

What Can Peer-to-Peer Do for Databases, and Vice Versa?

Steven Gribble Alon Halevy Zachary Ives Maya Rodrig Dan Suciu
{gribble, alon, zives, rodrig, sucIU}@cs.washington.edu

University of Washington
Seattle, WA USA

Abstract

The latest focus of the Internet community has centered around peer-to-peer systems like Napster, Gnutella, and Freenet. The grand vision — a decentralized community of machines pooling their resources to benefit everyone — is very compelling for many reasons: scalability, robustness, lack of need for administration, and even anonymity and resistance to censorship. Most existing peer-to-peer systems have focused on specific application domains (e.g. music files) or on providing file-system-like capabilities; these systems ignore the semantics of data. An important question for the database community is whether we can do better — how does peer-to-peer relate to our field, and what we can learn from and contribute to it? We tackle these questions, identify a number of potential research ideas in the overlap between data management and peer-to-peer systems, and present some preliminary fundamental results concerning these problems.

1 Introduction

A long-standing tenet of distributed systems is that the strength of a distributed system can grow as more hosts participate in it. Each participant has the potential of contributing data and computing resources (such as otherwise unused CPU cycles and storage) to the overall system. If this occurs, then the wealth of the community can scale with the number of participants. A peer-to-peer (P2P) distributed system is one in which participants rely on each other for service, rather than solely relying on dedicated and often centralized infrastructure. Instead of strictly decomposing the system into clients (which consume services) and servers (which provide them), peers in the system can elect to provide services as well as consume them. The membership of a P2P system is unpredictable, to some degree: service is provided by the peers that happen to be participating in it at any given time.

Many examples of P2P systems have emerged recently, most of which are wide-area, large-scale systems that provide content sharing [10], storage services [8], or distributed “grid” computation [3, 9]. Smaller-scale P2P systems also exist, such as federated, serverless file systems [2, 1] and collaborative workgroup tools [5]. The success of these systems has been mixed; some, such as Napster, have enjoyed enormous popularity and have continued to perform well at scale. Others, including Gnutella, have failed to attract a large community, possibly due to the combination of weak application semantics and technical flaws that limit its scaling.

Perhaps the most exciting possibility of peer-to-peer computing is that the desirable properties of the system can become *amplified* as new peers join: because of its decentralization, it is possible that the robustness, availability, and performance of the system grows with the number of peers. A more subtle possibility is that the richness and diversity of the system can similarly scale, since new peers can introduce specialized data or resources that the system was previously lacking. Decentralization also helps eliminate proprietary interests in the system’s infrastructure; instead of trust being placed in dedicated servers, trust is diffused over all participants in the system. The need for administration is diminished, since there is no dedicated infrastructure to manage. By routing requests through many peers and replicating content across many peers, the system might be able to hide the identity of content publishers and consumers, making it resilient against censorship.

The Challenges of Peer-to-Peer, and What Data Management Offers

Although the vision of P2P systems is grand, the technical challenges associated with them are immense, and as a result the realization of the vision has been elusive. Because the membership in the system is

ad-hoc and dynamic, it is very difficult to predict or reason about the location and quality of the system's resources. For example, the placement of data in content-sharing systems is often naïve: data placement is largely demand driven, with little regard given to network bandwidth, load, or historical trustworthiness of the peer on which the data is placed. Because the system is decentralized, any optimizations such as data placement must be done in a completely distributed manner; the system cannot necessarily presume the existence of a single oracle that coordinates the activity of all of the systems' peers. Furthermore, the dynamic nature of the system may impose fundamental limitations on its data consistency and availability: if the rate at which data changes in the system is high, then the overhead of maintaining globally accessible indexes may become prohibitive as the number of peers in the system grows.

Because P2P systems designers have to a large extent failed to overcome these challenges, the semantics provided by these systems is typically quite weak. In most content sharing systems, only popular content is readily accessible — yet content popularity seems to be driven by Zipf distributions, in which a large fraction of requests are directed to *unpopular* content. Similarly, current content sharing systems ignore problems such as updates to content, and they typically only support retrieval of objects by name.

At first glance, many of the challenges in designing P2P systems seem to fall clearly under the banner of the distributed systems community. However, upon closer examination, the fundamental problem in most P2P systems is the placement and retrieval of *data*. Not only does this make P2P a topic worthy of the database community's interest, but in fact data management techniques can be of great relevance to the P2P field. Indeed, current P2P systems focus strictly on handling semantics-free, large-granularity requests for objects by identifier (typically a name), which both limits their utility and restricts the techniques that might be employed to distribute the data. These current content sharing systems are largely limited to applications in which objects are large, opaque, and atomic, and whose content well-described by their name; for instance, today's P2P systems would be highly ineffective at content-based retrieval of text files or at fetching only the abstracts from a set of L^AT_EX documents. Moreover, they are limited to caching, prefetching, or pushing of content at the object level, and know nothing of overlap between objects.

