UNIVERSITY
OF TRENTO

DEPARTMENT OF INFORMATION AND COMMUNICATION TECHNOLOGY

38050 Povo – Trento (Italy), Via Sommarive 14
http://www.dit.unitn.it

THE CODB ROBUST PEER-TO-PEER DATABASE SYSTEM

Enrico Franconi, Gabriel Kuper, Andrei Lopatenko
and Ilya Zaihrayeu

April 2004

Technical Report # DIT-04-028

Also: accepted to the Second Workshop on Semantics in Peer-to-Peer and Grid Computing at the Thirteenth International World Wide Web Conference 18 May 2004, New York, USA

The coDB Robust Peer-to-Peer Database System

Enrico Franconi[1], Gabriel Kuper[2], Andrei Lopatenko[1,3], and Ilya Zaihrayeu[2]

[1] Free University of Bozen–Bolzano, Faculty of Computer Science, Italy,
franconi@inf.unibz.it, lopatenko@inf.unibz.it
[2] University of Trento, DIT, Italy, kuper@acm.org, ilya@dit.unitn.it
[3] University of Manchester, Department of Computer Science, UK

Abstract. In this paper we give an overview of the coDB semantically well-founded 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. Correctness and termination of query answering is guaranteed also in the case of runtime change in the topology of the network.

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, compared with classical data integration systems. In [Franconi *et al.*, 2003] we emphasise the difference between P2P systems and standard classical logic-based integration systems, as for example described in [Lenzerini, 2002a].

Let's consider now an example explaining why the P2P semantics is different from the classical semantics given to data integration systems. Suppose we have three distributed databases. The first one (DB_1) is the municipality's internal database, which has a binary table `Citizen-1` which contains the name of the citizen and the marital status (with values *single* or *married*). The second one

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

(DB_2) is a public database, obtained from the municipality's database, with two unary tables `Male-2` and `Female-2`. The third database (DB_3) is the Pension Agency database, obtained from a public database, with the unary table `Citizen-3` and a binary table `Marriage-3` (stating that two people are married). The three databases are interconnected by means of the following rules:

$1 : \texttt{Citizen-1}(x, y) \Rightarrow 2 : (\texttt{Male-2}(x) \vee \texttt{Female-2}(x))$
(this rule connects DB_1 with DB_2)

$2 : \texttt{Male-2}(x) \Rightarrow 3 : \texttt{Citizen-3}(x)$
$2 : \texttt{Female-2}(x) \Rightarrow 3 : \texttt{Citizen-3}(x)$
(these rules connect DB_2 with DB_3)

In the classical logical model, the `Citizen-3` table in DB_3 should be filled with all of the individuals in the `Citizen-1` table in DB_1, since the following rule is logically implied:

$1 : \texttt{Citizen-1}(x) \Rightarrow 3 : \texttt{Citizen-3}(x)$

However, in a p2p system this is not a desirable conclusion. In fact, rules should be interpreted only for fetching data, and not for logical computation. In this example, the tables `Female-2` and `Male-2` in DB_2 will be empty, since the data is fetched from DB_1, where the gender of any specific entry in `Citizen-1` is not known. From the perspective of DB_2, the only thing that is known is that each citizen is in the view ($\texttt{Female-2} \vee \texttt{Male-2}$). Therefore, when DB_3 asks for data from DB_2, the result will be empty.
In other words, the rules

$2 : \texttt{Male-2}(x) \Rightarrow 3 : \texttt{Citizen-3}(x)$
$2 : \texttt{Female-2}(x) \Rightarrow 3 : \texttt{Citizen-3}(x)$

will transfer no data from DB_2 to DB_3, since no individual is known in DB_2 to be either definitely a male (in which case the first rule would apply) or definitely a female (in which case the second rule would apply). We only know that any citizen in DB_1 is either male or female in DB_2, and no reasoning about the rules should be allowed.

