Ein Replikationsschema für multiple Fragmentierungen mit überlappenden Fragmenten

Ferdinand Bollwein
Institute of Computer Science
University of Göttingen
Goldschmidtstraße 7
37077 Göttingen, Germany
ferdinand.bollwein@stud.uni-goettingen.de

Lena Wiese
Institute of Computer Science
University of Göttingen
Goldschmidtstraße 7
37077 Göttingen, Germany
wiese@cs.uni-goettingen.de

ABSTRACT

In diesem Artikel stellen wir ein Replikationsverfahren für verteilte Datenbanksysteme vor, das multiple Fragmentierungen derselben Datentabelle unterstützt. Solche multiplen Fragmentierungen können beispielsweise für eine flexible Anfragebeantwortung ausgenutzt werden. Die Besonderheit unseres Ansatzes liegt darin, dass bei der Replikation und Wiederherstellung der Tabellen die Überschneidungen von Fragmenten, die aus unterschiedlichen Fragmentierungen entstehen, berücksichtigt werden, um so die Anzahl der benötigten Server zu reduzieren. Wir betrachten insbesondere den Fall, bei dem mehr Fragmentierungen als der gewünschte Replikationsfaktor existieren, sodass nur ein Teil der Replikationsbedingungen notwendigerweise erfüllt werden müssen und die restlichen optional sind.

Keywords

Behälterproblem mit Konflikten (BPPC), Datenreplikationsproblem (DRP), Fragmentierung, Verteilte Datenbanksysteme, Ganzzahlige lineare Optimierung

1. EINLEITUNG

Um große Datenmengen in verteilten Datenbanksystemen zu speichern, werden diese normalerweise in kleinere Teilmengen fragmentiert und dann auf mehrere Server verteilt. Darüber hinaus werden, um bessere Verfügbarkeit und Fehlertoleranz zu garantieren, Kopien der Datensätze erstellt und auf unterschiedlichen Servern gespeichert. Bisherige Arbeiten zu diesem Thema konzentrieren sich meist nur auf eine einzelne optimale Fragmentierung der Daten. In unserem Ansatz hingegen betrachten wir multiple Fragmentierungen, um anschließend eine Replikation der Fragmente zu finden, die Überlappungen berücksichtigt und so die Anzahl der benötigten Server reduziert. Dies kann beispielsweise zur flexiblen Anfragebeantwortung benutzt werden, was in [Wie14, Wie15a, Wie15b] untersucht wird. In diesem Artikel werden wir diese Arbeiten nun erweitern, indem wir beim m-Kopien-Replikationsproblem mehr Fragmentierungen als der Replikationsfaktor zulassen ($r > m$), sodass einige Replikationsbedingungen notwendig und andere optional sind.

1.1 Verwandte Arbeiten

Fragmentierung relationaler Tabellen ist ein seit langer Zeit untersuchtes Problem und Teil von Standardlehrbüchern wie [ÖV11]. Einige Ansätze setzen vertikale Fragmentierung um und betrachten Affinität von Attributen (innerhalb einer vorgegebenen Menge von Anfragen) als Optimierungsmerkmal. Eine vergleichende Evaluation mehrerer Ansätze zur vertikalen Fragmentierung findet sich in [JPPD13]. Im Gegensatz zu diesen Ansätzen setzen wir auf horizontale Fragmentierung für große Datensätze. Mit Bezug auf horizontale Fragmentierung wird üblicherweise eine einzige optimale Fragmentierung gesucht. Beispiele dafür sind [BK15] für Multiple Query Optimization (MQO), [CZJM10] zur Partitionierung anhand eines Graphen über einer Menge von Anfragen oder [TPRH16] zur Reduzierung von Abhängigkeiten zwischen Partitionen. Im Gegensatz dazu toleriert unser Ansatz mehrere Fragmentierungen und passt die Replikation den Überlappungen an. Das heißt, die existierenden Ansätze zum Finden einer *einzelnen* horizontalen Fragmentierung können mit unserem Ansatz *kombiniert* werden. Damit verbessern wir die Laufzeit für Bereichsabfragen durch Vermeiden unnötiger Vereinigungsoperationen. Zahlreiche Datenbanksysteme bieten zwar die automatische Fragmentierung an (so etwa der IBM DB2 Database Advisor [ZRL$^+$04], der Vertica DBDesigner [VBC$^+$14] oder Oracles partitioning by reference [ECS$^+$08]), jedoch unterstützen auch sie nur jeweils eine einzige Fragmentierung.

Datenreplikation ist ein zentraler Aspekt in verteilten Datenbanksystemen. Eine Übersicht über Optimierungsstrategien und Forschungsfragen für Replikationsverfahren gibt es in [KPX$^+$11, SPTB14]. Keines dieser Verfahren betrachtet jedoch mehrere Fragmentierungen. Wir benutzen gemeinsame Teilfragmente, mit denen eine Fragmentierung aus einer anderen Fragmentierung wiederhergestellt werden kann.

