed and non-partitioned data set is measured (Fig. 6.8) and then the FVPD data set (sec. 6.1) based on SSDs is evaluated in Fig. 6.9.

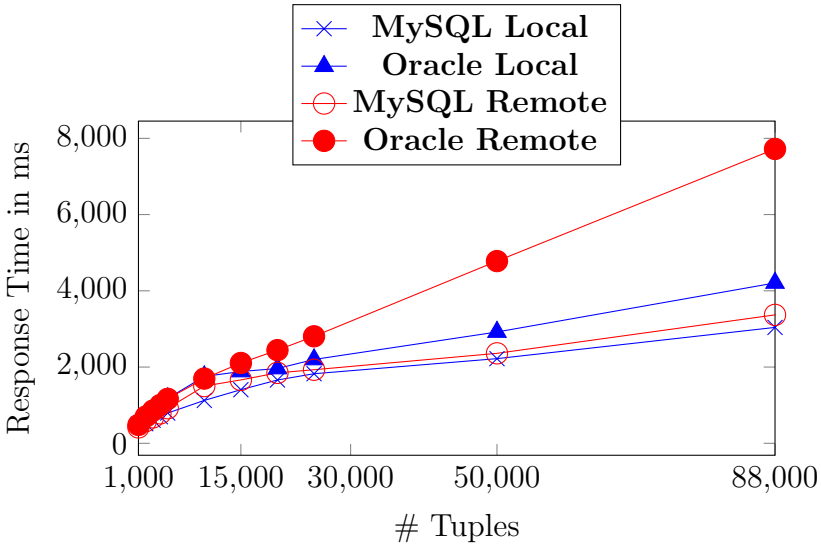

Figure 6.8: Initial SSD-Based Response Times

6.6.1 Conclusion

The initial SSD performance measurement in the *local* and in the *remote* environment (Fig. 6.8 compared to Fig. 6.1), showed that indeed the SSD as secondary storage improves the response times by ∼30% in a *local* MySQL, by ∼20% in a *local* Oracle, by ∼60% in a *remote* MySQL, and by ∼32% in a *remote*

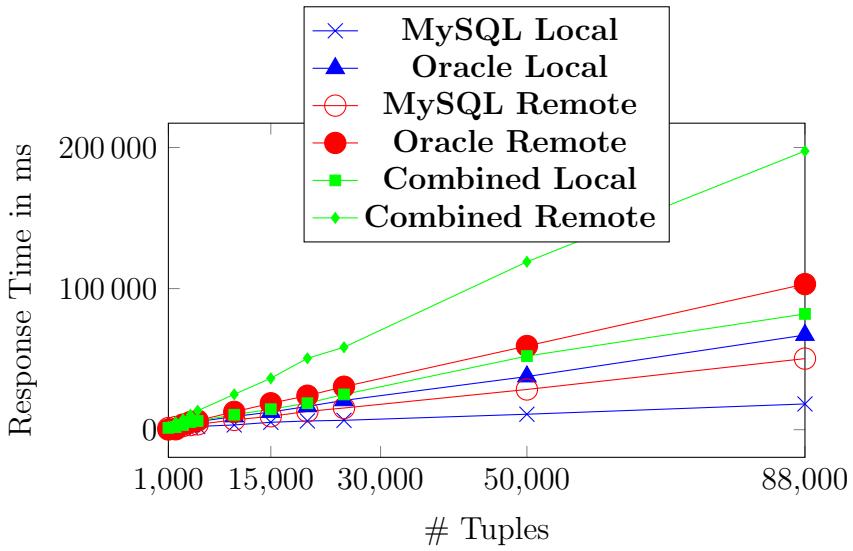

Figure 6.9: FVPD SSD-Based Response Times

Oracle environment (Fig. 6.8). These are promising results; however, they show the improvements based on a non-partitioned and non-distributed data set.

Transferred to a FVPD data set, the performance gains are depicted in Fig. 6.9 for the *local* and for the *remote* evaluation environment. Compared to the initial *SeDiCo* framework evaluation (Fig. 6.2), these values show a significant performance gain by factor ∼50 (MySQL *local*), by factor ∼68 (Oracle *local*), by factor ∼34 (MySQL *remote*) and by factor ∼52 (Oracle *remote*).

Analogous to these figures are the measurements for the *combined* usage of MySQL and Oracle. Here again, the slower database system is the bottleneck, but the SSD-based approach improved the *combined response times* by factor ∼64 for the *local* and by factor ∼41 for the *remote* setup. To sum up this section, it can be concluded that although the usage of SSDs yields to a remarkable performance gain, the SSD-based approach is still not feasible in practical usage scenarios. However, using SSDs rather than traditional HDDs is a good starting point for further approaches and thus, all following evaluations are performed with SSDs as secondary storage devices.

Chapter 7

Summarization of the Main Results

First of all, the hypotheses are recapitulated now and then they are either verified or rejected.

Hypothesis 0: The definition of a Fixed Vertically Partitioned Schema (FVPD) for relational databases improves the level of data security and data protection by separating (i.e. partitioning) and distributing logically coherent data to different storage locations.

This hypothesis can be verified. It was formally proved in Chapter 1. In addition to this, the *SeDiCo* framework as a concrete implementation of the FVPD methodology showed its technological feasibility.

With respect to the hypotheses 1-3, Table 7.1[1] states the aimed average response times of the non-partitioned and non-distributed data set, which was queried with Hibernate as the used ORM. The average response times used throughout this chapter are the average response times of the presented figures in the evaluation in Chapter 6, ranging from 1 - 288K tuples.

Table 7.1: Average Hibernate Response Time for a Non-FVPD Data Set in ms

Secondary Storage	MySQL Local	Oracle Local	MySQL Remote	Oracle Remote
HDD	1,626	1,701	2,050	2,991
SSD	1,253	1,415	1,267	2,256

Hypothesis 1 was stated as follows:

Hypothesis 1: Query Rewriting improves the response time to a level that is in the same order of magnitude as a non-partitioned and non-distributed scenario due to partitioned and parallelized query and join implementations.

