The Piazza Peer Data Management Project

Igor Tatarinov[1], Zachary Ives[2], Jayant Madhavan[1],
Alon Halevy[1], Dan Suciu[1], Nilesh Dalvi[1], Xin (Luna) Dong[1],
Yana Kadiyska[1], Gerome Miklau[1], Peter Mork[1]

[1]Department of Computer Science and Engineering
University of Washington, Seattle, WA 98195
{igor,jayant,alon,suciu,nilesh,lunadong,ykadiysk,gerome,pmork}@cs.washington.e du

[2]Department of Computer and Information Science
University of Pennsylvania, Philadelphia, PA 19103
zives@cis.upenn.edu

ABSTRACT

A major problem in today's information-driven world is that sharing heterogeneous, semantically rich data is incredibly difficult. Piazza is a *peer data management* system that enables sharing heterogeneous data in a distributed and scalable way. Piazza assumes the participants to be interested in sharing data, and willing to define pairwise mappings between their schemas. Then, users formulate queries over their preferred schema, and a query answering system expands recursively any mappings relevant to the query, retrieving data from other peers. In this paper, we provide a brief overview of the Piazza project including our work on developing mapping languages and query reformulation algorithms, assisting the users in defining mappings, indexing, and enforcing access control over shared data.

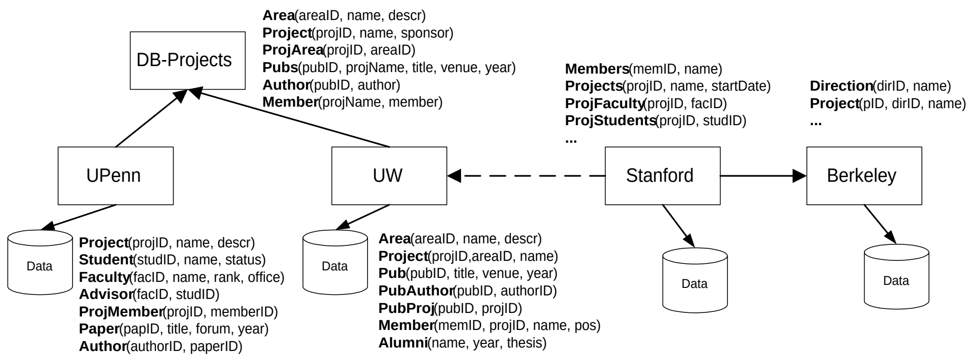
1. INTRODUCTION

A major problem in today's information-driven world is that sharing heterogeneous, semantically rich data is incredibly difficult. Authoring and publishing Web documents is very easy; peer-to-peer file sharing systems make file dissemination a simple job. No similar level of technology exists for meaningfully sharing information with different schemas and representations. This is not entirely surprising, since sharing semantically rich data is inherently a much harder problem: data in different schemas must somehow be *mapped* or related, queries are much richer, and security is an important consideration.

Yet, the benefits of semantic data sharing are quite compelling for many applications. Consider the problem of sharing scientific data. In the past, individual researchers tended to collect and analyze data in isolation, studying small-scale phenomena and keeping their data proprietary. Today, there is much interest in making data freely available for use by other researchers, generally with the goal of *integrating and aggregating* the data from multiple heterogeneous sources to get a bigger picture of the phenomena being studied. Examples include the SkyQuery project in astronomy [16] and the efforts of the Institute for Systems Biology [10]. Another emerging application that could greatly benefit from semantic data sharing is the Semantic Web [3, 8].

A number of past and current data management projects have explored limited aspects of the problem of sharing semantically rich data in a distributed, heterogeneous world. Mariposa [18] implemented distributed data sharing across the wide area, assuming heterogeneity of resources but not of schema. Research on data integration has addressed problems in mapping heterogeneous schemas with different capabilities, under the assumption that a *single* centralized mediated schema (or a hierarchy of mediated schemas) can be created. Research on the Semantic Web [3, 4] proposes using highly expressive, knowledge representation-based formalisms for describing data semantics — *ontologies* capture much more information than database schemas — but they have largely overlooked issues of translation or mediation between multiple heterogeneous ontologies.

