Schema Mediation in Peer Data Management Systems

Alon Y. Halevy Zachary G. Ives Dan Suciu Igor Tatarinov
University of Washington
Seattle, WA, USA 98195-2350
{alon,zives,suciu,igor}@cs.washington.edu

Abstract

Intuitively, data management and data integration tools should be well-suited for exchanging information in a semantically meaningful way. Unfortunately, they suffer from two significant problems: they typically require a comprehensive schema design before they can be used to store or share information, and they are difficult to extend because schema evolution is heavyweight and may break backwards compatibility. As a result, many small-scale data sharing tasks are more easily facilitated by non-database-oriented tools that have little support for semantics.

The goal of the peer data management system (PDMS) is to address this need: we propose the use of a decentralized, easily extensible data management architecture in which any user can contribute new data, schema information, or even mappings between other peers' schemas. PDMSs represent a natural step beyond data integration systems, replacing their single logical schema with an interlinked collection of semantic mappings between peers' individual schemas.

This paper considers the problem of schema mediation in a PDMS. Our first contribution is a flexible language for mediating between peer schemas, which extends known data integration formalisms to our more complex architecture. We precisely characterize the complexity of query answering for our language. Next, we describe a reformulation algorithm for our language that generalizes both global-as-view and local-as-view query answering algorithms. Finally, we describe several methods for optimizing the reformulation algorithm, and an initial set of experiments studying its performance.

1. Introduction

While databases and data management tools excel at providing semantically rich data representations and expressive query languages, they have historically been hindered by a need for significant investment in design, administration, and schema evolution. Schemas must generally be predefined in comprehensive fashion, rather than evolving incrementally as new concepts are encountered; schema evolution is typically heavyweight and may "break" existing queries. As a result, many people find that database techniques are obstacles to lightweight data storage and sharing tasks, rather than facilitators. They resort to simpler and less expressive tools, ranging from spreadsheets to text files, to store and exchange their data. This provides a simpler administrative environment (although some standardization of terminology and description is always necessary), but with a significant cost in functionality. Worse, when a lightweight repository grows larger and more complex in scale, there no easy migration path to a semantically richer tool.

Conversely, the strength of HTML and the World Wide Web has been easy and intuitive support for ad hoc extensibility — new pages can be authored, uploaded, and quickly linked to existing pages. However, as with flat files, the Web environment lacks rich semantics. That shortcoming spurred a movement towards XML, which allows data to be semantically tagged. Unfortunately, XML carries many of the same requirements and shortcomings as data management tools: for rich data to be shared among different groups, all concepts need to be placed into a common frame of reference. XML schemas must be completely standardized across groups, or mappings must be created between all pairs of related data sources.

Data integration systems have been proposed as a partial solution to this problem [11, 13, 3, 19, 9, 21]. These systems support rich queries over large numbers of autonomous, heterogeneous data sources by exploiting the semantic relationships between the different sources' schemas. An administrator defines a global *mediated schema* for the application domain and specifies semantic mappings between sources and the mediated schema. We get the strong semantics needed by many applications, and data sources can evolve independently — and, it would appear, relatively flexibly. Yet in reality, the mediated schema, the integrated part of the system that actually facilitates all information sharing, becomes a bottleneck in the process. Mediated schema design must be done carefully and globally; data sources cannot change significantly or they might violate the mappings to the mediated schema; concepts can only be added to the mediated schema by the central administrator. The ad hoc extensibility of the web is missing, and as a result many natural, small-scale information sharing tasks are difficult to achieve.

We believe that there is a clear need for a new class of data sharing tools that preserves semantics and rich query languages, but which facilitates ad hoc, decentralized sharing and administration of data and defining of semantic relationships. Every participant in such an environment should be able to contribute new data and relate it to existing concepts and schemas, define new schemas that others can use as frames of reference for their queries, or define new relationships between existing schemas or data providers. We believe that a natural implementation of such a system will be based on a peer-to-peer architecture, and hence call such a system a *peer data management system* (PDMS). (We comment shortly on the differences between PDMSs and P2P file-sharing systems). The vision of a PDMS is to blend the extensibility of the HTML web with the semantics of data management applications.

Example 1.1 The extensibility of a PDMS can best be illustrated with a simple example. Figure 1 illustrates a peer data management system for emergency services at the Oregon-Washington border (this will be a running example throughout the paper, so we only describe the functionality here). Unlike a hierarchy of data integration systems, a PDMS supports any arbitrary network of relationships between peers, but the true novelty lies in the PDMS's ability to exploit transitive relationships among peers' schemas. In the event of an earthquake, the peers drawn within the ellipse at the right of the figure may join the example PDMS. Mappings will be specified between the Earthquake Command Center (ECC) and the existing 911 Dispatch Center (9DC) — now, via transitive evaluation of semantic mappings, any queries over *either* the original 9DC or the ECC peer will make use of *all* of the source relations (hospital, fire, National Guard, and Washington State). □

Our contributions: We are building the Piazza PDMS, whose goal is to support decentralized sharing and administration of data in the extensible fashion described above. Piazza investigates many of the logical, algorithmic, and implementation aspects of peer data management. In this paper, we focus strictly on the problem of providing decentralized schema mediation, specifically on the topics of expressing mappings between schemas in such a system and answering queries over multiple schemas.

Research on data integration has provided a set of rich and well understood schema mediation languages upon which mediation in PDMSs can be built. The two commonly used formalisms are the *global-as-view* (GAV) approach used by [11, 13, 3], in which the mediated schema is defined as a set of views over the data sources; and the *local-as-view* (LAV) approach of [19, 9, 21], in which the contents of data sources ae described as views over the mediated schema. The semantics of the formalisms are defined in terms of *certain answers* to a query [1].

Porting these languages to the PDMS context poses two challenges. First, the languages are designed to specify relationships between a mediator and a set of data sources. In our context, they need to be modified to map between peers' schemas, where each peer can serve as both a data source and mediator. Second, the algorithms and complexity of query reformulation and answering in data integration are well understood for a *two-tiered* architecture. In the context of a PDMS, we would like to use the data integration languages to specify semantic relationships *locally* between small sets of peers, and answer queries *globally* on a network of semantically related peers. The key contributions of this paper are showing precisely when these languages can be used to specify local semantic relationships in a PDMS, and developing a query reformulation algorithm that uses local semantic relationships to answer queries in a PDMS.

We begin by describing a very flexible formalism, $\mathcal{PPL}$, (Peer-Programming Language, pronounced "people") for mediating between peer schemas, which uses the GAV and LAV formalisms to specify local mappings. We define the semantics of query answering for a PDMS by extending the notion of *certain answers* [1]. We present results that show the exact restrictions on $\mathcal{PPL}$ under which finding all the answers to the query can be done in polynomial time.

We then present a query reformulation algorithm for $\mathcal{PPL}$. Reformulation takes as input a peer's query and the formulas describing semantic relationships between peers, and it outputs a query that refers only to stored relations at the peers. Reformulation is challenging because peer mappings are specified locally, and answering a query may require piecing together multiple peer mappings to locate the relevant data. In uniform fashion, our algorithm interleaves both global-as-view and local-as-view reformulation techniques. The algorithm is guaranteed to yield all the certain answers when they are possible to obtain. We describe several methods for optimizing the reformulation algorithm and demonstrate its performance in a number of scenarios. Optimization of reformulation is a critical issue in the PDMS context because the algorithm may need to follow any path through semantically related peers, which may be as long as the diameter of the PDMS. Second, since data may be replicated in many peers, the branching factor of the algorithm may be high.