[1] As these measurements are conducted on a non-FVPD data set, a *combined* measurement with both databases simultaneously was not possible.

This hypothesis can be verified. Surprisingly, the average performance of *query rewriting* and its corresponding join algorithms mostly outperforms the basic performance metric (Table 7.1). Accordingly, Table 7.2 shows the performance values of all query mechanisms with the best achieved values in bold font.

Table 7.2: Average FVPD Response Time in ms

Approaches	**MySQL Local**	**Oracle Local**	**MySQL Remote**	**Oracle Remote**	**Combined Local**	**Combined Remote**
Initial *SeDiCo*	257,346	1,177,436	465,972	2,266,895	1,178,949	2,287,508[a]
Query Rewriting Nested-Loops Join	**955**	1,093	1,094	2,339	1,323	2,567
Query Rewriting Hash Join	**600**	755	716	1,944	979	1,959
Query Rewriting Sorted-Merge Join	**609**	766	729	1,946	993	2,255
Server-Based Caching Parallel Fetch	**2,020**	N/A[b]	2,094	N/A[b]	N/A[b]	N/A[b]
Server-Based Caching Lazy Fetch	**4,186**	N/A[b]	14,323	N/A[b]	N/A[b]	N/A[b]
Local Caching	**146**[c]	N/A[b]	365	N/A[b]	254	N/A[b]
Remote Caching	**229**	N/A[b]	434	N/A[b]	324	N/A[b]
SSD-Based	**5,160**	17,255	13,656	43,119	18,219	55,116

[a] note that this is the upper bound t_{upper} defined in Section 1

[b] As the queries are directly issued against the cache, the underlying database can be neglected. Therefore, only MySQL was used for this evaluation.

[c] note that this is the lower bound t_{lower} defined in Section 1, because here the local cache stores the already reconstructed relation $R(A)$

The figures depicted in Table 7.2 and Table 7.3 show that *query rewriting* is absolutely applicable in practical usage scenarios. Hence, the entire *SeDiCo* framework becomes a viable approach with respect to security and privacy in especially public cloud environments.

Table 7.3: Comparison of Hash and Sorted-Merge Join with Larger Data Sets in ms

#Tuples	Hash Join		Sorted-Merge Join	
	Build	**Probe**	**Sort**	**Merge**
1M	30,207	6,097	339	3,895
750K	10,167	1,671	578	3,859
500K	6,491	1,357	375	3,831
Average	**15,621**	**3,042**	**431**	**3,852**
Total	**18,063**		**4,283**	

Hypothesis 2 was originally stated as follows:

Hypothesis 2: Caching data improves the response time to a level that is in the same order of magnitude as a non-partitioned and non-distributed scenario due to the usage of In-Memory caches.

This hypothesis can be verified. For this evaluation, only the pure cache performance is important and thus Table 7.2 only focuses on the *local*, *remote* and *server-based* cache performance without the *cache warming* phase.

Hypothesis 3 With respect to the general research question, hypothesis 1 referred to the *SSD-based* approach and was stated as follows:

Hypothesis 3: Using Solid State Disks (SSDs) as distributed secondary storage devices for the FVPD data improves the response time to a level that is in the same order of magnitude as a non-partitioned and non-distributed scenario due to faster access times of the memory.

This hypothesis must be rejected. Although the response time gains achieved with SSDs were significant (cf. Section 6.6), the performance did not reach the same order of magnitude[2] as queries based on non-FVPD data sets. Nevertheless, the achieved performance values are listed in Table 7.2.

The evaluation further showed that every query mechanism has pros and cons and therefore, no clear recommendation can be given here. Table 7.4 summarizes these advantages and disadvantages.

This summary shows that *query rewriting* and *caching* proved the hypotheses of this work. Although the hypothesis concerning the *SSD-based* approach has to be rejected, even this approach promises performance improvements, however, the improvements were not as big as expected. Finally, it can be concluded that the

[2]except for the local MySQL measurement

Table 7.4: Query Mechanism Summary

Query Mechanism	Preserves *Security-by-Distribution*	Pros	Cons
Query Rewriting	yes	Fast response times	Large amount of client RAM memory necessary for the join algorithms (when large data volumes with many query matches are applied)
		Applicable in practical usage scenarios	Advantages of parallel fetch can only be applied on clients that have multiple cores (i.e. as many cores as there are partitions)
Caching		Fast response times	Additional cache coherence protocols, that affect the response time or the data consistency are required
		Applicable in practical usage scenarios	
Server-Based	yes	Dynamically scalable cache memories	Slower response times compared to query rewriting and local and remote caching
Local	yes	Fastest response time compared against the other approaches	Cache warming required at every start of the client (the more data, the more time-consuming is the cache warming) Cache requires large amount of client RAM (with large data volumes)
Remote	no	Cache warming must only be performed once at server start Dynamically scalable cache memories	Cache requires additional security and privacy measures
SSD-Based	yes	No conceptual, algorithmic or architectural *SeDiCo* framework changes required	Comparatively slow response times Inapplicable in practical usage scenarios because of the slow response times

results of *query rewriting* and *caching* are promising to further following *SeDiCo*'s vision of creating a *secure and distributed cloud data store* where the performance is in the same order of magnitude as traditional relational non-partitioned and non-distributed databases.

Chapter 8

Framework Application in Semantic Web Databases

The introduction of this section (in the full version of the thesis) outlines the history of the Semantic Web and its development. The thesis firstly gives a short introduction about the Semantic Web, its technologies and its basic notions. This introduction is omitted here due to the lack of space, but then the research problem is formulated. After that, the approach to transfer the *SeDiCo* distribution approach to RDF-based data is conceptualized, then implemented and finally evaluated and concluded. The concrete implementation, the correctness proof, the complexity analysis, the outlook and future work tasks, and relevant related work concerning this chapter (which are outlined in the thesis) are also omitted here due to the lack of space.