In Piazza, we focus on the problem of sharing semantically heterogeneous data in a distributed and scalable way. The participants in Piazza are data sources interested in sharing data. We start with the observation that participants will naturally prefer independent but related schemas for their data, and that their queries will typically be posed from the context of their preferred schema. Rather than requiring global agreement on a single unified schema, we provide query answering capabilities over an arbitrary network of local schemas and pairwise mappings between them. Our query answering algorithms take a query posed over any of these schemas and use the transitive closure of mappings to return all relevant data in that preferred schema. This achieves the schema mediation capabilities of a data integration system, but in a more extensible, decentralized way. As in peer-to-peer data sharing systems, we also allow any node to join the system and contribute resources (schemas, mappings, data, or computation) that improve the overall environment. Hence, we refer to our architecture as a *peer data management system* (PDMS) and a node as a *peer*.



DB-Projects
Area(areaID, name, descr)
Project(projID, name, sponsor)
ProjArea(projID, areaID)
Pubs(pubID, projName, title, venue, year)
Author(pubID, author)
Member(projName, member)
UPenn
UW
Stanford
Berkeley
Members(memID, name)
Projects(projID, name, startDate)
ProjFaculty(projID, facID)
ProjStudents(projID, studID)
...
Direction(dirID, name)
Project(pID, dirID, name)
...
Data
Project(projID, name, descr)
Student(studID, name, status)
Faculty(facID, name, rank, office)
Advisor(facID, studID)
ProjMember(projID, memberID)
Paper(papID, title, forum, year)
Author(authorID, paperID)
Data
Area(areaID, name, descr)
Project(projID,areaID, name)
Pub(pubID, title, venue, year)
PubAuthor(pubID, authorID)
PubProj(pubID, projID)
Member(memID, projID, name, pos)
Alumni(name, year, thesis)
Data
Data

Figure 1: A PDMS for a PDMS about database research. Arrows indicate mappings between the relations of the peers. DB-Projects **is a virtual, mediating peer that has no stored data. The figure illustrates how two semantic networks can be joined by establishing a single mapping between a pair of peers (**UW **and** Stanford **in this case).**

In order to provide these query and mediation capabilities at a level appropriate for scientific data sharing, Piazza must address a number of issues, as we describe in this paper. We begin in Section 2 by describing our approach to specifying schema mappings. Section 3 outlines Piazza's query answering algorithm. Since mappings are the basis of our system, but they may be tedious to create, we investigate techniques for facilitating mapping construction in Section 4. Section 5 discusses our initial work in indexing a semantically heterogeneous set of data. Finally, we briefly discuss security and access control in Section 6, and we conclude in Section 7.

2. SCHEMA MEDIATION

In contrast to a data integration environment, which has a tree-based hierarchy with data sources schemas at the leaf nodes and one or more *mediated schemas* as intermediate nodes, a peer data management system (PDMS) can support an arbitrary *graph* of interconnected schemas. Some of these schemas are defined virtually for purposes of querying and mapping. We call these *peer schemas*, and generally their relations (*peer relations*)[1] will have an open-world assumption (i.e., the data returned by querying these relations may be incomplete). Queries in the PDMS will be posed over the relations from a specific peer schema. A peer schema represents the peer's "view of the world" that is unlikely to be the same at different peers.

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 at other peers.

