Data Management in Large-scale P2P Systems[1]

Patrick Valduriez, Esther Pacitti

Atlas group, INRIA and LINA, University of Nantes – France
Patrick.Valduriez@inria.fr
Esther.Pacitti@lina.univ-nantes.fr

Abstract. Peer-to-peer (P2P) computing offers new opportunities for building highly distributed data systems. Unlike client-server computing, P2P can operate without central coordination and offer important advantages such as a very dynamic environment where peers can join and leave the network at any time; direct and fast communication between peers, and scale up to large number of peers. However, most deployed P2P systems have severe limitations: file-level sharing, read-only access, simple search and poor scaling. In this paper, we discuss the issues of providing high-level data management services (schema, queries, replication, availability, etc.) in a P2P system. This implies revisiting distributed database technology in major ways. We illustrate how we address some of these issues in the APPA data management system under development in the Atlas group.

1 Introduction

Data management in distributed systems has been traditionally achieved by distributed database systems [19] which enable users to transparently access and update several databases in a network using a high-level query language (e.g. SQL). Transparency is achieved through a global schema which hides the local databases' heterogeneity. In its simplest form, a *distributed database system* is a centralized server that supports a global schema and implements distributed database techniques (query processing, transaction management, consistency management, etc.). This approach has proved effective for applications that can benefit from centralized control and full-fledge database capabilities, e.g. information systems. However, it cannot scale up to more than tens of databases. Data integration systems [30] extend the distributed database approach to access data sources on the Internet with a simpler query language in read-only mode. Parallel database systems [31] also extend the distributed database approach to improve performance (transaction throughput or query response time) by exploiting database partitioning using a multiprocessor or cluster system. Although data integration systems and parallel database systems can scale up to hundreds of data sources or database partitions, they still rely on a centralized global schema and strong assumptions about the network.

[1] Work partially funded by project MDP2P (Massive Data in P2P) [15] of the ACI "Masses de Données" of the French ministry of research.

In contrast, peer-to-peer (P2P) systems adopt a completely decentralized approach to data sharing. By distributing data storage and processing across autonomous peers in the network, they can scale without the need for powerful servers. Popular examples of P2P systems such as Gnutella [8] and Kaaza [13] have millions of users sharing petabytes of data over the Internet. Although very useful, these systems are quite simple (e.g. file sharing), support limited functions (e.g. keyword search) and use simple techniques (e.g. resource location by flooding) which have performance problems. To deal with the dynamic behavior of peers that can join and leave the system at any time, they rely on the fact that popular data get massively duplicated.

Initial research on P2P systems has focused on improving the performance of query routing in the unstructured systems which rely on flooding. This work led to structured solutions based on distributed hash tables (DHT), e.g. CAN [24] and CHORD [27], or hybrid solutions with super-peers that index subsets of peers [32]. Although these designs can give better performance guarantees, more research is needed to understand their trade-offs between fault-tolerance, scalability, self-organization, etc.

Recently, other work has concentrated on supporting advanced applications which must deal with semantically rich data (e.g., XML documents, relational tables, etc.) using a high-level SQL-like query language, e.g. ActiveXML [2], Edutella [17], Piazza [29], PIER [9]. As a potential example of advanced application that can benefit from a P2P system, consider the cooperation of scientists who are willing to share their private data (and programs) for the duration of a given experiment. For instance, medical doctors in a hospital may want to share some patient data for an epidemiological study. Medical doctors may have their own, independent data descriptions for patients and should be able to ask queries like “age and last weight of the male patients diagnosed with disease X between day1 and day2” over their own descriptions.

Such data management in P2P systems is quite challenging because of the scale of the network and the autonomy and unreliable nature of peers. Most techniques designed for distributed database systems which statically exploit schema and network information no longer apply. New techniques are needed which should be decentralized, dynamic and self-adaptive.

In this paper, we discuss the main issues related to data management in large-scale P2P systems. we first recall the main principles behind data management in distributed systems and the basic techniques needed for supporting advanced functionality (schema management, access control, query processing, transaction management, consistency management, reliability and replication). Then we review P2P systems and compare the various architectures along several dimensions important for data management. we also discuss the state-of-the-art on data management in P2P systems. Finally, we illustrate how some of these issues (schema management, replication and query processing) are addressed in the context of APPA

(*Atlas Peer-to-Peer Architecture*), a P2P data management system which we are building.

