Part C: Project Description

1 Background and Motivation

One clear success of the internet has been the creation of new communities of people who share information, services, and resources in domains of mutual interest. These communities arise autonomously and undergo continuous change. Obvious examples are Internet groups focused on political causes, sports and recreational activities, local schools, music and books, investment activities, and so on. These communities exist without any centralized controlling authority; some disappear as quickly as they arrive, while others prosper. To the extent that they exist "physically" at all, they consist of a simple collection of cross-linked web pages, although some have more formal structures, such as peer-to-peer overlay networks based on communications protocols that clients use to join or interact.

Internet communities are enabled by the WWW, which uses a simple markup language (HTML) and access protocol (HTTP) to knit together a world-wide document retrieval and service system from a heterogeneous collection of autonomous computers. In some ways, however, the existing Web is extremely limited: specifically, it is focused on creating (mostly ASCII) documents and images for display. This presentation-oriented model provides no facility for *understanding* or *processing* the information transmitted. In particular, the Web presents no framework for sharing or interpreting structured or semi-structured *data*, and this makes many applications difficult or impossible. To attack this problem, we are proposing research that will substantially increase the *type* of information sharing, information processing, and cooperation that can occur on the Internet of the future.

At a high level, the Piazza project seeks to facilitate wide-scale data sharing among cooperating communities. By *data sharing*, we mean that participants can export and share schemas, base data, and data views constructed through integration over multiple distributed data sources. Topologically, we imagine the cooperating communities to be peer-oriented, self-organizing structures. That is, we assume that (1) each participant is both a client and a server, consuming and producing components of a distributed information database, and that (2) participants join and leave the communities at will. The goal of our research is to design, implement, and deploy the infrastructure on which such data-sharing communities can form and grow.

As an example of this approach, suppose that we are avid scuba divers and wish to create a web page with information on possible dive vacations. Our Web page will describe various diving sites as well as adjacent lodging, dive schools, restaurants, gear stores, low-cost airfares, etc. How could we construct such a page? The typical approach is to include a list of links to other relevant pages – some on dive schools, some on restaurants, some on finding airfares, and so on – a useful but unsatisfying solution. As an alternative, though, imagine instead the Piazza vision, in which the Internet is *data oriented*: i.e., airlines export data describing flights and prices; food snobs export data on great restaurants; dive schools export data describing locations, classes, availability, and fees. Imagine furthermore that this data could be indexed, located, and queried in a database-oriented fashion, producing new schemas and views. Other people in the data-sharing community could then grow this "database" by defining other new views, or by specifying semantic relationships between the schemas of existing views, permitting new data to be published and used in rich and unanticipated ways.

For example, on our scuba Web page, we could easily construct a list of dive classes available next weekend in places reachable from New York for under $200 that are close to 3-star restaurants. That list, a view constructed simply by a query over the various data sources, could even change dynamically with changes to its underlying base data. Once we export our new data view, others can apply it in *different* contexts – ones that we would never imagine – just as our use of the restaurant information for dive sites was not anticipated by the food snobs or airlines. In fact, the airlines might now import *our* new view, appropriately processed, to advertise their flights to the dive community.

The Piazza vision relates to and extends upon previous work in several closely related areas, and it is therefore important to clarify the crucial differences:

- *Data Integration Systems* [43, 100, 62, 33] permit queries over distributed heterogeneous data sources. However, data integration systems: (1) assume that a set of data sources exist and are defined *a priori*, (2) present a uniform interface to the data via a *mediator*, and (3) employ a *single* logical mediated schema at the mediator to specify the semantic mappings between the mediated schema and the schema of the data stored in the sources. In contrast, in Piazza all peers are equal, so a single logical mediated schema is undesirable, impractical, and perhaps even impossible to build. Within Piazza, queries at peers are always posed using the schema of the peer, and mediation and execution originate from the peer. Given the peer structure, a source may use a view created by another peer, which was derived in part from one of its own views, as we saw with the dive group and the airline data in the example above; this would be impossible in a conventional data integration system, because the base data sources and the mediator are *not* peers.

- *Distributed Databases* [86] similarly enable queries over distributed data. Distributed databases, however, assume that the data organization on different machines is *designed in advance*; i.e., the data and machines (and their characteristics) are known *a priori*, and the properties of the local workloads may also be known. Given that knowledge, the physical design of a distributed database application includes the explicit placement of data on the different machines: first the data is *fragmented* and then *allocated* optimally across machines [86]. Furthermore, it is assumed that the machines are similar and are running under central administration. While many of the distributed query processing and data distribution techniques developed for distributed databases will be relevant to Piazza, none of these assumptions apply in our environment, in which there is no central management, schemas are developed continuously and independently, and complete distributed knowledge is impossible to achieve.

- *The Semantic Web*, as described by Berners-Lee et al. [13], also relates to Piazza's data-oriented vision. However, their focus is, among other things, on agent technology and languages for expressing ontologies of data on the web. Our aim is combining our database and computer systems expertise to solve the underlying technical challenges in constructing a Piazza prototype on which applications can be built. Our challenges are different and significant; they lie in the areas of distributed query processing, query optimization, data indexing and search, network protocols, operating system resource management, data caching and placement, schema management, and data consistency, as several examples.

To enable our vision, Piazza will support a set of services to enhance the robustness and scalability of a wide-area instance of a data-sharing community. More specifically, Piazza will support efficient query processing, intelligent placement and caching of data to enhance performance and availability, incremental propagation of updates, and methods for specifying semantic relationships between views.

Given the ambition of the project, we intend to start simply, first building support for small-scale co-operating communities. Once our small-scale prototype is implemented, we will construct, instrument, and measure several applications executing on that platform. From these studies, we will design and build larger versions of Piazza intended to scale to larger, widely-distributed communities. Even at a small scale, Piazza presents a complex set of technical hurdles; however, we believe that the PIs bring together the right set of expertise in databases, query processing, semi-structured data (XML), operating systems, distributed systems, and networks to surmount these hurdles. Our intention is to unite technologies from the (often antagonistic) worlds of databases and computer systems in order to facilitate a new type of data-processing facility for cooperating communities on the Web.

Piazza is the first implementation of what we term a *Peer Data Management System*, or *PDMS*. In the following sections, we describe some of the technical issues in any PDMS, our proposed approach for Piazza, and the tasks we will carry out along the way.

2 Relationship to Previous Grants

Alon Halevy: Alon Halevy (until recently, Alon Levy) currently holds two NSF grants – A CAREER/PECASE grant (#9985114): *Flexible construction and management of web sites* ($500,000 for 5 years beginning in summer of 2000), and a single-PI grant (#9978567): *Efficient Query Processing for Data Integration* ($225,000, starting summer 1999 and ending in summer 2002). The CAREER grant is only tangentially related to the work described here.

The data integration grant studied several topics. First, we developed techniques for *adaptive query processing:* while traditional database optimizers make heavy use of statistics on their base data (e.g., cardinalities, selectivities), a data integration system may not have that meta-data about its data sources. We developed a query processor that starts with an initial query execution plan and refines it as more data streams in and we accumulate more meta-data [66, 63, 65]. Our results showed that despite the overhead associated with switching plans, we obtain better overall performance.

Second, we studied query processing for network-bound XML data [67, 64]. We developed the Tukwila System that employs the novel Xscan operator, which applies complex constraints to data as it streams in from the network. The system extended techniques from pipelined relational query processing to the context of XML to obtain a scalable query processor [67]. We developed and implemented an update language for XML [96] extending our previously developed query language for XML [28]. Finally, we developed scalable methods for query reformulation for data integration systems [90, 88] and studied some theoretical aspects of reformulation [84].

Earlier work of Halevy focused on the semantic issues of data integration [80, 79, 77, 37, 81]. While at AT&T Laboratories, Halevy developed the Information Manifold System that introduced the method to specifying semantic relationships between data sources known as *local-as-view* . A major thrust of Halevy's work combines techniques from the fields of Artificial Intelligence and Database Systems [42, 32, 76, 75, 10].

Steven Gribble: Steven Gribble is a Co-PI (along with Henry Levy and Anna Karlin) on NSF Grant 0121341 (ITR/SI: Adaptive Wide-Area Information Delivery, $870,000), a four-year grant that started in October 2001. Steven's past research includes robust Internet infrastructure, scalable Internet services, security, operating systems, mobile and wireless computing, networking, and peer-to-peer systems.

In [45], he developed cluster-of-workstation architectures to simplify the construction of highly-available Internet services, and in [52], he designed and implemented a new data storage platform geared towards high-concurrency workloads suitable for Internet servers. He has also performed many measurement studies of Internet workloads: in [51], he gathered and analyzed a 45-day long trace of web activity from UC Berkeley. In [94], he characterized the behavior of clients that participate in the Gnutella and Napster peer-to-peer systems, including their bottleneck bandwidths, availabilities, proclivity to share data, and tendency to misrepresent their capabilities.