Before we proceed, we would like to emphasize the following points. First, this paper is not concerned with how semantic mappings are generated: this is an entire field of investigation in itself (see [24] for a recent survey on schema mapping techniques). Second, while a PDMS is based on a peer-to-peer architecture, it is significantly different from a P2P file-sharing system (e.g., [22]). In particular, joining a PDMS is inherently a more heavyweight operation than joining a P2P file-sharing system, since some semantic relationships need to be specified. Our initial archi-

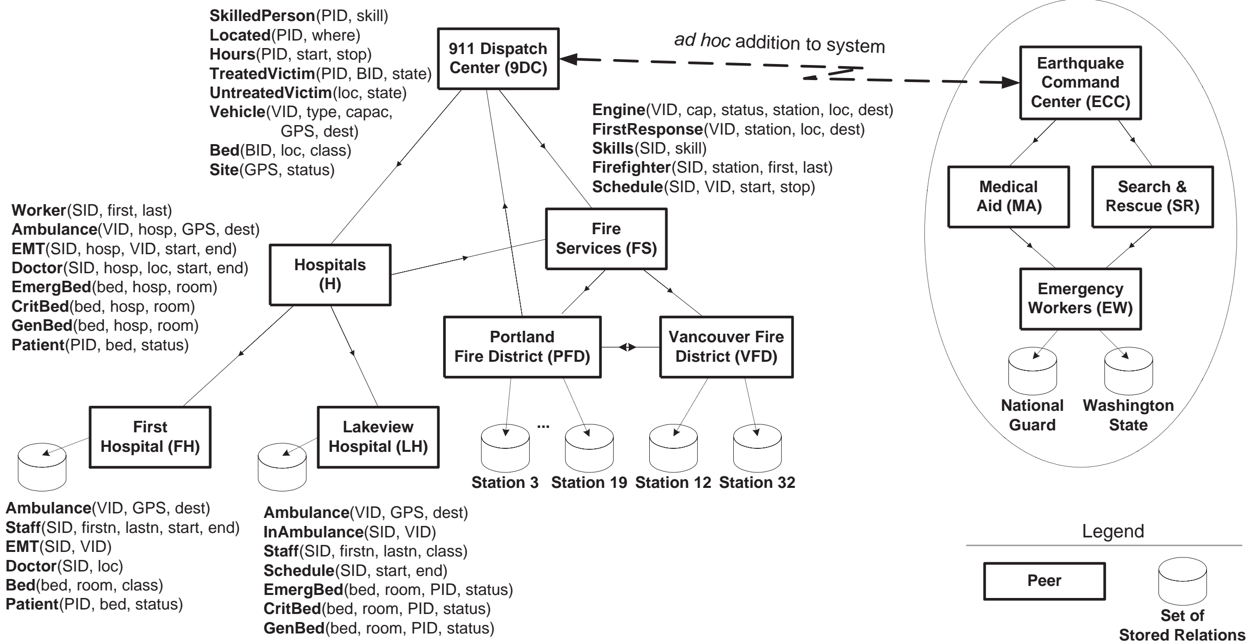

Figure 1. PDMS for coordinating emergency response in the Portland and Vancouver areas. Arrows indicate that there is (at least a partial) mapping between the relations of the peers. Stored relations are located at various fire stations and hospitals. The hospitals and fire districts run peers within the PDMS, publishing the stored relations for system use. Next, the Hospitals and Fire Services peers mediate between the incompatible schemas at the layer below. Finally, a 911 Dispatch Center provides a global view of all emergency services. In the event of an earthquake, a new Command Center and new relief workers can be added on an *ad hoc* basis, and they will be immediately integrated with existing services.

tecture focuses on applications where peers are likely to stay available the majority of the time, but in which peers should be able to join (or add new data) very easily. We anticipate there will be a spectrum of PDMS applications, ranging from more ad-hoc sharing scenarios to ones in which the membership changes less frequently or is restricted due to security or consistency requirements. Finally, we note that PDMS provide an infrastructure on which to build applications of the *Semantic Web* [4], which essentially share the vision of large-scale data sharing systems on the Web.

The paper is organized as follows. Section 2 formally defines the peer mediation problem and describes our mediation formalism. Section 3 shows the conditions under which query answering can be done efficiently in our formalism. In Section 4 we describe a query reformulation algorithm for a PDMS, and Section 5 describes the results of our experiments. Section 6 discusses related work and Section 7 concludes.

2. Problem definition

In this section, we present the logical formalisms for describing a PDMS and the specification of semantic mappings between peers. Our goal is to leverage the techniques for specifying mappings in data integration systems, extending them beyond the two-tiered architecture.

In our discussion, for simplicity of exposition we assume the peers employ the relational data model, although in our implemented system peers share XML files and pose queries in a subset of XQuery that uses set-oriented semantics. Our discussion considers select-project-join queries with set semantics, and we use the notation of conjunctive queries. In this notation, joins are specified by multiple occurrences of the same variable. Unless explicitly specified, we assume queries do not contain comparison predicates (e.g., $\neq$, $<$). Views refer to named queries.

We assume that each peer defines its own relational *peer schema* whose relations are called *peer relations*; a query in a PDMS will be posed over the relations from a specific peer schema. Without loss of generality we assume that relation and attribute names are unique to each peer.

Peers may also contribute data to the system, in the form of *stored relations*. Stored relations are analogous to data sources in a data integration system: all queries in a PDMS will be reformulated strictly in terms of stored relations that may be stored locally or on other peers. (Note that not every peer needs to contribute stored relations to the system, as some peers may strictly serve as logical mediators to other peers.) We assume that the names of stored relations are distinct from those of peer relations.

Example 2.1 Figure 1 illustrates many of the peer and source relations in an example PDMS for coordinating

emergency response: relations listed near the rectangles are peer relations, and those listed near the cylinders are source relations stored at the lowest-level peers. Lines between peers illustrate that there is a mapping (described later) between the relations of the two peers.

Stored relations containing actual data are provided by the hospitals and fire stations (the FH, LH, PFD, and VFD peers). The two fire-services peers (PFD and VFD) can share data because there are mappings between their peer relations. Additionally, the FS peer provides a uniform view of all fire services data. Similarly, H provides a unified view of hospital data. The 911 Dispatch Center (9DC) peer unites all emergency services data.

The flexibility of the PDMS (due to ability to evaluate transitive relationships between schemas) becomes evident when an earthquake occurs: an Earthquake Command Center (ECC) and other related peers join the system. Once mappings between the ECC and the existing 911 Dispatch Center are provided, queries over *either* the 9DC or ECC peers will be able to make use of *all* of the source relations. □

We note that when a peer submits a query, it may not always be interested in obtaining all possible data from anywhere in the PDMS. We ignore this issue in our discussion, and assume that restrictions on data sources can be specified via the user interface or that answers can be annotated appropriately for the user.