The rest of the paper is organized as follows. Section 2 recalls the main capabilities of distributed database systems. Section 3 discusses and compares P2P systems from the perspective of data sharing. Section 4 discusses data management in P2P systems. Section 5 introduces data management in the APPA system. Section 6 concludes.

2 Data Management in Distributed Systems

The fundamental principle behind data management is *data independence*, which enables applications and users to share data at a high conceptual level while ignoring implementation details. This principle has been achieved by *database systems* which provide advanced capabilities such as schema management, high-level query languages, access control, automatic query processing and optimization, transactions, data structures for supporting complex objects, etc.

A *distributed database* is a collection of multiple, logically interrelated databases distributed over a computer network. A *distributed database system* is defined as the software system that permits the management of the distributed database and makes the distribution *transparent* to the users [19]. Distribution transparency extends the principle of data independence so that distribution is not visible to users.

These definitions assume that each site logically consists of a single, independent computer. Therefore, each site has the capability to execute applications on its own. The sites are interconnected by a computer network with loose connection between sites which operate independently. Applications can then issue queries and transactions to the distributed database system which transforms them into local queries and local transactions (see Figure 1) and integrates the results. The distributed database system can run at any site s, not necessarily distinct from the data (i.e. it can be site 1 or 2 in Figure 1).

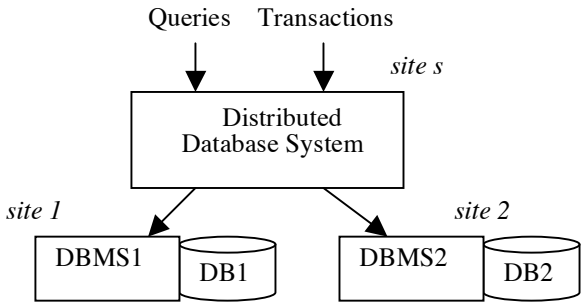

Figure 1. A distributed database system with two data sites

The database is physically distributed across the data sites by fragmenting and replicating the data. Given a relational database schema, for instance, fragmentation subdivides each relation into partitions based on some function applied to some

tuples' attributes. Based on the user access patterns, each of the fragments may also be replicated to improve locality of reference (and thus performance) and availability.

The functions provided by a distributed database system could be those of a database system (schema management, access control, query processing, transaction support, etc). But since they must deal with distribution, they are more complex to implement. Therefore, many systems support only a subset of these functions.

When the data and the databases already exist, one is faced with the problem of providing integrated access to heterogeneous data. This process is known as *data integration*: it consists in defining a *global schema* over the existing data and *mappings* between the global schema and the local database schemas. Data integration systems have received several names such as federated database systems, multidatabase systems and, more recently, mediators systems. In the context of the Web, mediator systems [30] allow general access to autonomous data sources (such as files, databases, documents, etc.) in read only mode. Thus, they typically do not support all database functions such as transactions and replication.

When the architectural assumption of each site being a (logically) single, independent computer is relaxed, one gets a *parallel database system* [31], i.e. a database system implemented on a tightly-coupled multiprocessor or a cluster. The main difference with a distributed database system is that there is a single operating system which eases implementation and the network is typically faster and more reliable. The objective of parallel database systems is high-performance and high-availability. High-performance (i.e. improving transaction throughput or query response time) is obtained by exploiting data partitioning and query parallelism while high-availability is obtained by exploiting replication.

The distributed database approach has proved effective for applications that can benefit from centralized control and full-fledge database capabilities, e.g. information systems. For administrative reasons, the distributed database system typically runs on a separate server and this reduces scale up to tens of databases. Data integration systems achieve better scale up to hundreds of data sources by restricting functionality (i.e. read-only querying). Parallel database systems can also scale up to large configurations with hundreds of processing nodes by relying on a single operating system. However, both data integration systems and parallel database rely on a centralized global schema.

3 P2P Systems

Peer-to-peer (P2P) systems adopt a completely decentralized approach to resource management. By distributing data storage, processing and bandwidth across all peers in the network, they can scale without the need for powerful servers. P2P systems have been successfully used for sharing computation, e.g. SETI@home [25], communication [11] or data, e.g. Gnutella [8] and Kaaza [13]. The success of P2P systems is due to many potential benefits: scale-up to very large numbers of peers, dynamic self-organization, load balancing, parallel processing, and fault-tolerance through massive replication. Furthermore, they can be very useful in the context of mobile or pervasive computing. However, existing systems are limited to simple

applications (e.g. file sharing), support limited functions (e.g. keyword search) and use simple techniques which have performance problems. Much active research is currently on-going to address the challenges posed by P2P systems in terms of high-level data sharing services, efficiency and security. When considering data management, the main requirements of a P2P system are [7]:

- **Autonomy**: an autonomous peer should be able to join or leave the system at any time without restriction. It should also be able to control the data it stores and which other peers can store its data, e.g. some other trusted peers
- **Query expressiveness**: the query language should allow the user to describe the desired data at the appropriate level of detail. The simplest form of query is key look-up which is only appropriate for finding files. Keyword search with ranking of results is appropriate for searching documents. But for more structured data, an SQL-like query language is necessary.
- **Efficiency**: the efficient use of the P2P system resources (bandwidth, computing power, storage) should result in lower cost and thus higher throughput of queries, i.e. a higher number of queries can be processed by the P2P system in a given time.
- **Quality of service**: refers to the user-perceived efficiency of the system, e.g. completeness of query results, data consistency, data availability, query response time, etc.
- **Fault-tolerance**: efficiency and quality of services should be provided despite the occurrence of peers' failures. Given the dynamic nature of peers which may leave or fail at any time, the only solution is to rely on data replication.
- **Security**: the open nature of a P2P system makes security a major challenge since one cannot rely on trusted servers. Wrt. data management, the main security issue is access control which includes enforcing intellectual property rights on data contents.

There are many different architectures and network topologies that are possible for P2P systems. Depending on the architecture, the above requirements are more or less difficult to achieve. For simplicity, we consider three main classes: unstructured, structured and super-peer. Unstructured and structured systems are also called "pure" P2P while super-peer systems are qualified as "hybrid". Pure P2P systems consider all peers equal with no peer providing special functionality.

In *unstructured systems*, the simplest ones, each peer can directly communicate with its neighbors. Figure 2 illustrates a simple unstructured system, each peer supporting the same *p2p* software. Autonomy is high since a peer only needs to know its neighbors to log in. Searching for information is simple: it proceeds by flooding the network with queries, each peer processing and redirecting the incoming queries to its neighbors. There is no restriction on the expressiveness of the query language which could be high. Such query routing based on flooding is general but does not scale up to large numbers of peers. Also, the incompleteness of the results can be high since some peers containing relevant data or their neighbors may not be reached because they are either off-line. However, since all peers are equal and able to replicate data, fault-tolerance is very high.

Initial research on P2P systems has focused on improving the performance of unstructured systems and led to *structured systems* based on distributed hash tables

(DHT), e.g. CAN [24] and CHORD [27]. A DHT system provides a hash table interface with primitives put(key,value) and get(key), where key is typically a file name and each peer is responsible for storing the values (file contents) corresponding to a certain range of keys. There is an overlay routing scheme that delivers requests for a given key to the peer responsible for that key. This allows one to find a peer responsible for a key in *O(log n)*, where *n* is the number of peers in the network. Because a peer is responsible for storing the values corresponding to its range of keys, autonomy is limited. Furthermore, DHT queries are typically limited to exact match keyword search. Active research is on-going to extend the capabilities of DHT systems to deal with more general queries such as range queries and join queries [9].

