Horizontal Fragmentation and Replication for Multiple Relaxation Attributes

Lena Wiese

Institute of Computer Science
University of Göttingen
Goldschmidtstraße 7
37077 Göttingen
Germany
lena.wiese@uni-goettingen.de

Abstract. The data replication problem (DRP) describes the task of distributing copies of data records (that is, database fragments) among a set of servers in a distributed database system. For the application of flexible query answering, several fragments can be overlapping (in terms of tuples in a database table). In this paper, we provide a formulation of the DRP for horizontal fragmentations with overlapping fragments; subsequently we devise a recovery procedure based on these fragmentations.

Keywords: Bin Packing Problem with Conflicts (BPPC), Data Replication Problem (DRP), Distributed Database, Fragmentation, Integer Linear Programming (ILP)

1 Introduction

When storing large-scale data sets in distributed database systems, these data sets are usually *fragmented* (that is, partitioned) into smaller subsets and these subsets are distributed over several database servers. Moreover, to achieve better availability and failure tolerance, copies of the data sets (the so-called *replicas*) are created and stored in a distributed fashion so that different replicas of the same data set reside on distinct servers.

In addition to technical requirements of data distribution, *intelligent query answering* mechanisms are increasingly important to find relevant answers to user queries. Flexible (or cooperative) query answering systems help users of a database system find answers related to his original query in case the original query cannot be answered exactly. *Semantic* techniques rely on taxonomies (or ontologies) to replace some values in a query by others that are closely related according to the taxonomy. This can be achieved by techniques of *query relaxation* – and in particular *query generalization*: the user query is rewritten into a weaker, more general version to also allow related answers.

In this paper we make the following contributions:

– instead of fixing a single relaxation attribute we allow multiple relaxation attributes which lead to several different fragmentations in which fragments from different fragmentations may share common tuples (they "overlap");

- we devise an m-copy replication scheme for the fragments ensuring the replication factor m by storing overlapping fragments on distinct servers;
- we state the replication problem as an optimization problem minimizing the number of occupied servers;
- we describe a recovery procedure for this kind of replication.

1.1 Organisation of the article

Section 2 introduces the main notions used in this article and gives an illustrative example. Section 3 defines the problem of data replication with overlapping fragments addressed in this article. Section 4 describes replication and recovery in a practical system. Related work is presented in Section 5 and Section 6 concludes this article with suggestions for future work.

2 Background and example

We provide background on query generalization, fragmentation and replication.

2.1 Query generalization

Query generalization has long been studied in flexible query answering [8]. Query generalization at runtime has been implemented in the CoopQA system [5] by applying three generalization operators to a conjunctive query. *Anti-Instantiation* (AI) is one query generalization operator that replaces a constant (or a variable occurring at least twice) in a query with a new variable y. In this paper we focus on replacements of constants because this allows for finding answers that are semantically close to the replaced constant. As the query language we focus on conjunctive queries expressed as logical formulas. We assume a logical language $\mathscr{L}$ consisting of a finite set of predicate symbols (denoting the table names; for example, *Ill*, *Treat* or P), a possibly infinite set *dom* of constant symbols (denoting the values in table cells; for example, *Mary* or a), and an infinite set of variables (x or y). A term is either a constant or a variable. The capital letter X denotes a vector of variables; if the order of variables in X does not matter, we identify X with the set of its variables and apply set operators – for example we write $y \in X$.

A query formula Q is a conjunction (denoted $\wedge$) of literals (consisting of a predicate and terms) with a set of variables X occurring freely; hence we write a query as $Q(X) = L_{i_1} \wedge \ldots \wedge L_{i_n}$. The Anti-Instantiation (AI) operator chooses a constant a in a query $Q(X)$, replaces one occurrence of a by a new variable y and returns the query $Q^{AI}(X, y)$ as the relaxed query. The relaxed query Q^{AI} is a deductive generalization of Q (see [5]).

