

# Queries and Updates in the coDB Peer to Peer Database System

Enrico Franconi<sup>†</sup>, Gabriel Kuper<sup>‡</sup>, Andrei Lopatenko<sup>†,§</sup>, Ilya Zaihrayeu<sup>‡</sup>

<sup>†</sup>Free University of Bozen–Bolzano, Faculty of Computer Science, Italy,  
franconi@inf.unibz.it, lopatenko@inf.unibz.it

<sup>‡</sup>University of Trento, DIT, Italy,  
kuper@acm.org, ilya@dit.unitn.it

<sup>§</sup>University of Manchester, Department of Computer Science, UK

## Abstract

In this short paper we present the coDB P2P DB system. A network of databases, possibly with different schemas, are interconnected by means of GLAV coordination rules, which are inclusions of conjunctive queries, with possibly existential variables in the head; coordination rules may be cyclic. Each node can be queried in its schema for data, which the node can fetch from its neighbours, if a coordination rule is involved.

## 1 Introduction

In the paper [Franconi *et al.*, 2003] we introduced a general logical and computational characterisation of peer-to-peer (P2P) database systems. We first defined a precise model-theoretic semantics of a P2P system, which allows for local inconsistency handling. We then characterised the general computational properties for the problem of answering queries to such a P2P system. Finally, we devised tight complexity bounds and distributed procedures in few relevant special cases. The basic principles of the characterisation given in [Franconi *et al.*, 2003] are: (a) the role of the coordination formulas between nodes is for data migration (as opposed to the role of logical constraints in classical data integration systems); (b) computation is delegated to single nodes (distributed local computation); (c) the topology of the network may dynamically

change; (d) local inconsistency does not propagate; (e) computational complexity can be low.

In the paper [Franconi *et al.*, 2004] we thoroughly analysed a distributed procedure for the problem of local database update in a network of database peers, as defined in [Franconi *et al.*, 2003]. The problem of local database update is different from the problem of query answering. Given a P2P database system, the answer to a local query may involve data that is distributed in the network, thus requiring the participation of all nodes at query time to propagate in the direction of the query node the relevant data for the answer, taking into account the (possibly cyclic) coordination rules bridging the nodes. On the other hand, given a P2P database system, a “batch” update algorithm will be such that all the nodes consistently and optimally propagate all the relevant data to their neighbours, allowing for subsequent local queries to be answered locally within a node, without fetching data from other nodes at query time. The update problem has been considered important by the P2P literature; most notably, recent papers focused on the importance of data exchange and materialisation for a P2P network [Fagin *et al.*, 2003; Daswani *et al.*, 2003].

The coDB P2P DB system we present here implements the above ideas in a very general fashion. A network of databases, possibly with different schemas, can be interconnected by means of GLAV coordination rules, which are inclusions of conjunctive queries, with possibly existential variables in the head. Each node can be queried in its schema for data, which the node can fetch from its neighbours, if a coordination rule is involved. Note that rules can be cyclic, i.e., a fix-point computation may be needed among the nodes in order to get all the data that is needed to answer a query. In the abovementioned papers we have showed the correctness of the procedures we have implemented in the coDB system.

coDB supports *dynamic* networks: even if nodes and coordination rules appear or disappear during the

---

*Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, requires a fee and/or special permission from the Endowment.*

**Proceedings of the 30th VLDB Conference,  
Toronto, Canada, 2004**

This work has been partially supported by the EU projects Sewasie, KnowledgeWeb, and Interop.

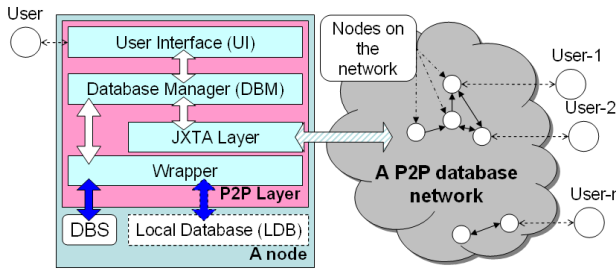


Figure 1: First level architecture

computation, the proposed algorithm will eventually terminate with a sound and complete result (under appropriate definitions of the latter, see [Franconi *et al.*, 2004]).

