Generalization of an Algorithm for Checking Consistency of Mapping Tables in a Peer-to-Peer System

Md. Anisur Rahman
School of Info. Tech. &Engineering
University of Ottawa
Ontario, Canada
anis@mcrlab.uottawa.ca

Illuju Kiringa
School of Info. Tech. &Engineering
University of Ottawa
Ontario, Canada
kiringa@site.uottawa.ca

Abdulmotaleb El Saddik
School of Info Tech. & Engineering
University of Ottawa
Ontario, Canada
abed@discover.uottawa.ca

Abstract

In a peer-to-peer system, each peer designs its own schema autonomously and stores data without considering anything about the data of other peers. The peers may need to interact with one another for relevant data. Since the involved databases do not share any global schema, interaction among them can not be done by data integration. To facilitate the interaction among the heterogeneous databases, techniques for data coordination are necessary. These techniques need a mapping of data between different databases. Data mapping mechanisms often rely on simple value searches to locate data of interest. Different peers may use different values to identify or describe the same data. To accommodate this, data mapping is done with the help of "mapping tables" that associate corresponding data values from different peer domains.

It is possible to reason about the mapping tables and check their consistency by treating them as constraints (called mapping constraints) on the exchange of information between peers. These constraints are not on the peers themselves, but only on the way in which search values are translated. An algorithm already exists for checking consistency of mapping tables. The algorithm assumes that the peers can be arranged in a linear path P_1, P_2,..., P_n such that peer P_i stores mapping tables between its data items and the data items of the next peer P_{i+1}. The algorithm doesn't work in a situation where such a linear path doesn't exist among the peers. In this paper we relaxed the assumption of linear path and modified the algorithm without affecting its time complexity. The modified algorithm works irrespective of the existence of linear path among peers.

Keywords

Peer-to-Peer Database, Data Coordination, Mapping Table

1. INTRODUCTION

Peer-to-Peer systems are characterized by the heterogeneity of data sources and absence of common schema. To find and exchange data when there is no agreement on the logical design of data, one may focus on data values and how values correspond [3]. To search data in peers with different universes, mapping tables are introduced, which store correspondence between values. They can be either binary tables containing pairs of identifiers from two different sources or tables mapping multiple attributes. Mapping tables permit searching across peers but don't restrict their operation beyond the agreement on values expressed in the tables, so they are appropriate for use in P2P systems [2].

Kementsietsidis et al. [1] illustrate how automated tools can help managing mapping tables between multiple sources by inferring new mappings (that is new entries in a mapping table). They showed that by treating mapping tables as constraints (called mapping constraints) on the exchange of information between peers it is possible to reason about them and check their consistency. The consistency problem is the problem of determining if a given mapping constraint formula is consistent and the inference problem is the problem of verifying if a set of mapping constraint formula implies another mapping constraint formula. An algorithm has been proposed in [1] for solving consistency and inference problem. To reduce the complexity of the algorithm they imposed some restrictions as follows:

- Only conjunctions of mapping constraints are considered. i.e. given a set of mapping constraints $\Sigma = \{\mu_1, \mu_2, \ldots, \mu_n\}$, Σ is said to be consistent if $\mu_1 \wedge \mu_2 \wedge \ldots \wedge \mu_n$ is consistent
- The algorithm is based on the notion of a path. A path θ is a list $P_1, P_2, \ldots, P_n$ of peers such that peer P_i stores mapping tables between its data items and the data items of the next peer in the path P_{i+1}, where i ranges from 1 to n-1.
- The length of the path is assumed to be small.
- The number of mapping constraints per peer is assumed to be fixed and small.
- The arity of the mapping constraints are also assumed to be fixed.

In situations where the above restrictions can be met, the given algorithm can efficiently compute the *cover* (definition of cover is given in next section) of a set of mapping constraints and thus can solve both consistency and inference problem. However, in practical situations, it is not always possible to satisfy all the above conditions. In this paper we tried to relax some of those restrictions and give a more generalized algorithm.