As a running example, we consider a hospital information system that stores illnesses and treatments of patients as well as their personal information (like address and age) in the following three database tables:

Ill	*PatientID*	*Diagnosis*
	8457	Cough
	2784	Flu
	2784	Asthma
	2784	brokenLeg
	8765	Asthma
	1055	brokenArm

Info	*PatientID*	*Name*	*Address*
	8457	Pete	Main Str 5, Newtown
	2784	Mary	New Str 3, Newtown
	8765	Lisa	Main Str 20, Oldtown
	1055	Anne	High Str 2, Oldtown

The query $Q(x_1, x_2, x_3) = \mathit{Ill}(x_1, \mathit{Flu}) \wedge \mathit{Ill}(x_1, \mathit{Cough}) \wedge \mathit{Info}(x_1, x_2, x_3)$ asks for all the patient IDs x_1 as well as names x_2 and addresses x_3 of patients that suffer from both flu and cough. This query fails with the given database tables as there is no patient with both flu and cough. However, the querying user might instead be interested in the patient called Mary who is ill with both flu and asthma. We can find this informative answer by relaxing the query condition *Cough* and instead allowing other related values (like Asthma) in the answers. An example generalization with AI is $Q^{AI}(x_1, x_2, x_3, y) = \mathit{Ill}(x_1, \mathit{Flu}) \wedge \mathit{Ill}(x_1, y) \wedge \mathit{Info}(x_1, x_2, x_3)$ by introducing the new variable y. It results in an non-empty (and hence informative) answer: $\mathit{Ill}(2748, \mathit{Flu}) \wedge \mathit{Ill}(2748, \mathit{Asthma}) \wedge \mathit{Info}(2748, \mathit{Mary}, \text{`}\mathit{New\ Str\ 3}, \mathit{Newtown}\text{'})$. Another answer obtained is the fact that Mary suffers from a broken leg as: $\mathit{Ill}(2748, \mathit{Flu}) \wedge \mathit{Ill}(2748, \mathit{brokenLeg}) \wedge \mathit{Info}(2748, \mathit{Mary}, \text{`}\mathit{New\ Str\ 3}, \mathit{Newtown}\text{'})$ which is however an overgeneralization.

2.2 Clustering-based fragmentation

Query generalization at runtime is highly inefficient. That is why we propose a clustering-based fragmentation that preprocesses data into fragments of closely related values (with respect to a relaxation attribute). This clustering-based fragmentation has two main advantages:

- it enables efficient query relaxation at runtime by returning all values in a matching fragment as relevant answers
- it reduces the amount of servers contacted during query answering in a distributed environment because only one server (containing the matching fragment) has to process the query while other servers can process other queries.

Here we need semantic guidance to identify the set of relevant answers that are close enough to the original query. In previous work [12], a clustering procedure was applied to partition the original tables into fragments based on a *single relaxation attribute* chosen for anti-instantiation. For this we used a notion of similarity between to constants; this similarity can be deduced with the help of an ontology or taxonomy in which the values are put into relation. Finding the fragments is hence achieved by grouping (that is, *clustering*) the values of the respective table column into clusters of closely related values and then splitting the table into fragments according to the clusters found. For example, clusters on the *Diagnosis* column can be made by differentiating between fractures on the one hand and respiratory diseases on the other hand. These clusters then lead to two fragments of the table *Ill* that could be assigned to two different servers:

Server 1:

Respiratory	*PatientID*	*Diagnosis*
	8457	Cough
	2784	Flu
	2784	Asthma
	8765	Asthma

Server 2:

Fracture	*PatientID*	*Diagnosis*
	2784	brokenLeg
	1055	brokenArm

Server 1 can then be used to answer queries related to respiratory diseases while Server 2 can process queries related to fractures. The example query $Q(x_1, x_2, x_3) = \mathit{Ill}(x_1, \mathit{Flu}) \wedge \mathit{Ill}(x_1, \mathit{Cough}) \wedge \mathit{Info}(x_1, x_2, x_3)$ will then be rewritten as $Q^{Resp}(x_1, x_2, x_3, y) = \mathit{Respiratory}(x_1, \mathit{Flu}) \wedge \mathit{Respiratory}(x_1, \mathit{Cough}) \wedge \mathit{Info}(x_1, x_2, x_3)$ and redirected to Server 1 where only the fragment Respiratory is used to answer the query. In this way only the informative answer containing asthma is returned – while the one containing broken leg will not be generated.