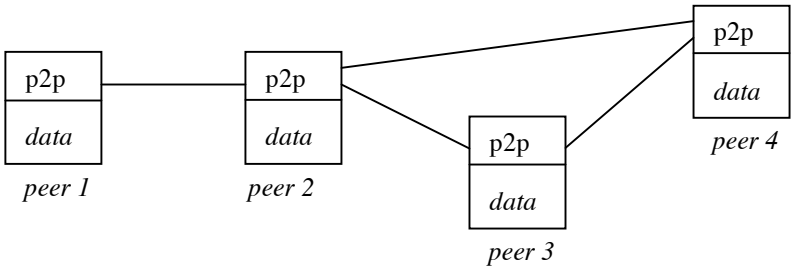

Figure 2. P2P unstructured network

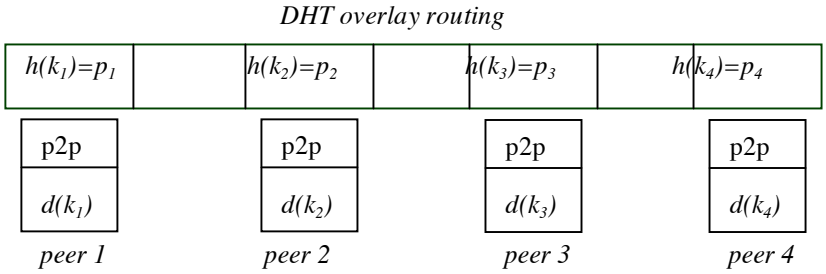

Figure 3. DHT network

Super-peer P2P systems are hybrid between pure systems and client-server systems. Unlike pure systems, peers are not all equal. Some peers, the super-peers, act as dedicated servers for some other peers and can perform complex functions such as indexing, query processing, access control and meta-data management. Using only one super-peer reduces to client-server with all the problems associated with a single server. For instance, Napster [16] which became famous for exchanging pirated music files used a central super-peer which made it easier to shut it down. Super-peers can also be organized in a P2P fashion and communicate with one another in sophisticated ways. Thus, unlike client-server systems, global information is not necessarily centralized and can be partitioned or replicated across all super-peers. Figure 4 illustrates a super-peer network that shows the different communication paths peer2super-peer (p2sp) and super-peer2super-peer (sp2sp). The main advantage of

super-peer is efficiency and quality of service. A requesting peer simply sends the request, which can be expressed in a high-level language, to its responsible super-peer which can then find the relevant peers either directly through its index or indirectly using its neighbor super-peers. Access control can also be better enforced since directory and security information can be maintained at the super-peers. However, autonomy is restricted since peers cannot log in freely to any super-peer. Fault-tolerance is typically low since super-peers are single points of failure for their sub-peers.

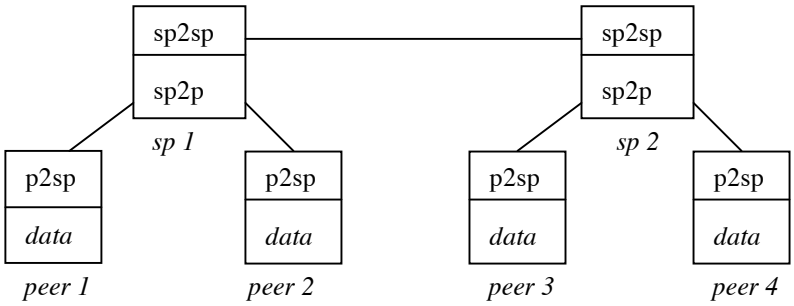

Figure 4. Super-peer network

Table 1 summarizes how the requirements for data management are possibly attained by the three main classes of P2P systems. This is a rough comparison to understand the respective merits of each class. For instance, "high" means it can be high. Obviously, there is room for improvement in each class of systems. For instance, fault-tolerance can be made higher in super-peer by relying on replication and fail-over techniques.

Requirements	**Unstructured**	**Structured**	**Super-peer**
Autonomy	high	low	average
Query expressiveness	"high"	low	"high"
Efficiency	low	high	high
QoS	low	high	high
Fault-tolerance	high	high	low
Security	low	low	high

Table 1. Comparison of P2P systems

4 Data Management in P2P Systems

Advanced P2P applications must deal with semantically rich data (e.g., XML documents, relational tables, etc.). To address these applications, we need functions similar to those of distributed database systems. In particular, users should be able to

use a high-level query language to describe the desired data. But the characteristics of P2P systems create new issues. First, the dynamic and autonomous nature of peers makes it hard to give guarantees about result completeness and makes static query optimization impossible. Second, data management techniques need to scale up to high numbers of peers. Third, the lack of centralized authority makes global schema management and access control difficult. Finally, even when using replication, it is hard to achieve fault-tolerance and availability in the presence of unstable peers. Most of the work on sharing semantically rich data in P2P systems has focused on schema management, and query processing and optimization. However, there has been very little work on replication, transactions and access control.

Schema management and query processing are generally addressed together for a given class of P2P system. Peers should be able to express high-level queries over their own schema without relying on a centralized global schema. Thus the main problem is to support decentralized schema mapping so that a query on one peer's schema can be reformulated in a query on another peer's schema. In PeerDB [18], assuming an unstructured network, schema mapping is done on the fly during query processing using information retrieval techniques. Although flexible, this approach limits query expressiveness to keyword search. Furthermore, query routing relies on flooding which can be inefficient. In PIER [9], a DHT network, the focus is on scaling up query processing to very large configurations assuming that de-facto standard schemas exist. However, only exact-match and equijoin queries are supported. In Edutella [17], a hybrid system, RDF-based schema descriptions are provided by super-peers. Thus, SQL-like query processing can be done by super-peers using distributed database techniques. Piazza [29] proposes a more general, network-independent, solution to schema management that supports a graph of pair-wise mappings between heterogeneous schema peers. Algorithms are proposed to reformulate a query in Xquery on a peer's schema into equivalent queries on the other peers' schemas. ActiveXML [2] is a general P2P system based on active XML documents, i.e. XML documents with embedded Web service calls in XQuery. Query processing in ActiveXML relies on a cost model which helps evaluating distributed queries and deciding which data and services to replicate.