Henry Levy: Henry Levy was Co-PI (along with Anna Karlin) on NSF Grant CCR-9632769, "Distributed Resource Management in Workstation and PC Clusters." The grant of $202,408 began in July 1996 and terminated in June 1998. He was co-PI (along with Anna Karlin, Tom Anderson, and Richard Ladner) on a follow-on NSF grant of $870,336, number EIA-9870740, "Resource Management in Local-Area and Wide-Area Networks," which ran from July 1998 through June 2001.

Our results were in two major environments – local-area networks and the World-Wide Web – and have implications for both.

Our local-area network research explored how changing network technologies could be exploited in operating system design. In particular, we designed and implemented a global memory management system (GMS) for workstation clusters [39], demonstrating that we could reduce access to VM pages or files by transferring them from memory of network nodes rather than local disk (our GMS implementation can read a remote memory page 50 times faster than a local disk page, given a gigabit network). Using GMS as a basis, we studied the impact of page size on a global memory system. Our results [68] showed that current page sizes are suboptimal for high-speed network traffic. Using analytical models, we studied the advantages of using server load as well as page-age information for page replacements in clusters with globally managed memory; in particular, we showed the extent to which server load can degrade remote-memory latency, and that deviating from LRU does not effect application performance significantly [102]. We then showed how prefetching over the network could achieve significant speedups in a global memory environment [103].

Our wide-area work focused on characterizing document access patterns in the World-Wide Web. For example, looking at the University of Washington as a collection of 200 diverse groups (departments, museums, colleges, etc.), we studied the extent to which document access patterns were determined by group membership [106]. We then used our UW traces, along with simultaneous traces we collected at Microsoft, to examine the potential for cooperation among Web proxies serving both large and small organizations. Our results showed that inter-proxy cooperation has benefits only in the small scale, and that large-scale cooperation has quickly diminishing returns [105]. We also studied the temporal properties of streaming-media accesses over the Web and the extent to which temporal overlap of streams permits efficient caching or real-time sharing techniques [20]. Finally, we showed how to architect Web services for scalability, by designing and implementing Porcupine, a highly scalable and available Internet mail service [92, 93].

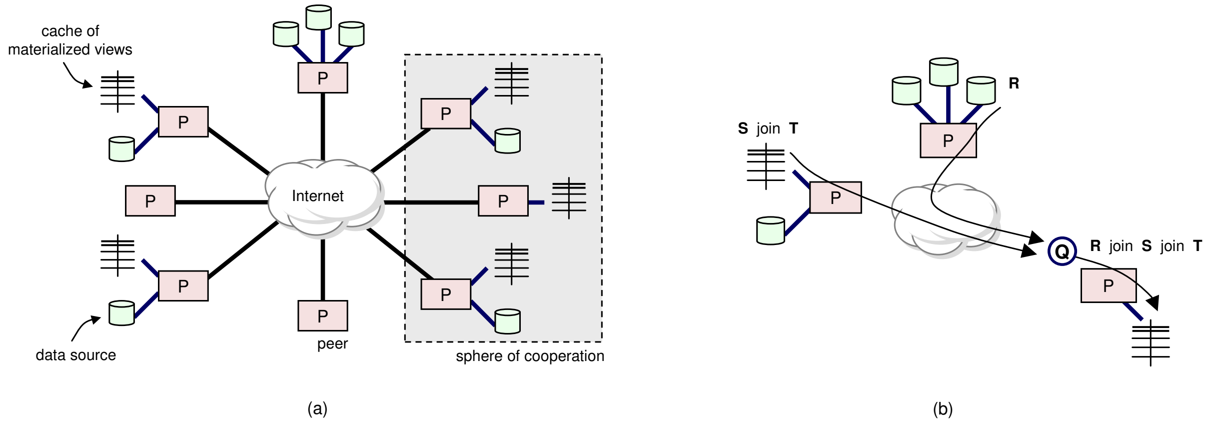


Figure 1: Piazza system architecture. (a) Piazza peers play any combination of three roles. They can act as data sources, providing base data to other peers. They can host caches of materialized views, optionally making that cache available to other peers. Finally, they can pose queries on the system, obtaining data locally, from a remote data origin, or from a remote materialized view cache. (b) A peer issues the query `R join S join T`; the base relation `R` is selected from a remote data origin, and the materialized view `S join T` is selected from a remote cache. The resulting view is cached locally.

Dan Suciu: Dan Suciu has one NSF grant: a CAREER grant *IIS-0092955*, 2001-2005, entitled *Efficient Management of Web Data*. Work on that grant is relevant, but complementary to the current proposal: it addresses four XML management tasks (publishing, transport, storage, and processing), each being relevant to the current proposal, but not addressing directly any of the current goals. As part of the CAREER project, Suciu is currently developing an XML toolkit for application-level processing of stream XML data [49], which could be used as part of the Piazza infrastructure. In the toolkit, Suciu is deploying new techniques based on lazy deterministic finite automata and *stream indexes* that can process XML streams at guaranteed throughput.

In prior work Suciu focused on XML publishing. He studied the problem of virtual XML publishing from relational databases in [41] and that of materialized XML publishing in [40]. XML publishing is an integral part of the Piazza system, since we use XML as the common data model, while most of the peer's data will be stored in different formats. In other work, Suciu made contributions to XML query language design [29], XML compression [83], indexes for semistructured data [83], and storage of XML and semistructured data [30]. Most of this work has relevance to Piazza. He is a coauthor of the book *Data on the Web: from Relations to Semistructured Data and XML* [1].

3 Proposed Research

In this section we describe the specific research issues that we plan to address in this project. We begin with a high-level overview of the system and the research issues.

3.1 Overview of Piazza

The goal of Piazza, or any PDMS in general, is to enable users to share data across local or wide-area networks in an ad-hoc, highly dynamic distributed architecture. In Piazza, every machine can serve both as a client (by posing queries) and as a server (by providing shared data to other peers that request it). To answer a query, a peer will use a combination of its own data and data obtained from other peers. In addition to data, peers may also share computational resources for more efficient query evaluation.

Figure 1 illustrates the core elements of the Piazza architecture. In an application of Piazza there are likely to be several kinds of peers, which will be distinguished along several dimensions. Some peers will own much of the base data that is used by other peers; some peers will have little data or data of interest to only very few others. Some peers will be more stable, trusted, and available. Other peers will be more volatile and transient. In some applications we may identify additional structure to the collection of peers;

as a simple example, the physical properties of the network or administrative boundaries may partition the peers into *spheres* of cooperating peers that are physically closer to each other. Spheres may also be created based on common interests in data. We use the term *sphere of cooperation* to denote a set of peers that have a higher degree of ability or commitment to sharing of resources.

Piazza's dynamic, ad-hoc, peer-based architecture for data management presents numerous research challenges that span the disciplines of databases, algorithms, systems, and theory. Here, we give a high-level description of some of the issues that are addressed in more detail in the following sections of this proposal.

To share data, peers need some mechanism for *describing* what data they store. In Piazza, peers advertise their contents with a set of view definitions. A view definition is a description of a query, the results of which are stored at the peer. The view definitions describing the data at each peer are stored in an index structure. When a query is asked by a peer, this index is consulted to check where relevant data is stored. The location of all relevant data is fed into a query processing engine that determines how to answer the query the most efficiently. We discuss efficient query processing in Section 3.2 and issues concerning indexing large number of views in a distributed environment in Section 3.4.

For performance and availability purposes, peers also replicate data locally. However, in our context, it does not make sense to cache data at a granularity of pages. Instead, we need a *semantic cache*, where the elements being cached are views over the data. Intuitively, we would like to cache queries that are asked often or whose results are useful in order to more efficiently evaluate other commonly asked queries. For example, in Figure 1(b), a peer executes a query by selecting base data from one remote peer, and joining it with a cached materialized view on another peer. The query results form a new materialized view, which are cached locally. We use the term *data placement problem* to refer to the questions of which views to replicate and where to replicate them. The data placement problem is discussed in Section 3.3.

In many cases, peers will have similar or identical schemas, but in general, they will have different schemas. There will be a set of mechanisms for describing the semantic relationships between the schemas of the different peers. A peer poses a query using its own schema, and if answering the query requires fetching data from other peers, then the appropriate peer translations are invoked. Mediating between schemas and the problems arising in answering queries with disparate schemas are discussed in Section 3.7.

Clearly, data will change frequently. When it does, cached data becomes invalidated. One option is to recompute the views that are cached. But a more sophisticated approach is to distribute *updategrams*, which contain only the changes in the data. In Section 3.5 we describe the issues involved.

As we previously mentioned, we expect that some peers will host data that is more popular than other peers. As the popularity of a peer's data grows, that peer must be able to handle an increasing query workload; this introduces resource management, scalability, and availability issues. In Section 3.8, we describe the research agenda we will pursue related to the design, architecture, and implementation of Piazza peers.