These limitations arise because the P2P world is lacking in the areas of *semantics*, *data transformation*, and *data relationships*, yet these are some of the core strengths of the data management community. Queries, views, and integrity constraints can be used to express relationships between existing objects and to define new objects in terms of old ones. Complex queries can be posed across multiple sources, and the results of one query can be materialized and used to answer other queries.

Building upon the techniques listed above, the data management community can provide the expertise necessary to develop better solutions to the *data placement problem* at the heart of any P2P system design: data must be placed in strategic locations and then used to improve query performance. Our community will benefit from the results, as new query processing systems can leverage the increased scalability, reliability, and performance of a successful P2P architecture.

We now proceed to define the data placement problem in more detail and identify the impact of P2P design dimensions on this problem. We conclude this paper with a description of the Piazza system, which we are building at the University of Washington to investigate data placement schemes for peer-to-peer domains with dynamic membership, data, and workloads.

2 Data Placement for Peer-to-Peer

We define the data placement problem for a P2P system as follows. Assume we are given a set of cooperating nodes connected by a network (typically, but not necessarily, the Internet) that has limited bandwidth on each link. Nodes know about and exchange data with a collection of participating peers, and they may serve any of four roles. The first of these is a *data origin*, which provides original content to the system and is the authoritative source of that data. As a *storage provider*, a peer stores materialized views (consuming available disk resources), and as a *query evaluator*, it uses a portion of its CPU resources to evaluate the set of queries forming its *workload*. As *query initiators*, peers act as clients in the system and pose new queries. (A node may initiate new queries on behalf of a query it is attempting to evaluate.)

The overall cost of answering a query 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. The *data placement problem* is to distribute data and work so the full query workload is answered with lowest cost under the existing resource and bandwidth constraints.

While a cursory glance at the data placement problem suggests many similarities with multi-query optimization in a distributed database, there are substantial and fundamental differences. For example, in the general case, a P2P system has no centralized schema and no central administration. Moreover, as we shall see in the next section, the data placement problem can come in many forms, depending on the design of the underlying P2P system.

Peer-to-Peer Design Choices Affecting Data Placement

While the globally optimal peer-to-peer concept is conceptually simple to define for an ideal environment, in practice any P2P system will have certain limitations. These compromises are due to factors such as constrained bandwidth and resources, message propagation delays, and so on. Some important dimensions that affect the data placement problem include:

Scope of decision-making: A major factor is the scale at which query processing and view materialization decisions are made. At one extreme, all queries in the entire system are optimized together, using complete knowledge of the available materialized views, resources, and network bandwidth constraints — this poses all of the challenges of multi-query optimization, plus a number of additional difficulties. In particular, work must be distributed globally across many peers, and decisions must be made about when and where to materialize results for future use. At the other end of the spectrum, every decision is made on a single-node, single-query basis — this is the familiar problem of query optimization for distributed data. Clearly, a good query optimization and data placement strategy will be much more beneficial to the global system than the local one; yet decisions are likely to be much more expensive to make on the global scale, so any real system will likely be forced to work within a smaller scope.

Extent of knowledge sharing: Related to the above problem is the question of how much knowledge is available to the system during its query optimization process. In particular, the first step in choosing a query evaluation strategy is likely to be identifying which nodes have materialized views that can speed query processing. A simple technique would be to use a centralized catalog of all available views and their locations, analogous to the central directory used by Napster. Yet this model introduces a single point of failure and a potential scalability bottleneck. Alternatively, one may attempt to replicate the complete catalog at all peers, but this requires too much update traffic to be feasible. A third solution might be to construct a hierarchical organization, much like those in DNS or LDAP: a peer first contacts a “known” site holding some fragment of the global catalog, and if the requested data cannot be resolved there, the request is forwarded to a peer higher up in the hierarchy. We discuss a fourth technique when we present the Piazza system later in this paper. A basic challenge in any such scheme is to achieve a reasonable degree of consistency as the number of peers in the system grows, as the placement of data changes, and as data is updated.

Heterogeneity of information sources: Data may originate at a few authoritative sources, or alternatively, every participant might be allowed (or expected) to contribute data to the community. The level of heterogeneity of the data influences the degree to which a system can ensure uniform, global semantics for the data. A P2P system might impose a single schema on all participants to enforce uniform, global semantics, but for some applications this will be too restrictive. Alternatively, a limited number of data sources and schemas may be allowed, so traditional schema and data integration techniques will likely apply (with the restriction that there is no central authority). The case of fully heterogeneous data makes global semantic integration extremely challenging.