1.2 Übersicht

In Abschnitt 2 werden die Hintergründe zu Fragmentierung und Datenverteilung beschrieben. Darüber hinaus stellen wir das Replikationsproblem für eine einzelne Fragmentierung vor. Anschließend erweitern wir in Abschnitt 3 das

28^{th} GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken), 24.05.2015 - 27.05.2015, Nörten-Hardenberg, Germany.

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

Table 1: Beispiel Krankheitstabelle

Replikationsproblem um multiple Fragmentierungen. Dabei gehen wir zunächst auf den Fall ein, bei dem der Replikationsfaktor der Anzahl der Fragmentierungen entspricht, um dann den interessanteren Fall, bei dem die Anzahl der Fragmentierungen größer als der Replikationsfaktor ist, zu behandeln. Abschnitt 4 schließt den Artikel mit einer Zusammenfassung und Vorschlägen für zukünftige Arbeiten ab.

2. HINTERGRUND

Zunächst werden hier kurz Vorarbeiten zu Fragmentierung und Datenverteilung vorgestellt. Desweiteren geben wir Einblick in das Replikationsproblem für eine einzelne Fragmentierung. Als laufendes Beispiel werden wir im Folgenden ein Informationssystem eines Krankenhauses verwenden, bei dem die ID der Patienten zusammen mit deren Krankheit gespeichert wird (Tabelle 1).

2.1 Fragmentierung

Im Folgenden werden wir ein Datenreplikationsschema für horizontale Fragmentierung vorstellen. Eine wichtige Eigenschaft horizontaler Fragmentierungen ist die Korrektheit. Diese beinhaltet drei Eigenschaften:

- *Vollständigkeit*: Jedes Tupel der ursprünglichen Datentabelle ist in einem Fragment enthalten.
- *Rekonstruierbarkeit*: Die Vereinigung aller Fragmente resultiert in der ursprünglichen Datentabelle.
- *Redundanzfreiheit:* Kein Tupel ist in zwei Fragmenten gleichzeitig enthalten.

Wir werden im Folgenden Vollständigkeit und Redundanzfreiheit für unsere intelligente Replikation ausnutzen.

2.2 Datenverteilung als Behälterproblem

In verteilten Datenbanksystemen werden Datensätze auf verschiedenen Servern gespeichert. Das Datenverteilungsproblem – ohne Replikation – kann daher als Behälterproblem (bin packing problem, BPP) dargestellt werden:

- K Server entsprechen K Behältern
- Jeder Behälter besitzt eine maximale Kapazität W
- n Datensätze entsprechen n Objekten
- Jedes Objekt i besitzt ein Gewicht $w_i \leq W$
- Die Objekte sollen auf eine minimale Anzahl von Behältern aufgeteilt werden, ohne die maximale Kapazität W eines Behälters zu überschreiten

BPP kann folgendermaßen als ganzzahliges lineares Programm dargestellt werden:

$$\min \sum_{k=1}^{K} y_k \tag{1}$$

$$\text{s.d.} \sum_{k=1}^{K} x_{ik} = 1 \qquad i = 1, \ldots, n \tag{2}$$

$$\sum_{i=1}^{n} w_i x_{ik} \leq W \cdot y_k \qquad k = 1, \ldots, K \tag{3}$$

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

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

In diesem ILP benutzen wir x_{ik} als Indikatorvariable, die angibt, ob das Fragment f_i dem Server k zugewiesen wird. Die Bedingung (2) stellt dabei sicher, dass jedes Fragment genau einem Server zugewiesen wird. Hat y_k den Wert 1, so bedeutet dies, dass Server k benutzt wird, also mindestens ein Fragment darauf gespeichert wird. Durch Gleichung (3) wird garantiert, dass die Kapazität der benutzten Server nicht überschritten wird. Letztendlich wird in der Zielfunktion (1) die Anzahl der belegten Server minimiert.

Das Behälterproblem mit Konflikten (BPPC) ist eine Erweiterung des klassischen Behälterproblems. Dabei wird zusätzlich ein Konfliktgraph $G = (V, E)$ betrachtet, bei dem die Knoten V der Menge der Objekte entsprechen. In dem Graph existiert eine Kante (i, j), wenn die beiden Objekte i und j nicht in denselben Behälter gelegt werden dürfen. Um diese Bedingung einzuhalten kann das vorherige ganzzahlige lineare Problem um eine Bedingung erweitert werden:

$$x_{ik} + x_{jk} \leq y_k \qquad k = 1, \ldots, K, \tag{6}$$

Da y_k höchstens den Wert 1 hat, wird durch Bedingung (6) sichergestellt, dass nur entweder x_{ik} oder x_{jk} den Wert 1 hat, falls die Kante (i, j) im Konfliktgraphen existiert und Behälter k benutzt wird. Soll der Behälter k hingegen leer bleiben, so ist y_k gleich 0. In diesem Fall müssen auch x_{ik} und x_{jk} den Wert 0 annehmen.