3.2 Answering Queries in Piazza

The first step in developing a system like Piazza is to build the infrastructure for answering queries posed at the peers. There are several issues that make this a challenging task:

- **The choice of data model:** peers are going to be heterogeneous, and therefore it is impractical to assume a uniform data model (e.g., relational) across all data and peers.
- **Answering queries over network-bound data:** like data integration systems, we need to answer queries where the data is streaming over the network, rather than being stored on local disk. As a consequence, we need to answer queries in a setting where we have little knowledge about the underlying data sources (cardinalities, selectivities) and unpredictable data transfer rates over the network.
- **Answering queries using views:** Recall that the data stored at the sources is described using views. Hence, to answer a query, Piazza must *reformulate* a peer query onto the existing views.

As in several data integration systems [47, 9, 34], our solution to the data model problem is to use XML (eXtensible Markup Language) [107] as an underlying common data format for sharing data among peers. Note that this does *not* mean all peers must store their data in XML, but rather that they send and receive data in XML. For example XML may originate from relational databases; mapping relational data to XML is called *XML publishing* and has been investigated before, including by one of the PIs [41, 40]. In a sense, XML is a least common denominator data model – its appeal is in its simplicity and the fact that it can be used to encode virtually any kind of data (relational, object-oriented, structured documents, web pages). In the last two years, a committee of the World-wide Web Consortium has been developing XQuery [18], a proposal for a standard XML query language that has been greatly influenced by several earlier proposals, including XML-QL [28], which was developed by two of the PIs.

To answer queries over networked-bound data sources our query processor will be based on the paradigm of *adaptive query processing* [5, 85, 65, 101, 24]. In adaptive query processing, the query processor starts with an initial plan for executing the query that may be based on very few statistics on the underlying sources. As the data streams in and we learn more about the sources, the processor adapts its plan accordingly. As examples, the processor may change the order or the physical implementation of the joins (or other operations) in the plan.

The Tukwila System [66, 67], which we are developing at the University of Washington, will form the basis for Piazza. Tukwila is designed to answer queries posed in XQuery over network-bound sources and its features illustrate the benefits of adaptive query processing. In Figure 2(a) we show the performance of Tukwila using adaptive query processing. The graph shows several queries from the TPC-H benchmark, run when the query optimizer either has correct statistics on the data or has no statistics at all. In each case we compare Tukwila with a standard optimizer that uses a static query execution plan (we show the case in which the optimizer considers only left-linear plans and the case in which it considers all bushy plans). The graph shows that Tukwila never does worse than static plans (despite the overhead associated with changing plans in midstream), and that for some cases we obtain substantial savings.

Another key aspect of our approach is that we process data as it is *streaming* from the sources. Tukwila is the first query XML query processor for streaming XML data. Tukwila uses the Xscan operator that applies regular path expression constraints to streaming data and the double pipelined join that is especially appropriate for network-bound query processing [66]. In addition, Tukwila adapts techniques from relational pipelined query execution to the context of XML to achieve high performance. Tukwila substantially outperforms other query processors that need to first *load* all the data into a data structure in memory, and only then apply the query operators.

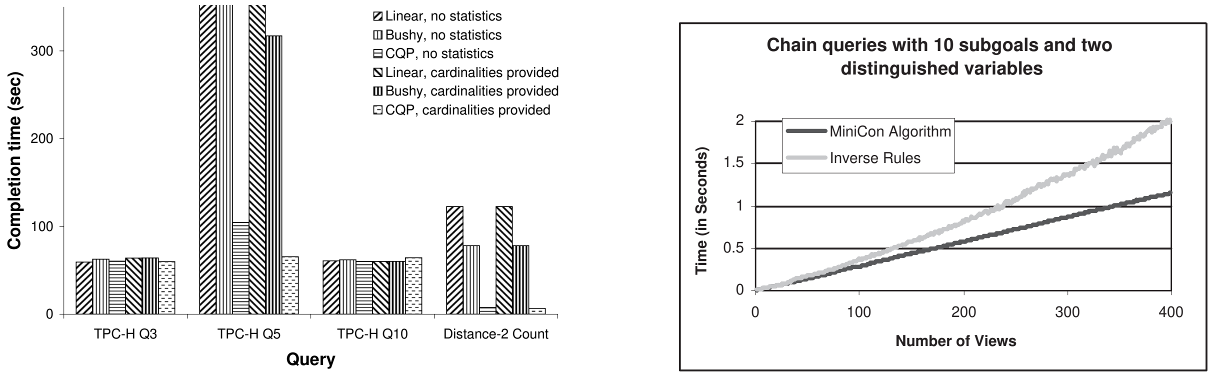
Answering queries using views requires subtle reasoning about the available data at the peers. For example, if a peer advertises that it stores the join of two relations R and S, then its data can be used for a query that joins R, S and T. However, that data, *by itself*, may not be enough to answer a query that simply requests R, since the join of R and S already lost the tuples of R that did not join with S. The topic of answering queries using views has received a significant amount of attention in recent years (Halevy surveys the work in [58]). The works on this topic have generally fallen into two camps: using views for more efficient query processing [108, 99, 11, 31, 110, 19, 48] and using views for describing data sources in data integration systems [80, 36, 73, 74], where the focus has been on the logical aspects of the problem. In Piazza, we need a combination of these two bodies of work. We use views to describe the data stored at the peers, but we also need to find the most efficient way to answer a query using a set of views because there will be many ways of obtaining the same answer.

Our approach will be based on extending the MiniCon Algorithm [89, 88] that we developed at the University of Washington. MiniCon is the first algorithm for answering queries using views that has been shown to scale up to large number of views. In Figure 2(b) we show the scaleup of MiniCon for the common case in which the views and queries are chain queries. The figure shows that MiniCon performs well even in the presence of hundreds of views, which is a scenario that is likely to happen in Piazza.

Tukwila and MiniCon put us in an excellent position to build a query answering system for Piazza. To combine them appropriately and address the challenges faced in Piazza, we will address the following issues:

- We will extend the MiniCon algorithm to handle views and queries in XQuery. In its full generality, answering XQuery queries using XQuery views is undecidable. However, the challenge is to identify important subsets of XQuery for which it is possible to build scalable algorithms for this problem. Our initial work on comparing queries that create nested structures [82] provides insight into such algorithms.

- Currently, Tukwila executes the entire query at one location after retrieving the data from the remote sources (though, when possible, it pushes computation to the data sources). We will extend Tukwila so it can decide to execute different parts of the query on different peers for improved efficiency. We note that



(a) First 5 tuples

(b) Total query time

Figure 2: The left graph shows the performance of Tukwila using *convergent query processing* (CQP) versus running Tukwila with static plans only. The queries (except for the last one) are taken from the TPC-H benchmark and involve select, project, join, grouping and aggregation. The graph shows that despite the overhead involved with reoptimizing the query and switching plans, the overall running times are still better than using a static plan. The right graph shows the performance of MiniCon compared to the Inverse-rules algorithm [35], and shows that MiniCon gracefully scales to hundreds of views.

developing such a strategy spans both the logical and physical aspects of query optimization, because the logical form of the plan may be different depending on where it is executing (and hence which views are available).

- Finally, when the index of sources is distributed, the time to locate a piece of data for query planning purposes may be significant. Hence, we will devise an execution strategy that interleaves query execution and data location.

3.3 Data Placement

The overall cost of answering a query in Piazza includes the transfer cost from the storage provider or data origin to the query evaluator, the cost of resources utilized at the query evaluator and other nodes, and the cost to transfer the results to the query initiator. Hence, one of the main opportunities for improving performance and sharing resources in a PDMS is by caching data at multiple peers such that it is *closer* to peers that need the data. The *data placement problem* is to distribute data so the full query workload is answered with lowest cost under the existing resource and bandwidth constraints. Data placement is a novel challenge raised by systems such as Piazza, and we expect it to be one of our main strands of work.

A specific case of the data placement problem appears in distributed and cooperative Web caching [15, 38, 46, 105], where the problem is optimal placement of requested Web pages within the caches. There, the data is cached at the granularity of Web pages and images; either a page is in the cache or it is not. The data placement problem for a PDMS is more complex, because views may contain some but not all of the data needed, and the system may decide to create views that provide intermediate results for many subsequent queries. In addition, multiple views may be combined to answer a query. The expressive query language and data model increases the opportunities for reuse of cached data, but on the other hand, the space required to cache some useful views may be larger (compared to Web pages). We will examine the tradeoffs here in detail.

A primitive version of the data placement problem is the problem of view selection for database systems or data warehouses [4, 21, 53, 55, 97, 109, 8, 54, 59, 70]. In the former case, the problem is to select a set of views to materialize in a database such that the performance of a workload of queries will improve. In a sense, this is the data placement problem with a single peer. In [22] we provide a theoretical analysis of the view selection problem and identify the pitfalls that cause the problem to be intractable. In a separate result [50] we show that even with a very simple data model, the data placement problem is already NP-complete. Hence, our focus will be on developing algorithms that work well in practice, and evaluating them in real settings.