2.1. A Mapping Language for PDMSs

Obviously, the power of the PDMS lies in its ability to exploit semantic mappings between peer and stored relations. In particular, there are two types of mappings that must be considered: (1) mappings describing the data within the stored relations (generally with respect to one or more peer relations), and (2) mappings between the schemas of the peers. At this point it is instructive to recall the formalisms used in the context of data integration systems, since we build upon them in defining our mapping description language.

2.1.1 Mappings in Data Integration

Data integration systems provide a uniform interface to a multitude of data sources through a logical, virtual *mediated* schema. (The mediated schema is virtual in the sense that it is used for posing queries, but not for storing data.) Mappings are established between the mediated schema and the relations at the data sources, forming a two-tier architecture in which queries are posed over the mediated schema and evaluated over the underlying source relations. A data integration system can be viewed as a special case of a PDMS.

Two main formalisms have been proposed for schema mediation in data integration systems. In the first, called *global-as-view* (GAV) [25, 11, 13, 3], the relations in the mediated schema are defined as views over the relations in the sources. In the second, called *local-as-view* (LAV) [19, 9, 21], the relations in the sources are specified as views over the mediated schema. In fact, in many cases the source relations are said to be *contained* in a view over the mediated schema, as opposed to being exactly equal to it. We illustrate both below.

Example 2.2 The 911 Dispatch Center's SkilledPerson peer relation, which mediates Hospital and Fire Services relations, may be expressed using a GAV-like definition. The definition specifies that SkilledPerson in the 9DC is obtained by a union over the H and FS schemas. Note in our examples, that peer relations are named using a peer-name:relation-name syntax.

$$
\begin{array}{ll}
\mathsf{9DC:SkilledPerson(PID, ``Doctor")} & :- \\
\quad \mathsf{H:Doctor(SID, h, l, s, e)} & \\
\mathsf{9DC:SkilledPerson(PID, ``EMT")} & :- \\
\quad \mathsf{H:EMT(SID, h, vid, s, e)} & \\
\mathsf{9DC:SkilledPerson(PID, ``EMT")} & :- \\
\quad \mathsf{FS:Schedule(PID, vid),} & \\
\quad \mathsf{FS:1stResponse(vid, s, l, d),} & \\
\quad \mathsf{FS:Skills(PID, ``medical")} &
\end{array}
$$

We may use the LAV formalism to specify the Lakeview Hospital peer relations as views over mediated Hospital relations. The LAV formalism is especially useful when there are many data sources that are related to a particular mediated schema. In such cases, it is more convenient to describe the data sources as views over the mediated schema rather than the other way around. In our scenario, H may eventually mediate between many hospitals, and hence LAV is appropriate for future extensibility. The following illustrates LAV mappings for one of the hospitals.

$$
\begin{array}{ll}
\mathsf{LH:CritBed(bed, hosp, room, PID, status)} & \subseteq \\
\quad \mathsf{H:CritBed(bed, hosp, room),} & \\
\quad \mathsf{H:Patient(PID, bed, status)} & \\
\mathsf{LH:EmergBed(bed, hosp, room, PID, status)} & \subseteq \\
\quad \mathsf{H:EmergBed(bed, hosp, room),} & \\
\quad \mathsf{H:Patient(PID, bed, status)} & \\
\mathsf{LH:GenBed(bed, hosp, room, PID, status)} & \subseteq \\
\quad \mathsf{H:GenBed(bed, hosp, room),} & \\
\quad \mathsf{H:Patient(PID, bed, status)} &
\end{array}
$$

□

The fundamental difference between the two formalisms is that GAV specifies how to extract tuples for the mediated schema relations from the sources, and hence query answering amounts to view unfolding. In contrast, LAV is source-centric, describing the contents of the data sources. Query answering requires algorithms for answering queries using views [14], but in exchange LAV provides greater extensibility: the addition of new sources is less likely to require a change to the mediated schema.

Our goal in $\mathcal{PPL}$ is to preserve the features of both the GAV and LAV formalisms, but to extend them from a two-tiered architecture to our more general network of interrelated peer and source relations. Semantic relationships in a PDMS will be specified between pairs (or small sets) of peer (and optionally source) relations. Ultimately, a query over a given peer relation may be reformulated over source relations on any peer in the transitive closure of peer mappings.

2.1.2 Mappings for PDMSs

We now present the $\mathcal{PPL}$ language, which uses the data integration formalisms locally. First we formally define our two types of mappings, which we refer to as storage descriptions and peer mappings.

Storage descriptions: Each peer contains a (possibly empty) set of *storage descriptions* that specify which data it actually stores by relating its stored relations to one or more peer relations. Formally, a storage description of the form $A : R = Q$, where Q is a query over the schema of peer A and R is a stored relation at the peer. The description specifies that A stores in relation R the result of the query Q over its schema.

In many cases the data that is stored is not *exactly* the definition of the view, but only a subset of it. As in the context of data integration, this situation arises often when the data at the peer may be incomplete (this is often called the *open-world assumption* [1]).[1] Hence, we also allow storage descriptions of the form $A : R \subseteq Q$. We call the latter descriptions *containment descriptions* versus *equality descriptions*.

Example 2.3 An example storage description might relate stored doctor relations at First Hospital to the peer relations.

$$\begin{array}{lll}\mathsf{doc(sid, last, loc)} & \subseteq & \mathsf{FH : Staff(sid, f, last, s, e),} \\ & & \mathsf{FH : Doctor(sid, loc)} \\ \mathsf{sched(sid, s, e)} & \subseteq & \mathsf{FH : Staff(sid, f, last, s, e),} \\ & & \mathsf{FH : Doctor(sid, loc)}\end{array} \quad \Box$$

Peer mappings: *Peer mappings* provide semantic glue between the schemas of different peers. We have two types of peer mappings in $\mathcal{PPL}$. The first are *inclusion* and *equality* mappings (similar to the concepts for storage descriptions). In the most general case, these mappings are of the form $Q_1(\bar{A}_1) = Q_2(\bar{A}_2)$, (or $Q_1(\bar{A}_1) \subseteq Q_2(\bar{A}_2)$ for inclusions) where Q_1 and Q_2 are conjunctive queries with the same arity and $\bar{A}_1$ and $\bar{A}_2$ are *sets* of peers. Query Q_1 (Q_2) can refer to any of the peer relation in $\bar{A}_1$ ($\bar{A}_2$, resp.). Intuitively, such a statement specifies a semantic mapping by stating that evaluating Q_1 over the peers $\bar{A}_1$ will always produce the same answer (or a subset in the case of inclusions) as evaluating Q_2 over $\bar{A}_2$. Note that since $\mathcal{PPL}$ allows queries on both sides of the equation, they can accommodate both GAV and LAV-style mappings (and thus we can express any of the mappings from Section 2.1.1).