8.1 Problem Formulation

The author of this work proposes to include security and privacy into to context of *linked data* (LD). Generally, as the name LD suggests, data should be linked. However, it might not be clear at first sight which data are confidential or sensitive or even worse, which data might become confidential and sensitive when they are combined with other data. Thus, the proposed FVPD approach to increase the level of security and privacy might also become viable in the context of LD. Furthermore, in relevant literature no approach has dealt with security and privacy in this context so far. Above that, it is worth mentioning that not even one of the W3C standards or recommendations considered security, privacy or performance in LD. Indeed, there are 2 papers (Rakhmawati et al., 2013) and (Betz et al., 2012) that mention copyright, data ownership and security, but only in a small section. Therefore, the challenge of privacy and security is considered as neglected so far.

Finally, this section is driven by a hypothesis that is stated as follows:

Relational data, with respective mappings exposed as RDF data and published via SPARQL endpoints are vertically partitioned and distributed according to the FVPD methodology. Thus, the FVPD approach improves the level of security and privacy through physical and logical data distribution at comparable response times which are in the same order of magnitude as in a non-partitioned and non-distributed data set.

8.2 Approach

The general approach is to expose relational data as SPARQL endpoints with the FVPD approach incorporated, as illustrated in Fig. 8.1 and Fig. 8.2.

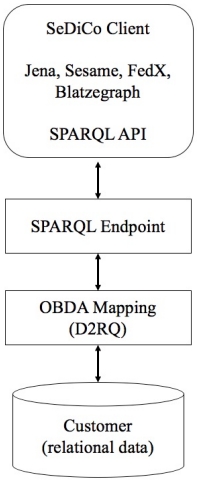

Figure 8.1: TPC-W *CUSTOMER* Table As SPARQL Endpoint

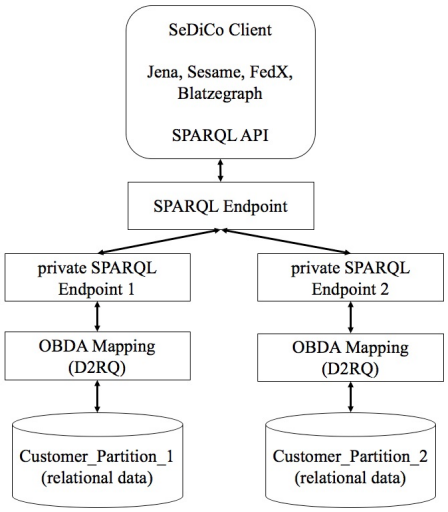

Figure 8.2: FVPD TPC-W *CUSTOMER* Partitions As SPARQL Endpoints

8.3 Evaluation

8.3.1 Evaluation Environment

For the evaluation of the before-mentioned Semantic Web frameworks and the underlying FVPD approach, the same evaluation environment as outlined in sec. 6.1 was used.

8.3.2 Local SPARQL 1.0 Evaluation

This section illustrates the measured values for the *local* non-FVPD (Fig. 8.3) and the FVPD-based (Fig. 8.4) evaluation.

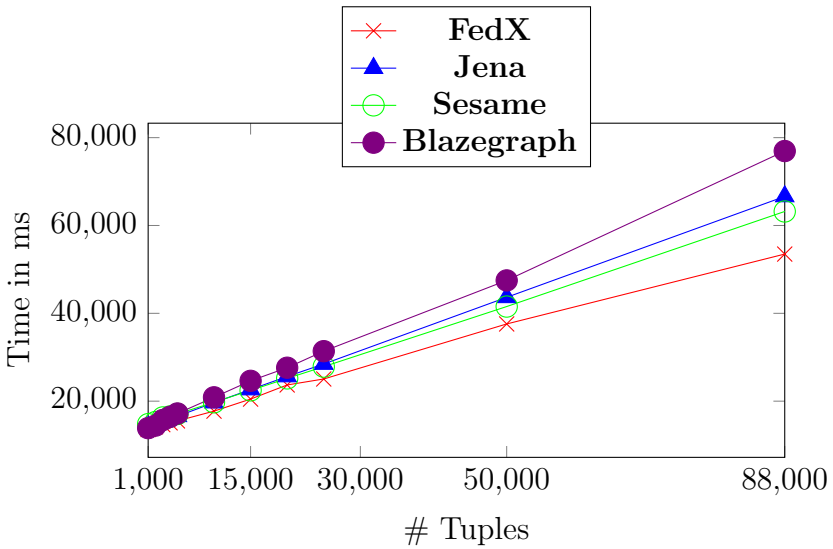

Figure 8.3: Local Non-FVPD OBDA Framework Evaluation

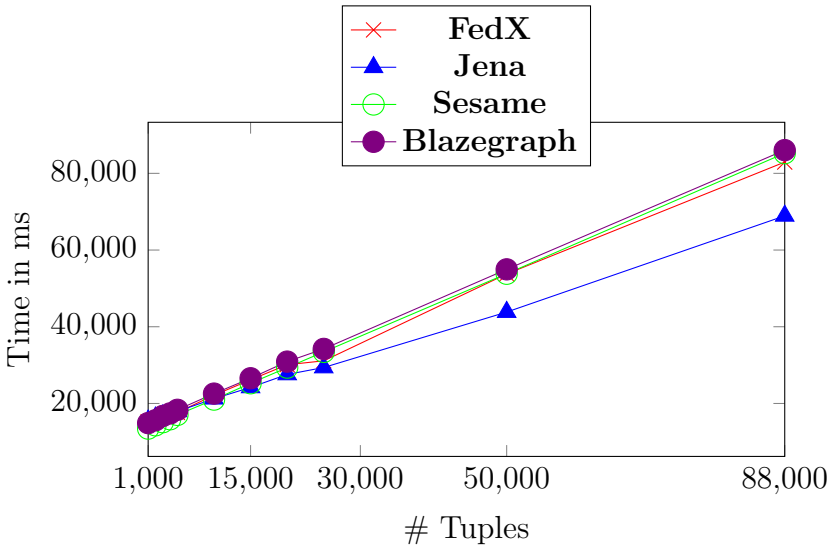

Figure 8.4: Local FVPD OBDA Framework Evaluation

8.3.3 Remote SPARQL 1.0 Evaluation

Accordingly, his section illustrates the measured values for the *remote* non-FVPD (Fig. 8.5) and the FVPD-based (Fig. 8.6) evaluation.