Data placement can be used to increase availability in addition to increasing performance. As the number of cached views of data across the system grows, the availability of the data in those views grows. It is possible to reconstruct views from origin data; it is also possible to reconstruct views by combining data from several other related cached views elsewhere in the system. Calculating the degree of availability of data in Piazza is therefore complex, and as a consequence, so is data placement for availability.

We will explore the problem space of data placement in several steps, as outlined below. In each case, we will develop algorithms for data placement, provide an analysis of the algorithms and conduct experiments in settings that include up to several hundred peers, and consider a variety of workloads. Furthermore, we also consider which indexes to build on the data in each of these cases.

- We begin with the simple case, in which we only consider caching queries that have been asked before. Queries will only be cached at the peer that originated the query, and space will be managed using some eviction policy.
- Next we will consider caching queries that have not explicitly been asked by any peer. These are queries that would be useful in computing many frequently asked queries, and can be derived by analyzing common subexpressions in a workload
- We will study the effect of updates on the data placement policy. Now the tradeoff also includes the probability that a cached query will become out of date, and the cost of updating it.
- The topology of the network introduces another dimension to the data placement problem. If there are spheres of cooperation within the network (or if there are significant variations on the time it takes to transfer data between pairs of nodes), then we also want to consider in our data placement algorithms the workload of the entire sphere of cooperation and the resources of the sphere.
- We will explore the relationship between placement for performance, and placement for availability. We will devise techniques for calculating the availability of data, given the degree of replication of the origin data, the availability of the peers hosting the data, and the reconstructability of the data from materialized views that exist in the system.
- Finally, there is a subtle interaction between data placement and distributed query evaluation. By exploiting previously cached queries, we obtain some of the effect of distributed query evaluation. A view cached at a peer and used later by another peer is akin to a computation that is done for the subsequent query at the first peer. We will study to what extent distributed query evaluation can be *achieved* by intelligent data placement. We will also study the effect of updates on the interaction between these two problems.

3.4 View Indexing

The views that are available to the query processors at the peers need to be made known to them. There can be, perhaps, hundreds of thousands of different views scattered over the peers as a result of the data placement activity (Section 3.2), but, for a given query, only a small fraction (say, a few hundreds) may be relevant at all. A *view index* is a data structure that keeps track of all available views in the community, and that can be searched in order to find the few that may be relevant to a given query. When a query is posed by any peer in the system, the view index is first consulted to retrieve the relevant views, and these are then fed into the query optimization algorithm. The view index will play a central role in Piazza, and it poses three major technical challenges.

The first is that we need to index view *definitions* rather than data. A definition is a query expression, and there has been no index structure developed so far for indexing expressions. Commercial database systems index *data*, and usually from an ordered domain (such as numbers or strings): the favored indexing method is the B^+ tree. For more complex domains various extensions have been proposed, most notably R-trees [56] and its variations for the domain of high dimensional data. None, however, indexes query expressions as opposed to data. To highlight the complexity of the problem, we notice here that in two quite restricted special cases, the view indexing problem becomes related to well studied (and difficult) data indexing problems. In one case all view definitions are restricted to be selection queries with inequality operators over the same table and attribute: then indexing views becomes similar to indexing intervals,

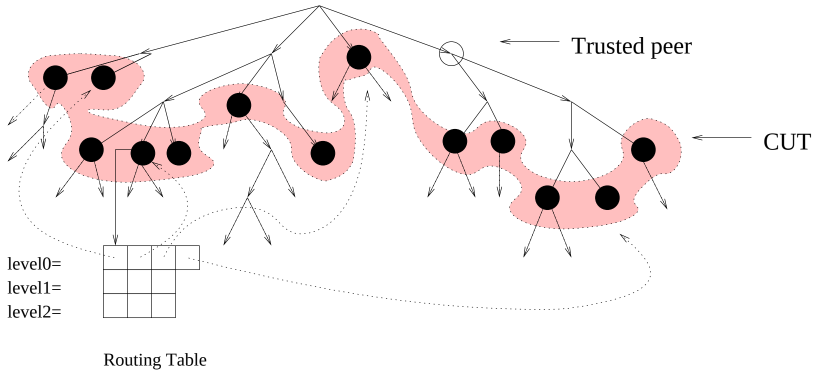


Figure 3: A technique for converting any search tree to a distributed search tree in a PDMS system. Nodes from a *cut* in the tree are assigned to peers. The cut does not need to be balanced. A search may start at any node in the cut. Each such node has a routing table to guide the search, with one row for each level above the node. When peers become too loaded, or when more peers join the system, nodes can split (i.e. the cut migrates down) and, optionally, more trusted peers may retain control of nodes above the cut.

for which some data structures exists in the field of computational geometry (e.g. range trees [91]). For example view definitions like $\sigma_{30 \leq cholesterol \leq 80}(Patient)$, $\sigma_{55 \leq cholesterol \leq 90}(Patient)$ would be translated into the intervals $[50, 80]$, $[55, 90]$ and these are stored in the index. In another extreme case all view definitions are restricted to be selections where the predicate is a conjunction of simple atomic conditions. In this case we can translate each view into a finite set of atomic conditions. Indexes structures specialized for sets have been proposed before, e.g. RD-trees [60]. For example a view definition like $\sigma_{highCholesterol \wedge young}(Patient)$ becomes the set $\{highCholesterol, young\}$. These two examples suggests that view indexing needs to be as general as indexing intervals and indexing sets, but, of course, much more general than both.

The second challenge is that the index needs to be distributed over a set of unreliable peers. Distributed index structures have been proposed recently for peer-to-peer systems (Plaxton trees [87], Chord [95]), but they have two shortcomings for applications like Piazza. First, in all these systems the keys being indexed are atomic, without any semantics. For example in Plaxton trees the keys are integers between 0 and $2^n - 1$. If other values need to be indexed (say, strings) then they need to be hashed to the key space. The only searches supported are exact matches, which makes them inadequate for view indexing. A second shortcoming is that in their approach to redundancy all peers are considered equal. In Piazza we assume that some peers are more trusted or more reliable than others.

Finally, the third challenge is that in Piazza often an inexact answer suffices and may be preferable to an exact answer that may take a much longer time to compute. For example, if the query is $\sigma_{flightTime<5h}(ScubaFoodSnobsFlights)$ and a nearby node holds the view $\sigma_{flightTime<4h}(ScubaFoodSnobsFlights)$ the latter may be a sufficient answer, and preferable over a search of the entire community for the few missing answers whose *time* is between $4h$ and $5h$. One approach to model inexact answers is to use *precision* (the fraction of the data in the view that is relevant to the query) and *recall* (fraction of the data items requested by the query that are contained in the view), in the sense used in information retrieval [6]. An exact search strategy corresponds to a recall of 100% while an approximate, more flexible strategy corresponds to a lower recall. Such an approach, however, creates a new kind of challenges for the index data structure and search algorithm, to reason about the views' extent and overlap, in order to estimate their precision and recall.

In some initial investigations we considered a few approaches to the centralized view indexing problem. One is to map views into a space of *feature sets*, and build an index for the latter (using, for example, RD-trees [60]). A view is relevant to a query if its set of features is included in (or includes, depending on the particular feature set) that of the query. For example, in the case of views expressed by conjunctive queries, possible choices for feature sets are: the set of all relation names occurring in the view definition, or the set of all joins in the view definition, or the entire query graph (encoded as a set). We are currently experimenting with this and other approaches.

We are also addressing the distribution challenge and have designed a method for transforming any tree-based index structure to a distributed index. The method applies both to balanced tree structures (e.g., B^+ trees, R trees, or, more generally, those considered in GiST [60]) and unbalanced structures, like Patricia trees. In the *distributed* version of the tree structure, peers hold only nodes from a *cut* in the tree. A cut is any set of nodes that separate the root from the leaves, and every peer assigned to a node in the cut stores the entire information in that subtree. Fig. 3 illustrates the idea. A search proceeds in a style that generalizes Plaxton trees [87]: any peer in the cut may be contacted to start a search, and the search proceeds along the cut, in a "horizontal" manner: this is different from a search on the centralized version, which proceeds "vertically" from the root. For that, each node in the cut holds a *routing table*, that contains one row for every ancestor, and one entry in that row for each child of that ancestor. A search hops from cut node to cut node, at each hop narrowing down the subtree where the value is located.

In this project will will conduct research addressing the following issues:

- We will design, implement, and evaluate a centralized indexing structure for views, defined by *conjunctive queries* with range predicates. The index should scale to a hundreds of thousand of views, perhaps more. In addition to the search time, we will evaluate how effective the index is in selecting views *only* relevant to a given query.[1]
- We will design, and implement a distributed indexing structure for an arbitrary search trees. In an initial version, we plan to scale to tens of peers. We will evaluate the method and understand its shortcomings. In a second stage we will design an improved techniques that scales to more peers, perhaps hundreds or more.
- We will investigate more relaxed search algorithms in the distributed indexing structure that trade off recall for response time. We will implement such algorithms on the distributed index structure, and evaluate their performance gain.