2.3 Data distribution as a Bin Packing Problem

In a distributed database system data records have to be assigned to different servers. The **data distribution problem** – however not considering replication yet – is basically a Bin Packing Problem (BPP) in the following sense:

- K servers correspond to K bins
- bins have a maximum capacity W
- n data records correspond to n objects
- each object has a weight (a capacity consumption) $w_i \leq W$
- objects have to be placed into a minimum number of bins without exceeding the maximum capacity W

This BPP can be written as an integer linear program (ILP) as follows – where x_{ik} is a binary variable that denotes whether fragment/object i is placed in server/bin k; and y_k denotes that server/bin k is used (that is, is non-empty). Moreover, each server/bin has a maximum capacity W and each fragment/object i has a weight w_i that denotes how much capacity the item consumes. As a simple example, W can express how many rows (tuples) a server can store and w_i is the row count of fragment i.

$$\text{minimize} \sum_{k=1}^{K} y_k \qquad \text{(minimize number of bins)} \tag{1}$$

$$s.t. \sum_{k=1}^{K} x_{ik} = 1, \;\; i = 1, \ldots, n \quad \text{(each object assigned to one bin)} \tag{2}$$

$$\sum_{i=1}^{n} w_i x_{ik} \leq W y_k, \; k = 1, \ldots, K \quad \text{(capacity not exceeded)} \tag{3}$$

$$y_k \in \{0, 1\} \;\; k = 1, \ldots, K \tag{4}$$

$$x_{ik} \in \{0, 1\} \; k = 1, \ldots, K, i = 1, \ldots, n \tag{5}$$

An extension of the basic BPP, the Bin Packing with Conflicts (BPPC) problem, considers a conflict graph $G = (V, E)$ where the node set $V = \{1, \ldots, n\}$ corresponds to the set of objects. A binary edge $e = (i, j)$ exists whenever the two objects i and j must *not* be placed into the same bin. In the ILP representation, a further constraint (Equation 9) is added to avoid conflicts in the placements.

$$\text{minimize} \sum_{k=1}^{K} y_k \quad \text{(minimize number of bins)} \tag{6}$$

$$s.t. \sum_{k=1}^{K} x_{ik} = 1, \;\; i = 1, \ldots, n \quad \text{(each object assigned to one bin)} \tag{7}$$

$$\sum_{i=1}^{n} w_i x_{ik} \leq W y_k, \; k = 1, \ldots, K \quad \text{(capacity not exceeded)} \tag{8}$$

$$x_{ik} + x_{jk} \leq y_k \; k = 1, \ldots, K, \forall (i, j) \in E \quad \text{(no conflicts)} \tag{9}$$

$$y_k \in \{0, 1\} \; k = 1, \ldots, K \tag{10}$$

$$x_{ik} \in \{0, 1\} \; k = 1, \ldots, K, \; i = 1, \ldots, n \tag{11}$$

Several results were obtained regarding hardness and approximation of bin packing with conflicts. BPPC can basically be regarded as a combination of a vertex coloring and the basic BPP: first of all, compute a coloring of the conflict graph such that items of different color cannot be placed in the same bin, then solve one classical BPP instance for each color.

3 Overlaps and multiple relaxation attributes

So far, for the taxonomy-based clustering approach only a *single* relaxation attribute has been considered in [12]. There it is proposed that, when doing m-way replication, simply m copies of the fragments obtained for the single relaxation attribute are replicated; this corresponds to solving a BPPC instance where the conflict graph states that copies of the same fragment cannot be placed on the same server. In this paper we want to generalize this procedure to multiple relaxation attributes. This has the following advantages:

- The system can answer queries for several relaxation attributes.
- The intelligent replication procedure reduces storage consumption and hence the amount of servers that are needed for replication.

In order to support flexible query answering on multiple columns, one table can be fragmented multiple times (by clustering different columns); that is, we can choose more than one relaxation attribute. In this case, several fragmentations will be obtained. More formally, if α relaxation attributes are chosen and clustered, then we obtain α fragmentations F_l $(l = 1 \ldots \alpha)$ of the same table; each fragmentation contains fragments $f_{l,s}$ where index s depends on the number of clusters found: if n_l clusters are found, then $F_l = \{f_{l,1}, \ldots, f_{l,n_l}\}$.