In order to explain the importance of cyclic rules, suppose now to have an additional cyclic pair of rules connecting DB_1 and DB_3 as follows:

$1 : \texttt{Citizen-1}(x, \text{“married”}) \Rightarrow 3 : \texttt{Marriage-3}(x, y)$
$3 : \texttt{Marriage-3}(x, y) \Rightarrow 1 : \texttt{Citizen-1}(x, \text{“married”}) \ \wedge$
$1 : \texttt{Citizen-1}(y, \text{“married”})$

These rules serve the purpose to synchronise the people who are known to be married in DB_1 (by means of the `Citizen-1` table) with the people who are known to be married in DB_3 (by means of the `Marriage-3` table). Suppose that it is known in DB_1 that only John is married, and nothing in known in DB_3 about marriages. In the classical logical model, a query to DB_3 asking for the non existence of some married person different from John will get a negative

answer[1]. In a P2P setting, we actually expect a positive answer, since the only information that is fetched is about John.

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 survey 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 [Calvanese *et al.*, 2003]. 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.

The coDB P2P DB system is *robust* in the sense that it supports *dynamic* networks: even if nodes and coordination rules appear or disappear during the computation, the proposed algorithm will eventually terminate with a sound and complete result (under appropriate definitions of the latter).

Note that our approach is novel with respect to the known P2P and data integration literature for several reasons. As we have shown, our P2P semantics is different from the classical (first-order logic) data integration semantics. In the context of P2P systems, are only two other approaches which deal in a well founded way with cycles in the coordination rules [Calvanese *et al.*, 2003; Serafini and Ghidini, 2000]. The acyclic case (e.g., [Halevy *et al.*, 2003]) is relatively simple – a query is propagated through the network until it reaches the leaves of the network. The work in [Calvanese *et al.*, 2003] uses a notion of semantics similar to the semantics introduced in [Franconi *et al.*, 2003], but it describes only a global algorithm, that assumes a central node where all computation is performed. The paper [Serafini and Ghidini, 2000] describes a local algorithm to compute query answers, but it does not allow real GLAV coordination rules (with existential variables in the head). The algorithms in coDB support such variables, in a similar fashion to the global non-distributed algorithm of [Calvanese *et al.*, 2003; 2004]. None of the above approaches support dynamic networks. The body

[1] Note that the semantics of a query is the *certain answer* semantics.

of work about distributed data replication uses possibly similar techniques, such as "lazy replication" and "epidemic algorithms" [Holliday *et al.*, 2003; Wuu and Bernstein, 1984], which are not directly applicable since they rely on a different semantics of the mappings; we plan to look deeper into these techniques which may be useful for optimisation purposes.

2 P2P database systems

Our P2P framework is based on the logical model of [Franconi *et al.*, 2003].

Definition 1 (Local database) *Let I be a nonempty finite set of indexes $\{1, 2, \ldots, n\}$, and C be a set of constants. For each pair of distinct i, $j \in I$, let L_i be a first-order logic without function symbols, with signature disjoint from L_j but for the shared constants C. A* local database *DB_i is a theory on the first order language L_i.*

Nodes are interconnected by means of coordination rules. A coordination rule allows a node i to fetch data from its neighbour nodes $j_1, \ldots, j_m$.

Definition 2 (Coordination rule) *A* coordination rule *is an expression of the form*

$$j_1 : b_1(\mathbf{x}_1, \mathbf{y}_1) \wedge \cdots \wedge j_k : b_k(\mathbf{x}_k, \mathbf{y}_k) \Rightarrow i : h(\mathbf{x}, \mathbf{y})$$

where $j_1, \ldots, j_k$, i are distinct indices, each $b_l(\mathbf{x}_l, \mathbf{y}_l)$ is a formula of L_{j_l}, and $h(\mathbf{x}, \mathbf{y})$ is a formula of L_i, and $\mathbf{x} = \mathbf{x}_1 \cup \cdots \cup \mathbf{x}_k$.

Note that we are making the simplifying assumption that the equal constants mentioned in the various nodes refer to equal objects, i.e., that they play the role of URIs (Uniform Resource Identifiers). Other approaches consider *domain relations* to map objects between different nodes [Serafini *et al.*, 2003], and we plan to consider such extensions in future work.

A P2P system is just the collection of nodes interconnected by (possibly cyclic) rules.