3.5 Managing Updates in PDMS

One of the significant concerns in building a system such as Piazza is the efficient management of updates to the data. Since the data at the peers are interlinked in complex ways, one update to the data may have far reaching effects. Furthermore, when data is replicated for purposes of performance and availability, managing updates becomes even harder.

A conservative approach is that whenever some data changes, we mark any (complete or partial) replica of the data or any view that depends on the data as out-of-date. We can either decide to recompute the view from the new version of the data, or to wait until the next time that a peer requests something from the view to recompute. In both cases, when the views are recomputed the cost will be significant.

The approach we follow is to identify updates as first-class citizens in our system. Whenever some data changes, we summarize the changes as an *updategram*. Intuitively, the updategram is much smaller than the entire data. If we manipulate updategrams in sophisticated ways, we can maintain the data in the system up-to-date with much less cost.

In some cases, for simple views, one can maintain views over data using only updates. For example, views that involve only selection and projection can usually maintained by examining the updategram. However, when queries involve joins, this is no longer the case. Consider a simple join of two relations $R \bowtie S$, and suppose R published an updategram. It is not possible to update the join without accessing the relation S. Even so, updating $R \bowtie S$ will be much cheaper than computing it from scratch.

If the $R \bowtie S$ is a common subexpression in many queries, we can do better. Given an updategram for R, we can compute a *booster* for S: the booster includes all of the tuples of S that may appear in a derivation of $R \bowtie S$ with a tuple in the updategram of R. The booster is computed once, but its results, can be used in different places. In particular, from the updategram of R and the booster of S we can compute an updategram for $R \bowtie S$.

Another ripe problem area is that of maintaining the integrity of data as it is replicated across the wide-area, especially in the face of updategram and booster propagation. In addition to indexing views and data, there will be many situations in which it is important for a peer that uses data to have confidence that the data is intact. We intend on exploring conventional cryptographic techniques for verifying the integrity of replicated views; for example, signed hashes of views should allow all peers to independently verify its

[1] The feature set implementation that we are currently exploring will return false positives.

integrity. To handle updates, we plan to extend the notion of an updategram to include a signed hash of the updated view.

In our work, we will publish updategrams and boosters in the system and use them for efficient maintenance of views. Specifically, we will do the following:

- We will develop an algebra of updategrams and boosters. The algebra will answer the questions of (1) when can a view be updated using only updategrams of its subexpressions, (2) how updategrams and boosters can be combined to generate additional updategrams, (3) given a set of updategrams and boosters, what are the logically correct ways of using them for answering a given query.
- We will develop a cost-based solution to using updategrams in Piazza: given a peer that stores a certain version of a view and a database of updategrams and boosters, what is the cheapest sequence of computations that will bring the view up to date. We will implement and test the cost-based algorithm we develop.
- We will develop techniques for expressing and certifying the integrity of origin data and cached views. We will extend our techniques to include validating the integrity of updategrams and boosters, so that data integrity can be assured across updates.
- Finally, we will consider the problem of updategram-placement. If we know which updategrams are good to compute, we must then determine the most effective places on the network to send and store them.

3.6 Content Indexing and Search

When a PDMS such as Piazza becomes popular, an enormous amount of data will be made available by its peers. One goal of a PDMS should be to illuminate the so-called "dark matter" on the Web that is hidden behind database front-ends. According to estimates in [7], the amount of dark data is at least 10 times as large as the data conventionally indexed by search engines. Data is only useful if it can be found, and if Piazza is to succeed, we will need to provide a search mechanism that is the analog of Web keyword searching to help people find relevant data.

In the Web, search engines provide a mechanism for locating content. Unfortunately, Web search engines are inefficient (as they must proactively crawl data), incomplete (since they rely on hyperlink structure to locate data), and stale (since they do not know when data has changed). Because of the rich structure of data and names in Piazza, we believe that we can overcome these deficiencies.

Updategrams and boosters in Piazza capture precisely the information necessary to compute changes in data. Rather than forcing search engines to crawl base relations and views, we intend on pushing updategrams and boosters to search engines. Changes that do not need to be immediately reflected in the search engine can be delayed, while more important changes can be immediately pushed. We intend to leverage existing research in efficiently updating search engine indices [26, 14, 98] to efficiently incorporate updategrams and boosters upon arrival at the engine.

We also have the opportunity to explore query paradigms that are richer than simple keyword search; the semantics exposed by the data in a PDMS could be exploited to improve search results. In particular, since people will be searching for data sources as well as data, they should be able to pose queries that combine properties of the data as well as properties of the schema. However, people should also be able to pose queries without knowing any schemas or structure.

The structure of the data in a PDMS can also be exploited to provide richer ranking functions in search engines. In [16, 71, 17], the hyperlink structure of the Web is used to augment the ranking of search engine results based on the notion of authoritative data sources, and hubs that point to them. In Piazza, the view structure is similar to the hyperlink structure, except that instead of only having pointers to related data, we also have knowledge of the queries that were used to compute data, and therefore the manner in which the data is being *used*.

Our research agenda therefore includes:

- Devising algorithms for pushing content updates into search engines, and efficiently incorporating those updates into their indices. Note that we must be able to prevent a peer from overwhelming a search engine with work from updates, and therefore we will need to consider rate-limiting techniques.

- Exploring techniques for mining the semantics of data, and the semantics of the view structure defined within a PDMS to improve the quality and precision of search.

3.7 Managing Schema Heterogeneity

Traditional database technology works only in the context of a centralized, well understood schema. In contrast, an important objective of Piazza is to include a diversity of peers, without imposing a centralized schema. Clearly each peer will have some schema for the data it contributes. The challenge is to offer peers the tools to mediate between them without centralized control. This is a more challenging problem than that encountered in data integration systems and heterogeneous database systems that also faced the problem of schema heterogeneity. The solutions in those contexts involved designing a single logical *mediated* schema, against which queries are posed. The mediated schema is logical in the sense that the data was still stored in the local schemas of the data sources, but the system was able to translate queries posed over the logical schema into a set of queries over the local schemas. In our context, a centralized logical mediated schema is undesirable, and often, impractical, because it still implies a certain level of centralized control.

Hence, an important goal of Piazza is the design of languages for mediating between schemas of different peers. Our approach is to leverage techniques developed in the data integration literature. Those systems explored two main approaches to such mappings: Global-as-view [47, 57, 3], where the mediated schema is defined as a set of views over the data sources, and Local-as-view [80, 36, 73, 74], where the contents of data sources were described as views over the mediated schema (for a comparison between the approaches see [62, 100, 78, 43]). These two approaches alone would not suffice in our context because we do not have a single mediated schema (and clearly, we do not want to specify mappings between every possible pair of peers).

We note that the focus in Piazza is on relatively simple languages for representing the data itself. In contrast, work in the context of the semantic web [13] has developed a more expressive language, DAML [61], for expressing ontologies over data, and hence richer sets of constraints on the data. In previous work [75, 76], we established some of the fundamental relationships between database query languages and expressive knowledge representation formalisms. So although expressive knowledge representation is not within the scope of Piazza, we will try to forge connections to the work on the semantic web wherever applicable.

To support a more ad-hoc architecture in Piazza we specify semantic relationships between peers using local *peer descriptions.* Most commonly, a peer description will specify semantic relationships between sets of relations of two different peers. However, peer descriptions can also describe a many-to-one relationship between peers or even many-to-many relationship. In addition to peer descriptions, peers also provide *storage descriptions* that specify which sets of data they actually store locally. Answering a peer query is enabled by piecing together multiple peer descriptions and obtaining the relevant data from the appropriate peers. Our specific objectives are the following:

- Study a spectrum of languages for expressing peer and storage descriptions. We will study the relative expressive power of such languages, and define the semantics of answering queries in such a context (generalizing the notion of *certain answers* [2]).
- Develop efficient algorithms for answering queries that involve sophisticated translations between peer schemas.
- Consider several fundamental semantic properties of a PDMS: when are two sets of peer descriptions equivalent, in the sense that they return the same answer for *any* query? We note that this is a considerably different problem from the equivalence of two *given* queries, a problem that has been studied thoroughly in the database literature. The problem of PDMS equivalence is a cornerstone to studying other properties, such as PDMS minimization (which peers can be eliminated without losing information), and PDMS consistency (are there satisfiable data instances for the PDMS).
- Extend the treatment of peer descriptions to the case where peers have integrity constraints (e.g., functional dependencies, inclusion dependencies), but these constraints are not necessarily consistent across peers (and hence, using another peer's data is even more subtle).

- Consider probabilistic peer and storage descriptions: often it is impossible to state exactly which data a peer stores. For example, a peer may contain the set of publications about database systems, but all that can be said about the set is that the probability that a given publication is at the peer is 0.8. Furthermore, probabilities may also be required at the schema level: the class of database papers and operating systems papers is not disjoint, but may overlap with a certain probability. We will extend the probabilistic representations proposed in [42, 72] to the Piazza setting.

