Robust Data Exchange for Unreliable P2P Networks

Duc Minh Le, Andrew Charles Smith and Peter McBrien
Dept. of Computing, Imperial College London,
Exhibition Road, London, SW7 2AZ

dmle,acs203,pjm@doc.ic.ac.uk

Abstract

The aim of this work is to provide a robust way for peers with heterogeneous data sources to exchange information in an unreliable network. We address this problem in two ways. Firstly, we define a set of application-layer data exchange protocols to facilitate the discovery of and the communication between peers. Secondly, we provide a query processing component with a cache-driven query processor that allows peers on the network to cache queries and their results on demand. The data caches are used to give partial or complete answers to a query if the original data sources are unavailable.

1. Introduction

P2P systems that go beyond simple file sharing and that integrate schemas making use of the semantic information contained in them are growing in number [1], [2], [3], [4]. These systems allow structured global repositories of data to be created. Such a repository may be dynamic with peers leaving and joining the P2P network all the time. They may also be made up of heterogeneous data sources.

We can imagine, for example, a number of groups of field researchers tracking the movements of migrating animals in different regions of the world. They record their observations on PDAs which have an intermittent network link to a local base station where a database of all results for a region is kept. The base station has a permanent network connection to other base stations around the world. The field researchers switch off their PDAs when they are not in use resulting in peers leaving and joining the network and the different base stations do not necessarily use the same schema or data modelling language to store their data, making the data sources heterogeneous.

Our work has focused on addressing the issues above by focusing on how robust data exchange may occur between peers in a global P2P network. In particular, we have developed and demonstrated a technique whereby peers with heterogeneous data sources can integrate with each other and then exchange queries and data in a robust way. The techniques and implementation are an extension to AUTOMED [5] and the existing P2P protocols were developed to be robust against peer and link failure, with particular emphasis on supporting various query processing options and data caching. As far as we are aware there is no other work specifically aimed at providing robust data integration and exchange in heterogeneous P2P systems.

The remainder of this paper is structured as follows. We give an overview of our P2P data integration and exchange methodology in Section 2. We then present, in Section 3, the set of protocols we developed to allow data exchange to be performed over a P2P network. Section 4 describes the cache-drive query processing component which enables peers to cache queries and their results, enabling the system to answer queries when failures occur in the network. Section 5 offers some conclusions and directions for further work.

2. P2P Data Integration and Exchange

In this section we will briefly discuss the integration and exchange of heterogeneous data in a P2P network and specifically how we do this with AUTOMED. We can summarise the operation as follows: To perform a data integration, an AUTOMED peer P_n will integrate a number of data sources $DS_a, DS_b, \ldots$ into some global schema GS_n, and then transform that global schema into one or more public schemas $PS_s, PS_t, \ldots$ that are made available to other peers on the network. An AUTOMED P2P network is thus formed by peers which relate to each other via some common public schemas.

This flexible method of describing the logical association between peers, data sources and public schemas is used in two operational ways to control the interaction between peers:

- P_x may send a query on PS_s to peer P_y, requesting that P_y evaluate the result. P_y may then transform the query into one on its local data sources, and pass back the results to P_x. For example P_1 may execute queries, evaluating the results against DS_1 and DS_2. It may then ask for a remote execution of the query on P_2, and obtain an answer from DS_3.
  In addition, P_y may act as a **broker**, passing on the query to any other peers it knows support PS_s, or any other public schemas PS_t that P_y has a pathway between PS_s and PS_t. For example, the query from P_1 having been passed to P_2 may be transformed into a query on PS_3, and then the logical association with P_3 is used to pose the query on PS_3 on P_3, and obtain answers from DS_4 and DS_5.
- P_x may request that P_y send the pathways from PS_s to the local data sources of P_y over to P_x. Peer P_x may store these pathways in its local repository, and evaluate queries directly of the remote data sources.
  For example, P_1 may ask for the pathway on P_2 from PS_2 to DS_3, and add that information into its local repository. This means it may then perform the translation of queries from PS_2 to P_2, placing no load on P_2

The integration and exchange of data between heterogeneous sources requires some **common data model** (**CDM**) [6], capable of accurately expressing all the information stored in each data source. The CDM we use is the **hypergraph data model** (**HDM**) [7], [8]. The HDM is very flexible and has been used to describe many of the most popular data modelling languages (DMLs) in use today such as SQL, XML and the ER model. However, we use the CDM only when necessary. In the standard **mediator** approach to data integration [9], data sources are always converted into the CDM when used in a data integration system. In our approach, we only convert into the CDM if the data sources use different DMLs. Hence, we reduce the complexity of the system by only converting between DMLs where necessary.