Data replication in the presence of updates and transactions remains an open issue. The data sharing P2P systems like Gnutella and Kaaza deal with static, read-only files (e.g. music files) for which update is not an issue. Freenet [6] partially addresses updates which are propagated from the updating peer downward to close peers that are connected. However, peers that are disconnected do not get updated. ActiveXML [2] supports the definition of replicated XML fragments as Web service calls but does not address update propagation. Update is addressed in P-Grid [1], a structured network that supports self-organization. The update algorithm uses rumor spreading to scale and provides probabilistic guarantees for replica consistency. However, it only considers updates at the file level in a mono-master mode, i.e. only one (master) peer can update a file and changes are propagated to other (read-only) replicas.

Advanced applications are likely to need more general replication capabilities such as various levels of replication granularity and multi-master mode, i.e. whereby the same replica may be updated by several (master) peers. For instance, a patient record may be replicated at several medical doctors and updated by any of them during a visit of the patient, e.g. to reflect the patient's new weight. The advantage of multi-

master replication is high-availability and high-performance since replicas can be updated in parallel at different peers. However, conflicting updates of the same data at different peers can introduce replica divergence. Then the main problem is to assure replica consistency. In distributed database systems [19], synchronous replication (e.g. Read-Once-Write-All) which updates all replicas within the same transaction enforces mutual consistency of replicas. However, it does not scale up because it makes use of distributed transactions, typically implemented by 2 phase commit. Preventive replication [22] can yield strong consistency, without the constraints of synchronous replication, and scale up to large configurations. However, it requires support for advanced distributed services and a high speed network with guaranteed maximum time for message reception as is the case in cluster systems. This assumption does not hold for P2P systems. A more practical solution is optimistic replication [20] which allows the independent updating of replicas and divergence until reconciliation. However, existing optimistic replication solutions do not address important properties of P2P systems such as self-organization.

5 Data Management in the APPA system

To illustrate data management in large-scale P2P systems, we introduce the design of the APPA system [3]. The main objectives of APPA are scalability, availability and performance for advanced applications. APPA has a layered service-based architecture. Besides the traditional advantages of using services (encapsulation, reuse, portability, etc.), APPA is network-independent so it can be implemented over different P2P networks (unstructured, DHT, super-peer, etc.). The main reason for this choice is to be able to exploit rapid and continuing progress in P2P networks. Another reason is that it is unlikely that a single P2P network design will be able to address the specific requirements of many different applications. Furthermore, different P2P networks could be combined in order to exploit their relative advantages, e.g. DHT for key-based search and super-peer for more complex searching.

There are three layers of services in APPA: P2P network, basic services and advanced services. The P2P network layer provides network independence with services that are common to all P2P networks : peer id assignment, peer linking and key-based storage and retrieval. The basic services layer provides services for peer management and communication over the network layer:

- **P2P data management:** stores and retrieves P2P data (e.g. meta-data, index data) by key in the P2P network.
- **Peer management:** provides support for peer joining (and rejoining) and for storage, retrieval and removal of peer ids.
- **Peer communication:** enables peers to exchange messages (i.e. service calls) even with disconnected peers using a persistent message queue.
- **Group membership management:** allows peers to join an abstract *group*, become *members* of the group and send and receive membership notifications. This is similar but much weaker than group communication [5].

- **Consensus module:** allows a given set of peers to reach agreement on a common value despite failures.

The advanced services layer provides services for semantically rich data sharing including schema management, replication, query processing, caching, security, etc. using the basic services. To capitalize on Web service standards, the shared data are in XML format (which may be interfaced with many data management systems) and the query language is XQuery. In addition, we assume each peer has data management capabilities (e.g. a DBMS) for managing its local XML data, possibly through a traditional wrapper interface.

The APPA services are organized differently depending on the underlying P2P network. For instance, in the case of a DHT network, the three service layers are completely distributed over all peers. Thus, each peer needs to manage P2P data in addition to its local data. In the case of a super-peer network, super-peers provide P2P network services and basic services while peers provide only the advanced services. APPA is being implemented using the JXTA framework [12] which provides a number of abstractions to P2P networks. In particular, JXTA provides global information sharing and group management on top of unstructured and DHT networks. Furthermore, it allows to organize some peers as super-peers.