The second kind of peer mappings are called *definitional mappings*. They are datalog rules whose relations (both head and body) are peer relations. Formally, as long as a peer relation appears only once in the head of a definitional description, such mappings can be written as equalities. We include definitional mappings in order to obtain the full power of GAV mappings. We distinguish definitional mappings for the following reasons:

- as we show in Section 3, the complexity of answering queries when equality mappings are restricted to being definitional is more attractive than the general case, and
- definitional mappings can easily express disjunction: e.g., $P(x) : -P_1(x)$ and $P(x) : -P_2(x)$ means that P is the union of P_1 and P_2 (while the pair of mappings $P(x) = P_1(x)$ and $P(x) = P_2(x)$ means that P, P_1 and P_2 are equal).

In summary, a PDMS N is specified by a set of peers $\{P_1, ..., P_n\}$, a set of peer schemas $\{S_1, ..., S_m\}$ and a mapping function from peers to schemas, a set of stored relations $\mathcal{R}_i$ at each peer P_i, a set of peer mappings $\mathcal{L}_N$, and a set of storage descriptions $\mathcal{D}_N$. The storage descriptions and peer mappings provided by a peer P_i may reference stored or peer relations defined by other peers, so any peer can extend another peer's relations or use its data.

2.2. Semantics of $\mathcal{PPL}$

Given the peer and stored relations, their mappings, and a query over some peer schema, the PDMS needs to answer the query using the data from the stored relations. To formally specify the problem of query answering, we need to define the semantics of queries. We show below how the notion of *certain answers* [1] from the data integration context can be generalized to our context. Our goal is to formally define what is the set of answers to a query Q posed over the relations of a peer A. The challenge arises because the peer schemas are virtual; in fact, some data may only exist partially, if at all, in the system.

Formally, we assume that we are given a PDMS N and an instance for the stored relations, D, i.e., a set of tuples $D(R)$ for each stored relation $R \in (\mathcal{R}_1 \cup \ldots \cup \mathcal{R}_n)$. A *data instance* I for a PDMS N is an assignment of a set of tuples to each relation in each peer (both the peer and stored relations). We denote by $I(R)$ the set of tuples assigned to the relation R by I, and we denote by $Q(I)$ the result of computing the query Q over the extensional data in I.

[1] Sometimes it may be possible to describe the exact contents of a data source with a more refined query, but very often this cannot be done.

To define certain answers, we will consider only the data instances that are consistent with the specification of N:

Definition 2.1 (**Consistent data instance**) A data instance I is said to be *consistent* with a PDMS N and an instance D for N's stored relations if:

- For every storage description in $\mathcal{D}_N$, if it is of the form $A : R = Q_1$ ($A : R \subseteq Q_1$), then $D(R) = Q_1(I)$ ($D(R) \subseteq Q_1(I)$).
- For every peer description in $\mathcal{L}_N$:
  - if it is of the form $Q_1(\mathcal{A}_1) = Q_2(\mathcal{A}_2)$, then $Q_1(I) = Q_2(I)$,
  - if it is of the form $Q_1(\mathcal{A}_1) \subseteq Q_2(\mathcal{A}_2)$, then $Q_1(I) \subseteq Q_2(I)$,
  - if it is a definitional description whose head predicate is p, then let $r_1, \ldots, r_m$ be all the definitional mappings with p in the head, and let $I(r_i)$ be the result of evaluating the body of r_i on the instance I. Then, $I(p) = I(r_1) \cup \ldots \cup I(r_m)$. □

Intuitively, a data instance I is consistent with N and D if it describes *one* possible state of the world (i.e., extension for each of the peer relations) that is allowable given the data and peer mappings and D. We define the certain answers to be those that hold in *every* possible consistent data instance:

Definition 2.2 (**Certain answers**) Let Q be a query over the schema of a peer A in a PDMS N, and let D be an instance of the stored relations of N. A tuple $\bar{a}$ is a *certain answer* to Q if $\bar{a}$ is in $Q(I)$ for every data instance that is consistent with N and D. □

Note that in the last bullet of Definition 2.1 we did not require that the extension of p be the least-fixed point model of the datalog rules. However, since we defined certain answers to be those that hold for *every* consistent data instance, we actually do get the intuitive semantics of datalog for these mappings.

Query answering: Now we can define the query answering problem: given a PDMS N, an instance of the stored relations D and a query Q, find all certain answers of Q.

Section 3 considers the computational complexity of query answering, and section 4 describes an algorithm for finding all the certain answers.

3. Complexity of Query Answering

This section establishes the basic results on the complexity of finding the certain answers in a PDMS. The complexity will depend on the restrictions we impose on peer mappings in $\mathcal{PPL}$. The computational complexity of finding all certain answers is well understood for the data integration context with a two-tiered architecture of a mediator and a set of data sources [1]. The key contribution of this section is to show the complexity of query answering in the global context of a PDMS, when the data integration formalisms are used locally.

The focus of our analysis is on data complexity — the complexity of query answering in terms of the total size of the data stored in the peers. Typically, the complexity of query answering is either polynomial, Co-NP-hard but decidable, or undecidable. In the polynomial case, it is often possible to find a *reformulation* of the query into a query that refers only to the stored relations. The reformulated query is then further optimized and then executed. In the latter two cases, it is not possible to find *all* certain answers efficiently; but it is possible to develop an efficient reformulation algorithm that does not provide all certain answers, but which *only* returns certain answers.

A basic result: We begin by showing that cyclicity of peer mappings plays a significant role in the complexity of answering queries.

Definition 3.1 (**Acyclic inclusion peer mappings**) A set $\mathcal{L}$ of inclusion peer mappings in $\mathcal{PPL}$, is said to be acyclic if the following directed graph is acyclic. The graph contains a node for every peer relation mentioned in $\mathcal{L}$. There is an arc from the node corresponding to R to the node corresponding to S if there is a peer description in $\mathcal{L}$ of the form $Q_1(\bar{\mathcal{A}}_1) \subseteq Q_2(\bar{\mathcal{A}}_2)$ where R appears in Q_1 and S appears in Q_2. □

The following theorem characterizes two extreme cases of query answering in PDMS:

Theorem 3.1 *Let N be a PDMS specified in $\mathcal{PPL}$.*

1. *The problem of finding all certain answers to a conjunctive query Q, for a given PDMS N, is undecidable.*
2. *If N includes only inclusion peer and storage descriptions and the peer mappings are acyclic, then a conjunctive query can be answered in polynomial time data complexity.*

The difference in complexity between the first and second bullets shows that the presence of cycles is the culprit for achieving query answerability in a PDMS (note that equalities automatically create cycles). In a sense the theorem also establishes a limit on the arbitrary combination of the formalisms of LAV and GAV. The proof is based on a reduction from the implication problem for functional and inclusion dependencies ([2], Theorem 9.2.4).

The second bullet points out a powerful schema mediation language for PDMS for which query answering can be done efficiently. It shows that LAV and GAV style reformulations can be chained together arbitrarily, and extends the results of [10], which combined one level of LAV followed by one level of GAV.