For example, clusters on the `Diagnosis` column can be made by differentiating between fractures on the one hand and respiratory diseases on the other hand. And additionally, clusters on the patient ID can be obtained by dividing into rows with ID smaller than 5000 and those with ID larger than 5000.

Respiratory	*PatientID*	*Diagnosis*
	8457	Cough
	2784	Flu
	2784	Asthma
	8765	Asthma

Fracture	*PatientID*	*Diagnosis*
	2784	brokenLeg
	1055	brokenArm

IDlow	*PatientID*	*Diagnosis*
	2784	Flu
	2784	brokenLeg
	2784	Asthma
	1055	brokenArm

IDhigh	*PatientID*	*Diagnosis*
	8765	Asthma
	8457	Cough

We assume that each of the clusterings (and hence the corresponding fragmentation) is *complete*: every value in the column is assigned to one cluster and hence every tuple is assigned to one fragment. We also assume that each clustering and each fragmentation are *non-redundant*: every value is assigned to exactly one cluster and every tuple belongs to exactly one fragment (for one clustering); in other words, the fragments inside one fragmentation do not overlap.

However, fragments from two *different* fragmentations (for two different clusterings) may overlap. For example, both the *Respiratory* as well as the *IDhigh* fragments contain the tuple $\langle 8457, Cough \rangle$. Due to completeness, every tuple is contained in exactly one of the fragments of each of the α fragmentations: for any tuple j, if α relaxation attributes are chosen and clustered, then in any fragmentation F_l $(l = 1 \dots \alpha)$ there is a fragment $f_{l,s}$ such that tuple $j \in f_{l,s}$.

3.1 Data replication for overlapping fragments

The main contribution of this paper is to analyze intelligent data replication schemes with *multiple* relaxation attributes while at the same time minimizing the amount of data copies – and hence reducing overall storage consumption of the underlying flexible query answering system. The approach is as follows:

- Apply the above clustering heuristics to *any* of the α relaxation attributes.
- Based on each clustering obtain a complete fragmentation of the given table.
- Fragments of different fragmentations (for different clusterings) overlap.
- Ensure replication factor m for tuples by considering these overlaps in BPPC.

While in the standard BPP and BPPC representations usually disjoint fragments and exactly m copies are considered, we extend the basic BPPC as follows:

Conjecture 1. With our intelligent replication procedure, less data copies (only m copies of each tuple) have to be replicated hence reducing the amount of storage needed for replication as opposed to conventional replication approaches that replicate m copies for each of the α fragmentations F_l (which results in αm copies of each tuple).

We argue that m copies of a tuple suffice with an advanced recovery procedure: that is, for every *tuple* j we require that it is stored at m *different* servers for backup purposes but these copies of j may be contained in different fragments: one fragmentation F_l can be recovered from fragments in any other fragmentation F'_l (where $l \neq l'$). From here on we assume that there are **exactly** m relaxation attributes (that is, $\alpha = m$); in case there are less than m relaxation attributes, some of the existing fragmentations are simply duplicated; in case there are more than m relaxation attributes, the remaining fragmentations can be stored on arbitrary servers. We hence consider the following problem:

Definition 1 (Data replication problem with overlapping fragments (overlap-DRP)). *Given m fragmentations $F_l = \{f_{l,1}, \ldots, f_{l,n_l}\}$ and replication factor m, for every tuple j there must be fragments f_{l,i_l} (where $1 \leq l \leq m$ and $1 \leq i_l \leq n_l$) such that $j \in f_{1,i_1} \cap \ldots \cap f_{\alpha,i_m}$ and these fragments are all assigned to different servers.*