To deal with semantically rich data, APPA supports decentralized schema management. Our solution takes advantage of the collaborative nature of the applications we target. We assume that peers that wish to cooperate, e.g. for the duration of an experiment, are likely to agree on a *Common Schema Description* (CSD). Our experience with scientific applications taught us this assumption is realistic [28]. Given a CSD, a peer schema can be specified using views. This is similar to the local-as-view approach in data integration [14] except that, in APPA, queries at a peer are expressed against the views, not the CSD. The peer schemas are stored as P2P data using the key-based storage and retrieval module, where the key is a combination of attribute and relation.

To focus on collaborative applications, we follow the small world assumption [10] which allows us to deal with groups of peers of manageable size. In this context, we can assume that the P2P system is self-organized [4] which yields fast communication between peers of the same group. Our replication model is based on the *lazy multi-master* scheme of distributed databases [21] which we transpose here to P2P. Multi-master replication allows a group of peers to update the same replicas, thus improving availability and performance. However, conflicting updates of the same data at different peers can introduce replica divergence. To solve this problem, we adapt log-based reconciliation [23] to address the properties of P2P systems. The original solution works as follows. Assuming each peer in the group holds a replica r, users locally execute *tentative actions* on r which respect some constraints and, record these actions in a local replication log. Periodically, all the logs are merged together by some peer in a *global log L*. Then the *reconcile* algorithm can be executed by that peer using L in order to produce a *best schedule* (an interleaved order of actions from different peers) which respects all the defined constraints. The best schedule is then applied at each peer to update r, possibly undoing some local tentative actions. This solution assures eventual consistency among replicas: if all users stop submitting actions then mutual consistency is achieved among all peers holding r [25]. However, this centralized solution is not well suited for P2P systems because decisions must be

taken in a distributed way in order to preserve peers' autonomy and eventual consistency must be assured. Furthermore, it does not consider the case of peers joining or leaving a group which may impact the scheduling of actions.

In our solution, we use the P2P data management service (henceforth common storage) to log the tentative actions executed by each peer that updates *r*. The reconcile algorithm works as follows. Whenever a peer *p* updates *r*, the effects of the tentative action is immediately reflected locally and the corresponding tentative action (henceforth action) is logged in the common storage in the *action log* (see Figure 5). Thus, all actions may be eventually seen by the other peers of the group, even those that may be disconnected.

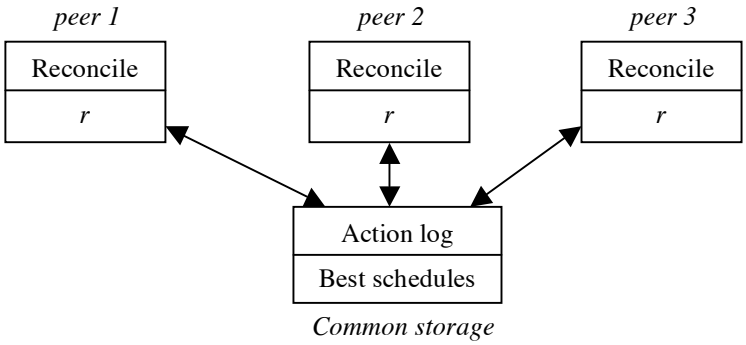

Figure 5. Distributed Reconciliation in APPA

To manage the action log, the peers of the involved group agree (using the consensus module) on a time interval Δt in which the log actions must be grouped together to form a *log unit*. Hence, a log unit *l* holds an unordered set of actions performed by any peer of the group to update *r* during a time interval Δt. Thus, the action log keeps a set of log units that are reconciled on demand (a peer reconciles the log unit he is involved) and, whenever log units do not conflict (for instance updates on different objects), they may be reconciled in parallel. Thus, whenever a peer *p* wishes to reconcile its actions *wrt.* the other peers' actions, it locally executes the reconcile algorithm using a complete log unit (stored in the action log) as input and produces the corresponding *best schedule unit*.

Our replication solution guarantees eventual consistency among replicas [26,25]. It is completely distributed and respects the autonomy of peers. Furthermore, information about replicas is systematically captured and can be exploited for other services, e.g. query processing.

Query processing in APPA deals with schema-based queries and considers data replication. Given a user query *Q* on a peer schema, the objective is to find the minimum set of relevant peers (query matching), route *Q* to these peers (query routing), collect the answers and return a (ranked) list of answers to the user. Since the relevant peers may be disconnected, the returned answers may be incomplete. Depending on the QoS required by the user, we can trade completeness for response time, e.g. by waiting for peers to get connected to get more results.