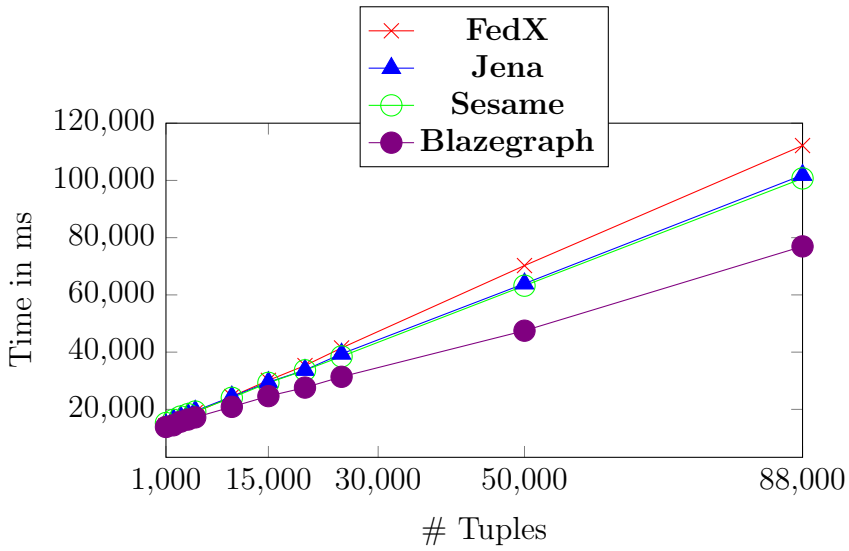

Figure 8.5: Remote Non-FVPD OBDA Framework Evaluation

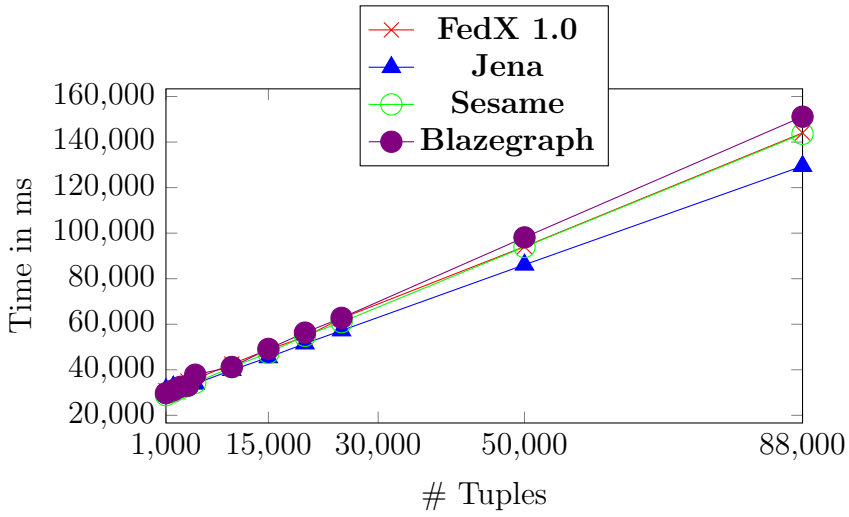

Figure 8.6: Remote FVPD OBDA Framework Evaluation

8.4 Conclusion

To conclude the evaluation, it can be noted that the results measured in (Haase et al., 2010) could not be confirmed. Surprisingly, the query performance for the *local* setup over the evaluated federation is similar (except for FedX) compared to the single endpoint implementation with a non-partitioned and non-distributed data set. Although, in the *remote* setup the average query performance degrades by factor ∼2 (Fig. 8.5 compared against Fig. 8.6), it can be noted that the federation does not have such an high impact on the average performance as it was elaborated in (Haase et al., 2010). However, it has to be mentioned that in this

evaluation a comparatively small data set was used compared to e.g. DBPedia (∼7 billion triples) or ∼110,000K triples in (Haase et al., 2010). Moreover, the evaluation in this section focused on an federation of only 2 different SPARQL endpoints, whereas the evaluation in (Haase et al., 2010) focused on 12 different endpoints. Hence it must be assumed that with a growing number of triples and a growing number of partitions (or federations) the performance degrades will also increase. Here, further measurements that take a problem-driven data distribution (with respect to the number of vertical partitions), a corresponding number of different endpoints and a use-case driven data volume into consideration are necessary. Above that, finding an adequate benchmark that takes these issues into account is another challenging task, which is therefore considered as a future work challenge.

Above that, (Haase et al., 2010) determined that complex queries are more effected by performance losses (averagely by factor ∼3) than simple queries. This could be confirmed in this evaluation with the separated view of SPARQL 1.0 (simple queries) and SPARQL 1.1 (complex queries). Here, the figures above show that the simple SPARQL 1.0 queries outperform the complex SPARQL 1.1 queries averagely by factor ∼10 for both, the *local* (Fig. 8.4) and the *remote* (Fig. 8.6) setup. Hence, the join can be identified as the crucial factor for the performance degrade and thus, especially complex queries could benefit from further optimizations with respect to this expensive operation.