In dieser Arbeit werden wir diese Darstellung von (BPPC) benutzen, um eine m-Kopien-Replikation für eine einzelne Fragmentierung, später aber auch für multiple Fragmentierungen, einer Tabelle zu verwirklichen.

2.3 Replikation für eine Fragmentierung

In [Wie15b] wird der Fall einer einzelnen Fragmentierung mit m-Kopien-Replikation betrachtet. Dies bedeutet, dass für jedes Fragment jeweils m Kopien angelegt werden und diese jeweils auf unterschiedliche Server verteilt werden müssen. Formal kann das Datenreplikationsproblem mit m Kopien (m-copy-DRP) folgendermaßen definiert werden:

Definition 1. Sei m der gewünschte Replikationsfaktor, sei $F = \{f_1, \ldots, f_n\}$ eine korrekte Fragmentierung einer Tabelle. Weiter seien $F^1, \ldots, F^m$ die Mengen der Kopien von F. Für jedes Fragment $f_i \in F$ ist eine Verteilung der m Kopien $f_i^1 \in F^1, \ldots, f_i^m \in F^m$ gesucht, sodass die Kopien alle auf unterschiedliche Server verteilt sind.

Dieses Problem entspricht der Lösung eines BPPC Problems, bei dem die Bedingungen, dass alle Kopien auf verschiedene Server verteilt werden, durch Kanten im Konfliktgraph dargestellt werden. Der Konfliktgraph hat folgende Form: Die Knotenmenge ist $V = \bigcup_{l=1}^{m} F^l$ und die Kantenmenge ist $E = \left\{ \left(f_i^l, f_i^{l'} \right) | i = 1, \ldots, n; l = 1, \ldots, m; l' < l \right\}$.

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

Fracture	*PatientID*	*Diagnosis*
	2784	brokenLeg
	1055	brokenArm

Table 2: Fragmentierung auf dem Attribut Diagnosis

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

IDhigh	*PatientID*	*Diagnosis*
	8765	Asthma
	8457	Cough

Table 3: Fragmentierung auf dem Attribut PatientID

3. ÜBERLAPPUNGEN UND MULTIPLE FRAGMENTIERUNGEN

In diesem Abschnitt werden wir dieses Replikationsschema nun auf multiple Fragmentierungen erweitern. Dabei wird einerseits der Speicherbedarf durch eine intelligente Replikationsstrategie reduziert und somit die Anzahl der benötigten Server minimiert und andererseits die Möglichkeit geschaffen, durch unterschiedliche Fragmentierungen flexibel auf die Bedürfnisse des Nutzers zu reagieren. Dies kann, wie bereits zuvor erwähnt, zur flexiblen Anfragebeantwortung verwendet werden, um mehrere verschiedene Relaxationsattribute zuzulassen (siehe [Wie14, Wie15a, Wie15b]).

Formal betrachten wir r Fragmentierungen $F^1, \ldots, F^r$ derselben Tabelle. Jede Fragmentierung F^l $(1 \leq l \leq r)$ besteht aus Fragmenten $f^l_1, \ldots, f^l_{n_l}$, wobei n_l von der jeweiligen Fragmentierung abhängt.

In unserem Beispiel könnte eine Fragmentierung darin bestehen, die Spalte *Diagnosis* in Atemwegserkrankungen und Knochenbrüche zu unterteilen (Tabelle 2). Zusätzlich könnte man eine weitere Fragmentierung anhand der Spalte *PatientID* erstellen, bei der die IDs in Werte kleiner als 5000 und Werte größer als 5000 unterteilt werden (Tabelle 3).

3.1 Datenreplikation für überlappende Fragmente

Wir stellen nun ein intelligentes Datenreplikationsschema für multiple Fragmentierungen vor, bei dem die Anzahl der Kopien von Tupeln reduziert werden soll, um so den Gesamtspeicherbedarf zu minimieren.

Bei m-copy-DRP (Abschnitt 2.3) betrachteten wir lediglich disjunkte Fragmente und jeweils m Kopien davon. Diese Annahmen werden nun folgendermaßen verändert: Fragmente verschiedener Fragmentierungen dürfen überlappen. Daher ist es im Allgemeinen nicht nötig, m Kopien jedes Fragments zu speichern, um jedes Tupel m mal zu replizieren. Daher schlagen wir, um den Speicherbedarf zu reduzieren, ein intelligentes Replikationsschema vor und fordern, dass lediglich jedes Tupel m mal repliziert wird und nicht jedes Fragment.