Cyclic PDMSs: Acyclic PDMSs may be too restrictive for practical applications. One particular case of interest is *data replication*: when one peer maintains a copy of the data stored at a different peer. For example, referring to Fig. 1, the Earthquake Command Center may wish to replicate the 911 Dispatch Center's `Vehicle` table for reliability, using an expression such as:

$$\mathsf{ECC}:\mathsf{vehicle}(\mathsf{vid},\mathsf{t},\mathsf{c},\mathsf{g},\mathsf{d}) \quad = \quad \mathsf{9DC}:\mathsf{vehicle}(\mathsf{vid},\mathsf{t},\mathsf{c},\mathsf{g},\mathsf{d})$$

This example illustrates that we need equality in order to express data replication, which introduces a cyclic PDMS (the two relations mutually include each other's contents). While in general query answering is undecidable, it becomes decidable when equalities are projection-free, as in this example. The following theorem shows an important special case where query answering is tractable, and two additional cases where it is decidable.

Theorem 3.2 *Let N be a PDMS for which all inclusion peer mappings are acyclic, but which may also contain equality peer mappings.*

1. *if the following two conditions hold: (1) whenever a storage or peer description in N is an equality description, it does not contain projections, and (2) a peer relation that appears in the head of a definitional description does not appear on the right-hand side of any other description, then the query answering problem is in polynomial time.*

2. *if the conditions of the previous bullet hold, except that some equality storage descriptions contain projections, then the data complexity of the query answering problem is co-NP complete.*

3. *if the conditions of the first bullet hold, except that some of the queries on the right-hand side of the peer mappings may be unions of conjunctive queries, the data complexity of query answering is co-NP complete.*

Note that the first bullet in the theorem also allows definitional mappings to be disjunctive, if there are multiple mappings with the same head predicate. The conditions of this bullet describe the most relaxed conditions under which query answering is tractable, and extends the results of [1] for purely LAV mappings. The algorithm described in the next section will find all the certain answers under these conditions. The two subsequent bullets show that relaxing the conditions of the first bullet cause the query answering problem to be intractable.

Adding comparison predicates: Many applications will make extensive use of comparison predicates in peer mappings. Comparison predicates are especially useful when many peers model the same type of data, but they are distinguished on ranges of certain values of attributes (e.g., author names, years of publication, price ranges, geographic location). The following theorem shows what happens when comparison predicates are introduced into the peer mappings of a PDMS. We note that the algorithm we describe in the next section finds all the certain answers when the PDMS satisfies the conditions of the first bullet.

Theorem 3.3 *Let N be a PDMS satisfying the same conditions as the first bullet of Theorem 3.2, and let Q be a conjunctive query.*

1. *if comparison predicates appear only in storage descriptions or in the bodies of definitional mappings, but not in Q, then query answering is in polynomial time.*

2. *otherwise, if either the query contains comparison predicates or comparison predicates appear in non-definitional peer mappings, then the query answering problem is co-NP complete.*

Summary: with arbitrary use of the data integration formalisms in a PDMS, query answering is undecidable. However, this section has shown that there is a powerful subset of $\mathcal{PPL}$ in which query answering is tractable. The subset allows both the LAV and GAV mediation languages, and it supports a limited form of cycles in the peer mappings and as well as limited use of comparison predicates.

4. Query Reformulation Algorithm

In this section we describe an algorithm for query reformulation for PDMSs. The input of the algorithm is a set of peer mappings and storage descriptions and a query Q. The output of the algorithm is a query expression Q' that *only* refers to stored relations at the peers. To answer Q we need to evaluate Q' over the stored relations. The precise method of evaluating Q' is beyond the scope of this paper, but we note that recent techniques for adaptive query processing [16] are well suited for our context.

The algorithm is sound and complete in the following sense. Evaluating Q' will always *only* produce certain answers to Q. When all the certain answers can be found in polynomial time (according to Section 3), Q' will produce all certain answers.

4.1. Algorithm overview

Before we describe the details of the algorithm, we first provide some intuition on its working and the challenges it faces. Consider a PDMS in which all peer mappings are definitional (similar to GAV mappings in data integration). In this case, the algorithm is a simple construction of a rule-goal tree: goal nodes are labeled with atoms of the peer relations, and rule nodes are labeled with peer mappings. We begin by expanding each query subgoal according to the relevant definitional peer mappings in the PDMS. When none of the leaves of the tree can be expanded any further, we use the storage descriptions for the final step of reformulation in terms of the stored relations.

At the other extreme, suppose all peer mappings in the PDMS are inclusions in which the left-hand side has a single atom (similar to LAV mappings in data integration). In this case, we begin with the query subgoals and apply an algorithm for answering queries using views (e.g., [14]). We apply the algorithm to the result until we cannot proceed further, and as in the previous case, we use the storage descriptions for the last step of reformulation.

The first challenge of the complete algorithm is to combine and interleave the two types of reformulation techniques. One type of reformulation replaces a subgoal with a set of subgoals, while the other replaces a set of subgoals with a single subgoal. The algorithm will achieve this by building a rule-goal tree, while it simultaneously marks certain nodes as covering not only their parent node, but also their uncle nodes (as described in the example below).

Example 4.1 To illustrate the rule-goal tree,[2] Figure 2 shows an example for a simple query. We begin with the query, Q, which asks for firefighters with matching skills riding in the same engine. Q is expanded into its three subgoals, each of which appears as a goal node. The SameEngine peer relation (indicating which firefighters are assigned to the same engine) is involved in a single definitional peer description (r_0), hence we expand the SameEngine goal node with the rule r_0, and its children are two goal nodes of the AssignedTo peer relation (each specifying an individual fire fighter's assignment).

The Skill relation is involved in an inclusion peer description (r_1). Hence, we expand Skill(f1,s) with the rule node r_1, and its child is a goal node of the relation SameSkill. This "expansion" is of different nature because of the LAV-style reformulation. Intuitively, we are reformulating the Skill(f1,s) subgoal to use the left-hand side of r_1. The right-hand side of r_1 includes two subgoals of Skill (with the appropriate variable patterns), so we also mark r_1 as covering its *uncle* node. (In the figure, this annotation is indicated by a dashed line.) Since the peer relation Skill is involved in a single peer description, we do not need to expand the subgoal Skill(f2,s) any further. Note, however, that we must apply description r_1 a second time with the head variables reversed, since SameSkill may not be symmetric (because it is $\subseteq$ rather than $=$).

At this point, since we cannot reformulate the peer mappings any further, we consider the storage descriptions. We find stored relations for each of the peer relations in the tree (S_1 and S_2), and produce the final reformulation. Reformulations of peer relations into stored relations can also be either in GAV or LAV style. In this simple example, our reformulation involves only one level of peer mappings, but in general, the tree may be arbitrarily deep. □

[2]More precisely, we actually build a rule-goal DAG, as illustrated in the example.

The second challenge we face is that the rule-goal tree may be huge. First, the tree may be very deep, because it may need to follow any path through semantically related peers. Second, the branching factor of the tree may be large because data is replicated at many peers. Hence, it is crucial that we develop effective methods for pruning the tree and for generating first solutions quickly. It is important to emphasize that while many sophisticated methods have been developed for constructing rule-goal trees in the context of datalog analysis (e.g., [15, 26]), the focus in these works has been developing termination criteria that provide certain guarantees, rather than optimizing the construction of the tree itself.