In Section 2, we introduce the main terminology. Section 3 presents the existing algorithm for checking consistency of mapping tables in different peers. In section 4 we show how some of the assumptions of the algorithm can be relaxed. Section 5 offers the conclusion and points to future work.

2. BASIC CONCEPTS

We introduce basic concepts taken from [1]. A *Mapping Table m* from a set of attributes X to a set of attributes Y is a finite set of mappings over $X \cup Y$.

A sample mapping table relating gene data in one source to the related protein data in another source (where the gene is said to encode for the protein) is shown in Figure 1.

GDB_id		SwProt_id
G1		P9
G1		Q62
G2		P40

Figure1. Mapping table

The mapping table is not necessarily a function, there may be many proteins related to a gene. Even a mapping table relating gene identifiers may be many to many.

Mapping constraints of the form $\mu : X \xrightarrow{m} Y$ is said to be satisfied by a tuple *t* if *X-value* and *Y*-value of *t* are associated in mapping table *m*. A relation *r* is said to satisfy a mapping constraint if every tuple of the relation satisfies the later.

Mapping constraint formulas (MCF) can contain conjunction, disjunction or negation of mapping constraints. The grammar of the language for MCFs is defined as follows:

MCF := (MCF $\wedge$ MCF) | (MCF $\vee$ MCF) | $\neg$MCF | μ

where μ is an atomic mapping constraint.

A definition of what it means for a tuple to satisfy a MCF Φ, defined over a set of mapping constraints is given below.

- If $\Phi = \mu$, then $t \models \Phi$ iff $t \models \mu$.
- If $\Phi = \neg \Phi_1$, then $t \models \Phi$ iff it is not true that $t \models \Phi_1$
- If $\Phi = \Phi_1 \wedge \Phi_2$, then $t \models \Phi$ iff $t \models \Phi_1$ and $t \models \Phi_2$.
- If $\Phi = \Phi_1 \vee \Phi_2$, then $t \models \Phi$ iff $t \models \Phi_1$ or $t \models \Phi_2$.

To infer new mapping tables means to find the set of all mapping tables that are valid over a specific set of attributes. To do this, knowledge from the mapping tables available in the network of peers should be combined. Mapping constraints can be used to infer additional mappings that are not explicitly represented in any peer.

The *consistency problem* is the problem of determining if a given mapping constraint formula is consistent and the *inference problem* is the problem of verifying if a set of mapping constraint formulas imply another mapping constraint formula. The consistency and inference problems for mapping constraint formulas are equivalent. To check whether a set of mapping constraint formula Σ implies Φ it can be verified whether $\neg\Phi \wedge \bigwedge_{\varphi \in \Sigma} \varphi$ is not consistent. Consistency problem for mapping constraints is NP-complete.

A valuation ρ over a mapping table *m* is a function that maps each constant value in *m* to itself and each variable *v* of *m* to a value in the intersection of the domains of the attributes where *v* appears. Furthermore, if *v* appears in an expression of the form *v* - *S*, then $\rho(v) \notin S$.

Extension of a mapping constraint *ext*(μ) is a set of valuation of the tuples of the mapping table. A mapping constraint $\mu : X \xrightarrow{m} Y$ is said to be a *cover of a set of mapping constraints* Σ over set of attributes *U* if:

1. Σ is consistent if and only if there exists t$\in$ ext(μ); and
2. For every mapping constraint $\mu' : X \xrightarrow{m'} Y$, $\Sigma \models \mu'$ if and only if *ext*(μ) $\subseteq$ *ext*(μ').

Partition is the key concept of the algorithm in [1]. If P and P' be two peers and $\Sigma_{P,P'}$ be the set of constraints between these two peers, a *partition* Π is a subset of $\Sigma_{P,P'}$ and is constructed as follows. Consider a graph $G_{P,P'} = (V,E)$, where V contains one vertex for each constraint in $\Sigma_{P,P'}$ and there is an edge between two constraints if their attribute overlap. Partition Π contains all the constraints of $\Sigma_{P,P'}$ whose corresponding vertices belong to the same connected component of $G_{P,P'}$.