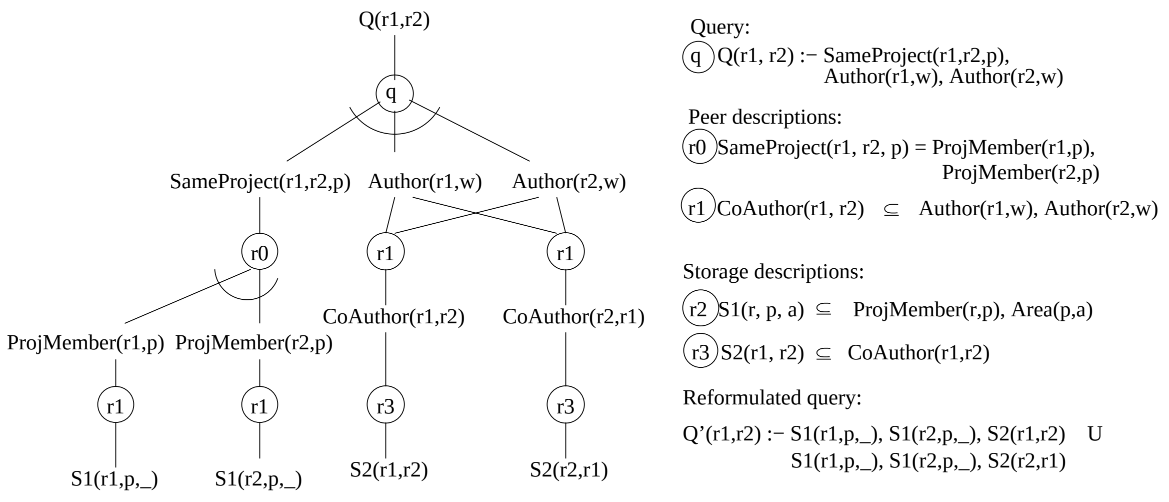
Figure 1 shows a simplified example of a PDMS for sharing database research-related data. The novelty of the PDMS lies in its ability to exploit transitive relationships among the mapping edges between peers' schemas. The figure shows that two semantic networks can be fully joined together with only a few mappings between similar members of each semantic network (in our example, we only required a single mapping). The new mapping (dashed line) from Stanford to UW enables any query at any of the five peers to access data at *all* other peers through transitive evaluation of semantic mappings. Importantly, we can add our mappings between the most similar nodes in the two semantic networks; this is typically much easier than attempting to map a large number of highly dissimilar schemas into a single mediated schema (as in conventional data integration).

There are two types of schema mappings in Piazza. A mapping that relates two or more peer schemas is called a *peer description*, whereas a mapping that relates a stored schema to a peer schema is called a *storage description*. Peer descriptions define the correspondences between the "views of the world" at different peers. Storage descriptions, on the other hand, map the data stored at a peer into the peer's view of the world. Thus, storage descriptions are similar to data source descriptions in a data integration system.

Two main formalisms have been proposed for schema mediation in data integration systems. In the first, called *global-as-view* (GAV) [6], the relations in the mediated schema are defined as views over the relations in the sources. In the second, called *local-as-view* (LAV) [13], the relations in the sources are specified as views over the mediated schema.

Piazza combines and generalizes the two data integration formalisms, and it extends them to the XML world in a way that keeps evaluation tractable. Two kinds of peer descriptions are supported: *equality* and *inclusion* descriptions. Peer descriptions have the following form: $Q_1(\mathcal{P}_1) = Q_2(\mathcal{P}_2)$, (or $Q_1(\mathcal{P}_1) \subseteq Q_2(\mathcal{P}_2)$ for inclusions) where Q_1 and Q_2 are conjunctive queries with the same arity and $\mathcal{P}_1$ and $\mathcal{P}_2$ are *sets* of peers. Intuitively, our mapping statement specifies

[1]We use the relational model to simplify the discussion. The implemented Piazza system is based on XML: peer schemas are defi ned in XML Schema, queries and mappings are specifi ed in an XQuery-based language. We refer the interested reader to a more detailed description of the system in [8].



Query:
q Q(r1, r2) :− SameProject(r1,r2,p), Author(r1,w), Author(r2,w)

Peer descriptions:
r0 SameProject(r1, r2, p) = ProjMember(r1,p), ProjMember(r2,p)
r1 CoAuthor(r1, r2) ⊆ Author(r1,w), Author(r2,w)

Storage descriptions:
r2 S1(r, p, a) ⊆ ProjMember(r,p), Area(p,a)
r3 S2(r1, r2) ⊆ CoAuthor(r1,r2)

Reformulated query:
Q'(r1,r2) :− S1(r1,p,_), S1(r2,p,_), S2(r1,r2) U
S1(r1,p,_), S1(r2,p,_), S2(r2,r1)

Figure 2: An example reformulation DAG fo the database research domain.

a semantic mapping by stating that evaluating Q_1 over the peers $\mathcal{P}_1$ will always produce the same answer (or a subset in the case of inclusions) as evaluating Q_2 over $\mathcal{P}_2$.