Comparing the measured *RDF-based* results against the *SSD-based* TPC-W benchmark results in sec. 6.6 shows that the average query performance degrades by factor ∼22 in the *local* non-FVPD setup. Interestingly, in the *local* FVPD setup, the performance degrades only by factor ∼5. Moreover, the measurements for the *remote* setup are almost similar with performance degrades by factor ∼28 for the non-FVPD and by factor ∼4 for the FVPD data set. Thus, on the one hand, compared to a traditional database access via JDBC or via ORM, there is a significant performance loss that has to be considered in real-world scenarios. On the other hand however, accessing RDF-based data offers a schema free and more flexible way of querying data or even finding unknown relations, links or other information between data with the usage of inference engines.

Contrasting the *local* and the *remote* figures above, it can be concluded that although the integration of the FVPD approach decreases the query performance

by averagely factor ∼2 (in the *remote* setup)[1], it is a viable approach if further optimizations are taken into consideration. Such optimizations could involve strategies and techniques depicted in sec. 4.2 and sec. 4.1 of this work, namely *caching* or *query rewriting*. Transferring such concepts to the proposed FVPD approach results in both, faster query times and a privacy and security-aware way of storing such data in RDF-based form.

To sum up the conclusion with respect to the hypothesis expressed in the problem formulation of sec. 8.1, it can be stated that it can be verified. Finally, the query performance is in the same order of magnitude as the results in this section show. Hence, for the given scenario the FVPD approach with data exposed as SPARQL endpoints is a feasible approach with respect to the query performance.

[1] whereas in the *local* setup the performance is similar

Summary and Outlook

Summary

The *SeDiCo* framework, developed by the author of this work and the student works listed in Section 8.4, splits database relations and distributes the partitions across (ideally) different clouds. Moreover, the framework aims at avoiding vendor lock-ins with respect to the used database systems and with respect to the cloud providers through the usage of database abstraction with Hibernate as an Object-relational Mapper (ORM) and through cloud abstraction with *jclouds* as a cloud abstraction layer on the IaaS service layer.

Although *SeDiCo* increases the level of security and privacy, it leads to tremendous performance losses, as the FVPD data have to be joined together again before they are actually accessed. Despite the technological feasibility, the framework was not usable in practical usage scenarios due to the tremendous performance degrade. Hence, this thesis focused on this performance challenge and conceptualized, implemented and evaluated *query mechanisms* for the FVPD partitions. For this, the research problem was defined as a minimization problem with respect to the response time. Moreover, the entire FVPD approach was formalized and proved according to Codds relational model (Codd, 1970).

Furthermore, the current state-of-the-art regarding the key concepts (i.e. security and privacy, Cloud Computing, Database Abstraction (ORM), Query Rewriting, Caching and Benchmarking) were elaborated. Based on this, 3 query mechanisms were conceptualized and developed. Accordingly, the query mechanisms were evaluated against a non-partitioned and non-distributed data set as well as against *SeDiCo*s FVPD implementation. The evaluation, based on the TPC-W benchmark showed that *query rewriting* and *caching* improve the caching compared to the initial *SeDiCo* implementation by factors. Surprisingly, both query mechanisms improved the caching to a level that is in the same order of magnitude as queries against a non-distributed and non-partitioned setup.

This thesis showed that a combination of the query mechanisms would also a viable approach to achieve additional performance gains. The evaluation of this sustainable idea is in the focus of the future work regarding the *SeDiCo* develop-

ment. This leads to the final conclusion that the developed query mechanisms have a remarkable impact on the response time and that this work serves as a guideline for interested practitioners and shows which query mechanism promises which performance gains. Another aspect is the fact that the field of database research is not bound to a concrete application domain and as database workloads cannot be generalized, the *SeDiCo* framework has to be tested and evaluated against various domains and application scenarios. Therefore, this thesis developed a basic performance metric, which can be used to test and evaluate further scenarios and applications and to determine how the FVPD approach affects the response time. The thesis also contains an analysis of the usage of the framework in a Semantic Web application scenario in which the FVPD approach is applied. The preliminary response time evaluation results are promising, however, further optimization strategies have to be elaborated, applied and evaluated in order to reach similar performance results with traditional relational database queries.

List of Publications Related to the Thesis

This section lists (in order of their appearance) the author's publications and their thematic focus to the respective thesis chapters. It can be noted that central aspects and ideas of all chapters have been successfully published and presented at national and international conferences and journals. Finally, this thesis added substantial extensions to them and integrated and structured the publications in a central and thematically focused framework. A corresponding summary of all listed works can be found in the full version of the thesis.

No.	Publication	Ref. to Thesis Chapter
1	Kohler J.; Specht T.; Simov K.: An Approach for a Security and Privacy-Aware Cloud-Based Storage of Data in the Semantic Web. In: Proc. of The First IEEE International Conference on Computer Communication and the Internet (ICCCI 2016). Wuhan, China.	8
2	Werner S., Kohler J.; Specht T.; Simov K.: Cache Synchronization in a Vertically Distributed Cloud Database Environment. In: Proc. of AKWI 2016 - Arbeitskreis Wirtschaftsinformatik an Fachhochschulen, September 2016. Brandenburg, Germany.	4, 5, 6