3.8 The Architecture of a Piazza Peer

Externally, the behavior of a Piazza peer is that of a network-accessible system that makes a set of views available for selection and retrieval, issues queries to other nodes, and caches views locally based on some data placement policy. Internally, a Piazza peer must manage disk, memory, network, and computational resources, and must balance these resources across externally and locally originating queries. We anticipate that some peers will receive a much greater volume of external queries than others; these popular peers may host commonly accessed data and views, or they may be acting as dedicated view caches for large populations (much like an institutional Web cache).

3.8.1 Resource Management

In Web servers, resource management is largely an issue of admission control, rather than one of scheduling policy. While some policies can have an effect on task latency [25], each Web request involves a relatively small amount of work, and thus the main challenge of servers is that of gracefully handling periods of excess load rather than the careful scheduling of resources across tasks. In contrast, we expect that the amount of work that a Piazza node must perform will vary greatly depending on the nature of the query being executed. For externally originating queries, the range of data being selected as well as the presence or absence of an index may have a dramatic effect on the amount of work the node must perform.

An initial issue we will consider is extending the work on adaptive query processing at the University of Washington [66, 63, 65] to handle scheduling of a large number of concurrent queries on a Piazza node. Recall that Tukwila blurs the line between query optimization and execution, and using the same ideas we can view the execution plans of all currently executing queries as resources that can be dynamically adjusted.

Another issue that we wish to address is the fair allocation of resources in a Piazza node. We believe that the query is the right level of abstraction on which to allocate resources: certain classes of queries should have priority over others. These query classes can be defined in terms of the views over which they are defined or the nodes from which they originate. Class-based queueing [44] or hierarchical fair queueing [27, 12] in networking systems provide mechanisms for allocating network bandwidth across classes of flows; we want to explore similar mechanisms for allocating query processing resources across classes of queries. Once we have built such a mechanism, we will use it to explore how to adaptively adjust the priority of locally originating queries relative to external queries. Additionally, this will let us investigate the effects of locally greedy optimization strategies, versus those that attempt to optimize globally across a sphere of cooperation.

Denial of service is a growing concern for networked servers, and it is entirely possible for the Piazza system to be attacked in this manner. Unlike the Web, a single query issued by an untrustworthy peer could potentially cause terabytes of data to be transferred across hundreds of peers. Both local and global mechanisms will be necessary to limit the amount of work done by any individual query. Query optimization techniques that estimate the amount of work done by a query should make it possible to cap local resource consumption, and we hope to build upon work at the University of Washington on protocol containment in active networks [104] to enforce global work containment across Piazza peers.

3.8.2 External Interfaces and Protocols

As part of the Piazza architecture, we must design a protocol through which peers communicate. The protocol will expose a set of operations that can be invoked on a peer, and it will also define the data and metadata that is passed in and returned by each operation. Some of the operations that a peer may support include returning an index of cached views, returning an index about origin data, and selecting from a view or origin data. To make a local data placement decision, retrieved data should include metadata indicating how long the data is likely to remain valid. Where possible, indexes of cached views and origin data should also include the join selectivity of the data (to help remote peers formulate efficient query plans across views).

Additionally, when a source peer requests data from a destination peer, we believe that the source should pass in the complete query that caused it to select the destination; this will help the destination peer reason about why its data is popular, and possibly enable it to materialize more useful views proactively. For example, assume that peer P1 contains the relations R1 and R2. Furthermore, assume that P2 issues the query R1 $\bowtie$ R2 where $x < 7$, and that P3 issues the query R1 $\bowtie$ R2 where $x > 5$. Any further queries of a similar structure but non-contained selection predicates will retrieve relations R1 and R2 from P1 and duplicate the effort of calculating the join. If P1 were informed that the queries executing on peers that retrieve data from it had the common subexpression R1 $\bowtie$ R2, P1 could proactively materialize the join.

The transport layer over which the data flows from a Piazza peer is another topic of interest. Clearly data must be reliably transported, but in most situations, the order in which the data is transmitted doesn't need to be preserved. Therefore, TCP may not have the ideal data transport semantics for our purposes. Update propagation provides another opportunity for exploring transport in Piazza. If a large number of views could benefit from an updategram, it will be more efficient to form an overlay multicast [69, 23] tree across the peers caching those views than to individually unicast them.

3.8.3 Scalability and Availability

As the data and materialized views served by a Piazza peer become more popular, that peer must be able to scale in terms of the traffic it can handle, and it must also have high availability. Clusters of workstations provide a natural platform for scalable, highly available systems, if those systems have a large amount of coarse-grained parallelism [45]. In Piazza, query workloads and data are both potential sources of parallelism, implying that both computation and data can be partitioned and replicated across the cluster.

Given that queries can be heterogeneous in the amount of work performed to satisfy them, we anticipate that data partitioning and load balancing in a clustered Piazza node will be a rich problem space. In addition to the familiar (but still challenging!) problem of placing data with a cluster to optimize a given workload, in Piazza we will also have to factor in the cost of distributing and applying updates within the cluster. Similarly, given a set of updategrams, instead of bringing all replicas of a view up to date it may be more efficient to update a single view replica and instead replicate the updategrams for availability.

In many scalable systems, manageability ultimately becomes the bottleneck. A significant part of management complexity is the dynamic configuration of the system in response to the addition or removal of nodes, or fluctuation in workload. Configuration includes including the delegation of responsibility, the placement of data and computation, and the efficient generation and distribution of metadata such as indexes. To maintain manageability, this configuration must must be as automated as possible [92]; a significant challenge in building a scalable Piazza node will be identifying all of the configuration decisions, and building mechanisms and algorithms to automate them.

In summary, we plan to explore the architecture and implementation of piazza peers, specifically focussing on the following issues:

- The design of resource management algorithms and of fairness mechanisms across queries.
- The design of scalable, highly available Piazza peers implemented using clusters of workstations, including consideration of automated management of these peers.
- The design of the protocols and transports that facilitate data exchange between peers, including the operations that a peer supports, and the metadata passed with the operations and their return values to enhance functionality.

3.9 Summary

This proposal has outlined some of the high-level issues in Piazza, a long-range research effort aimed at creating a global-scale *data-oriented* information facility, which we have labelled a Peer Data Management System (PDMS). Our goal is to facilitate sharing of complex data among cooperating communities on the Internet. The Piazza vision is broad, but we will focus on the design and implementation of a key set of problems underlying that vision. In the short run, we will create an experimental framework for developing and analyzing small- to medium-scale data-sharing applications. In the context of this framework, some of the specific deliverables will be the following:

- We will architect the Piazza PDMS and its peer nodes, including both local and global resource management algorithms and fairness policies. We will design and implement the infrastructure on which we can create a highly scalable and available PDMS.
- We will develop algorithms for efficiently answering queries in a PDMS that exploit local data as well as remote data. Our algorithms will handle data in multiple formats, using XML as the common data model, and will make use of powerful local query processors where possible. Unlike traditional database systems, which rely heavily on detailed statistics, Piazza's query plans must be adaptive and heuristic, because complete knowledge in a wide-area distributed system is impossible to achieve.
- We will develop protocols for automatically placing data views on nodes of a PDMS for improved query throughput. The cached views will be either results of explicit query requests or computed queries that are expected to speed up future requests. The data placement protocols will consider the observed workload on the system, the structure and performance of the network, and the tradeoff between local efficiency of a node and global efficiency.
- We will develop efficient algorithms for propagating updates to maintain consistency in a PDMS. The algorithms will be based on the concept of *updategrams*, which Piazza will treat as first class citizens in query processing. Our data placement algorithms will consider placement of updategrams.
- We will create methods for indexing large numbers of queries, and performing fast retrievals based on a variety of criteria. We will also develop distributed data structures for indexing large numbers of queries.
- We will develop languages for describing semantic relationships between schemas of different peers, and reformulating peer queries to exploit data from other peers.
- We will create methods for searching large spaces of semantically rich data and augmenting the ranking of the results by exploiting the semantics. We will also invent new methods for summarizing the contents of structured databases that are amenable to search engines, and proactively notifying the search engines of appropriate updates to the data.
- We will design the network protocols and transport systems that facilitate data exchange between peers in our system. We will develop replication algorithms to increase data availability in the face of failures and techniques to enforce data integrity.

Overall, we will create the technical basis for a PDMSand will implement an experimental platform on which applications can be constructed and studied. We will begin at a small scale, using computer clusters we've constructed in our building, moving to the larger scale as the infrastructure develops.

This project includes significant challenges, but has significant potential for impact as well. The challenges cross all areas of traditional databases, semi-structured data, distributed systems, operating systems, and networks. We believe that the PIs combined experience in all of these areas gives us the ability to resolve these challenges, using recent advances that we and many others have developed. In the end, our goal is to greatly expand the potential for data management in the next-generation Internet environment.

References

[1] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web : From Relations to Semistructured Data and Xml*. Morgan Kaufmann, 1999.

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

[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] S. Agrawal, S. Chaudhuri, and V. Narasayya. Automated selection of materialized views and indexes in Microsoft SQL Server. In *Proc. of VLDB*, pages 496–505, Cairo, Egypt, 2000.