To avoid ambiguity, we prefix relation names at each peer with the peer name. The following statement is an example of an equality mapping between two peers: UW (University of Washington) and DBProjects (a Database-Projects peer):

```
DBProjects:Member(pName, member) =
   UW:Member(mid, pid, member), UW:Project(pid, pName)
```

A storage description can also be either an equality ($P : R = Q$) or an inclusion ($P : R \subseteq Q$); here Q is a query over the schema of peer P and R is a stored relation at the peer. As in the context of data integration, an inclusion description implies that that the data at the peer may be *incomplete*, which corresponds to the *open world assumption* [1]. The following storage description defines the stored relation students at peer UPenn in terms of UPenn's peer relations:

```
UPenn:student(sid, name, advisor) ⊆ UPenn:Student(sid, name),
   UPenn:Advisor(sid, fid), UPenn:Faculty(fid, advisor)
```

The set of mappings of a PDMS defines its *semantic network* (or *topology*). Optimizing the topology of a PDMS is an interesting research problem. Some of the possible optimization criteria include: eliminating redundant mappings, reducing the diameter of a PDMS (to reduce information loss in query reformulation), and identifying semantically unreachable peers. Analyzing the semantic network of a PDMS requires the ability to compose mappings which is a challenging problem on its own (see [15] for initial results).

3. QUERYING

Query reformulation is perhaps the single most important aspect of query processing in a PDMS, since it is crucial for PDMS's ability to answer user queries. In this section, we outline the query reformulation algorithm implemented in Piazza. (We note again that we limit our discussion to the relational case; the implemented system uses an XML query reformulation algorithm that we describe in [8].) 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 refers to stored relations *only*. 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 [11] are well suited for our context.

Before we describe the algorithm, two points need to be mentioned. First, by introducing an auxiliary relation (view) a description of the form $Q_1(\mathcal{P}_1) = Q_2(\mathcal{P}_2)$ can be rewritten as two simpler descriptions: $Q_1(\mathcal{P}_1) = V$ and $V = Q_2(\mathcal{P}_2)$. Second, an equality description can be rewritten a pair of inclusion descriptions. Hence, we can assume that all descriptions are of the form $V \subseteq Q(\mathcal{P})$ or $V \supseteq Q(\mathcal{P})$.

To provide some intuition for the algorithm, consider a PDMS in which all peer mappings are of the form $V \supseteq Q(\mathcal{P})$. This case is similar to unfolding GAV mappings in data integration. The algorithm proceeds by constructing a simple rule-goal tree [19]: 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 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 of the form $V \subseteq Q(\mathcal{P})$. In this case (that is similar to LAV mappings in data integration), we begin with the query subgoals and apply an algorithm for answering queries using views [7]. 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.

A major challenge of the reformulation algorithm is to combine and interleave the two types of reformulation techniques. One type of reformulation (unfolding) replaces a subgoal with a set of subgoals, while the other (rewriting) replaces a set of subgoals with a single subgoal. As a result, the output of the algorithm can be a DAG rather than a tree as illustrated by the following example.

Figure 2 shows the reformulation DAG for a simple query, Q, which asks for researchers who have worked on the same project and also co-authored a paper. We begin by expanding Q into its three subgoals, each of which appears as a goal node. The SameProject peer relation (indicating which researchers work on the same project) is involved in peer description r_0, hence we expand the SameProject goal node with the rule r_0, and its children are two goal nodes of the ProjMember peer relation (each specifying the projects an individual researcher is involved in).

The Author relation is involved in an inclusion peer description (r_1). We expand Author(r1,w) with the rule node r_1, and its child becomes a goal node of the relation CoAuthor. This "expansion" is of different nature because of the LAV-style reformulation. Intuitively, we are reformulating the Author(r1,w) subgoal to use the left-hand side of r_1. Note that we must reformulate both Author subgoals at once because r_1 projects out the join variable w. This step is based on Minicon, an efficient algorithm for query rewriting [7]. Now we must apply description r_1 a second time with the head variables reversed, since CoAuthor may not be symmetric (because r_1 is an inclusion rather than equality).

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. Other challenges that we address when constructing a reformulation DAG in Piazza are avoiding redundant work through memoization and pruning [9] and choosing an optimal reformulation order.

4. CONSTRUCTING MAPPINGS

In the previous two sections, we have described Piazza mappings and how they are used to evaluate queries. Clearly, another important question is where mappings come from and how they are created. In Piazza, our goal is also to develop tools and techniques that vastly simplify and assist in mapping creation.

Mappings between schemas can be constructed in a two-step process. The first phase is *Schema Matching*: discovering a match, or a set of correspondences, which identify similar elements in schemas that are to be mapped. For example, a match between the Berkeley and UW projects (see Figure 1) will include the correspondence: Berkeley.Direction $\sim$ UW.Area. Such correspondences are statements of similarity and have little or no semantics. The second phase of mapping construction takes these correspondences as inputs, and it uses a combination of automatic techniques and human intervention (as in the Clio system [20]) to provide a precise mapping.