Query processing proceeds in four main phases: (1) query reformulation, (2) query matching, (3) query optimization and (4) query decomposition and execution. Phases 1, 2 can be done using techniques found in other P2P systems. However, phases 3 and 4 are new because of data replication. The optimization objective is to minimize the amount of redundant work (on replicas) while maximizing load balancing and query parallelism. This is done by statically selecting an optimal set of relevant replicas from which on-line peers are dynamically selected at run time based on load. Query decomposition and execution exploits parallelism using intermediate peers. Since some relevant peers may have only subsets of relations in Q, query decomposition produces a number of subqueries (not necessarily different), one for each peer, together with a composition query to integrate, e.g. through join and union operations, the intermediate results [30]. Finally, Q is sent to each peer which (if connected) reformulates it on its local schema (using the peer mappings), executes it and sends back the results to the sending peer which integrates the results. Result composition can also exploit parallelism using intermediate peers. For instance, let us consider relations r_1 and r_2 defined over CSD r and relations s_1 and s_2 defined over CSD r, each stored at a different peer, and the query *select * from r, s where r.a=s.a and r.b=2 and s.c=5* issued a peer q. A parallel execution strategy for Q is shown in Figure 6.

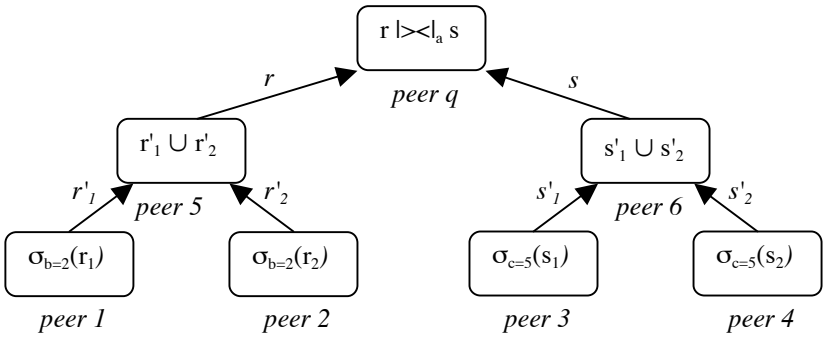

Figure 6. Example of parallel execution using intermediate peers

This strategy exhibits independent parallelism between peers 1-4 (the select (σ) operations can all be done in parallel) and peers 5-6 (the union operations can be done in parallel). It can also yield pipelined parallelism. For instance, if the left-hand operand of an intermediate peer is smaller than the right-hand operand, then it would be entirely transferred first so the other operand could be pipelined thus yielding parallelism between peers 2-5-q and peers 4-6-q. Parallel execution strategies improve both the query response time and the global efficiency of the P2P system.

6 Conclusion

P2P systems adopt a completely decentralized approach to data sharing. By distributing data storage and processing across autonomous peers in the network, they

can scale without the need for powerful servers. Although very useful, these systems are too simple and limited for advanced applications.

Advanced P2P applications such as scientific cooperation must deal with semantically rich data (e.g., XML documents, relational tables, etc.). Supporting such applications requires significant revisiting of distributed database techniques (schema management, access control, query processing, transaction management, consistency management, reliability and replication). When considering data management, the main requirements of a P2P system are autonomy, query expressiveness, efficiency, quality of service, and fault-tolerance. Depending on the P2P network architecture (unstructured, structured DHT, or hybrid super-peer), these requirements are more or less difficult to achieve. Unstructured networks have better fault-tolerance but can be quite inefficient because they rely on flooding for query routing. Hybrid systems have better potential to satisfy high-level data management requirements. However, DHT systems are best for key-based search and could be combined with super-peer networks for more complex searching.

Most of the work on sharing semantically rich data in P2P systems has focused on schema management, and query processing. However, there has been very little work on replication, transactions and access control. To illustrate some of these issues we introduced the APPA data management system which we are building. APPA has a P2P network independent architecture and supports decentralized schema management, optimistic replication based on log reconciliation and query processing that exploits replication for parallelism and load balancing.

Research on data management in P2P systems is only beginning. Much more work is needed to revisit distributed database techniques for large-scale P2P systems. The main issues have to deal with schema management, high-level query processing, transaction support and replication, and security. Furthermore, it is unlikely that all kinds of data management applications are suited for P2P systems. Typical applications which can take advantage of P2P systems are probably light-weight and involve some sort of cooperation. Characterizing carefully these applications is important and will be useful to produce performance benchmarks.