3	Kohler, J.; Simov, K.; Specht, T.: Analysis of the Join Performance in Vertically Distributed Cloud Databases. In: International Journal of Adaptive, Resilient and Autonomic Systems (IJARAS), 6(2), 2016	1, 2, 3, 4, 5, 6
4	Kohler J.; Specht T.: Analysis of Cache Implementations in a Vertically Distributed Cloud Data Store. In: Proc. of The 3rd IEEE World Conference on Complex Systems, November 2015. Marrakech, Morocco.	4, 5, 6
5	Kohler J.; Simov K.; Fiech A.; Specht T.: On The Performance Of Query Rewriting In Vertically Distributed Cloud Databases. In: Proc. of The International Conference Advanced Computing for Innovation ACOMIN 2015, November 2015. Sofia, Bulgaria.	1, 2, 3, 4, 5, 6
6	Kohler J.; Specht T.: Dynamic Software-Based Scaling In Private Clouds. In: Proc. of AKWI 2015 - Arbeitskreis Wirtschaftsinformatik an Fachhochschulen, September 2015. Luzern, Switzerland.	3
7	Kohler J.; Specht T.: A Performance Comparison Between Parallel And Lazy Fetching in Vertically Distributed Cloud Databases. In: Proc. of The International Conference on Cloud Computing Technologies and Applications - CloudTech 2015, June 2015. Marrakesh, Morocco.	4, 5, 6
8	Kohler J.; Specht T.: Performance Analysis of Vertically Partitioned Data in Clouds Through a Client-Based In-Memory Key-Value Store Cache. In: Proc. of The 8th International Conference on Computational Intelligence in Security for Information Systems, June 2015. Burgos, Spain.	1, 2, 3, 4, 5, 6
9	Kohler J.; Specht T.: Vertical Query-Join Benchmark in a Cloud Database Environment. In: Proc. of The 2nd World Conference on Complex Systems, November 2014. Agadir, Morocco.	1, 2, 3, 4, 5, 6
10	Kohler J.; Specht T.: Analysis of the Join-Problem in Vertically Distributed Databases (in German). In: Proc. of AKWI 2014 - Arbeitskreis Wirtschaftsinformatik an Fachhochschulen, September 2014. Regensburg, Germany.	1, 2, 3, 4
11	Velikova D.; Kohler J.; Gerten R.: Case Study On Financing And Business Development Processes In Technopreneurship. In: Proc. of European Conference on Innovation and Entrepreneurship (ECIE) 2014, September 2014. Belfast, Ireland.	2
12	Kohler J.; Specht T.: Vertical Update-Join Benchmark in a Cloud Database Environment. In: Proc. of WiWiTa 2014. Wismarer Wirtschaftsinformatiktage. June 2014.	4, 5, 6

13	Kohler J.; Specht T.: A Marketplace for the Cloud: Comparison of Data Stores Through QoS/SLA. In: Technologien fuer digitale Innovationen. Springer Verlag 2014. Wiesbaden, Germany.	3
14	Mueller P.; Kohler J.; Specht T.: A Vertical Data Distribution Approach in the Cloud. (in German) In: eJournal of AKWI - Arbeitskreis Wirtschaftsinformatik. Februar 2014. Luzern. ISSN: 2296-4592. http://akwi.hswlu.ch	1, 2, 3, 4, 5, 6
15	Kohler J.; Specht T.: A Marketplace for an Efficient and Transparent XaaS-Evaluation. (in German) In: Proc. of AKWI - Arbeitskreis Wirtschaftsinformatik an Fachhochschulen, September 2013. Friedberg, Germany.	3
16	Kohler J.: *SeDiCo* - Towards a Framework for a Secure and Distributed Cloud Data Store. In: Proc. of Chip-To-Cloud Security Forum, September 2012. Nice, France.	2

List of Theses Supervised by the Author

The following list presents all bachelor theses that were developed during the development of the *SeDiCo* framework supervised by the author. These works represent small development and testing tasks that were helpful for the author to implement and evaluate the FVPD approach.

1. Seminar Paper 06/2016

   Lorenz, Richard: Conceptualization, Implementation and Evaluation of a Vertical Partitioning Approach for NoSQL Document Stores. (in German)

2. Bachelor-Thesis 09/2015

   Taenzer, Martin: Trusted Cloud: A Way Towards a Secure and Trustworthy Cloud. (in German)

3. Bachelor-Thesis 09/2015

   Werner, Stefan: Conceptualization, Implementation and Evaluation of Cache Synchronization Mechanisms in a Vertically Partitioned Cloud Database Application. (in German)

4. Bachelor-Thesis 08/2015

   Heiler, Daniel: Parallel Data Access Through Query-Rewriting in a Vertically Distributed Cloud Database Environment

5. Bachelor-Thesis 07/2015

   Atilgan, Tunahan: Evaluation of Semantic Service Registries for Web Services. (in German)

6. Bachelor-Thesis 05/2015

   Sahin, Huzeyfe: Integration of a Middle-tier Database Cache into a Vertically Partitioned Cloud Architecture. (in German)

7. Bachelor-Thesis 05/2015

   Schmidt, Sonny: Conceptualization and Implementation of an Automated Horizontal Scaling Platform for Cloud-based Database Cluster. (in German)

8. Bachelor-Thesis 01/2015

   Eslengert, Igor: Conceptualization and Implementation of a Web-based and Automated Cloud Service Level Agreement Directory for the IaaS Layer. (in German)

9. Bachelor-Thesis 10/2014

   Hlipala, Christof: Conceptualization and Implementation of a Dynamic Scaling Mechanism for CloudStack. (in German)

10. Bachelor-Thesis 07/2013

    Mueller, Patrick: Vertical Data Partitioning in a Distributed Cloud Architecture. (in German)

11. Bachelor-Thesis 07/2013

    Kaiser, Leon: Framework Evaluation of Cloud Abstraction Layers for the Integration of Distributed Data. (in German)

Approbation of the Results

Research Papers. The above-mentioned list of the author's publication is the result of research papers that were created and published before and during the course of this thesis. All published papers include the attendance and the presentation of the work at the respective conference by the author. The thesis added substantial extensions to these works as outlined in more detail below. Based on the promising results of this thesis further research papers are planned with respect to the topics mentioned in the outlook. Concrete planned works focus on the adaption of the FVPD approach to NoSQL databases (key-value, column, document, and graph stores).

Student Theses. The same holds for the above-mentioned list of theses supervised by the author. These seminar and bachelor theses were conducted during the course of the thesis to investigate further topics that are related (but not in the central focus) to the thesis. Similarly, further student works (and also master theses) are planned based on the results of this thesis and with respect to the author's role as lecturer at the University of Applied Sciences in Mannheim.