Inferred partitions involve constraints of more than one peer. A partition graph $G_\Pi = (V_\Pi, E_\Pi)$ is constructed where V_Π contains one vertex for each partition in the union of partitions of the associated peers. There is an edge between two partitions if their attributes overlap. Each connected component of graph G_Π become an inferred partition. Example of partition and inferred partition will be given in the next section.

3. EXISTING ALGORITHM

The existing algorithm takes as input a path θ of peers $P_1, P_2, P_3, \ldots P_n$, a set of mapping constraints Σ, and two sets of attributes *X* and *Y* in peers P_1 and P_n. Output is a mapping constraint that is a cover of Σ.To check whether a set of mapping constraints Σ infers a new mapping constraint μ', the algorithm is run to find the cover μ and it is checked whether *ext*(μ) $\subseteq$ *ext*(μ')**.** To check whether Σ is consistent it is enough to run the algorithm to find the cover μ and check whether *ext*(μ) is nonempty.

The algorithm runs on top of a prototype peer-to-peer data management system in which each peer manages a collection of data. Each peer autonomously chooses a logical design and physical organization for the data.. The algorithm distributes the computation among the peers, and delivers the results in a streaming fashion (as soon as they are available). The algorithm has two phases: information gathering phase and the computation phase. The algorithm can be presented by the following example.

Let us consider a path $\theta = P_1, P_2, P_3, P_4$ of four peers. The mapping constraints Σ in this path are shown in figure 2. Peer P_1 has attributes A_i (i$\in$ [1,6]), P_2 has attributes B_i (i$\in$ [1,6]), P_3 has attributes C_i (i$\in$ [1,4]), P_1 has attributes D_i (i$\in$ [3,4]). The aim of the algorithm is to compute the cover μ between the attributes of per P_1 and peer P_4.

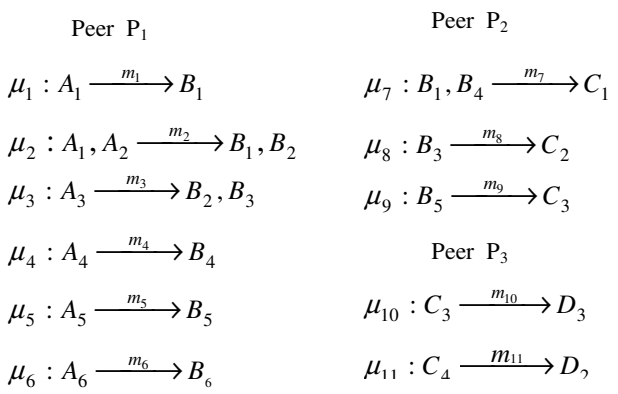

Figure 2. A path θ = P_1, P_2, P_3, P_4 of four peers

During the information gathering phase peers compute their partitions and send attributes for each of their partitions to the next peer. The next peer computes its own partitions and using information received from the previous peer it computes inferred partitions. Each partition will contain attributes which overlap. This process will propagate to the last peer. e.g. Figure 3 depicts the partitions of peers P_1 and Figure 4 shows inferred partitions over P_1 and P_2.