We illustrate this with our example. Assume that 5 rows is the maximum capacity W of each server and assume a replication factor 2. In a conventional replication approach, all fragments are of approximately the same size and do not overlap. Hence, the conventional approach would replicate all fragments (*Respiratory*, *Fracture*, *IDhigh*, *IDlow*) to two servers each. then assign the *Respiratory* fragment (with 4 rows) to one server $S1$ and a copy of it to another server $S2$. Now the *Fracture* fragment (with 2 rows) will not fit on any of the two servers; its two replicas will be stored on two new servers $S3$ and $S4$. For storing the *IDlow* fragment (with 4 rows), the conventional approach would need two more servers $S5$ and $S6$. The *IDhigh* fragment (with 2 rows) could then be mapped to servers $S3$ and $S4$. The conventional replication approach would hence require at least six servers to achieve a replication factor 2.
In contrast, our intelligent replication approach takes advantage of the overlapping fragments so that **three** servers suffice to fulfill the replication factor 2; that is, the amount of servers can be substantially reduced if a more intelligent replication and recovery scheme is used that respects the fact that several fragments overlap and that can handle fragments of differing size to optimally fill remaining server capacities. This allows for better self-configuration capacities of the distributed database system. First we observe how one fragment can be recovered from the other fragments: Fragment *Respiratory* can be recovered from fragments *IDlow* and *IDhigh* (because *Respiratory*= (*IDlow* $\cap$ *Respiratory*) $\cup$ (*IDhigh* $\cap$ *Respiratory*)); Fragment *Fracture* can be recovered from fragment *IDlow* (because *Fracture*= (*IDlow* $\cap$ *Fracture*)); Fragment *IDlow* can be recovered from fragments *Respiratory* and *Fracture* (because *IDlow*= (*IDlow* $\cap$ *Respiratory*) $\cup$ (*IDlow* $\cap$ *Fracture*)); Fragment *IDhigh* can be recovered from fragment *Respiratory* (because *IDhigh*= (*IDhigh* $\cap$ *Respiratory*)). Hence, we can store fragment *Respiratory* on server $S1$, fragment *IDlow* on server $S2$, and fragments *Fracture* and *IDhigh* on server $S3$ and still have replication factor 2 for individual *tuples*.

We now show that our replication problem (with its extensions to overlapping fragments and counting replication based on tuples) can be expressed as an

advanced BPPC problem. Let J be the amount of tuples in the input table, m be the number of fragmentations, K the total number of available servers and n be the overall number of fragments obtained in all fragmentations. In the ILP representation we keep the variables y_k for the bins and x_{ik} for fragments – to simplify notation we assume that $i = 1 \dots n$ where $n = |F_1| + \dots + |F_m| = n_1 + \dots + n_m$: all fragments are numbered consecutively from 1 to n even when they come from different fragmentations. In addition, we introduce K new variables z_{jk} for each the tuple j such that $z_{jk} = 1$ if the tuple j is placed on server k; we maintain a mapping between fragments and tuples such that if fragment i is assigned to bin k, and j is contained in i, then tuple j is also assigned to k (see Equation (15)); the other way round, if there is no fragment i containing j and being assigned to bin k, then tuple j neither is assigned to k (see Equation (16)); and we modify the conflict constraint to support the replication factor: we require that for each tuple j the amount of bins/servers used is at least m (see Equation (17)) to ensure the replication factor.

$$\text{minimize} \sum_{k=1}^{K} y_k \qquad \text{(minimize number of bins)} \tag{12}$$

$$\text{s.t.} \sum_{k=1}^{K} x_{ik} = 1, \quad i = 1, \dots, n \qquad \text{(each fragment } i \text{ assigned to one bin)} \tag{13}$$

$$\sum_{i=1}^{n} w_i x_{ik} \leq W y_k, \quad k = 1, \dots, K \qquad \text{(capacity not exceeded)} \tag{14}$$

$$z_{jk} \geq x_{ik} \text{ for all } j : j \in i \qquad \text{(tuple } j \text{ in bin when fragment } i \text{ is)} \tag{15}$$

$$z_{jk} \leq \sum_{(i:j \in i)} x_{ik} \qquad \text{for all } j \qquad \text{(tuple not in bin when no fragment is)} \tag{16}$$

$$\sum_{k=1}^{K} z_{jk} \geq m \qquad \text{for all } j \qquad \text{(replication factor } m \text{ on tuples)} \tag{17}$$

$$y_k \in \{0, 1\} \quad k = 1, \dots, K \tag{18}$$

$$x_{ik} \in \{0, 1\} \quad k = 1, \dots, K, \; i = 1, \dots, n \tag{19}$$

$$z_{jk} \in \{0, 1\} \quad k = 1, \dots, K, \; j = 1, \dots, J \tag{20}$$

This ILP will find a valid solution to overlap-DRP.