Research Projects & Funding. During the course of the thesis, the *SeDiCo* idea with its FVPD approach was funded by the above-mentioned institutions.

Closely related to the results of the thesis and to the above-mentioned topics, further research projects in cooperation with partners from industries and other research groups in national as well as international contexts should be acquired. The results of the thesis build an excellent foundation for additional project proposals in order to further pursue the FVPD idea of *SeDiCo* with funded projects.

Invited Talks. Furthermore, the author was invited by several institutions to talk and present new ideas about Cloud Security based on Partitioned and Distributed Databases. These talks are listed as follows:

1. 10/2016:

   SeDiCo - Query Optimization in Vertically Distributed Databases. Mannheimer Informatik-Kolloquium. Mannheim, Germany 2016. (in German).

2. 12/2014:

   SeDiCo - Towards a Framework for a Secure and Distributed Cloud Data Store. Mannheimer Informatik-Kolloquium. Mannheim, Germany 2014. (in German).

3. 07/2014:

   SeDiCo - Towards a Framework for a Secure and Distributed Cloud Data Store. German Chamber of Industry and Commerce Frankfurt. Frankfurt a. M., Germany 2014. (in German).

4. 01/2014:

   Business Process Modeling Repository. Foundations and Challenges of Change in Ontologies and Databases. University Bolzano, Italy 2014.

5. 07/2013:

   How to Handle Complex Data with Distributed Data Systems in the Cloud. Big Data Conference. Mannheim, Germany 2013. (in German).

6. 01/2013:

   Vertical Database Partitioning in the Cloud. Commit-Workshop Datenmanagement. University of Applied Sciences, Mannheim 2013. (in German).

7. 05/2012:

   Towards a Framework for a Distributed and Secure Cloud Datastore. WiWiTa 2012. Wismarer Wirtschaftsinformatiktage. Wismar 2012. (in German).

The dissemination of this thesis might generate additional attention to *SeDiCo* and its FVPD methodology. Hence, further invited talks might also become possible in the near future.

Key Scientific and Applied Scientific Contributions

With respect to the tasks defined in the introduction of this work, it can be concluded that all tasks have successfully been accomplished with this thesis (cf. Tab. 8.2).

Table 8.2: Successfully Conducted Tasks

Number	Task	Ref. to Thesis Chapter
1	The definition of a methodology for creating an FVPD schema for relational data and a proof of the correctness of the methodology	1
2	The conceptualization of adequate query mechanisms for relational FVPD data sets	4
3	The implementation of these relational query mechanisms in Java	5
4	The evaluation of these relational query mechanisms in terms of their response time	6
5	The comparison of all developed relational query mechanisms against each other and against the initial *SeDiCo* implementation	7
6	The application of the FVPD methodology in the Semantic Web with Resource Description Framework-based (RDF-based) data	8

The expected results and the contribution to the current state-of-the-art can be concluded as follows:

Contribution 1: Definition of a *Security-by-Distribution* Principle for Relational Databases

This thesis picked up the *Security-by-Distribution* approach and formalized it to substantiate it with the relational calculus to have a proper formal theoretical background. This was then used to prove the correctness of the principle and of all developed query mechanisms. After that, the evaluation results were used to formulate the research problem and to develop the thesis' hypotheses. The TPC-W benchmark used in these works was also used to evaluate the query mechanisms in the presented thesis. With the verification of all hypotheses and the formal proofs, this thesis was able to prove that the developed query mechanisms indeed are able to enhance the response time.

Contribution 2: Development of Vertical Query Mechanisms

Due to the above-mentioned performance degrades while querying the vertically distributed data, this thesis developed 3 query strategies to tackle these issues. This thesis also formalized the strategies and with this, substantiated them with a formal background (i.e. the relational calculus). Hence, the correctness of all approaches could be proved and their complexity could be analyzed.

Contribution 3: Integration of Query Mechanisms into the *SeDiCo* Framework

Based on the above-mentioned results, the author was able to integrate them into the overall *SeDiCo* framework, such that it can be applied in a unique and transparent way.

This thesis developed a detailed evaluation with a detailed comparison and a discussion of all query mechanisms.

Contribution 4: Response Time Evaluation and Query Mechanism Classification

With respect to the before-mentioned results, this thesis picked up all evaluations and extended them with a detailed overview and interpreted and concluded the evaluation figures. The query mechanisms could be classified and concrete recommendations can now be given which mechanism is suitable in which use case scenario or for which database workload.

Contribution 5: Transfer of the *SeDiCo* Approach to other Databases

The Semantic Web as an interesting application domain could be identified during the course of this thesis. Here, the adoption of the *Security-by-Distribution* approach to RDF-based data sets which are queried with SPARQL were challenging issues. Accordingly, this thesis conducted a deep analysis of the theoretical background (i.e. relational calculus which is also used for SPARQL as it is closely related to SQL), added a formal correctness proof, and considered its complexity. Hence, it could be proved that the *Security-by-Distribution* approach is also applicable in other application domains.

Contribution 6: Further (Indirect) Related Work

This thesis subsumed the challenging topics *Cloud Computing*, *Security-by-Distribution*, *Performance*, *Scalability*, and *Reliability* under the *SeDiCo* umbrella and adapted the discussed issues to the focus of this thesis.

All in all, the author (and the respective co-authors) were able to publish a total of 16 research papers; the author was able to supervise a total of 11 student works (i.e. bachelor theses and seminar works), and there were 7 invited talks given by the author.

This leads to the outlook where the attention to future work tasks and interesting further research work is drawn.

Outlook

Query performance, besides data security and privacy, is a key issue in today's applications either they are based on physical hardware servers or on virtualized cloud infrastructures. The results of thesis showed that there are viable approaches to achieve an adequate response times. Finally, this pushes the *SeDiCo* framework towards more practical usage scenarios, which involves further research topics, outlined in more detail in the full version of the thesis: Primary Key Challenges, Reporting in Distributed Databases, Partitioning and Distributing at Runtime, Row-based Security, Mobile Platform Integration, CRUD Performance Evaluation, Data Encryption, Data Classification, Data Partitioning, and NoSQL.