Dynamicity of participants: Some P2P systems, such as [8], assume a fixed set of nodes in the system. However, one of the greatest potential strengths of P2P systems is when they eschew reliance on dedicated infrastructure and allow peers to leave the system at will. Even under these conditions, participants typically have broadly varying availability characteristics. Some peers are akin to servers: their membership in the system stays largely static. Others have much more dynamic membership, joining and leaving the system at will. In a configuration where original data is distributed uniformly across the network, including on nodes that frequently disappear, it may become impossible to reliably access certain items. At the other extreme, if all data is placed or cached only on the set of static “servers,” the system will have greatly reduced flexibility and performance (this configuration is equivalent to yesterday’s web, prior to proxy caches and content distributors such as Akamai). An intermediate approach places all original content on the consistently

available nodes to provide availability, but replicates or caches data at the dynamic peers.

Data granularity: The data within a P2P system can be accessible at many degrees of granularity. At the *atomic granularity level*, data consists of a collection of indivisible objects, *e.g.*, one complete MP3 file. For data placement at this level, we have to either place an entire object at a peer, or not at all; this is the semantics currently supported by today's P2P systems. At the *hierarchical granularity level*, sets of objects can be grouped into larger objects, thus forming hierarchies. For example, multiple MP3 files may be grouped into an album, and albums into collections; for the data placement problem at this level, we can now either place a single file or the entire album at a peer. Finally, with *value based granularity*, data objects are aggregated from many atomic (or hierarchical) values. For example, tuples in a relation consist of values. The data placement problem has now a new dimension: data can be restructured and integrated before being placed.

Degrees of replication: Data items can be replicated at will, only sparingly, or not at all. Obviously, a large degree of replication improves query time and efficiency, but makes updates much harder, and also increases the retrieval complexity (as we will discuss later). Maintaining consistency over replicated objects is a well known difficult database problem [4]. A typical solution, which is quite acceptable for P2P, is to have each object be owned by a single master, which is solely responsible for its freshness. Still, propagating the updates to all peers holding copies of that object may be difficult since we need to find all those copies.

Freshness and update consistency: There are many possible ways of propagating updates from the data origins to intermediate nodes that have materialized views of this data. Some possible solutions would be invalidation messages pushed by the server, or client-initiated validation messages; however, both of these incur overhead that limits scalability. Another approach is a timeout/expiration-based protocol, as employed by DNS and web caches. This approach has a lower overhead, at the cost of providing much looser guarantees about freshness and consistency. Still, this is a much stronger guarantee than what P2P currently gives us, which is no guarantee at all.

It should not be surprising that the data placement problem is intractable at the extreme points of each of the dimensions listed above. In fact, we can show that even the simplest form of the problem is NP-complete.

Complexity of the Data Placement Problem

The simplified form of the problem can be defined as follows. Assume a model in which each peer node n_j has associated storage B_j and query workload Q_j . Every network link imposes a cost per unit of data transferred.

Data model and query capabilities: Assume a data model in which every object o_a is atomic, consumes s_a units of space, and is identified by object identifier oid_a . Our query language supports one form of request, *object queries*: given object identifier oid_a , return object o_a .

The appropriate choice of data to maintain at a particular node is highly dependent on the set of queries we expect to be given. We model our expected queries by a query workload, which is a set of queries $\mathcal{Q} = Q_1, \dots, Q_m$, where each query Q_i has an associated non-negative weight, w_i . The weight describes the relative frequency of Q_i within the workload. We require that the weights sum up to 1 ($\sum_{1 \leq i \leq m} w_i = 1$).

Data placing: data placement is the assignment of a set of objects to be stored at each peer in the network. Data may be replicated — objects may be stored at several nodes. A data placement may be described extensionally (*i.e.*, a set of oid's at each peer), or by a set of views for each peer, whose evaluation would return the objects stored at the peer.

The *cost* of data placing is more subtle to define and is context-specific; we define it here for our simple case of object queries. Given an object query $q(o_a)$ at a node n_j , the cost is zero if o_a is stored at n_j . Otherwise, we assume that we know which is the cheapest node n_{cheap} to get the object from, and the cost is the sum of the costs on the edges between n_j and n_{cheap} . The cost of the workload at node n_j is the weighted sum of the costs of its constituent queries. The cost of a data placement is the sum of the costs of the workloads of the peers in the network.