3.2 Reducing the amount of variables

The ILP representation in the previous section is highly inefficient and does not scale to large amounts of tuples: due to the excessive use of z-variables, for large J finding a solution will take prohibitively long. Indeed, in the given representation, we have K y-variables, $n \cdot K$ x-variables, and $J \cdot K$ z-variables where usually $J \gg n$. That is why we want to show now that it is possible to focus on the x-variables to achieve another ILP representation for overlap-DRP:

for any tuple j such that j is contained in two fragments i and i' (we assume that $i < i'$ to avoid isomorphic statements in the proof), it is sufficient to ensure that the two fragments are stored on two different servers. In other words, for the $(m \cdot (m-1))/2$ pairs of overlapping fragments i and i', we can make them mutually exclusive in the ILP representation; that is, in the ILP representation we have to satisfy $(m \cdot (m-1))/2$ equalities of the form $x_{ik} + x_{i'k} = 1$ to make them pairwise conflicting.

Theorem 1. *If for any two fragments i and i' such that $i \cap i' \neq \emptyset$ there hold $(m \cdot (m-1))/2$ equations of the form $x_{ik} + x_{i'k} = 1$ where $i < i'$, $i = 1, \ldots, n-1$, $i' = 2, \ldots, n$ and $k = 1, \ldots, K$, then it holds for any tuple j that $\sum_{k=1}^{K} z_{jk} \geq m$.*

Proof. First of all, for every tuple j there are m fragments i such that $j \in i$ due to completeness of the m fragmentations. Now we let I be the set of these m fragments. Then for any two $i, i' \in I$ we have $j \in i \cap i'$ by construction. Due to Equation (13), for every $i \in I$ there must be exactly one bin k such that $x_{ik} = 1$ and for all other i^* it holds that either $x_{ik} + x_{i^*k} = 1$ (if $i < i^*$) or $x_{i^*k} + x_{ik} = 1$ (if $i^* < i$) so that none of these fragments is assigned to bin k. Hence, m bins are needed to accommodate all fragments in I. Due to Equation (15), we assure that when $x_{ik} = 1$ then also $z_{jk} = 1$ for the given j and any $i \in I$. Hence $\sum_{k=1}^{K} z_{jk} \geq m$ (Equation 17) holds.

Instead of considering all individual tuples j, we can now move on to considering only overlapping fragments (with non-empty intersections) and requiring the $(m \cdot (m-1))/2$ equations to hold for each pair of overlapping fragments. We transform the previous ILP representation into the one that enforces a conflict condition for any two overlapping fragments. This coincides with the conventional BPPC representation, where the conflict graph is built over the set of fragments (as the vertex set) by drawing an edge between any two fragments that overlap.

Definition 2 (Conflict graph for overlap-DRP). *The conflict graph $\mathscr{G}^{DRP} = (V, E)$ is defined by $V = F_1 \cup \ldots \cup F_m$ (one vertex for each fragment inside the m fragmentations) and $E = \{(i, i') \mid i, i' \in V \text{ and } i \cap i' \neq \emptyset\}$ (an undirected edge between fragments that overlap).*

Continuing our example, we have a conflict graph over the fragments *Respiratory*, *Fracture*, *IDlow* and *IDhigh* with an edge between *Respiratory* and *IDlow*, and an edge between *Respiratory* and *IDhigh*, and an edge between *Fracture* and *IDhigh*. The ILP representation for overlap-DRP looks now as follows:

$$\text{minimize} \sum_{k=1}^{K} y_k \qquad \text{(minimize number of bins)} \tag{21}$$

$$\text{s.t.} \sum_{k=1}^{K} x_{ik} = 1, \;\; i = 1, \ldots, n \quad \text{(each fragment } i \text{ assigned to one bin)} \tag{22}$$

$$\sum_{i=1}^{n} w_i x_{ik} \leq W y_k, \; k = 1, \ldots, K \quad \text{(capacity not exceeded)} \tag{23}$$