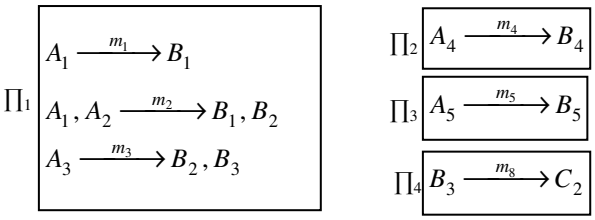

Figure 3. Partitions of Peer P_1

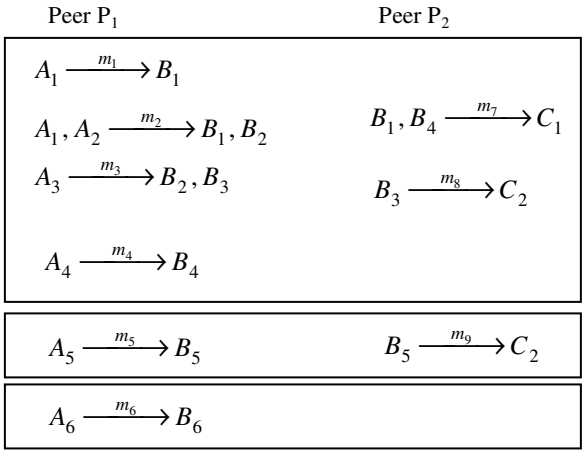

Figure 4. Inferred partition over P_1 and P_2

The computation phase starts from the penultimate peer where cover between last and the second last peers is computed. The cover only involves the attributes of the two peers which appear in the inferred partition. The mappings belonging to the cover are streamed to the peer to the left. Using the information from the inferred partitions that peer determines with which of its own partitions the incoming stream of mappings should be associated. Then it uses this information to compute a cover between the last peer in the path and itself. A local algorithm is used to compute the cover. The mappings of this cover are then passed to the peer which is a predecessor to it along the path. This process continues until the first peer gets the cover between the second peer and the last peer. In the final step the first peer generates cover between its own attributes and those of the last peer in the path.

4. GENERALIZATION OF THE ALGORITHM

The algorithm in the previous section makes some assumptions to reduce computation complexity. In this section we show that some of the assumptions can be relaxed to generalize the algorithm without affecting its complexity. We show how consistency among the mapping tables of some peer can be determined when the peers do not form a linear path. We also show how to deal with a long path.

4.1 RELAXING THE ASSUMPTION OF EXISTENCE OF A LINEAR PATH

Let us consider some cases where the peers don't form a single linear path as depicted in the following figures. An arrow between two peers in the figures indicates that the peer at the tail of the arrow contains mapping tables between the attributes of the two peers connected by the arrow.

Case1: Dealing with a Tree-like path

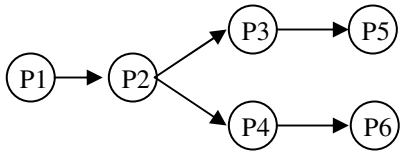

Figure 5. Tree-like path

In this case the peers form a tree. But we can get two linear paths from the tree as P1-P2-P3-P5 and P1-P2-P4-P6 and check the consistency independently along the two paths. If the cover between P1 and P5 along the path P1-P2-P3-P5 is nonempty and the cover between P1 and P6 along the path P1-P2-P4-P6 is also nonempty, we can conclude that all the mapping tables in the peers P1, P2, P3, P4, P5 and P6 are consistent. Since P3 and P5 are not linked to P4 and P6, there is no way for them to be mutually inconsistent. For this reason no cross-checking is necessary among the peers of two different branches of a tree-like structure among the peers.

Case 2: Dealing with a Graph-like path

A Graph-like path is shown in Figure 6 in the following page.

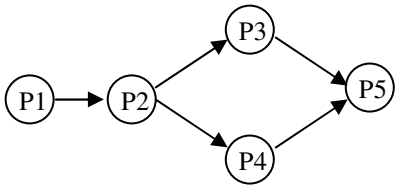

Figure 6. Graph-like path

In this case the solution is not so simple. Though the peers P3 and P4 are not directly connected, they may be mutually inconsistent. To show this by an example let the relations and the mapping tables of the peers be as in Figures 7 and 8:

P1	P2	P3	P4	P5
A	**B**	**C**	**D**	**E**
a1	b1	c1	d1	e1
a2	b2	c2	d2	e2

Figure 7. Relations of the peers

P1		P2		P3		P4	
A	**B**	**B**	**C**	**B**	**D**	**C**	**E**	**D**	**E**
a1	b1	b1	c1	b1	d2	c1	e1	d1	e1
a2	b2	b2	c2	b2	d1	c2	e2	d2	e2

Figure 8. Mapping tables in the peers

For finding the cover between P1 and P5 there are two alternative paths: P1-P2-P3-P5 and P1-P2-P4-P5. The covers along the two paths are:

A	**E**
a1	e1
a2	e2

Figure 9. Cover along the path P1-P2-P3-P5

A	**E**
a1	e2
a2	e1

Figure 10. Cover along the path P1-P2-P4-P5

The two covers themselves are mutually inconsistent. This means that when we consider all the mapping tables at a time they are inconsistent though we were able to find a non-empty cover between the end peers along two different paths. In situations where there exist alternative paths between two nodes, the algorithm can be used to find cover between the two nodes along all possible paths. Then we can check whether the covers along different paths are themselves consistent. If yes, we can conclude that all the mapping tables are consistent, otherwise they are inconsistent. This extra checking can help the algorithm to cope with a graph-like path as given in case 2.