Definition 2.1 (Static data placement and lookup problem) Given a graph G describing a network of peers, the static data placement and lookup problem is to perform data placement with optimal cost, where queries are zero-cost object lookups. \square

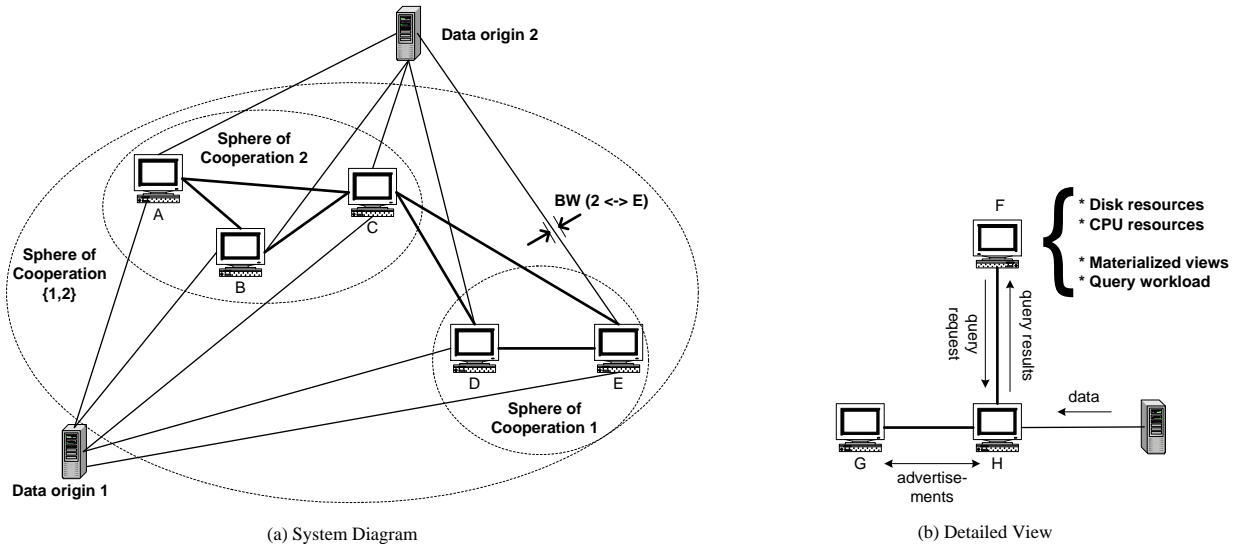


Figure 1: Architecture of the Piazza system. Data origins serve original content; peer nodes (A-E) cooperate to store materialized views and answer queries, but have limited disk and CPU resources. Nodes are connected by bandwidth-constrained links, and advertise the presence of materialized views to propagate knowledge. Finally, nodes belong to *spheres of cooperation* with which they share resources; these spheres may be nested within successively larger spheres.

We observe that special cases and slight variations on this problem occur in several data management contexts. A very simple version of this problem was considered in the context of data placement in distributed databases (see [7] for a survey). View-selection for data warehouses is a very specific instance of the data-placement problem, where the network includes only two nodes, the database and the warehouse.

In our initial theoretical investigation of the data placement and lookup problem, we have shown the following result:

Theorem 2.1 *Let G be a graph describing a peer-to-peer network. Then, the static data placement and lookup problem is NP-complete, even if all the queries in the workloads in G are object queries.*

The proof of the theorem is based on a reduction from the vertex-cover problem. The theorem shows that even in a very restricted case, the problem is intractable. Our setting is especially simple because it does not even consider non-trivial queries over the data. It is important to note that the NP-hardness is in the size of the network.

This theorem should not dampen our enthusiasm regarding the data placement problem — quite the contrary. The challenge is to find more specific settings in which to study the problem, where the network and workloads have interesting properties that can be exploited.

A version of the problem that seems especially interesting is the *dynamic data placement problem*, which includes dynamic data, dynamic query workloads, and dynamic peer membership. A solution to this problem is required to build a decentralized, globally distributed P2P query processor. Similar needs arise in the context of data management for ubiquitous computing [6]. Here, data is both integrated and accessed from many devices (desktops, laptops, PDAs, cell phones), and each of these devices has a local store but can also retrieve data at different rates from various points on the network.