$$x_{ik} + x_{i'k} \leq y_k \; k = 1, \ldots, K, \, i \cap i' \neq \emptyset \text{ (overlapping fragments } i, i') \quad (24)$$

$$y_k \in \{0, 1\} \; k = 1, \ldots, K \quad (25)$$

$$x_{ik} \in \{0, 1\} \; k = 1, \ldots, K, \, i = 1, \ldots, n \quad (26)$$

4 Experimental Study

Our prototype implementation – the OntQA-Replica system – runs on a distributed SAP HANA installation which is an in-memory database system and hence needs a good replication strategy that also reduces the amount of servers needed. The example data set consists of a table that resembles a medical health record and is based on the set of Medical Subject Headings (MeSH [11]). The table contains as columns an artificial, sequential tuple ID, a random patient ID, and a disease chosen from the MeSH data set as well as the concept identifier of the MeSH entry. We varied the table sizes during our test runs. The smallest table consists 56,341 rows, a medium-sized table of 1,802,912 rows and the largest of 14,423,296 rows. A clustering is executed on the MeSH data based on the concept identifier (which orders the MeSH terms in a tree); in other words, entries from the same subconcept belong to the same cluster. One fragmentation (the "clustered" fragmentation) was obtained from this clustering and consists of 117 fragments; these fragments have a column called clusterid. Another fragmentation (the "range-based" fragmentation) is based on ranges of the patient ID and consists of 6 fragments for the small table, 19 for the medium-sized table and 145 for the large table; these fragments have a column called rangeid.

For the replication procedure, first the overlapping fragments (the "conflicts") are identified by using `SELECT DISTINCT clusterid, rangeid FROM` c_i `JOIN` c_i `ON (`r_j`.tupleid=`r_j`.tupleid)` for each clustered fragment c_i and each range-based fragment r_j.

Afterwards from the conflicts the overlap-DRP ILP is generated and solved. For the small table, the input had 1820 constraints on 1240 variables; for the medium-sized table, the input had 5720 constraints on 1370 variables; for the large table, the input had 43520 constraints on 2630 variables. Based on the ILP solution, the fragments are moved to different servers by using `ALTER TABLE` c_i `MOVE TO '`*severname*`' PHYSICAL`.

To enable recovery, a lookup table is maintained that stores for each clusterid the tupleids of those tuples that constitute the clustered fragment. The recovery procedure was executed on the range-based fragmentation to recover the clustered fragmentation by running `INSERT INTO` c_i `SELECT * FROM` $r_1, \ldots, r_m$ `JOIN lookup on (lookup.tupleid =` c_i`.tupleid) WHERE lookup.clusterid=`i for each cluster i. The runtimes obtained are shown in Figure 1.

5 Related Work

There is a long history of fragmentation approaches for the relational data model. Most approaches consider workload-aware fragmentation (see for example, [1, 4,

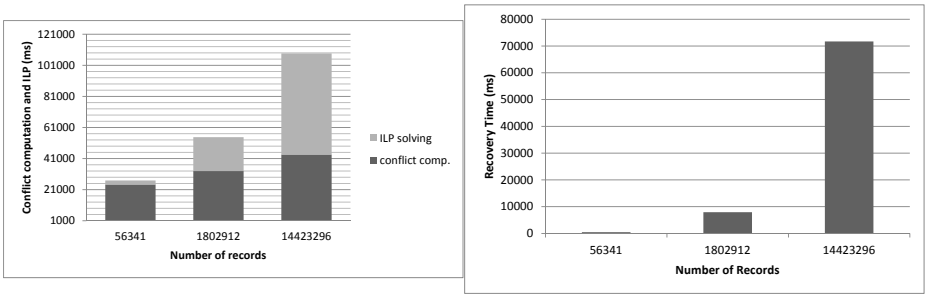

Fig. 1. Runtimes of replication computation and recovery

9]) that optimize distribution of data for a given workload of queries. However none of these approaches consider semantical similarity of values inside a fragment as is needed for our approach of query relaxation.