Dabei argumentieren wir, dass m Kopien eines Tupels für ein intelligentes Wiederherstellungsverfahren genügen: Jedes Tupel j soll als Sicherungskopie auf m verschiedenen Servern gespeichert sein, diese Kopien dürfen sich jedoch in unterschiedlichen Fragmenten befinden. Das bedeutet, dass jede Fragmentierung F^l aus den Fragmenten einer anderen Fragmentierung $F^{l'}$ wiederhergestellt werden kann.

Wir unterscheiden im Folgenden drei Fälle: Zuerst nehmen wir an, dass die Anzahl der Fragmentierungen gleich der Anzahl der geforderten Kopien ist $(r = m)$. Für den zweiten Fall, bei dem die Anzahl der Fragmentierungen kleiner als die Anzahl der Kopien ist $(r < m)$, können einige der Fragmentierungen einfach kopiert werden. Den interessanteren Fall, dass die Zahl der Fragmentierungen größer als die Zahl der geforderten Kopien ist $(r > m)$, werden wir in einem weiteren Abschnitt behandeln. Zunächst geben wir allerdings eine formale Definition für das Datenreplikationsproblem mit überlappenden Fragmenten (overlap-DRP):

Definition 2. Seien $F^l = \left\{f^l_1, \ldots, f^l_{n_l}\right\}$ für $l = 1, \ldots, r$ Fragmentierungen derselben Datentabelle und m der geforderte Replikationsfaktor. Für jedes Tupel j sind Fragmente f_{i_l} mit $1 \leq l \leq m$ und $1 \leq i_l \leq n_l$ gesucht, sodass $j \in f^1_{i_1} \cap \ldots \cap f^m_{i_m}$ und diese Fragmente müssen auf unterschiedliche Server verteilt werden.

Wir werden diese Definition nun an dem Beispiel veranschaulichen. Dazu nehmen wir an, dass die maximale Kapazität W der Server 5 Tupel beträgt und setzen einen Replikationsfaktor 2 voraus. Desweiteren seien Fragmentierungen wie in den Tabellen 2 und 3 gegeben, was zu den Fragmenten *Respiratory*, *Fracture*, *IDhigh*, *IDlow* führt. Im m-copy-Replikationsschema würde jedes Fragment jeweils auf zwei Servern gespeichert: Dafür werden mindestens 6 Server benötigt. Mit unserem intelligenten Replikationsschema, das Überlappungen von Fragmenten ausnutzt, können wir eine Lösung konstruieren, die lediglich 3 Server benötigt:

- Zunächst speichern wir das Fragment *Respiratory* auf Server $S1$.
- Anschließend legen wir das Fragment *IDlow* auf Server $S2$.
- Die Fragmente *Fracture* und *IDhigh* werden zusammen auf Server $S3$ gespeichert.

Dadurch erhalten wir den Replikationsfaktor 2 für jedes Tupel und können dennoch jedes Fragment aus den anderen zurückgewinnen:

- Fragment *Respiratory* kann aus den Fragmenten *IDlow* und *IDhigh* zurückgewonnen werden, denn *Respiratory* = (*IDlow* ∩ *Respiratory*) ∪ (*IDhigh* ∩ *Respiratory*)
- Fragment *Fracture* kann aus *IDlow* zurückgewonnen werden, denn *Fracture* = (*IDlow* ∩ *Fracture*)
- Fragment *IDlow* kann aus *Respiratory* und *Fracture* rekonstruiert werden, denn *IDlow* = (*IDlow* ∩ *Respiratory*) ∪ (*IDlow* ∩ *Fracture*)
- Fragment *IDhigh* kann aus dem Fragment *Respiratory* zurückgewonnen werden, denn *IDhigh* = (*IDhigh* ∩ *Respiratory*)

Nun werden wir erarbeiten, wie overlap-DRP als erweitertes BPPC Problem dargestellt werden kann und dazu ein ganzzahliges lineares Programm formulieren. Hierfür bezeichne J die Anzahl der Tupel der Eingabetabelle, m den Replikationsfaktor, K die Anzahl der zur Verfügung stehenden Server und n die Gesamtzahl aller Fragmente aller Fragmentierungen. Dies führt zu folgendem ganzzahligen linearen Programm:

$$\min \sum_{k=1}^{K} y_k \tag{7}$$

$$\text{s.d.} \sum_{k=1}^{K} x_{ik} = 1 \qquad i = 1, \ldots, n \tag{8}$$

$$\sum_{i=1}^{n} w_i x_{ik} \leq W \cdot y_k \qquad k = 1, \ldots, K \tag{9}$$

$$z_{jk} \geq x_{ik} \qquad \forall j : j \in f_i \tag{10}$$

$$z_{jk} \leq \sum_{(i:j \in f_i)} x_{ik} \qquad k = 1, \ldots, K, j = 1, \ldots, J \tag{11}$$