Definition 3 (P2P system) *A* peer-to-peer (P2P) system *is a tuple of the form $MDB = \langle LDB, CR \rangle$, where $LDB = \{DB_1, \cdots, DB_n\}$ is the set of local databases, and CR is the set of coordination rules.*

A user accesses the information hold by a P2P system by formulating a query at a specific node.

Definition 4 (Query) *A* local query *is a first order formula in the language of one of the local databases DB_i.*

The semantics of a P2P system and of queries is derived from the general logical framework presented in [Franconi *et al.*, 2003]. In the current version of the coDB system, we assume that all nodes are relational databases; coordination

rules may contain conjunctive queries in both the head and body (without any safety assumption and possibly with built-in predicates in the body); the body involves only one node per rule. Under these assumptions, computing query answers is reducible to data fetching [Franconi *et al.*, 2003; Calvanese *et al.*, 2003].

To describe the P2P networks we introduce the notion of a *dependency edge* between nodes of a P2P network.

Definition 5 *There is a* dependency edge *from a node i to node j, if there is a coordination rule with head at node i and body at node j.*

Note that the direction of a dependency edges is the opposite to that of the rules. The direction of a rule is the direction in which data is transfered, whereas the dependency edge has the opposite orientation. In this paper we use $\mathcal{MDB}$ to denote a P2P system, using terms such as *P2P system* or *a network*; please note that we consider the general case when the network is *cyclic*. $\mathcal{I}$ is used to denote a set of all nodes in given $\mathcal{MDB}$, $\mathcal{C}$ denotes the set of all coordination rules, and $\mathcal{L}$ the set dependency edges between nodes in a network derived from $\mathcal{C}$. Subsets of $\mathcal{I}$ are denoted by $\mathcal{A}$. We assume that $\mathcal{I}$, $\mathcal{L}$, and $\mathcal{C}$ are always finite sets.

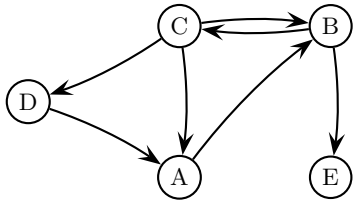
Definition 6 *A dependency path for a node i is a path $\langle i_1, i_2, \ldots, i_n \rangle$ of dependency edges, such that 1) $i_1 = i$; 2) $\langle i_1, \ldots, i_{n-1} \rangle$ is a simple path (no one node appears twice).*

Definition 7 *A maximal dependency path for a node i is a dependency path such that if we add any node to the path, the result will not be a dependency path. In this paper, when we describe dependency paths for a node i, we omit the first node (i).*

As an example, consider a P2P system with the follow schemas and rules:

$A : a(X,Y)$	$r1 : E : e(X,Y) \rightarrow B : b(X,Y)$
$B : b(X,Y)$	$r2 : B : b(X,Y), b(Y), Z \rightarrow C : c(X,Z)$
$C : c(X,Y), f(X)$	$r3 : C : c(X,Y), c(Y,Z) \rightarrow B : b(X,Z)$
$D : d(X,Y)$	$r4 : B : b(X,Y), b(X,Z), X \neq Z \rightarrow A : a(X,Y)$
$E : e(X,Y)$	$r5 : A : a(X,Y) \rightarrow C : f(X)$
	$r6 : A : a(X,Y) \rightarrow D : d(Y,X)$
	$r7 : D : D(X,Y), D(Y,Z) \rightarrow C : c(X,Y)$

The dependency edges and the maximal dependency paths for the example are:



#	path	#	path	#	path	#	path
A	ABCA	B	BE	C	BE	D	ABE
A	ABE	B	BCAB	C	BC	D	ABCD
A	ABCB	B	BCB	C	DABC	D	ABCB
A	ABDA	B	BCDAB	C	ABC	D	ABCA
				C	ABE		

3 Dynamic behaviour of the P2P network

One of the distinctive characteristics of P2P systems is that the network can vary dynamically. Assume that the network $\mathcal{MDB}$ consist initially of a set of nodes $\mathcal{I}$, and that $\mathcal{C}$ is an initial set of coordination rules with $\mathcal{L}$ being the initial set of dependency edges. We model network dynamicity by adding/removing coordination rules between nodes, and therefore deletion of a node is modelled by deleting all coordination rules that relate to this node. With respect to query answering and update, adding/removing nodes with coordination rules is easily seen to be equivalent to the assumption that all nodes are present from the start, and that only coordination rules are changed.