Bin packing is one of the classical NP-complete problems and it has been shown to be APX-hard (it is not approximable with a ratio less than 1.5; see [6]). As BPP is a special case of the more general BPPC, these properties carry over to BPPC as well. Some variants of classical bin packing have been surveyed in [2]. One of the primary sources of BPPC is [6]. However, as the number of fragments we consider in our overlap-DRP is comparably low, these complexity theoretic considerations usually do not affect the practical implementation and any off-the-shelf ILP solver will find an optimal solution.

There is also related work on specifying resource management problems as optimization problems. An adaptive solution for data replication using a genetic algorithm is presented in [7]; they also consider transfer cost of replicas between servers. Virtual machine placement is a very recent topic in cloud computing [10, 3]. However, these specifications do not address the problem of overlapping resources as we need for the query relaxation approach in this article.

6 Conclusion and Future Work

We presented and analyzed a data replication problem for a flexible query answering system. It provides related answers by relaxing the original query and obtaining a set of semantically close answers. The proposed replication scheme allows for fast response times due to materializing the fragmentations. By solving an ILP representation of the data replication problem, we minimize the overall number of servers used for replication. In this paper the focus lies on supporting multiple relaxation attributes that lead to multiple fragmentations of the same table. A minimization of the number of servers is due to the fact that one fragmentation can be recovered from other fragmentations based on overlapping fragments. The experimental evaluation shows sensible performance results.

Future work has to mainly address dynamic changes in the replication scheme. Deletions and insertions of data lead to changing fragmentations sizes and hence an adaptation of the server allocations might become necessary (similar to [7]). The use of adaptive methods will be studied where (a large part of) a previous solution might be reused to obtain a new solution. Another approach is to compute the common subfragments (intersections) of overlapping fragments and use these subfragments as a unit of replication. Copies of these subfragments will hence be distributed among the servers.

6.1 Acknowledgements

The author gratefully acknowledges that the infrastructure and SAP HANA installation for the test runs was provided by the Future SOC Lab of Hasso Plattner Institute (HPI), Potsdam.

References

1. Agrawal, S., Narasayya, V., Yang, B.: Integrating vertical and horizontal partitioning into automated physical database design. In: Proceedings of the 2004 ACM SIGMOD international conference on Management of data. pp. 359–370. ACM (2004)
2. Coffman Jr, E.G., Csirik, J., Leung, J.Y.T.: Variants of classical one-dimensional bin packing. Handbook of Approximation Algorithms and Meta-Heuristics, Francis and Taylor Books (CRC Press), London (2007)
3. Goudarzi, H., Pedram, M.: Energy-efficient virtual machine replication and placement in a cloud computing system. In: IEEE 5th International Conference on Cloud Computing (CLOUD). pp. 750–757. IEEE (2012)
4. Grund, M., Krüger, J., Plattner, H., Zeier, A., Cudre-Mauroux, P., Madden, S.: Hyrise: a main memory hybrid storage engine. Proceedings of the VLDB Endowment 4(2), 105–116 (2010)
5. Inoue, K., Wiese, L.: Generalizing conjunctive queries for informative answers. In: Flexible Query Answering Systems. pp. 1–12. Springer (2011)
6. Jansen, K., Öhring, S.: Approximation algorithms for time constrained scheduling. Information and Computation 132(2), 85–108 (1997)
7. Loukopoulos, T., Ahmad, I.: Static and adaptive distributed data replication using genetic algorithms. Journal of Parallel and Distributed Computing 64(11), 1270–1285 (2004)
8. Michalski, R.S.: A theory and methodology of inductive learning. Artificial Intelligence 20(2), 111–161 (1983)
9. Özsu, M.T., Valduriez, P.: Principles of distributed database systems. Springer Science & Business Media (2011)
10. Shi, W., Hong, B.: Towards profitable virtual machine placement in the data center. In: Fourth IEEE International Conference on Utility and Cloud Computing (UCC). pp. 138–145. IEEE (2011)
11. U.S. National Library of Medicine: Medical subject headings, http://www.nlm.nih.gov/mesh/
12. Wiese, L.: Clustering-based fragmentation and data replication for flexible query answering in distributed databases. Journal of Cloud Computing 3(1), 1–15 (2014)