[5] R. Avnur and J. M. Hellerstein. Eddies: Continuously adaptive query processing. In *SIGMOD '00*, 2000.

[6] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley, 1999.

[7] P. Bailey, N. Craswell, and D. Hawking. Chart of darkness: Mapping a large intranet. Technical report, Commonwealth Scientific and Industrial Research Organization (CSIRO), Department of Mathematical and Information Sciences, 2000.

[8] E. Baralis, S. Paraboschi, and E. Teniente. Materialized views selection in a multidimensional database. In *Proc. of VLDB*, pages 156–165, 1997.

[9] C. K. Baru, A. Gupta, B. Ludäscher, R. Marciano, Y. Papakonstantinou, P. Velikhov, and V. Chu. XML-based information mediation with MIX. In *SIGMOD '99*, pages 597–599, 1999.

[10] C. Beeri, A. Y. Levy, and M.-C. Rousset. Rewriting queries using views in description logics. In *Proc. of PODS*, pages 99–108, Tucson, Arizona., 1997.

[11] R. Bello, K. Dias, A. Downing, J. Feenan, J. Finnerty, W. Norcott, H. Sun, A. Witkowski, and M. Ziauddin. Materialized views in Oracle. In *Proc. of VLDB*, pages 659–664, 1998.

[12] J. C. R. Bennett and H. Zhang. Hierarchical packet fair queueing algorithms. *IEEE/ACM Transactions on Networking*, 5(5):675–689, 1997.

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

[14] E. Brown, J. Callan, and W. Croft. Fast incremental indexing for full-text information retrieval. In *Proceedings of the 20th Very Large Data Base Conference (VLDB'94)*, Santiago, Chile, 1994.

[15] P. Cao, J. Zhang, and K. Beach. Active cache: Caching dynamic contents on the web. In *Middleware '98*, Sept. 1998.

[16] S. Chakrabarti, B. Dom, D. Gibson, J. Kleinberg, S. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. Hypersearching the web. *Scientific American*, June 1999.

[17] S. Chakrabarti, B. Dom, D. Gibson, J. Kleinberg, P. Raghavan, and S. Rajagopalan. Automatic resource compilation by analyzing hyperlink structure and associated text. In *Proceedings of the 7th World-Wide Web Conference*, 1998.

[18] D. Chamberlin, D. Florescu, J. Robie, J. Simeon, and M. Stefanascu. XQuery: A query language for XML. Technical report, World Wide Web Consortium, February 2001. Available from http://wwww.w3.org/TR/xquery/.

[19] S. Chaudhuri, R. Krishnamurthy, S. Potamianos, and K. Shim. Optimizing queries with materialized views. In *Proc. of ICDE*, pages 190–200, Taipei, Taiwan, 1995.

[20] M. Chesire, A. Wolman, G. Voelker, and H. Levy. Measurement and analysis of a streaming media workload. In *Proceedings of the 3nd USENIX Symposium on Internet Technologies and Systems*, March 2001.

[21] R. Chirkova and M. Genesereth. Linearly bounded reformulations of conjunctive databases. In *Proc. of DOOD*, pages 987–1001, 2000.

[22] R. Chirkova, A. Halevy, and D. Suciu. A formal perspective on the view selection problem. In *Proc. of VLDB*, 2001.

[23] Y.-H. Chu, S. G. Rao, and H. Zhang. A case for end system multicast. In *Measurement and Modeling of Computer Systems*, pages 1–12, 2000.

[24] R. L. Cole. A decision theoretic cost model for dynamic plans. *IEEE Data Engineering Bulletin*, 23(2):34–41, 2000.

[25] M. Crovella, B. Frangioso, and M. Harchol-Balter. Connection scheduling in web servers. In *USENIX Symposium on Internet Technologies and Systems (USITS '99)*, Boulder, CO, USA, October 1999.

[26] D. Cutting and J. Pedersen. Optimizations for dynamic inverted index maintenance. In *Proceedings of the 13th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1990.

[27] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. In *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 1–12, Austin, TX, September 1989.

[28] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu. A query language for XML. http://www.research.att.com/ mff/xml/w3c-note.html, 1998.

[29] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu. A query language for XML. In *Proceedings of the Eights International World Wide Web Conference (WWW8)*, pages 77–91, Toronto, 1999.

[30] A. Deutsch, M. Fernandez, and D. Suciu. Storing semistructured data with STORED. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 431–442, 1999.

[31] A. Deutsch, L. Popa, and V. Tannen. Physical data independence, constraints and optimization with universal plans. In *Proc. of VLDB*, pages 459–470, 1999.

[32] A. Doan, P. Domingos, and A. Halevy. Reconciling schemas of disparate data sources: a machine learning approach. In *Proc. of SIGMOD*, 2001.

[33] R. Domenig and K. Dittrich. An overview and classification of mediated query systems. *SIGMOD Record*, 28(3):63–72, September 1999.

[34] D. Draper, A. Y. Halevy, and D. S. Weld. The nimble xml data integration system. In *Proc. of ICDE*, pages 155–160, 2001.

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

[36] O. M. Duschka and M. R. Genesereth. Query planning in infomaster. In *Proceedings of the ACM Symposium on Applied Computing*, pages 109–111, San Jose, CA, 1997.

[37] O. M. Duschka and A. Y. Levy. Recursive plans for information gathering. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, pages 778–784, 1997.

[38] L. Fan, P. Cao, J. Almeida, and A. Z. Broder. Summary cache: A scalable wide-area web cache sharing protocol. In *Proc. of ACM SIGCOMM '98*, August 1998.

[39] M. Feeley, W. Morgan, F. Pighin, A. Karlin, H. Levy, and C. Thekkath. Implementing global memory management in a workstation cluster. In *Proceedings of the 15th ACM Symposium on Operating Systems Principles*, December 1995.

[40] M. Fernandez, A. Morishima, and D. Suciu. Efficient evaluation of XML middle-ware queries. In *Proceedings of ACM SIGMOD Conference on Management of Data*, Santa Barbara, 2001.

[41] M. Fernandez, D. Suciu, and W. Tan. SilkRoute: trading between relations and XML. In *Proceedings of the WWW9*, pages 723–746, Amsterdam, 2000.

[42] D. Florescu, D. Koller, and A. Levy. Using probabilistic information in data integration. In *Proc. of VLDB*, pages 216–225, Athens, Greece, 1997.

[43] D. Florescu, A. Levy, and A. Mendelzon. Database techniques for the world-wide web: A survey. *SIGMOD Record*, 27(3):59–74, September 1998.

[44] S. Floyd and V. Jacobson. Link-sharing and resource management models for packet networks. *IEEE/ACM Transactions on Networking*, 3(4):365–386, 1995.

[45] A. Fox, S. D. Gribble, Y. Chawathe, E. A. Brewer, and P. Gauthier. Cluster-based scalable network services. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles*, St.-Malo, France, Oct 1997.

[46] S. Gadde, M. Rabinovich, and J. Chase. Reduce, reuse, recycle: An approach to building large internet caches. In *Proceedings of the Workshop on Hot Topics in Operating Systems (HotOS)*, May 1997.

[47] 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.

[48] J. Goldstein and P.-A. Larson. Optimizing queries using materialized views: a practical, scalable solution. In *Proc. of SIGMOD*, pages 331–342, 2001.

[49] T. J. Green, M. Onizuka, and D. Suciu. Processing xml streams with deterministic automata and stream indexes, 2001. submitted for publication.

[50] 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.

[51] S. D. Gribble and E. A. Brewer. System design issues for internet middleware services: Deductions from a large client trace. In *Proceedings of the 1997 USENIX Symposium on Internet Technologies and Systems (USITS 97)*, Monterey, California, USA, December 1997.

[52] S. D. Gribble, E. A. Brewer, J. M. Hellerstein, and D. Culler. Scalable, distributed data structures for internet service construction. In *Proceedings of the 4th USENIX Symposium on Operating System Design and Implementation (OSDI 2000)*, San Diego, California, USA, October 2000.

[53] H. Gupta. Selection of views to materialize in a data warehouse. In *Proc. of ICDT*, pages 98–112, 1997.

[54] H. Gupta, V. Harinarayan, A. Rajaraman, and J. D. Ullman. Index selection for OLAP. In *Proc. of ICDE*, pages 208–219, 1997.

[55] H. Gupta and I. S. Mumick. Selection of views to materialize under a maintenance cost constraint. In *Proc. of ICDT*, pages 453–470, 1999.

[56] A. Guttman. R-Trees: a dynamic index structure for spatial searching. In *Proc. ACM SIGMOD International Conference on Managemt of Data*, pages 47–57, Boston, June 1984.

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

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

[59] V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. In *Proc. of SIGMOD*, pages 205–216, 1996.

[60] J. Hellerstein, J. Naughton, and A. Pfeffer. Generalized search trees for database systems. In *Proceedings of VLDB*, pages 562–573, Zurich, 1995.

[61] I. Horrocks, F. van Harmelen, and P. Patel-Schneider. DAML+OIL. http://www.daml.org/2001/03/daml+oil-index.html, March 2001.