We define an atomic network change operation as follows.

- *addLink(i,j,rule,id)*: add the coordination rule *rule* from node j (the body) to node i (the head). *id* is the name of a rule, which should be unique for a given pair of nodes.
- *deleteLink(i,j,id)*: delete the coordination rule *id* between nodes i and j

Definition 8 *1. A* change $\mathbf{U}$ of a network $\mathcal{MDB}$ *is a sequence of atomic change operations over* $\mathcal{MDB}$.
2. A finite change *of a network is a finite sequence of atomic changes.*
3. An initial subchange $\mathbf{U}_1$ *of a change* $\mathbf{U}$ *is a initial prefix of* $\mathbf{U}$
4. A subchange $\mathbf{U}_{\mathcal{A}}$ *of* $\mathbf{U}$ *in respect to* $\mathcal{A} \subset \mathcal{I}$ *is a set of atomic operations of* $\mathbf{U}$*, relevant to* $\mathcal{A}$ *and ordered with the same order as in* $\mathbf{U}$

We assume that in the case of atomic change the network will be notified about the change in the following cases:

1. in case of *addLink(i,j,rule,id)*, the node i (which will be able to fetch data by this rule) gets a notification;
2. in case of *deleteLink(i,j,id)*, the node i (which will be unable to fetch data by this rule) gets a notification.

Definition 9 *1. A* sound answer *of a query* Q *in a network subject to runtime changes, is an answer to the query that is included in the result that we would obtain if we executed all the addLink statements before running* Q*, and did not execute the deleteLink statements at all.*

2. *A* complete answer *of a query Q in a network subject to runtime changes, is an answer to the query that contains the result that we would obtain if we executed all the deleteLink statements before running Q, and did not execute the addLink statements at all.*

The basic idea behind this definition is that we cannot know in advance what the state of the database will be at termination time. Therefore, in the definition we require that a sound and/or complete answer will be classically sound and/or complete with respect to the part of the database that is *unchanged*. The result with respect to the part that is changed will depend on the order and timing of the execution of the changes. In this sense, the answer to a query in a network subject to "small" changes will be still meaningful with respect to the majority of the data that resides in the stable parts of the network. The following theorem states that our update algorithm behaves well with respect to change.

Theorem 1 *1. For a finite runtime change of a network, the update algorithm will terminate, and it gives sound and complete answers to queries in the network subject to runtime changes.*
2. In the case of an infinite runtime change to the network, the update algorithm may not terminate.

However, in general we cannot assume that a network change is finite. In the general case, therefore, the nodes in the network may never reach the fix-point – or at least, we may not be able to show that they have reached a fix-point. We now give a condition on when a subset of nodes can reach a fix-point, even under infinite change of the whole network.

Definition 10 *1. A set of nodes $\mathcal{A}_1$ is separated from a set of nodes $\mathcal{A}_2$ in a P2P network $\mathcal{I}$ if there is no dependency path from any node in $\mathcal{A}_1$ that involves a node in $\mathcal{A}_2$.*
2. A set of nodes $\mathcal{A}_1$ is separated from a set of nodes $\mathcal{A}_2$ in P2P network $\mathcal{I}$ with respect to a change $\mathbf{U}$ if for any subchange of $\mathbf{U}$ there is no dependency path from a node in $\mathcal{A}_1$ involving a node in $\mathcal{A}_2$ in the network we obtain by applying that subchange.