## 2 The Architecture

We implement database peers on top of *JXTA* [Project *JXTA*, 2004]. *JXTA* specifies a set of protocols which provide implementation of basic, as well as rather sophisticated P2P functionalities. As basic functionalities we can distinguish: definition of a peer on a network; creation of communication links between peers (called pipes); creation of messages, which can envelope arbitrary data (e.g. code, images, queries); sending messages onto pipes, etc. Examples of more sophisticated functionalities provided by *JXTA* are: creation of peer groups; specification of services and their implementation on peers; advertising of network resources (i.e. peers, pipes, peer groups, services, etc.) and their discovery in a distributed, decentralised environment. *JXTA* has a number of advantages for developing P2P applications. It provides IP independent naming space to address peers and other resources, it is independent of system platforms (e.g. Microsoft Windows, Macintosh or UNIX) and networking platforms (e.g. Bluetooth, TCP/IP), it can be run on various devices such as PCs or PDAs, and provides support for handling firewalls and NATs. We have chosen *JXTA* since it already gives practically all basic building blocks for developing P2P applications and thus allow the developer to concentrate on implementation of specific functionalities a given application is required to provide.

The first level logical architecture of a node, inspired by [Bernstein *et al.*, 2002], is presented on Figure 1. A node consists of *P2P Layer*, *Local Database (LDB)* and *Database Schema (DBS)*. *DBS* describes part of *LDB*, which is shared for other nodes. *P2P Layer* consists of *User Interface (UI)*, *Database Manager (DBM)*, *JXTA Layer* and *Wrapper*. Nodes connect to a P2P database network by means of connecting to other peer(s), as it is schematically shown on Figure 1 (see the arrow from *JXTA Layer* to the network and arrows between nodes in the network).

By means of UI users can commence network

queries and updates, browse streaming results, start topology discovery procedures, and so on. Among other things, UI allows to control other modules of P2P Layer. For instance, user can modify the set of coordination rules w.r.t. other nodes, define connection details for Wrapper, etc. *DBM* processes both user queries and queries coming from the network, as well as global and query-dependent update requests. It is also responsible for processing of query results coming both from *LDB* and the network, as well as for processing of updates results coming from the network. Finally, *DBM* manages propagation of queries, update requests, query results and update results on the network. *JXTA Layer* is responsible for all node's activities on the network, such as discovering of previously unknown nodes, creating pipes with other nodes, sending messages containing queries, update requests, query results, etc. Wrapper manages connections to *LDB* and executes input database manipulation operations. This is a module which is adjusted depending on the underlying database. For instance, when *LDB* does not support nested queries, then this is the responsibility of Wrapper to provide this support. Yet another task of Wrapper is retrieval and maintenance of *DBS*.

The *LDB* rectangle stands for *RDBMS*. It has dashed border to mean that local database may be absent. Nevertheless *DBS* must always be specified in order to allow a node to participate on the network. In this situation a given node acts as a mediator for propagating of requests and data, and all required database operations (as join and project) are executed in Wrapper. The *DBS* rectangle has rounded corners because it represents a repository, where *DBS* is stored. Arrows between *UI* and *DBM* as well as arrows between *JXTA Layer*, *Wrapper* and *DBM* have the same graphical notation because they represent procedure calls between different execution modules. The arrow between *JXTA Layer* and the network has another notation because it represents communication supported by *JXTA*. The arrows connecting *Wrapper*, *DBS* and *LDB* have yet another notation because the communication they denote is *LDB* dependent.

Nodes may import data from their *acquaintances* using definitions of *coordination rules*. The head of a coordination rule is a conjunctive query which refers to some local relation at a given node, and the body is another conjunctive query (sharing some variables with the head) which refers to relations of an acquaintance. In data integration literature this kind of mapping between two schemas is called *Global-Local-As-View*, or *GLAV* [Lenzerini, 2002]. The body of a coordination rule may also contain a set of comparison predicates which specify constraints over the domain of particular attributes of the acquaintance's relations. In order to import data from a node's acquaintance using a given coordination rule definition, the acquaintance executes the coordination rule and sends the results back to that node.

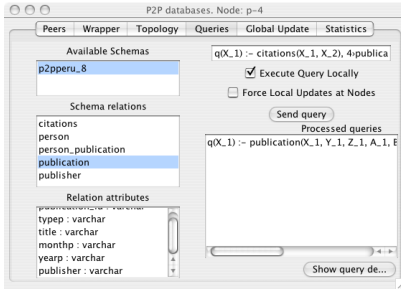


Figure 2: Query interface

A *global update* in a P2P database network is a process of updating nodes’ databases using *all* definitions of coordination rules they maintain. A global update is started when some (dedicated) node sends to all its acquaintances global update requests, containing definitions of appropriate coordination rules. These acquaintances execute the queries, respond with the query results, and propagate the global update to their acquaintances, and so on. The global update request propagation is stopped at some node if that node has no acquaintances to propagate the request, or if that node has already received this request message. For the purpose of global update identification, all global update request messages carry the same unique identifier generated at the node which started the global update procedure. We use JXTA to generate global updates identifiers.