3 Exploring Peer-to-Peer with the Piazza System

We conclude this paper with a description of our initial work in building the Piazza system (Figure 1), which focuses on the dynamic data placement problem mentioned above. Our architecture aims for scalability even with large numbers of nodes and moderately frequent updates. We assume a model in which data origins are distinct from the peers in the system, and are the only sources of updates to their data. All peer nodes

belong to *spheres of cooperation*, in which they pool their resources and make cooperative decisions. Each sphere of cooperation may in turn be nested within a successively larger sphere, with which it cooperates to a lesser extent. These spheres of cooperation will often occur within particular administrative boundaries (e.g. within a corporation or local ISP), and in many ways resemble a cooperative cache or nodes sharing a proxy on the web. Given this configuration, we focus on the following aspects of the data placement problem:

Query optimization exploiting commonalities and available data At the heart of our problem lies a variation of traditional multi-query optimization. Ideally, the Piazza system will take the current query workload, find commonalities among the queries, exploit materialized views whenever cost-effective, distribute work under resource and bandwidth constraints, and determine whether certain results should be materialized for future use (while considering the likelihood of updates to the data). For scalability reasons, we make these decisions at the level of a sphere of cooperation rather than on a global basis. In order to perform this optimization, Piazza must address two important sub-problems:

- **Propagating information about materialized views:** When a query is posed, the first step is to consider whether it can be answered using the data at “nearby” storage providers, and to evaluate the costs of doing so. This requires the query initiator to be aware of existing materialized views and properties such as location and data freshness. To facilitate this, we propagate information about materialized views using techniques derived from routing protocols [11]. In particular, a node advertises its materialized views to its neighbors. Each node consolidates the advertisements it receives and propagates them to its neighbors. Under constrained resources, any node can arbitrarily drop advertisements without jeopardizing system correctness — a query can always be posed in terms of the data origins. This routing protocol avoids the scalability problems caused by broadcasting every view materialization and those caused by broadcasting every query request.
- **Consolidating query evaluation and data placement:** A node may pose a query that cannot be evaluated with the data available from known peers. In this case, the data must be retrieved directly from the data origins. However, at any given point, there may be many similar un-evaluable queries within the same sphere of cooperation, and the sphere should choose an optimal strategy for evaluating all of them. Therefore, all un-evaluable queries are broadcast within the cluster; the cluster identifies commonalities among this query set, then assigns roles (evaluation of a query or subquery and/or materialization of results) to specific nodes based on the best overall expected cost.

Guaranteeing data freshness Since we wish to support dynamic data as well as dynamic workloads, Piazza must refresh materialized views when original data is updated. For the scalability reasons discussed in Section 2, we have elected to use expiration times on our data items, rather than a coherence protocol. This reduces network traffic and provides better guarantees than current P2P systems, but does not achieve the strong semantics of traditional databases.

Solutions to the problems listed above should be generally applicable not only within our system, but to any peer-to-peer-like distributed system that supports dynamic data and dynamic workloads. We believe the strategies we have adopted hold promise, and hope to further develop and experimentally validate them as we build the Piazza system. Our goal — a scalable, reliable, performant distributed query answering system leveraging both P2P ideas and data management techniques — seems within reach.

References

- [1] T. E. Anderson, M. Dahlin, J. M. Neeffe, D. A. Patterson, D. S. Roselli, and R. Wang. Serverless network file systems. In *SOSP 1995*, volume 29(5), pages 109–126, December 1995.
- [2] W. J. Bolosky, J. R. Douceur, D. Ely, and M. Theimer. Feasibility of a serverless distributed file system deployed on an existing set of desktop pcs. In *Proceedings of the International Conference on Measurement and Modeling of Computer Systems, 2000*, pages 34–43, June 2000.
- [3] How Entropia works. World-wide web: www.entropia.com/how.asp, 2000.
- [4] J. Gray, P. Helland, P. E. O’Neil, and D. Shasha. The dangers of replication and a solution. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Montreal, Quebec, Canada, June 4-6, 1996*, pages 173–182, 1996.

- [5] Groove. World-wide web: www.groove.net/, 2001.
- [6] Z. G. Ives, A. Y. Levy, J. Madhavan, R. Pottinger, S. Saroiu, I. Tatarinov, S. Betzler, Q. Chen, E. Jaslikowska, J. Su, and W. T. T. Yeung. Self-organizing data sharing communities with sagres. In *SIGMOD '00*, volume 29, page 582, 2000.
- [7] D. Kossman. The state of the art in distributed query processing. *ACM Computing Surveys*, September 2000.
- [8] J. Kubiawicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. Oceanstore: an architecture for global-scale persistent storage. In *ASPLoS 2000*, pages 190–201, November 2000.
- [9] About LEGION – the Grid OS. World-wide web: www.appliedmeta.com/legion/about.html, 2000.
- [10] Napster. World-wide web: www.napster.com, 2001.
- [11] A. S. Tanenbaum. *Computer Networks*. Prentice Hall PTR, 3rd edition, 1996.