Theorem 2 *If, for a network $\mathcal{I}$, a set of nodes $\mathcal{A}$ is separated from $(\mathcal{I} \setminus \mathcal{A})$ with respect to a (possibly infinite) change $\mathbf{U}$ over $\mathcal{I}$, and $\mathbf{U}_{\mathcal{A}}$ is finite, then the algorithm, when applied to a node in $\mathcal{A}$, terminates, yielding a sound and complete answer.*

Lemma 3 *For a finite runtime change of the network, the complexity of the update algorithm at each node is in 2EXPTIME with respect to the size of the change.*

4 The architecture of the coDB system

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

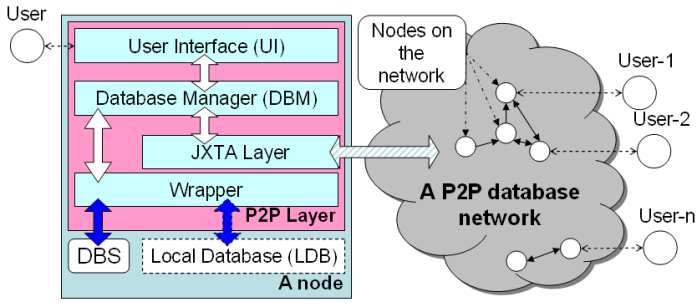


Fig. 1. First level architecture

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 the UI users can commence network queries and updates, browse streaming results, start topology discovery procedures, and so on. Among other things, th 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 (we use MySQL in coDB). 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 relations 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, 2002b]. 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 computes the coordination rule and sends the results back to that node.

A *global update* in a P2P database network is a process of updating nodes' databases using *all* definitions of coordination rules they maintain. Note that currently the coDB system only implements queries by first doing a global update and then by answering the query locally at the node at which the query itself was posed. 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 computes 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

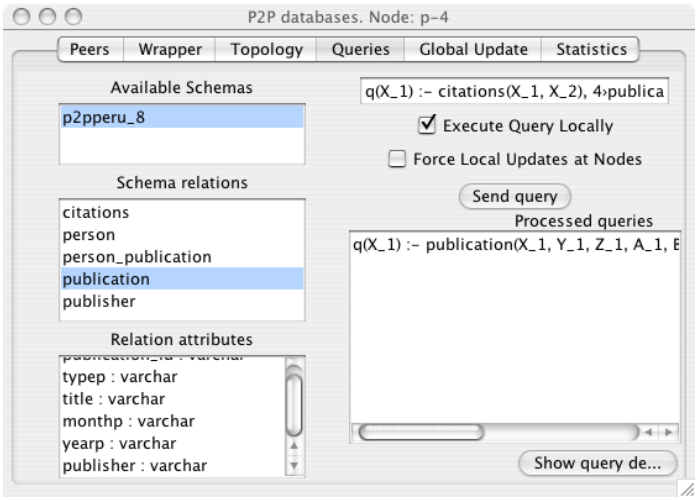


Fig. 2. Query interface

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.

5 The distributed algorithm

Herein we provide a concise description of the distributed 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 global update request, if the request ID is already known to that node.

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 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 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 dependency paths passing through it; b) all 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 discovery 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.

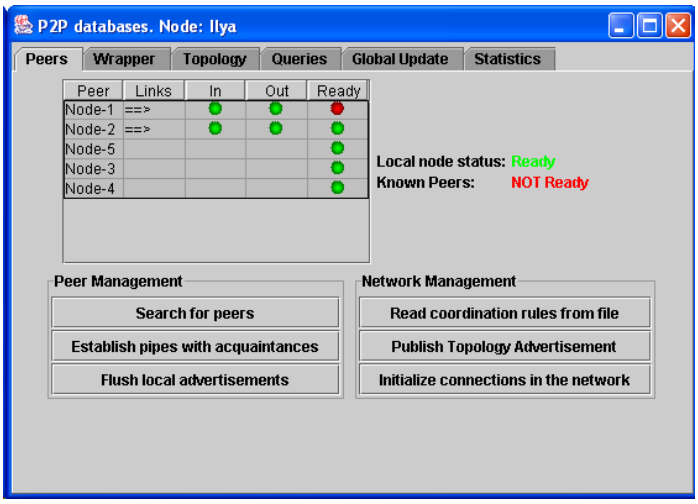


Fig. 3. Peer discovery window

6 Experiments

Preliminary experiments were done to measure the scalability of our approach with respect to the size and topology of the P2P network: 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: 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.