### 3 The Algorithm

Herein we provide a concise description of the global update algorithm [Franconi *et al.*, 2004]. In order to understand how nodes process incoming query results and when results propagation is complete, we introduce some additional notions. We call coordination rules, *incoming links* at some node, if these rules are used by some other (acquainted) nodes for importing data from that given node. We call coordination rules, *outgoing links* at some node, if that node uses these rules in order to import data from its acquaintances. We say that an incoming link is *dependent* on an outgoing link, or that an outgoing link is *relevant* for some incoming link, if the head of the outgoing link reference to a relation, which is referenced by a body subgoal of the incoming link.

Query propagation is being done using extension of “diffusing computation” approach [Lynch, 1996]. When node gets a query request, it answers it using local data immediately, and it forwards it through all outgoing links. Each query request is labelled by a sequence of IDs of nodes it passed through. A node does not propagate a query request, if its ID is contained in the label of query request.

Query results coming from an acquaintance via some outgoing link (say,  $O$ ) can be seen as an additional, possibly empty set of tuples (say,  $T$ ) for the

relations ( $R$ ) referenced by the head of this outgoing link. This, in turn, means that re-computing of incoming links, dependent on  $O$ , may produce new results for the acquainted nodes. For performance reasons, it is important to avoid duplication in producing and propagating data. Therefore we first remove from  $T$  those tuples which are already in  $R$ , and get the set of tuples  $T'$ . If the conjunctive query in the head of the rule contains existential variables, then fresh new marked null values are used in tuples of  $T'$ . Then,  $T'$  is added to  $R$ . Incoming links, which are dependent on  $O$ , are computed by substituting  $R$  by  $T'$ . The reason for that is avoiding producing query results which might have been already produced for these incoming links. For each incoming link  $i$  we get query results  $R_i$ . Afterwards, we delete from  $R_i$  those tuples which have been already sent to the incoming link, and then send remaining tuples onto  $i$ . The receiver node processes these results analogously and may evoke, in turn, further results’ propagation. Therefore, incoming data can be seen as a result of *transitive* propagation of query results via a path of nodes, which we call *update propagation path*. At each node in the path, we reconcile and store results sent to corresponding incoming links until global update processing is complete for that node.

The global update processing is finished for some node (we say that after this the node is in the state “closed”) if all outgoing links are in the state “closed”. Initially, when a node starts a global update propagation or receives a global update request message, it is in the state “open” and all its outgoing links (if any) are in the state “open”. An acquaintance closes an incoming link (and, respectively, outgoing link at some acquainted node) if all its outgoing links which are relevant for this incoming link are in the state “closed”. A node closes its outgoing link if a) it got query results for all the maximal paths<sup>1</sup> passing through it; b) all query results did not bring any new data to local database. When all outgoing links of a node are in the state “closed”, then the node is also in the state “closed”. The global update processing is complete, when all nodes are in the state “closed”. It is worth saying that our algorithm processes global update properly in the presence of cyclic dependencies and guarantees termination. Under a proper global update processing nodes update their databases with all data that can be retrieved from their acquaintances, taking into account transitive dependencies between incoming and outgoing links. After the termination of the algorithm each node contains a sound and complete set of data (with respect to the semantics given in [Franconi *et al.*, 2003]).

In addition to global updates handling and query answering at a node, *coDB* supports a topology discov-

<sup>1</sup>By *maximal dependency path* for a node  $I$  and query  $Q$ , we call a dependency path which a) originates in the node initiated  $Q$ , b) passes node  $I$ ; c) is simple; d) can not be extended to any other simple dependency path by adding nodes to the tail.

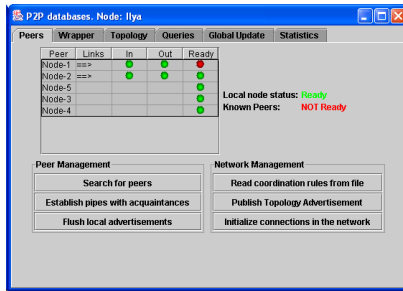


Figure 3: Peer discovery window

ery algorithm. When a node starts, it creates pipes with those nodes, w.r.t. which it has coordination rules, or which have coordination rules w.r.t. the given node. Several coordination rules w.r.t. a given node can use one pipe to send requests and data. If some coordination rules are dropped and a pipe is not assigned any coordination rule, then this pipe is also closed.