$$\sum_{k=1}^{K} z_{jk} \geq m \qquad j = 1, \ldots, J \tag{12}$$

$$y_k \in \{0, 1\} \qquad k = 1, \ldots, K \tag{13}$$

$$x_{ik} \in \{0, 1\} \qquad k = 1, \ldots, K, i = 1, \ldots, n \tag{14}$$

$$z_{jk} \in \{0, 1\} \qquad k = 1, \ldots, K, j = 1, \ldots, J \tag{15}$$

In dieser Formulierung verwenden wir die Variablen y_k und x_{ik} wie in den vorherigen ILPs. Um die Notation zu vereinfachen benutzen wir $i = 1, \ldots, n$, mit $n = |F^1| + \ldots + |F^m|$, nummerieren also alle Fragmente nacheinander von 1 bis n, auch wenn sie aus unterschiedlichen Fragmentierungen stammen. Wir führen zusätzlich K Indikatorvariablen z_{jk} für jedes Tupel j ein, mit $z_{jk} = 1$, falls Tupel j auf Server k gespeichert wird. Mithilfe von Gleichung (10) erreichen wir, dass wenn Fragment f_i auf Server k gespeichert ist und Tupel j in f_i enthalten ist, die entsprechende Variable z_{jk} ebenfalls gleich 1 ist. Umgekehrt erreichen wir durch Bedingung (11), dass wenn kein Fragment, das j enthält, auf Server k gespeichert wird, die Variable z_{jk} den Wert 0 annehmen muss. Durch die Nebenbedingung (12) erzwingen wir den Replikationsfaktor m für jedes Tupel j.

3.2 Reduktion der Anzahl der Variablen

Die Formulierung des ganzzahligen linearen Programms im vorherigen Abschnitt ist aufgrund der vielen z-Variablen höchst ineffizient für eine große Anzahl von Tupeln in einer Tabelle. Daher wollen wir nun zeigen, dass es möglich ist, sich lediglich auf die x-Variablen zu konzentrieren.

Zunächst wird in diesem Abschnitt nur der Fall betrachtet, bei dem der Replikationsfaktor gleich der Anzahl der Fragmentierungen ist ($r = m$), mit dem Fall $r > m$ befassen wir uns anschließend im nächsten Abschnitt.

Ist der Replikationsfaktor m gleich der Anzahl der Fragmentierungen r, so dürfen Fragmente f_i und $f_{i'}$ nicht auf demselben Server gespeichert werden, falls $f_i \cap f_{i'} \neq \emptyset$ (wir nehmen $i < i'$ an, um isomorphe Bedingungen zu vermeiden). Ansonsten kann für alle Tupel $j \in f_i \cap f_{i'}$ der Replikationsfaktor nicht erreicht werden. Andererseits ist diese Bedingung auch hinreichend, denn aufgrund der $r = m$ Fragmentierungen wird dadurch gewährleistet, dass jedes Tupel auf m verschiedenen Servern gespeichert ist. Diese Beobachtung führt zu folgender Vereinfachung des ganzzahligen linearen Programms:

f_1	tupleID	PatientID	Diagnosis
	1	2784	brokenLeg
	2	2784	Flu
	3	8765	Asthma
	4	8457	Cough

Table 4: Fragmentierung auf dem tupleID Attribut

$$\text{minimize} \sum_{k=1}^{K} y_k \tag{16}$$

$$\text{s.d.} \sum_{k=1}^{K} x_{ik} = 1 \qquad i = 1, \ldots, n \tag{17}$$

$$\sum_{i=1}^{n} w_i x_{ik} \leq W \cdot y_k \qquad k = 1, \ldots, K \tag{18}$$

$$x_{ik} + x_{i'k} \leq y_k \qquad k = 1, \ldots, K, f_i \cap f_{i'} \neq \emptyset \tag{19}$$

$$y_k \in \{0, 1\} \qquad k = 1, \ldots, K \tag{20}$$

$$x_{ik} \in \{0, 1\} \qquad k = 1, \ldots, K, i = 1, \ldots, n \tag{21}$$

Die Bezeichnungen in dieser Formulierung sind analog zu den vorherigen Formulierungen. Durch die Nebenbedingung (19) wird sichergestellt, dass der Replikationsfaktor m eingehalten wird, da Fragmente, f_i und $f_{i'}$, deren Schnittmenge nicht leer ist, auf unterschiedliche Server verteilt werden müssen.

3.3 Optionale Konflikte

In diesem Abschnitt betrachten wir nun den komplizierteren Fall, bei dem die Anzahl der Fragmentierungen größer als der geforderte Replikationsfaktor ist ($r > m$). In diesem Fall müssen, um den Replikationsfaktor einzuhalten, für jedes Tupel j nur m der r Fragmente, die j beinhalten, auf unterschiedliche Servern verteilt werden. Die restlichen $r - m$ Fragmente können beliebig, auch auf den bereits belegten Servern, gespeichert werden, was dazu beiträgt, den Speicherbedarf zu verringern.