Our current focus is on automated techniques for schema matching, *i.e.* the first phase of mapping construction. Our approach is characterized by two key properties: using an ensemble of individual heuristics and algorithms, and exploiting past experience.

Combining multiple types of evidence: There is a variety of evidence in schemas that can be exploited by different heuristics or algorithms. For example, the names of the schema elements, their data instances, their data types, any accompanying text descriptions, or similarity is schema structure. However each of these types of evidence is also typically noisy, and hence hard to exploit, *e.g.* names have abbreviations, synonyms, etc., and text descriptions are uncommon and often inconsistent. We propose a multi-strategy approach that will combine different approaches to exploit each of these evidences. Our approach is based on the machine learning technique called *stacking*. In [5], we demonstrated that such an approach can yield robust matching performance.

Exploiting past experience: Schema matching tasks are often repetitive. For example, all the schemas that are being matched for a particular application are typically in a single domain (*database projects* in Figure 1), and hence will have similar elements. Hence it should be possible to glean knowledge from known validated mappings among these schemas and reuse this information for matching new schema pairs. With this intuition, we are building a *corpus*-based schema matcher. A corpus has a collection of schemas, validated mappings, data instances, and other forms of meta-data. Statistics can be computed over schema elements in the corpus and be employed for schema matching. This corpus will evolve over time through a feedback process: the learned knowledge is applied to match new schemas, and the schemas and the eventual validated mapping are then assimilated back into the corpus. In [14] we present preliminary results of exploiting past experience to schema matching. In this work, classifiers are trained for each unique element belonging to some schema in the corpus. Future schema matching tasks are based on the following premise: two elements can be deemed to be similar if they cannot be distinguished from each other using the learned classifiers in the corpus. Our initial results are promising: they demonstrate good accuracy and strong evidence of evolution, and this opens an exciting avenue for further research.

Although schema mapping is important in the context of data sharing, it is equally important to have facilities for instance-level mapping, i.e., object mapping. Object correspondences are especially important in fields such as biology and medicine, where the same entity may have many different identifiers or names, or over time a given record may need to be merged with another (or split into two records).

Recent work by Miller et al. [12] exploits a set of related mappings to infer further object correspondences; we hope to go even further in inferring correspondences, using containment and structural relationship information between entities.

5. SEARCHING

A query can be evaluated in a PDMS by sending it (reformulated appropriately) to all the peers that might have answers. In such a scheme, it is absolutely vital that every query not flood the entire network. Our query reformulation algorithm devotes considerable effort towards pruning rewritings that are guaranteed to return no results (or redundant results). However, reformulation can only exploit information contained in the mapping definitions, whereas it would be desirable to exploit information about the actual *data* stored at the peers in order to identify the peers relevant to the user query. One way to address this problem is to build an index over the peers that is somehow aware of both schema and value mappings.

As a first step towards addressing this problem, we have begun development of an index structure that allows simple value lookup with partial match over structured attributes, across simple attribute-equivalence mappings. Selection predicates can often be mapped to a similar model, so we feel this is a good starting point. The key challenge is how to create a scalable index that returns, for any query, the set of relevant peers with as few false positives[2] as possible.

Index architecture: Our current index implementation is centralized: this is similar to a search engine on the Web, and unlike a distributed hash table (DHT) in P2P systems for file sharing. We plan to extend to a distributed context in the future, but there are two reasons for this initial design choice. First, the searches supported by our index are semantically rich, and cannot be supported by hashing. Second, in most applications of peer data management systems it is realistic to assume that some peer(s) are willing to offer the modest resources needed to service a Piazza index. Keeping the resource requirements at a minimum is critical in order to encourage volunteers.

The novelty of our index system is that participating peers will upload *summaries* of their data at different granularities. Unlike many sparse indexing techniques, in which the sparseness level is controlled by physical parameters such as page size, we allow each peer to specify data summaries at an appropriate granularity level. The peer also makes available to the index all its peer mappings, allowing the index engine to correlate attributes from different peers — thus supporting the simplest type of schema mappings. Peers periodically refresh their data summaries at the index.