[62] R. Hull. Managing semantic heterogeneity in databases: A theoretical perspective. In *Proc. of PODS*, pages 51–61, Tucson, Arizona, 1997.

[63] Z. Ives, A. Halevy, and D. .Weld. Convergent query processing. Submitted for publication, 2001.

[64] Z. Ives, A. Halevy, and D. Weld. Integrating network-bound xml data. *Data Engineering Bulletin 24(2)*, pages 20–26, 2001.

[65] Z. Ives, A. Levy, D. Weld, D. Florescu, and M. Friedman. Adaptive query processing for internet applications. *Data Engineering Bulletin 23(3)*, 2000.

[66] Z. G. Ives, D. Florescu, M. T. Friedman, A. Y. Levy, and D. S. Weld. An adaptive query execution system for data integration. In *SIGMOD '99*, pages 299–310, 1999.

[67] Z. G. Ives, A. Y. Halevy, and D. S. Weld. An XML query engine for network-bound data. Submitted for publication. Available at data.cs.washington.edu/integration/tukwila/xmlengine.pdf., 2001.

[68] H. Jamrozik, M. Feeley, G. Voelker, J. Evans, A. Karlin, H. Levy, and M. Vernon. Reducing network latency using subpages in a global memory environment. In *Proceedings of the 7th ACM Symposium on Architectural Support for Programming Languages and Operating Systems*, October 1996.

[69] J. Jannotti, D. Gifford, K. Johnson, M. Kaashoek, and J. Jr. Overcast: Reliable multicasting with an overlay network. In *Proceedings of the 4th USENIX Symposium on Operating System Design and Implementation (OSDI 2000)*, San Diego, California, USA, October 2000.

[70] H. J. Karloff and M. Mihail. On the complexity of the view-selection problem. In *Proc. of PODS*, pages 167–173, Philadelphia, Pennsylvania, 1999.

[71] J. M. Klienberg. Authoritative sources in a hyperlinked environment. In *Proceedings of the 9th ACM-SIAM Symposium on Discrete Algorithms*, 1998.

[72] D. Koller, A. Levy, and A. Pfeffer. P-Classic: a tractable probabilistic description logic. In *Proceedings of the AAAI Fourteenth National Conference on Artificial Intelligence*, 1997.

[73] C. T. Kwok and D. S. Weld. Planning to gather information. In *Proceedings of the AAAI Thirteenth National Conference on Artificial Intelligence*, pages 32–39, 1996.

[74] E. Lambrecht, S. Kambhampati, and S. Gnanaprakasam. Optimizing recursive information gathering plans. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, pages 1204–1211, 1999.

[75] A. Levy and M.-C. Rousset. Combining Horn rules and description logics in carin. *Artificial Intelligence*, 104:165–209, September 1998.

[76] A. Levy and M.-C. Rousset. Verification of knowledge bases based on containment checking. *Artificial Intelligence*, 101(1-2):227–250, 1998.

[77] A. Y. Levy. Obtaining complete answers from incomplete databases. In *Proc. of VLDB*, pages 402–412, Bombay, India, 1996.

[78] A. Y. Levy. Combining artificial intelligence and databases for data integration. In *Special issue of LNAI: Artificial Intelligence Today; Recent Trends and Developments, Lecture Notes in Computer Science, Volume 1600*, pages 249–268. Springer, 1999.

[79] A. Y. Levy, A. Rajaraman, and J. J. Ordille. Query answering algorithms for information agents. In *Proceedings of the National Conference on Artificial Intelligence*, pages 40–47, 1996.

[80] 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.

[81] A. Y. Levy, A. Rajaraman, and J. D. Ullman. Answering queries using limited external processors. In *Proc. of PODS*, pages 227–237, Montreal, Canada, 1996.

[82] A. Y. Levy and D. Suciu. Deciding containment for queries with complex objects and aggregations. In *Proc. of PODS*, Tucson, Arizona., 1997.

[83] H. Liefke and D. Suciu. XMill: an efficent compressor for XML data. In *Proceedings of SIGMOD*, pages 153–164, Dallas, TX, 2000.

[84] T. Millstein, A. Levy, and M. Friedman. Query containment for data integration systems. In *Proc. of PODS*, pages 67–75, Dallas, Texas, 2000.

[85] J. Naughton, D. DeWitt, D. Maier, A. Aboulnaga, J. Chen, L. Galanis, J. Kang, R. Krishnamurthy, Q. Luo, N. Prakash, R. Ramamurthy, J. Shanmugasundaram, F. Tian, K. Tufte, S. Viglas, Y. Wang, C. Zhang, B. Jackson, A. Gupta, and R. Chen. The Niagara Internet query system. *IEEE Data Engineering Bulletin*, June 2001.

[86] M. T. Ozsu and P. Valduriez. *Principles of Distributed Database Systems.* Prentice Hall, Upper Saddle River, New Jersey, 1999.

[87] C. Plaxton, R. Rajaraman, and A. Richa. Accessing nearby copies of replicated objects in a distributed environment. *Theory of Computing Systems*, 32:241–280, 1999.

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

[89] R. Pottinger and A. Levy. A scalable algorithm for answering queries using views. Submitted for publication, 1999.

[90] R. Pottinger and A. Levy. A scalable algorithm for answering queries using views. In *Proc. of VLDB*, pages 484–495, Cairo, Egypt, 2000.

[91] F. Preparata and R. P. M. Shamos. *Computational Geometry : An Introduction.* Springer Verlag, New York, 1991.

[92] Y. Saito, B. Bershad, and H. Levy. Manageability, availability, and performance in porcupine: A highly-scalable internet mail service. *ACM Transactions on Computer Systems*, 18(3), August 2000.

[93] Y. Saito and H. Levy. Optimistic replication for internet data services. In *Proceedings of the 14th International Symposium on Distributed Computing*, October 2000.

[94] S. Saroiu, P. K. Gummadi, , and S. D. Gribble. A measurement study of peer-to-peer file sharing systems. In *Proceedings of Multimedia Computing and Networking 2002 (MMCN'02)*, San Jose, California, USA, January 2002.

[95] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of SIGCOMM*. ACM, 2001.

[96] I. Tatarinov, Z. Ives, A. Halevy, and D. Weld. Updating XML. In *Proc. of SIGMOD*, pages 413–424, 2001.

[97] D. Theodoratos and T. Sellis. Data warehouse configuration. In *Proc. of VLDB*, pages 126–135, Athens, Greece, 1997.

[98] A. Tomasic, H. Garcia-Molina, and K. Shoens. Incremental updates of inverted lists for text document retrieval. In *SIGMOD '94*, pages 289–300, May 1994.

[99] O. G. Tsatalos, M. H. Solomon, and Y. E. Ioannidis. The GMAP: A versatile tool for physical data independence. *VLDB Journal*, 5(2):101–118, 1996.

[100] J. D. Ullman. Information integration using logical views. In *Proc. of ICDT*, pages 19–40, Delphi, Greece, 1997.

[101] T. Urhan and M. J. Franklin. Xjoin: A reactively-scheduled pipelined join operator. *IEEE Data Engineering Bulletin*, 23(2):27–33, 2000.

[102] G. Voelker, H. Jamrozik, M. Vernon, E. Lazowska, and H. Levy. Managing server load in global memory systems. In *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modelling of Computer Systems*, June 1997.

[103] G. Voelker, T. Kimbrel, E. Anderson, M. Feeley, J. Chase, A. Karlin, and H. Levy. Implementing cooperative prefetching and caching in a globally-managed memory system. In *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modelling of Computer Systems*, June 1998.

[104] A. Whitaker and D. Wetherall. Forwarding without loops in icarus. In *Submitted to IEEE INFOCOMM 2002*, New York, NY, USA, June 2002.

[105] A. Wolman, G. Voelker, N. Sharma, N. Cardwell, A. Karlin, and H. Levy. The scale and performance of cooperative web proxy caching. In *Proceedings of the 17th ACM Symposium on Operating Systems Principles*, December 1999.

[106] A. Wolman, G. Voelker, N. Sharma, T. Landry, M. Brown, D. Pinnell, N. Cardwell, A. Karlin, and H. Levy. Measurement and analysis of group-based web object sharing and caching. In *Proceedings of the 2nd USENIX Symposium on Internet Technologies and Systems*, October 1999.

[107] http://w3c.org/XML/, 1998.

[108] H. Z. Yang and P. A. Larson. Query transformation for PSJ-queries. In *Proc. of VLDB*, pages 245–254, Brighton, England, 1987.

[109] J. Yang, K. Karlapalem, and Q. Li. Algorithms for materialized view design in data warehousing environment. In *Proc. of VLDB*, pages 136–145, Athens, Greece, 1997.

[110] M. Zaharioudakis, R. Cochrane, G. Lapis, H. Pirahesh, and M. Urata. Answering complex SQL queries using automatic summary tables. In *Proc. of SIGMOD*, pages 105–116, 2000.