References

Apache. (2016). *Apache CloudStack.* Retrieved 2016-02-01, from `https://cloudstack.apache.org/`

Betz, H., Hose, K., & Sattler, K. (2012). Learning from the History of Distributed Query Processing. In *Third international workshop on consuming linked data (cold2012)* (Vol. 905, pp. 15–26). Boston, USA: CEUR-WS.

Carlton, D. (2013). *Cloud Computing 2014: Ready for Real Business?* Retrieved 2014-08-11, from `http://www.mscmalaysia.my/sites/all/themes/mscmalaysia/images/cloud\-page/cloud\-pdf/Morning\_2\_Gartner\_DarrylCarlton.pdf`

Codd, E. F. (1970). A relational model of data for large shared data banks. *Communications of the ACM*, *13*(6), 377–387. doi: 10.1145/362384.362685

Elmasri, R., & Navathe, S. B. (2015). *Fundamentals of Database Systems* (7th editio ed.). London, UK: Pearson Education, UK. doi: 10.1016/S0026-2692(97)80960-3

Gens, F., & Shirer, M. (2013). *IDC Forecasts Worldwide Public IT Cloud Services Spending to Reach Nearly $108 Billion by 2017 as Focus Shifts from Savings to Innovation.* Retrieved 2016-02-01, from `http://www.idc.com/getdoc.jsp?containerId=prUS24298013`

Grund, M., Cudre-Mauroux, P., & Madden, S. (2011). A Demonstration of HYRISE A Main Memory Hybrid Storage Engine. *Proceedings of the VLDB Endowment*, *4*(12), 1434–1437.

Haase, P., Mathäß, T., & Ziller, M. (2010). An evaluation of approaches to federated query processing over linked data. In *I-semantics '10 proceedings of the 6th international conference on semantic systems* (Vol. 5, pp. 1–9). Graz, Austria: ACM. doi: 10.1145/1839707.1839713

Hevner, A., & Chatterjee, S. (2010). *Design Research in Information Systems* (Vol. 22). Boston, MA: Springer US. doi: 10.1007/978-1-4419-5653-8

Hevner, A., March, S., Park, J., & Ram, S. (2004). Design Science in Information Systems Research. *MIS Quarterly*, *28*(1), 75–105. doi: 10.2307/25148625

Hewlett Packard. (2016). *Helios Eucalyptus Website.* Retrieved 2016-02-01, from `http://www8.hp.com/us/en/cloud/helion-overview.html`

Kohler, J., Simov, K., Fiech, A., & Specht, T. (2015). On The Performance Of

Query Rewriting In Vertically Distributed Cloud Databases. In *Proceedings of the international conference advanced computing for innovation acomin 2015.* Sofia, Bulgaria.

Kohler, J., Simov, K., & Specht, T. (2015). Analysis of the Join Performance in Vertically Distributed Cloud Databases. *International Journal of Adaptive, Resilient and Autonomic Systems (IJARAS)*, *6*(2). doi: 10.4018/IJARAS

Kohler, J., & Specht, T. (2012). SeDiCo - Towards a Framework for a Secure and Distributed Datastore in the Cloud. In *Proceedings of chip-to-cloud security forum 2012.* Nice, France.

Kohler, J., & Specht, T. (2014a). Ein Marktplatz für die Cloud: Vergleichbarkeit von Datenspeichern durch QoS-/SLA-Mapping. *Technologien für digitale Innovationen*, *1*(1).

Kohler, J., & Specht, T. (2014b). Vertical Query-Join Benchmark in a Cloud Database Environment. In *Proceedings of the 2nd ieee world conference on complex systems.* Agadir, Marocco.

Kohler, J., & Specht, T. (2014c). Vertical Update-Join Benchmark in a Cloud Database Environment. In *Proceedings of WiWiTa 2014 Wismarer Wirtschaftsinformatiktage. Wismar, Germany* (pp. 159–175). Wismar, Germany.

Kohler, J., & Specht, T. (2015a). Analysis of Cache Implementations in a Vertically Distributed Cloud Data Store. In *Proceedings of the 3rd ieee world conference on complex system.* Marrakesh, Morocco.

Kohler, J., & Specht, T. (2015b). Performance Analysis of Vertically Partitioned Data in Clouds Through a Client-Based In-Memory Key-Value Store Cache. In *Proceedings of the 8th international conference on computational intelligence in security for information systems.* Burgos, Spain: Springer.

Kohler, J., & Specht, T. (2015c). A Performance Comparison Between Parallel And Lazy Fetching in Vertically Distributed Cloud Databases. In *International conference on cloud computing technologies and applications - cloudtech 2015.* Marrakesh, Morocco: IEEE Computer Society.

Krueger, J., Grund, M., Zeier, A., & Plattner, H. (2010). Enterprise application-specific data management. In *Proceedings - ieee international enterprise distributed object computing workshop, edoc* (pp. 131–140). Vitoria, Brazil.

Rakhmawati, N. A., Umbrich, J., Karnstedt, M., Hasnain, A., & Hausenblas, M. (2013). Querying over Federated SPARQL Endpoints - A State of the Art Survey. *arXiv preprint arXiv:1306.1723*. Retrieved from `http://arxiv.org/abs/1306.1723`

RedHat. (2016). *ORM Hibernate Documentation.* Retrieved 2016-02-01, from http://hibernate.org/orm/documentation/5.0/

Son, J. H., & Kim, M. H. (2004). An adaptable vertical partitioning method in distributed systems. *Journal of Systems and Software*, *73*(3), 551–561. doi: 10.1016/j.jss.2003.04.002

TIOBE. (2016). *TIOBE Index.* Retrieved 2016-02-01, from http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html

Appendix A

List of Tables

Appendix B

List of Figures