It is obvious that the trick to deal with tree-like path or graph-like path does not cost much overhead and does not increase the overall time complexity.

4.2. RELAXING THE ASSUMPTION OF SHORT LENGTH OF PATH

One of the assumptions of the algorithm is that the length of the path of peers has to be small. This assumption is practical. But in case the path is long, the algorithm can be modified to manage the situation in a "divide and conquer" fashion. A long path can be subdivided into manageable sub paths as shown in Figure 11.

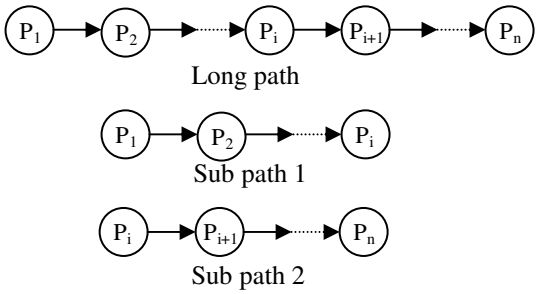

Figure 11. Dividing a long path into manageable parts

In the 1st phase the algorithm can be applied to find covers for each sub path. Cover of the covers of each sub path is then calculated by running the algorithm in the 2nd phase. e.g. consider a path $P_1,P_2,\ldots P_i,P_{i+1},\ldots P_n$. The path can be divided into two sub paths: $P_1,P_2,\ldots P_i$ and $P_i,P_{i+1},\ldots P_n$. Peer P_i is common to both sub paths. We can run the algorithm to compute the cover between P_1 and P_i along sub path 1. P_1 will store the cover of sub path 1. The algorithm may be run in parallel to find the cover between the peers P_i and P_n along sub path 2. Peer P_i would contain the cover of sub path 2. Now the algorithm can be run again to check whether the covers in P_1 and P_i (which were computed in the first phase) are consistent with each other. If they are consistent we can conclude that mapping tables in the whole path are consistent.

Division of a long path into manageable sub paths and combination of the partial results to get the final result demands very little overhead which is acceptable for the sake of generalization of the algorithm

5. CONCLUSIONS

The algorithm proposed in [1] solves problems of managing mapping tables efficiently under some assumptions. We tried to relax some of the assumptions to make it more general. The assumption about the existence of a linear path between peers was relaxed. In case when peers form a tree, we showed that in order to check the consistency of the mapping tables it is enough to check the consistency of mapping tables of each branch of the tree. When there exists more than one path between two peers it is necessary to calculate cover along all the paths and check whether the covers themselves are consistent.

Relaxing the assumption about path's length we showed that by dividing the path into sub-paths, running the cover algorithm within each sub-path, and checking the consistency of those covers, it is possible to use the proposed algorithm for a long path.

Only conjunctions of mapping constraints are considered in the algorithm. There is scope for modification of the algorithm to make it capable to cope with the disjunction of the mapping constraints too.

6. REFERENCES

[1] A.Kementsietsidis, M. Arenas and R.J. Miller, "Mapping Data in Peer-to-Peer Systems: Semantics and Algorithmic Issues", SIGMOD 2003.

[2] M.Arenas, V.Kantere, A.Kementsietsidis, I.Kiringa, R.Miller, J.Mylopoulos, "The Hyperion Project: From Data Integration to Data coordination", SIGMOD Record, Sept 2003.

[3] P.Bernstein, F. Guinchiglia, A. Kementsietsidis, I. Zaraeu,J. Mylopoulos, "Data Management for Peer-to-Peer Computing: A Vision". In WebDB, 2002.