## 4 The Demo

In the demo we will measure the performance of various networks arranged in different topologies: we need to start-up all the nodes, establish coordination rules between pairs of nodes, run a set of experiments and, finally, collect statistical information. In order to facilitate these tasks we provide some peer (called *super-peer*) with some additional functionalities. In particular, that peer can read coordination rules for *all* peers from a file and broadcast this file to all peers on the network. Once received this file, each peer looks for relevant coordination rules and creates necessary pipe connections. If a coordination rules file is received when a peer has already set up coordination rules and pipes, then it drops “old” rules and pipes, and creates new ones, where necessary. Thus, a super-peer can dynamically change the network topology at *runtime*. Each node shows to its user the other nodes it has pipes with, and w.r.t. which nodes it has incoming and outgoing links. It also shows which other nodes (not acquaintances) it has discovered with the help of JXTA (Figure 3).

For the purposes of collecting experimental data, each node has an additional statistical module. This module accumulates various information about global updates such as: total execution time of an update, number of query result messages received per coordination rule and the volume of the data in each message, longest update propagation path, and so on. During the lifetime of a network, each node accumulates this information. A super-peer has the possibility to collect, at any given time, statistical information from *all* nodes on the network. Then, the super-peer processes all incoming statistical messages, aggregates them and creates a final statistical report.

For intermediate nodes, global update processing is done on the background, transparently for the user.

Each node maintains a global update processing report and makes it available for the user on request. The report includes information about starting and finishing times of an update, volume of data transferred, which acquaintances have been queried and to which nodes query results have been sent.

## References

- [Bernstein *et al.*, 2002] P. Bernstein, F. Giunchiglia, A. Kementsietsidis, J. Mylopoulos, L. Serafini, and I. Zaihrayeu. Data management for peer-to-peer computing: A vision. *Workshop on the Web and Databases, WebDB*, 2002.
- [Calvanese *et al.*, 2003] Diego Calvanese, Elio Damaggio, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. Semantic data integration in p2p systems. In *Proc. of the VLDB International Workshop On Databases, Information Systems and Peer-to-Peer Computing (DBISP2P-2003)*, 2003.
- [Calvanese *et al.*, 2004] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Riccardo Rosati. Logical foundations of peer-to-peer data integration. In *Proc. of the 23rd ACM SIGACT SIGMOD SIGART Sym. on Principles of Database Systems (PODS-2004)*, 2004. To appear.
- [Daswani *et al.*, 2003] Neil Daswani, Hector Garcia-Molina, and Beverly Yang. Open problems in data-sharing peer-to-peer systems. In *ICDT 2003*, 2003.
- [Fagin *et al.*, 2003] Ronald Fagin, Phokion G. Kolaitis, Ren&#233; J. Miller, and Lucian Popa. Data exchange: Semantics and query answering. In *Proceedings of the 9th International Conference on Database Theory*, pages 207–224. Springer-Verlag, 2003.
- [Franconi *et al.*, 2003] Enrico Franconi, Gabriel Kuper, Andrei Lopatenko, and Luciano Serafini. A robust logical and computational characterisation of peer-to-peer database systems. In *Proc. of the VLDB International Workshop On Databases, Information Systems and Peer-to-Peer Computing (DBISP2P-2003)*, 2003.
- [Franconi *et al.*, 2004] Enrico Franconi, Gabriel Kuper, Andrei Lopatenko, and Ilya Zaihraeu. A distributed algorithm for robust data sharing and updates in p2p database networks. In *Proc. of the EDBT International Workshop on Peer-to-Peer Computing and Databases*, 2004.
- [Ghidini and Serafini, 1998] Chiara Ghidini and Luciano Serafini. Distributed first order logics. In Franz Baader and Klaus Ulrich Schulz, editors, *Frontiers of Combining Systems 2*, Berlin, 1998. Research Studies Press.
- [Halevy *et al.*, 2003] Alon Y. Halevy, Zachary G. Ives, Dan Suciu, and Igor Tatarinov. Schema mediation in peer data management systems. In *ICDE*, 2003.
- [Hellerstein, 2003] Joseph M. Hellerstein. Toward network data independence. *SIGMOD Rec.*, 32(3):34–40, 2003.
- [Kementsietsidis *et al.*, 2003] Anastasios Kementsietsidis, Marcelo Arenas, and Renee J. Miller. Mapping data in peer-to-peer systems: Semantics and algorithmic issues. In *Proceedings of the SIGMOD International Conference on Management of Data (SIGMOD’03)*, 2003.
- [Lenzerini, 2002] Maurizio Lenzerini. Data integration: A theoretical perspective. In Lucian Popa, editor, *Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 233–246, 2002.
- [Lynch, 1996] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., 1996.
- [Project JXTA, 2004] Project JXTA, 2004. See <http://www.jxta.org>.
- [Serafini *et al.*, 2003] Luciano Serafini, Fausto Giunchiglia, John Mylopoulos, and Philip A. Bernstein. Local relational model: A logical formalization of database coordination. In *CONTEXT 2003*, pages 286–299, 2003.