Acknowledgements

We wish to thank R. Akbarinia, V. Martins for their many inputs and fruitful discussions in the context of the APPA project, and S. Abiteboul and I. Manolescu for fruitful discussions in the context of the MDP2P project.

References

1. K. Aberer et al. P-Grid: a self-organizing structured P2P system. *SIGMOD Record, 32(3),* 2003.
2. S. Abiteboul et al. Dynamic XML documents with distribution and replication. *SIGMOD Conf.,* 2003.

3. R. Akbarinia, V. Martins, E. Pacitti, P. Valduriez. Replication and query processing in the APPA data management system. Submitted for publication, 2004.
4. E. Anceaume, M. Gradinariu, M. Roy. Self-organizing systems case study : peer-to-peer systems. *DISC Conf.,* 2003.
5. G. Chockler, I. Keidar, R. Vitenberg. Group communication specifications: a comprehensive study. *ACM Computing Surveys*, 33(427-469), 2001.
6. I. Clarke et al. Protecting Free Expression Online with Freenet. *IEEE Internet Computing, 6(1),* 2002.
7. N. Daswani, H. Garcia-Molina, B. Yang. Open problems in data-sharing peer-to-peer systems. *Int. Conf. on Database Theory*, 2003.
8. Gnutella. http://www.gnutelliums.com/.
9. R. Huebsch et al. Querying the Internet with PIER. *VLDB Conf.,* 2003.
10. A. Iamnitchi, M. Ripeanu, I. Foster. Locating data in (small world?) peer-to-peer scientific collaborations. *Int. workshop on P2P Systems (IPTPS),* 2002.
11. ICQ. http://www.icq.com/.
12. JXTA. http://www.jxta.org/.
13. Kazaa. http://www.kazaa.com/.
14. A. Levy, A. Rajaraman, J. Ordille. Querying heterogeneous information sources using source descriptions. *VLDB Conf.,* 1996.
15. MDP2P. http://www.sciences.univ-nantes.fr/info/recherche/ATLAS/MDP2P/.
16. Naspter. http://www.napster.com/.
17. W. Nejdl, W. Siberski, M. Sintek. Design issues and challenges for RDF- and schema-based peer-to-peer systems. *SIGMOD Record, 32(3),* 2003.
18. B. Ooi, Y. Shu, K-L. Tan. Relational data sharing in peer-based data management systems. *SIGMOD Record, 32(3),* 2003.
19. T. Özsu, P. Valduriez. *Principles of Distributed Database Systems*. 2nd Edition, Prentice Hall, 1999.
20. E. Pacitti, O. Dedieu. Algorithms for optimistic replication on the Web. *Journal of the Brazilian Computing Society*, *8(2),* 2002.
21. E. Pacitti, E. Simon: Update propagation strategies to improve freshness in lazy master replicated databases. *The VLDB Journal, 8(3-4),* 2000.
22. E. Pacitti, T. Özsu, C. Coulon. Preventive multi-master replication in a cluster of autonomous databases, *Euro-Par Conf.,* 2003), 2003.
23. N. Preguiça, M. Shapiro, C. Matheson. Semantics-based reconciliation for collaborative and mobile environments. *CoopIS Conf.*, 2003.
24. S. Ratnasamy et al. A scalable content-addressable network. *Proc. of SIGCOMM*, 2001.
25. SETI@home. http://www.setiathome.ssl.berkeley.edu/.
26. M. Shapiro. A simple framework for understanding consistency with partial replication. Technical Report, Microsoft Research, 2004.
27. I. Stoica et al. Chord: A scalable peer-to-peer lookup service for internet applications. *Proc. of SIGCOMM*, 2001.
28. A. Tanaka, P. Valduriez. The Ecobase environmental information system: applications, architecture and open issues. *ACM SIGMOD Record,* 3(5-6), 2000.
29. I. Tatarinov et al. The Piazza peer data management project. *SIGMOD Record 32(3),* 2003.
30. A. Tomasic, L. Raschid, P. Valduriez. Scaling access to heterogeneous data sources with DISCO. *IEEE Trans. on Knowledge and Data Engineering*, 10(5), 1998.
31. P. Valduriez: Parallel Database Systems: open problems and new issues. *Int. Journal on Distributed and Parallel Databases*, 1(2), 1993.
32. B. Yang, H. Garcia-Molina. Designing a super-peer network. *Int. Conf. on Data Engineering,* 2003.