Before proceeding, we recall the main aspect of algorithms for rewriting queries using views [23] that is germane to our discussion. Suppose we have the following query Q and views (we use the terminology of [23]):

$$\begin{array}{lll} Q(X,Y) & :- & e_1(X,Z), e_2(Z,Y), e_3(X,Y) \\ V_1(A,B) & :- & e_1(A,C), e_2(C,B) \\ V_2(D,E) & :- & e_3(X,Y), e_4(Y) \\ V_3(U) & :- & e_1(U,Z) \end{array}$$

To find a way of answering Q using the views, we first try to find a view that will *cover* the subgoal $e_1(X,Z)$ in the query. We realize that V_1 will suffice, so we create a *Minicon description* (MCD) for it. The MCD specifies that an atom $V_1(X,Y)$ will cover the subgoal $e_1(X,Z)$, but it also specifies that the atom will cover the first *two* subgoals in Q. Similarly, we create an MCD for V_2 and the third subgoal, and finally we combine the MCDs to produce the rewriting:

$$Q'(X,Y) :- V_1(X,Y), V_2(X,Y)$$

The important point to note is that the MCD may tell us that it covers more than the original subgoal for which it was created. Furthermore, MCDs will only be created when the views are guaranteed to be useful. For example, in the case

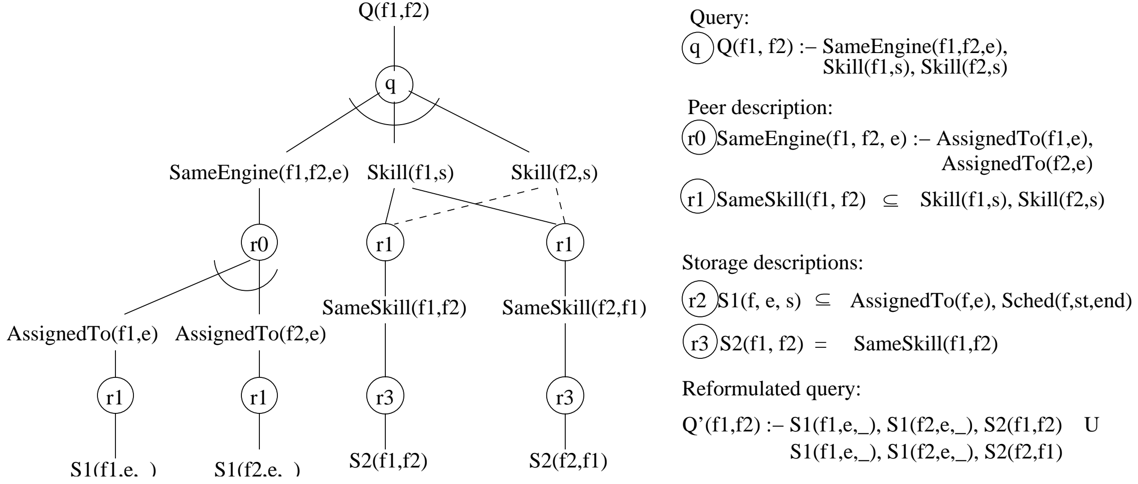

Figure 2. Reformulation rule-goal tree for Emergency Services domain. Dashed lines represent nodes that are included in the unc label (see text).

of V_3, since the variable Z is projected from the answer, the view is useless and an MCD will not be created.

We now describe the construction of the rule-goal tree in detail, deferring a discussion of the order in which we expand nodes in the tree. Later, we describe several methods for optimizing the tree's construction.

4.2. Creating the rule-goal tree

The algorithm takes as input a conjunctive query $Q(\bar{X})$ that is posed at some peer, and a set of peer mappings and storage descriptions in $\mathcal{PPL}$. We first describe the algorithm for the case in which there are no comparison predicates in the PDMS or the query.

Step 1: the algorithm transforms every equality description into two inclusion mappings. It then transforms every inclusion description of the form $Q_1 \subseteq Q_2$ into the pair: $V \subseteq Q_2$, and $V \ :- \ Q_1$, where V is a new predicate name that appears nowhere else in the peer mappings.

Step 2: the algorithm builds a rule-goal tree T. When a node n in T is a goal node, it has a label $l(n)$ which is an atom whose arguments are variables or constants. The label $l(n)$ of a rule node is a peer description (except that the child of the root is labeled with the rule defining the query). Finally, a rule node n that is labeled with an inclusion description also has a label $unc(n)$: this label always includes at least the father of n, but may also include nodes that are siblings of its father goal node. As described earlier, the reason for this label is that an MCD can cover more that the subgoal for which it was created.

The root of T is labeled with the atom $Q(\bar{X})$, and it has a single rule-node child whose children are the subgoals of the query. The tree is constructed by iterating the following step, until no leaf nodes can be expanded further.

Choose an arbitrary leaf goal node n in T whose label is $l(n) = p(\bar{Y})$, and p is not a stored relation. Perform all the expansions possible in the following two cases. In either case, never expand a goal node n with a peer description that was used on the path from the root to n. This guarantees termination of the algorithm even in a cyclic PDMS.

1. Definitional expansion: this is the case where peer relations appear in GAV-style mappings. If p appears in the head of a definitional description r, expand n with the definition of p. Specifically, let r' be the result of unifying $p(\bar{Y})$ with the head of r. Create a child rule n_r, with $l(n_r) = r'$, and create one child goal-node for n_r for every subgoal of r' with the corresponding label. Existential variables in r' should be renamed so they are fresh variables that do not occur anywhere else in the tree constructed thus far.

2. Inclusion expansion: this is the case where peer relations appear in LAV-style mappings. If p appears in the right-hand side of an inclusion description or storage description r of the form $V \subseteq Q_1$ (or $V = Q_1$), we do the following. Let $n_1, \ldots, n_m$ be the children of the father node of n, and $p_1, \ldots, p_m$ be their corresponding labels. Create an MCD for $p(\bar{Y})$ w.r.t. $p_1, \ldots, p_m$ and the description r. Recall that the MCD contains an atom of the form $V(\bar{Z})$ and the set of atoms in $p_1, \ldots, p_m$ that it covers.

Create a child rule node n_r for n labeled with r, and a child goal node n_g for n_r labeled with $V(\bar{Z})$. Set $unc(n_g)$ to be the set of subgoals covered by the MCD. Repeat this process for every MCD that can be created for $p(\bar{Y})$ w.r.t. $p_1, \ldots, p_m$ and the description r.

Step 3: we construct the solutions from T. The solution is a union of conjunctive queries over the stored relations. Each of these conjunctive queries represents one way of obtaining answers to the query from the relations stored at peers. Each of them may yield different answers unless we know that

some sources are replicas of others.

Let us consider the simple case, where only definitional mappings are used, first. The answer would be the union of conjunctive queries, each with head $Q(\bar{X})$ and a body that can be constructed as follows. Let T' be a subset of T where we arbitrarily choose a *single* child at every goal node, and for which all leaves are labeled by stored relations. The body of a conjunctive query is the conjunction of all the leaves of T'.

To accommodate inclusion expansions as well, we create the conjunctive queries as follows. In creating T's we still choose a single child for every goal node. This time, we do not necessarily have to choose *all* the children of a rule node n. Instead, given a rule node n, we need to choose a subset of the children $n_1, \ldots, n_l$ of n, such that $unc(n_1) \cup \ldots \cup unc(n_l)$ includes all of the children of n.