Finally, users perform searches by submitting queries to the index engine and retrieving answers.

The Logical Model: The index uses a very simple data model consisting of objects that contain sets of attribute-value pairs. Thus, a data object is:

$$d \ ::= \ [A_1 = v_1, A_2 = v_2, \ldots, A_n = v_n] \qquad (1)$$

where $A_1, \ldots, A_n$ are *attributes* and $v_1, \ldots, v_n$ are *atomic values*. The set of attributes is dynamic. Each peer may define its own attributes, and even two objects from the same peer may have different sets of attributes. If a peer mapping relates two attributes from different peers, this information is made available to the index in the form of an *attribute mapping*, which can have the form $A \subseteq B$ or $A = B$.

The index supports *partial-match* queries of the form:

$$q \ = \ [B_1 = w_1, B_2 = w_2, \ldots, B_p = w_p] \qquad (2)$$

Here $B_1, \ldots, B_p$ are attributes and $w_1, \ldots, w_p$ are constants. Each condition $B_i = w_i$ is called a *predicate*, hence the query consists of a conjunction of p predicates. The query q *matches* a data object d if d satisfies all predicates in the query. To satisfy some predicate $B_i = w_i$ the data object must contain an attribute value pair $A_j = v_j$ s.t. $A_j \subseteq B_i$ and $v_j = w_i$.

A peer may chose to export to the index each data item or summaries of collections of data items. The latter are like (1) only now the values $v_1, \ldots, v_n$ are replaced with patterns, describing constraints on the data. For example, a peer may export the following two summaries:

s_1 = `[patientName = "Ge%", stimulus = "%",`
`age = 55, image="%"]`
s_2 = `[name = "Por%", age IN [50, 70],`
`disease ="tuberculosis", type = "%"]`

The first data summary s_1 specifies a set of objects that have the attributes `patientName`, `stimulus`, and `image`, where the `patientName` begins with the letters `Ge` and the age is 55, but the `stimulus` is unspecified. This could mean either that there are several objects at that peer matching this pattern, or that there is a single such object but for which the peer does not want to provide the precise value for `patientName` or `stimulus`: however, it is willing to specify an approximation for them. Similarly, s_2 specifies four attributes, one of which has a completely specified value, the others only partially specified. The pattern for `age` is a range.

Note that while individual peers may have very few attributes, the total number of attributes can be large and it can grow dynamically. Current techniques for answering such partial-match queries (e.g. [2]) need to know the total number of dimensions a priori and even then cannot efficiently handle data with even hundreds of dimensions. We present a new indexing scheme that scales with the number of attributes of individual peers and searches in time proportional to the size of the query result.

Implementation: The index maintains a main table `RD(Oid, Attr, Val)` containing all attribute value pairs together with the object id where they occur together. This table

[2]Peers that do not have any answers for the query.

could be sufficient to answer all partial match queries, for example, query (2) translates into a p-way self-join on RD. However, this is quite inefficient when none of the base predicates is very selective[3]. Indeed, the best query execution plan that a relational engine has in such a case amounts essentially to a linear scan. To avoid that, the index uses auxiliary tables at various *levels*. The level k index has the schema

$$\mathtt{L_k}(\mathtt{Oid}, \mathtt{Attr_1}, \mathtt{Val_1}, \ldots, \mathtt{Attr_k}, \mathtt{Val_k})$$

and contains k-tuples of attribute values pairs that occur in the same data item. Since there are $\binom{n}{k}$ possible tuples for each object (like (1)), the level k index stores only a subset of such tuples: namely only those for which the number of objects that contain this tuple is significantly smaller (by a fixed factor) than the number of objects containing $(\mathtt{Attr_1}, \mathtt{Val_1}, \ldots, \mathtt{Attr_{k-1}}, \mathtt{Val_{k-1}})$ (the attributes are ordered lexicographically). This keeps the size of the level k indexes small, and L_k usually becomes empty after few levels. A query (2) starts with a lookup in L_p (which is executed as a clustered index lookup); if the answer is empty, then it proceeds with a lookup in L_{p-1}, etc. Each query is thus answered after at most p lookups. The answer may contain false positives, but their fraction is guaranteed to be below a fixed factor chosen by the administrator.