Wir werden dies an einem kleinen Beispiel illustrieren. Angenommen wir haben eine Fragmentierung F, die nur das Fragment f beinhaltet, eine Fragmentierung F' mit Fragmenten f'_1 und f'_2 und eine dritte Fragmentierung F'' mit den Fragmenten f''_1 und f''_2. Dabei überlappt f_1 mit allen anderen vier Fragmenten: $f_1 \cap f'_1 \neq \emptyset$, $f_1 \cap f'_2 \neq \emptyset$, $f_1 \cap f''_1 \neq \emptyset$ und $f_1 \cap f''_2 \neq \emptyset$. Wir veranschaulichen diese Situation in einem leicht modifizierten Beispiel unseres Krankenhausszenarios in den Tabellen 4, 5 und 6. Für f_1 gibt es also 4 Konfliktbedingungen und es ist nicht klar, welche davon eingehalten werden sollten, um einerseits 2-Kopien-Replikation für jedes Tupel in f_1 sicherzustellen und andererseits die Anzahl der Server zu minimieren. Betrachten wir das Beispiel nun mit konkreten Werten. Wir nehmen eine maximale Kapazität der Server von $W = 6$ an, das Gewicht von f_1 ist $w_1 = 4$, die Gewichte von f'_1 und f'_2 sind $w'_1 = w'_2 = 2$, das Gewicht von f''_1 ist $w''_1 = 1$ und das Gewicht von f''_2 ist gleich $w''_2 = 3$. Wir diskutieren nun einige Optionen, wie diese Fragmente verteilt werden könnten:

- Angenommen wir speichern f_1 auf dem Server $S1$. Um **alle** zuvor genannten Konfliktbedingungen einzuhal-

f_1'	tupleID	PatientID	Diagnosis
	1	2784	brokenLeg
	2	2784	Flu
f_2'	tupleID	PatientID	Diagnosis
	3	8765	Asthma
	4	8457	Cough

Table 5: Fragmentierung auf dem PatientID Attribut

f_1''	tupleID	PatientID	Diagnosis
	1	2784	brokenLeg
f_2''	tupleID	PatientID	Diagnosis
	2	8457	Cough
	3	2784	Flu
	4	8765	Asthma

Table 6: Fragmentierung auf dem Diagnosis Attribut

ten, müssen f_1' und f_2' auf einen zweiten Server $S2$ gelegt werden und f_1'' und f_2'' müssen auf einem dritten Server gespeichert werden. Dies resultiert also in einer 3-Kopien-Replikation.

- Angenommen wir fordern lediglich 2-Kopien-Replikation. Wir können also versuchen, manche der überlappenden Fragmente auf demselben Server zu speichern, so lange die Replikationsbedingung erfüllt ist. Legen wir f_1 und f_1'' auf einen Server $S1$, so passen f_1' und f_2'' auf einen zweiten Server $S2$. Fragment f_2' muss dann auf einem dritten Server $S3$ gespeichert werden. Wir erzielen also 2-Kopien-Replikation für alle Tupel, benötigen aber dennoch weiterhin drei Server.

- Tatsächlich ist es aber auch möglich, die benötigte Anzahl der Server auf zwei zu reduzieren, aber dennoch 2-Kopien-Replikation zu erzielen. Hierfür speichern wir f_1 und f_1' auf einem Server $S1$. Auf einem zweiten Server $S2$ ist nun ausreichend Speicherplatz für die restlichen Fragmente f_1'', f_2' und f_2'' vorhanden.

Aus diesem Beispiel wird deutlich, dass die Entscheidung, welche Konfliktbedingungen eingehalten werden sollen und welche optional sind, sehr schwer ist. Im Folgenden wird nun die Frage beantwortet, wie diese optionalen Konfliktbedingungen in unser ganzzahliges lineares Programm integriert werden können.

Formal gesehen betrachten wir wiederum r Fragmentierungen $F^l = \{f_1^l, \ldots, f_{n_l}^l\}$ und für jedes Tupel j gibt es Fragmente $f_{i_l}^l$ für $1 \leq l \leq r$ und $1 \leq i_l \leq n_l$, sodass $j \in f_{i_1}^1 \cap \ldots \cap f_{i_r}^r$. Sind $f_{i_1}^1, \ldots, f_{i_r}^r$, Fragmente aus den r unterschiedlichen Fragmentierungen, so definieren wir den Begriff des *gemeinsamen Teilfragments* als den Durchschnitt $f_{i_1}^1 \cap \ldots \cap f_{i_r}^r$. In unserem Beispiel ergeben sich folgende nichtleere gemeinsame Teilfragmente:

$$f_1 \cap f_1' \cap f_1'' = \{t_1\}$$
$$f_1 \cap f_1' \cap f_2'' = \{t_2\}$$
$$f_1 \cap f_2' \cap f_2'' = \{t_3, t_4\}$$