Remark 4.1 We note that in some cases, an MCD may cover cousins or uncles of its father node, not only its own uncles. For brevity of exposition, we ignore this detail in our discussion. However, we note that we do not compromise completeness as a result. In the worst case, we obtain conjunctive rewritings that contain redundant atoms. □

Incorporating comparison predicates: as we stated earlier, comparison predicates provide a very useful mechanism for specifying constraints on domains of stored relations or peer relations, and therefore exploiting them can lead to significant pruning of the tree. When the query or the peer mappings and storage descriptions include comparison predicates we modify the algorithm as follows. We associate with each node n a constraint-label $c(n)$. The constraint label describes the conjunction of comparison predicates that are known to hold on the variables in $l(n)$.

As we build T, constraints get added and propagated to child nodes. Specifically, suppose we expand a node n with a definitional description r, and let $c_1 \wedge \ldots \wedge c_m$ be the comparison predicates in r. Then we set $c(r)$ to be $c(n) \wedge c_1 \wedge \ldots \wedge c_m$, and the labels of its children are the projections of $c(r)$ on the variables of the child.[3] When we expand a goal node with an inclusion peer description then the MCD will be created w.r.t. the constraints in the parent and in the peer description. Finally, we do not expand a node in the tree if its label is not satisfiable (this implies that it can only yield the empty set of answers to Q).

In step 3, when we construct the conjunctive queries, we add to them the conjunction of their constraint labels. If the resulting conjunctive query in unsatisfiable, we discard it. Note that constraints can also be propagated *up* the tree (in the same spirit at the predicate move-around algorithm [18]), thereby detecting additional unsatisfiable labels during the construction of the tree.

[3]When a conjunction of constraints is projected on a subset of the variables, the result may be a disjunction of constraints. The algorithm can either choose to manipulate such disjunctions or approximate them by the least subsuming conjunctions.

4.3. Optimizations

As explained earlier, a major challenge for reformulation in the context of PDMS is optimizing the construction of the rule-goal tree. Up to this point we described which nodes need to be in the tree. We now briefly describe several optimization opportunities for this context. Several optimizations can immediately be borrowed from techniques developed for evaluation of datalog and logic programs, but *lifted* from the data level to the expression level: (1) memoization of nodes, (2) detection of dead ends and useless paths. Note that in the presence of comparison predicates, a node n can become unreachable is if its constraint label $c(n)$ is unsatisfiable. This may occur because the stored relations we have access to certain data that is known to be disjoint from what is requested in the query.

A more subtle case in which useless paths can be detected is as follows. Suppose we have two sibling goal nodes with labels $p_1(\bar{X})$ and $p_2(\bar{Y})$, and suppose that p_1 appears in a *single* inclusion peer description of the form $V(\bar{Z}) \subseteq p_1(\bar{X}), p_2(\bar{Y})$, and that predicate p_2 appears on the right-hand side of numerous inclusion peer mappings. In this case, the only way to reformulate p_1 will be through V, and V already satisfies the subgoal $p_2(\bar{Y})$. Hence, there is no need to explore any of the other ways of reformulating p_2: they are all redundant.

While these optimizations have significant potential, the challenge is to build the tree in an *order* that most exploits them. The goal is to find the dead ends as early as possible to maximize the pruning. Our algorithm employs a priority scheme in expanding nodes: it assigns every node a certain priority based on how likely it is to yield useful pruning. Finally, we note that in many contexts, there will be a large number of reformulations, and hence an important optimization is to generate the *first* reformulations quickly so query execution can begin (in the spirit of [8]).

5. Experiments

This section describes an initial set of experiments concerning the performance of our reformulation algorithm. Currently, the major impediment to performing experiments at this point is the lack of existing PDMS to test on. Hence, our experiments are based on a workload generator that produces PDMS for several reasonable topologies.

The parameters to the generator are: (1) the number of peers $\mathcal{R}$ in the system, and (2) the expected diameter $\mathcal{L}$ of the PDMS (i.e., the longest chain of peer mappings that can be constructed). Intuitively, the diameter of the PDMS will correspond to the number of levels of goal nodes in the tree. We call each such level a stratum, and to create the PDMS,

we assign a number of peers to each stratum. The generator also controls the ratio of definitional versus inclusion peer mappings. Finally, the right-hand sides of the peer mappings are chain queries over a set of relations that was selected randomly from the stratum below (for definitional mappings) and above (for inclusions). In our figures, each data point is generated from the average of 100 runs.

Figure 3 shows the size of the tree (number of nodes) as a function of the number of strata, and the percent of definitional peer mappings (in the figure, %dd denotes the percent of definitional mappings). As shown, with 8 strata, the size of the tree grows to 30,000 nodes. On average, the algorithm generates nodes at a rate of 1,000 per second (with relatively unoptimized code). We note that the size of the tree grows with the relative percent of definitional mappings. The reason for this is that we get more peer relations that are defined as unions of conjunctive queries, and hence a higher branching factor in the tree.

Figure 4 shows that despite the large trees, the first rewritings can be found efficiently. For example, even with a diameter of 8, finding the first few rewritings can be done in under 3 seconds. Hence, we believe that in practice our algorithm will scale gracefully to large PDMS.

The main conclusions from our experiments are the following. First, the key bottleneck of the algorithm is the time to find the rewritings from the rule-goal tree (step 3), whereas step 2 scales up to rather large trees. Hence, an important issue is to tune the algorithm to produce the first rewritings as quickly as possible. Second, the main factor determining the size of the rule-goal tree is the diameter of the PDMS. In contrast, the number of peers at every stratum has a relatively little effect, because it is usually the case that most of them are irrelevant to a given query.

6. Related Work

The idea of mediating between different databases using local semantic relationships is not new. Federated databases and cooperative databases have used the notion of inter-schema dependencies to define semantic relationships between databases in a federation (e.g., [20]). In previous proposals, it was assumed that each database in the federation stored data, and hence the focus was on mapping between the stored relations in the federation. Our work differs in several ways. First, the scale of a PDMS is assumed to be much larger and its structure more ad hoc. Joining and leaving a PDMS should be much easier than in a federated database. As a consequence, the relationships between the peers are much looser. Second, peers can play different roles — some provide data, others provide integration services between other peers, and some provide both. As a result, we need to be able to map both relationships among stored relations and among conceptual relations (i.e., extensional vs. intentional relations). Third, our focus is on algorithms for chaining through multiple peer mappings in order to locate data relevant to a query.

In [12] we described some of the challenges involved in building a PDMS, focusing on *intelligent data placement*, a technique for materializing views at nodes in the network in order to improve performance and availability. In [17] the authors study a variant of the data placement problem, and focus on intelligently caching and reusing queries in an OLAP environment. Recently, [5] described *local relational models* as a formalism for mediating between different peers in a PDMS, and a sound and complete algorithm for answering queries using the formalism, but do not describe the expressive power of the formalism compared to previous ones in the data integration literature.

Description logics offer an alternative formalism for specifying peer relationships [7, 6]. We chose conjunctive queries for our formalism mostly because we believe that the join, selection and projection operations are the fundamental core necessary for expressing useful queries.