6. SECURITY AND ACCESS CONTROL

Although the goal and emphasis of Piazza is data sharing, in practice, peers are almost never willing — or even legally able — to share their data in an uncontrolled way. Usually a data owner will wish to grant different access rights to different peers in the system. Standard access control methods are poorly suited for the PDMS setting because they generally protect data by keeping it behind a secure server and processing queries on behalf of clients. In forthcoming work [17] we have developed techniques for publishing a single data instance in a protected form. The published data is encrypted such that multiple peers can use a single replica, but are restricted in their access to parts of the data in accordance with the owners' preferences. Data owners in Piazza can specify access control policies declaratively and generate data instances that enforce them. Access control rights can be dynamically modified simply by exchanging cryptographic keys.

7. CONCLUSIONS

In this paper we have presented a brief overview of the Piazza project and some of its focal points. To this point, our efforts have generally revolved around defining the means of answering queries in this system: mapping definition, query reformulation, assistance in mapping creation, indexing data, and restricting the use of data to those with permission. Our current implementation already provides many useful capabilities, and we have a running prototype of a PDMS for academic research.

However, a great deal of exciting future work remains. We would like to explore richer and more expressive means of defining mappings and defining semantic information: two interesting directions are to bring Piazza to a more knowledge representation-based, ontology-driven world, as in most Semantic Web efforts (we discuss our initial efforts in [8]); and to add a probabilistic, approximation-based semantics to our mapping language. The challenge in both of these areas is maintaining tractability of query reformulation. We are also interested in the problem of analyzing the semantic network of a PDMS and developing efficient caching strategies to speed up query execution.

[3]If users specify multiple predicates, then it is quite likely that none of them alone is very selective.

REFERENCES

[1] S. Abiteboul and O. Duschka. Complexity of answering queries using materialized views. In *PODS*, 1998.

[2] J. L. Bentley. Multidimensional binary search trees used for associative searching. *CACM*, 18(9):509–517, 1975.

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

[4] M. Dean, D. Connolly, F. van Harmelen, J. Hendler, I. Horrocks, D. McGuinness, P. Patel-Schneider, and L. Stein. OWL web ontology language 1.0 reference, 2002. Manuscript available from http://www.w3.org/2001/sw/WebOnt/.

[5] A. Doan, P. Domingos, and A. Halevy. Reconciling Schemas of Disparate Data Sources: A Machine Learning Approach. In *SIGMOD*, 2001.

[6] 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), March 1997.

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

[8] A. Halevy, Z. Ives, P. Mork, and I. Tatarinov. Piazza: Data Management Infrastructure for Semantic Web Applications. In *WWW*, 2003.

[9] A. Halevy, Z. Ives, D. Suciu, and I. Tatarinov. Schema Mediation in Peer Data Management System. In *ICDE*, 2003.

[10] Institute for Systems Biology. http://www.systemsbiology.org.

[11] Z. Ives, A. Halevy, and D. Weld. Integrating Network-Bound XML Data. *IEEE Data Engineering Bulletin*, 24(2), 2001.

[12] A. Kementsietsidis, M. Arenas, and R. Miller. Mapping Data in Peer-toPeer Systems: Semantics and Algorithmic Issues. In *VLDB*, 2003.

[13] A. Levy, A. Rajaraman, and J. Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. In *VLDB*, 1996.

[14] J. Madhavan, P. Bernstein, K. Chen, A. Halevy, and P. Shenoy. Corpus-based Schema Matching. In *Workshop on Information Integration on the Web at IJCAI*, 2003.

[15] J. Madhavan and A. Halevy. Composing Mappings among Data Sources. In *VLDB*, 2003.

[16] T. Malik and A. Szalay. Skyquery: A web service approach to federate databases. In *Proceedings of CIDR*, 2003.

[17] G. Miklau and D. Suciu. Controlling Access to Published Data Using Cryptography. In *VLDB*, 2003.

[18] M. Stonebraker, P. M. Aoki, W. Litwin, A. Pfeffer, A. Sah, J. Sidell, C. Staelin, and A. Yu. Mariposa: A Wide-Area Distributed Database System. *VLDB Journal*, 5(1):48–63, 1996.

[19] J. Ullman. *Database and Knowledge-Base Systems*, volume 2. Addison-Wesley, 1989.

[20] L.-L. Yan, R. Miller, L. Haas, and R. Fagin. Data Driven Understanding and Refi nement of Schema Mappings. In *SIGMOD*, 2001.