Für solche nichtleeren gemeinsamen Teilfragmente erhält man paarweise Konfliktbedingungen der Form $x_{i_l k} + x_{i_{l'} k} \leq y_k$ für $1 \leq l \leq r$ und $0 < l' < l$. Greifen wir das Beispiel für das gemeinsame Teilfragment $\{t_1\}$ wieder auf, ergibt das die paarweisen Konfliktbedingungen:

$$x_{1k} + x_{1k}' \leq 1$$
$$x_{1k} + x_{1k}'' \leq 1$$
$$x_{1k}' + x_{1k}'' \leq 1$$

Um 2-Kopien-Replikation zu garantieren, muss nur eine dieser Bedingungen erfüllt sein. Um dem gerecht zu werden führen wir neue c-Indikatorvariablen für jede dieser Bedingungen ein:

$$x_{1k} + x_{1k}' \leq 1 + c_{1k}$$
$$x_{1k} + x_{1k}'' \leq 1 + c_{2k}$$
$$x_{1k}' + x_{1k}'' \leq 1 + c_{3k}$$

Diese Variablen haben die folgende Bedeutung: Sind die c-Variablen gleich 0, dann ist die Konfliktbedingung erfüllt und die beiden Fragmente werden nicht zusammen auf demselben Server k gespeichert. Ist die c-Variable hingegen gleich 1, ist die Konfliktbedingung nicht erfüllt und die beiden Fragmente können zusammen auf Server k gespeichert werden. Um die m-Kopien-Replikation zu erzwingen fordern wir, dass die Summe der c-Variablen höchstens $r - m$ ist, was praktisch bedeutet, dass höchstens $r - m$ Bedingungen verletzt werden dürfen und mindestens m Bedingungen erfüllt werden. In unserem Beispiel führt dies zur Bedingung $c_{1k} + c_{2k} + c_{3k} \leq 1$. Dieses Konzept wenden wir nun für beliebige Werte von r und m mit $r > m$ an:

$$\min \sum_{k=1}^{K} y_k \tag{22}$$

$$\text{s.d.} \sum_{k=1}^{K} x_{ik}^l = 1 \qquad i = 1, \ldots, n_l, \quad l = 1, \ldots, r \tag{23}$$

$$\sum_{i=1}^{n} \sum_{l=1}^{r} w_i^l x_{ik}^l \leq W \cdot y_k \qquad k = 1, \ldots, K \tag{24}$$

$$x_{i_1 k}^l + x_{i_2 k}^{l'} \leq 1 + c_{ll'k}^s \qquad k = 1, \ldots, K, \; l = 1, \ldots, r, \; 0 < l' < l, \; f_{i_2 k}^{l'} \cap f_{i_1 k}^l \cap g_s \neq \emptyset, \; s = 1, \ldots, S \tag{25}$$

$$\sum_{\substack{l=1 \\ 0<l'<l}}^{r} c_{ll'k}^s \leq r - m \qquad k = 1, \ldots, K, \; s = 1, \ldots, S \tag{26}$$

$$y_k \in \{0, 1\} \qquad k = 1, \ldots, K \tag{27}$$

$$x_{ik}^l \in \{0, 1\} \qquad k = 1, \ldots, K, \; i = 1, \ldots, n_l, \; l = 1, \ldots, r \tag{28}$$

$$c_{ll'k}^s \in \{0, 1\} \qquad k = 1, \ldots, K, \; i = 1, \ldots, n_l, \; l = 1, \ldots, r, \; 0 < l' < l \tag{29}$$

Da wir r Fragmentierungen der Form $F^l = \{f_1^l, \ldots, f_{n_l}^l\}$ für $l = 1, \ldots, r$ betrachten, gibt es die Indikatorvariablen x_{ik}^l, die angeben, ob Fragment i aus der Fragmentierung l auf Server k gespeichert werden. Jedes Fragment f_i^l besitzt ein gewisses Gewicht w_i^l. Mit S bezeichnen wir die Gesamtzahl der nichtleeren gemeinsamen Teilfragmente und mit g_s für $s = 1, \ldots, S$ die Teilfragmente selbst. Die Bedingungen (25) und (26) garantieren zusammen, dass mindestens m Fragmente jedes gemeinsamen Teilfragments g_s auf unterschiedlichen Servern gespeichert werden.

4. ZUSAMMENFASSUNG UND AUSBLICK

In diesem Artikel haben wir das Datenreplikationsproblem für multiple Fragmentierungen betrachtet. Um Speicherplatz zu sparen, wurde ein intelligentes Replikationsschema vorgestellt, bei dem unnötige Kopien von Tupeln vermieden werden. Anschließend haben wir das Problem als ganzzahliges lineares Programm formuliert und dabei versucht die Anzahl der Variablen zu reduzieren, um eine bessere Lösbarkeit zu ermöglichen.