7. Conclusions

The concept of the peer data management system emphasizes not only an ad-hoc, scalable, distributed peer-to-peer computing environment (which is compelling from a distributed systems perspective), but it provides an easily extensible, decentralized environment for sharing data with rich semantics. This is in contrast to data integration systems, which have a centralized mediated schema and administrator, and which, in our experience, impede small, point-to-point collaborations.

We presented a solution to schema mediation in peer data management systems. We described $\mathcal{PPL}$, a flexible mediation scheme for PDMSs, which uses previous mediation formalisms at the local level to form a network of semantically related peers. We characterized the theoretical limitations on answering queries in $\mathcal{PPL}$-PDMSs. Next, we described a query reformulation algorithm for $\mathcal{PPL}$. The primary contribution of the algorithm is that it combines both LAV- and GAV-style reformulation in a uniform fashion, and it is able to chain through multiple peer descriptions to reformulate a query. We described optimization methods for reformulation, and some experimental results that show its utility. The final result is a practical solution for schema mediation in PDMS.

Future research includes reconciling peers with inconsistent integrity constraints, and considering richer constraint languages at the peers. More generally, peer data management is a very rich domain that creates a wealth of new problems, such as how to replicate data, how to reconcile inconsistent data, and optimization across multiple peers.

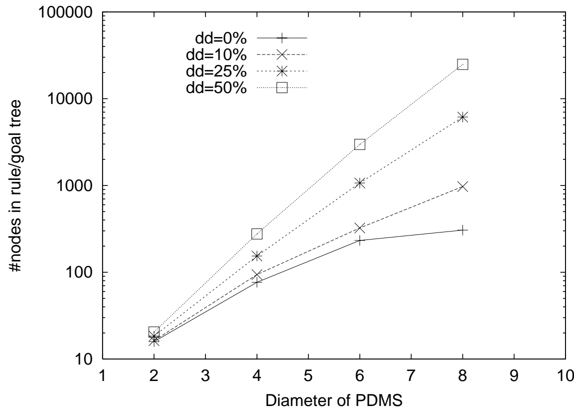

Figure 3. The size of the rule/goal tree for different diameters of a 96-peer PDMS.

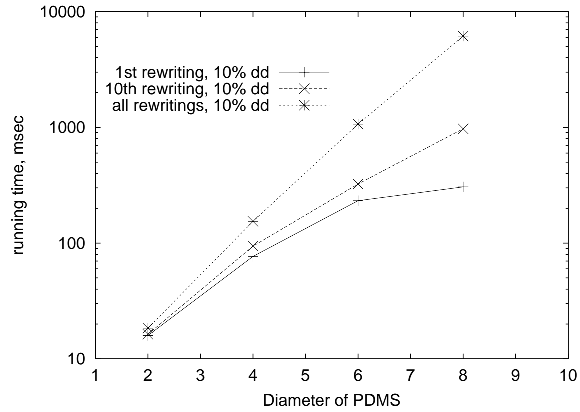

Figure 4. The time to first answers (96 peers).

References

[1] S. Abiteboul and O. Duschka. Complexity of answering queries using materialized views. In *Proc. of PODS*, pages 254–263, Seattle, WA, 1998.

[2] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison Weseley, 1995.

[3] S. Adali, K. Candan, Y. Papakonstantinou, and V. Subrahmanian. Query caching and optimization in distributed mediator systems. In *Proc. of SIGMOD*, pages 137–148, Montreal, Canada, 1996.

[4] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, May 2001.

[5] P. Bernstein, F. Giunchiglia, A. Kementsietsidis, J. Mylopoulos, L. Serafini, and I. Zaihrayeu. Data management for peer-to-peer computing : A vision. In *ACM SIGMOD WebDB Workshop 2002*, 2002.

[6] D. Calvanese, D. G. Giuseppe, and M. Lenzerini. Ontology of Integration and Integration of Ontologies. In *DL*, 2001.

[7] T. Catarci and M. Lenzerini. Representing and using interschema knowledge in cooperative information systems. *Journal of Intelligent and Cooperative Information Systems*, pages 55–62, 1993.

[8] A. Doan and A. Halevy. Efficiently ordering query plans for data integration. In *Proc. of ICDE*, 2002.

[9] O. M. Duschka and M. R. Genesereth. Answering recursive queries using views. In *Proc. of PODS*, pages 109–116, Tucson, Arizona., 1997.

[10] M. Friedman, A. Levy, and T. Millstein. Navigational plans for data integration. In *Proceedings of the National Conference on Artificial Intelligence*, 1999.

[11] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, and J. Widom. The TSIMMIS project: Integration of heterogeneous information sources. *Journal of Intelligent Information Systems*, 8(2):117–132, March 1997.

[12] S. Gribble, A. Halevy, Z. Ives, M. Rodrig, and D. Suciu. What can databases do for peer-to-peer? In *ACM SIGMOD WebDB Workshop 2001*, 2001.

[13] L. Haas, D. Kossmann, E. Wimmers, and J. Yang. Optimizing queries across diverse data sources. In *Proc. of VLDB*, Athens, Greece, 1997.

[14] A. Y. Halevy. Answering queries using views: A survey. *VLDB Journal*, 10(4), 2001.

[15] A. Y. Halevy, I. Mumick, Y. Sagiv, and O. Shmueli. Static analysis in datalog extensions. *Journal of the ACM*, 48(5):971–1012, September 2001.

[16] Z. G. Ives, A. Y. Halevy, and D. S. Weld. Integrating network-bound XML data. *IEEE Data Engineering Bulletin Special Issue on XML*, 24(2), June 2001.

[17] P. Kalnis, W. Ng, B. Ooi, D. Papadias, and K. Tan. An adaptive peer-to-peer network for distributed caching of olap results. In *Proc. of SIGMOD*, 2002.

[18] A. Y. Levy, I. S. Mumick, and Y. Sagiv. Query optimization by predicate move-around. In *Proc. of VLDB*, pages 96–107, Santiago, Chile, 1994.

[19] A. Y. Levy, A. Rajaraman, and J. J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proc. of VLDB*, pages 251–262, Bombay, India, 1996.

[20] W. Litwin, L. Mark, and N. Roussopoulos. Interoperability of multiple autonomous databases. *ACM Computing Surveys*, 22 (3):267–293, 1990.

[21] I. Manolescu, D. Florescu, and D. Kossmann. Answering xml queries on heterogeneous data sources. In *Proc. of VLDB*, pages 241–250, 2001.

[22] Napster. World-wide web: www.napster.com, 2001.

[23] R. Pottinger and A. Halevy. Minicon: A scalable algorithm for answering queries using views. *VLDB Journal*, 2001.

[24] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB Journal*, 10(4):334–350, 2001.

[25] J. M. Smith, P. A. Bernstein, U. Dayal, N. Goodman, T. Landers, K. Lin, and E. Wong. Multibase – integrating heterogeneous distributed database systems. In *Proceedings of the National Computer Conference*, pages 487–499. AFIPS Press, Montvale, NJ, 1981.

[26] D. Srivastava and R. Ramakrishnan. Pushing constraint selections. In *Proc. of PODS*, pages 301–315, San Diego, CA., 1992.