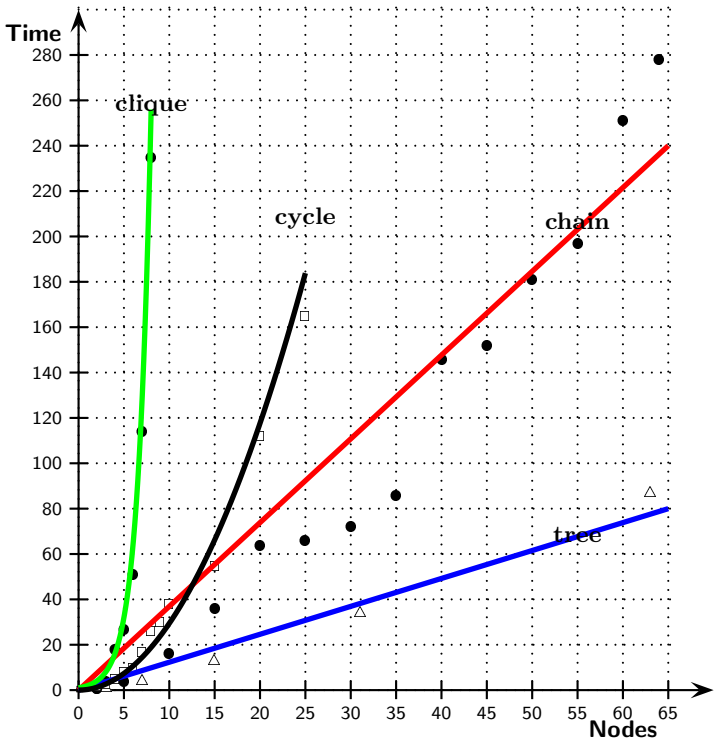


Fig. 4. Update time with respect to the size of the network.

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.

64 nodes participated to the preliminary experiments. The local relational databases are based on DBLP data (`http://dblp.uni-trier.de/xml`) and contained about 6400 records about publications (about 100 per node), organised in 3 different relational schemas. The data distribution is such that there is no intersection between initial data in neighbour nodes. Four types of topologies have been considered: trees, chains, simple cycles, and cliques. The first two topologies don't contain cyclic rules, while the last two do contain cyclic rules.

By looking at the execution time with respect to the size of the network (see Figure 4) we see, as expected, a linear dependency in the case of non-cyclic networks. In the case of the simple cyclic structure, we see a quadratic increase of evaluation time with respect to the size of the network, as expected—since the =re is a unique cycle in the network. Experiments with a clique topology,

show an exponential blow-up in the evaluation time, due to the fact that the cycles grow at least exponentially in the size of the network for this topological structure.

7 Conclusions

In this paper we have given an overview of the coDB P2P DB system. Important aspects of our system are: (1) it is semantically well-founded, (2) it allows for cyclic GLAV coordination rules with built-in predicates, (3) supports batch updates or queries on-the-fly, (4) correctness and termination of query answering is guaranteed also in the case of runtime change in the topology of the network.

Our main focus for the near future will be to work on optimisations in the various algorithms involved, to experiment with various topologies, to identify special cases where the complexity becomes lower.

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, R. 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.

[Holliday *et al.*, 2003] JoAnne Holliday, Robert C. Steinke, Divyakant Agrawal, and Amr El Abbadi. Epidemic algorithms for replicated databases. *IEEE Trans. Knowl. Data Eng.*, 15(5):1218–1238, 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, 2002a] Maurizio Lenzerini. Data integration: a theoretical perspective. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 233–246. ACM Press, 2002.

[Lenzerini, 2002b] 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 and Ghidini, 2000] Luciano Serafini and Chiara Ghidini. Using wrapper agents to answer queries in distributed information systems. In *Proceedings of the First Biennial Int. Conf. on Advances in Information Systems (ADVIS-2000)*, 2000.

[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.

[Wuu and Bernstein, 1984] G. T. Wuu and A. J. Bernstein. Efficient solutions to the replicated log and dictionary problems. In *Proc. of 3rd ACM Symp. Principles of Distributed Computing*, pages 233–242, 1984.