Solche multiplen Fragmentierungen könnten beispielsweise benutzt werden, um Datenbankanfragen flexibel zu beantworten. Diese Anwendung wurde bereits in vorherigen Arbeiten für ein einzelnes Relaxationsattribut (eine einzelne Fragmentierung) behandelt, lässt sich aber auf natürliche Art und Weise auf mehrere Relaxationsattribute erweitern.

Eine genaue Komplexitätsanalyse des Verfahrens kann noch vorgenommen werden. Es wird aber deutlich, dass für den Fall $r > m$ eine zusätzlich Schwierigkeit (über das BPPC) darin liegt, die Überlappungen (nicht-leeren Schnittmengen) zwischen den Fragmenten zu finden.

In zukünftigen Arbeiten sollten vor Allem dynamische Veränderungen im Replikationsschema untersucht werden. Hinzufügen und Entfernen von Daten führt zu Veränderungen der Größen der Fragmente und daher könnte eine Umverteilung der Daten auf den Servern notwendig werden.

Zudem beschreiben die Autoren von C-Store [**?**] die Möglichkeit verschiedene Projektionen (also vertikale Fragmente) in verschiedene Segmente (horizontale Fragmente) aufzuteilen ohne jedoch ein genaues Verteilungsverfahren anzugeben. Eine solche Form der hybriden Fragmentierung als ILP darzustellen ist eine weitere zukünftige Fragestellung.

5. REFERENCES

[BK15] Ladjel Bellatreche and Amira Kerkad. Query interaction based approach for horizontal data partitioning. *International Journal of Data Warehousing and Mining (IJDWM)*, 11(2):44–61, 2015.

[CZJM10] Carlo Curino, Yang Zhang, Evan P. C. Jones, and Samuel Madden. Schism: a workload-driven approach to database replication and partitioning. *Proceedings of the VLDB Endowment*, 3(1):48–57, 2010.

[ECS+08] George Eadon, Eugene Inseok Chong, Shrikanth Shankar, Ananth Raghavan, Jagannathan Srinivasan, and Souripriya Das. Supporting table partitioning by reference in oracle. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1111–1122. ACM, 2008.

[JPPD13] Alekh Jindal, Endre Palatinus, Vladimir Pavlov, and Jens Dittrich. A comparison of knives for bread slicing. *Proceedings of the VLDB Endowment*, 6(6):361–372, 2013.

[KPX+11] Qifa Ke, Vijayan Prabhakaran, Yinglian Xie, Yuan Yu, Jingyue Wu, and Junfeng Yang. Optimizing data partitioning for data-parallel computing. In *13th Workshop on Hot Topics in Operating Systems, HotOS XIII*. USENIX Association, 2011.

[ÖV11] M. Tamer Özsu and Patrick Valduriez. *Principles of Distributed Database Systems, Third Edition*. Springer, 2011.

[SPTB14] Michael Stonebraker, Andrew Pavlo, Rebecca Taft, and Michael L Brodie. Enterprise database applications and the cloud: A difficult road ahead. In *Cloud Engineering (IC2E), 2014 IEEE International Conference on*, pages 1–6. IEEE, 2014.

[TPRH16] Alexandru Turcu, Roberto Palmieri, Binoy Ravindran, and Sachin Hirve. Automated data partitioning for highly scalable and strongly consistent transactions. *Parallel and Distributed Systems, IEEE Transactions on*, 27(1):106–118, 2016.

[VBC+14] Ravi Varadarajan, Vivek Bharathan, Ariel Cary, Jaimin Dave, and Sreenath Bodagala. Dbdesigner: A customizable physical design tool for vertica analytic database. In *Data Engineering (ICDE), 2014 IEEE 30th International Conference on*, pages 1084–1095. IEEE, 2014.

[Wie14] Lena Wiese. Clustering-based fragmentation and data replication for flexible query answering in distributed databases. *Journal of Cloud Computing*, 3(1):1–15, 2014.

[Wie15a] Lena Wiese. Horizontal fragmentation and replication for multiple relaxation attributes. In *Data Science (30th British International Conference on Databases)*, pages 157–169. Springer, 2015.

[Wie15b] Lena Wiese. Ontology-driven data partitioning and recovery for flexible query answering. In *Database and Expert Systems Applications*, pages 177–191. Springer, 2015.

[ZRL+04] Daniel C Zilio, Jun Rao, Sam Lightstone, Guy Lohman, Adam Storm, Christian Garcia-Arellano, and Scott Fadden. Db2 design advisor: integrated automatic physical database design. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pages 1087–1097. VLDB Endowment, 2004.