Once any necessary conversion into the CDM has been done, a **mapping language** is needed to create mappings from each data source to a **global schema**. We use the **Both-As-View** (**BAV**) [10] approach to data integration, which describes *bi-directional* mappings, called **transformation pathways**, between schemas. Hence, once we have a definition of $PS_s \rightarrow PS_t$, we may automatically infer the mapping $PS_t \rightarrow PS_s$, and hence move data and queries between the two schemas. As such, AUTOMED using a combination of BAV and the HDM, provides a powerful framework on which to do heterogeneous data integration in a P2P network [4].

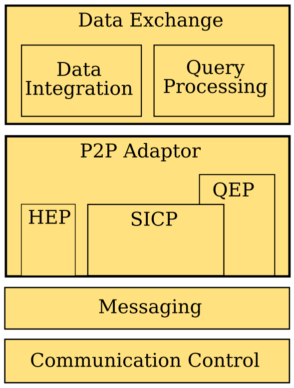

Figure 1. The AutoMed P2P Protocol Suite

3. Data and Meta-Data Exchange

Figure 1 illustrates the application-layer protocol suite which allows AUTOMED peers to function in a P2P network.

This protocol suite has the following components:

- The **communication control** layer provides an abstraction of the standard TCP/IP transport layers. As a result, all the peer's operations can be performed without being aware of the underlying transport protocol being used. The default transport protocol is the connectionless UDP, although this can be changed to the connection oriented TCP if required.
- The **messaging** layer defines the application message format, which encapsulates both the control information and the application data to be exchanged between peers. The general message format is described in Table 1.
  Each message is represented as an XML document, and the content field itself contains (sub) XML documents that represents either meta-data or a data object being exchanged.
- **P2P adaptor** layer consists of three protocols. The **hello protocol** (**HEP**) is used by peers to join the network. Peers can only start exchanging public schemas and data when they have

Message field	Description	Usage
performative	Action to be carried out in the context of this message	Used by the recipient to determine the mode of communication, e.g. unicast or multicast, advertising or requesting information
context	What the content relates to	Typically mapped to the name of an application process or component, e.g. authentication, query execution, and so on
sender	Name of the sending peer	Identifies the sender
receiver	Name of the target peer	Identifiers the receiver
mesgID	Unique ID given to each message	Randomly generated by the sender and may be reused by the receiver in the inReplyTo field of its replied message
inReplyTo	Optional field which specifies the value of the mesgID field of the received message	Taken from the value of the mesgID of the request message
content	Zero or more (name,value) pairs	Can be used to store both the primitive and object data

Table 1. The Data Exchange Message Format

finished the HEP process. The **schema information control protocol** (**SICP**) is responsible for distributing and managing the information about public schemas, peer-to-schema associations, transformation pathways, and data models. The **query execution protocol** (**QEP**) defines standard functions for executing queries among peers on the network.

- **data exchange** layer is the application layer which uses the services provided by the data integration and query processing components to allow peers exchange data.

The P2P adaptor layer is responsible for defining the standard protocols for establishing the P2P network, exchanging public schema metadata, and routing queries. An AUTOMEDP2P network consists of a set of peers connected via one or more directory service peers. The **Hello Protocol** (HEP), is used by peers to locate a directory service on the network for registration. At start-up, a peer sends a hello (uni- or multi-cast) message to discover a directory service in its neighbourhood network. The designated directory service peer (DSP) confirms its presence by replying with a hello_ack message. Once registered, a peer periodically exchanges keep_alive messages with its DSP to confirm that it is active.

The **Schema Information Control Protocol** (SICP) defines a set of functions that enable a peer to publish its public schemas and transformation pathways to the directory service, so they can be looked up and acquired by other peers. The SICP can only be initialised after a peer has successfully registered itself to the network. First, a peer P_a may publish its public schemas to the DSP using the schema_advertise message. Next, the DSP may forward to P_a a pathway_request message asking for the transformation pathways to the data sources which are attached to the published public schemas. For each pathway that P_a has in the local repository, it sends a pathway_reply message, which has details about the pathway, back to the DSP.

Finally, the **Query Execution Protocol (QEP)** is responsible for routing a query over the P2P network. To do this, it defines three key functions: send_query, send_group_query, and execute_query. The first two functions are used to route a query from a query (source) peer to one or a number of target peers. A query is posed on a peer's public schema. To identify which peers to send the query to, the query peer first determines the target schemas of the public schema using the associated transformation pathways as defined by the SICP. It then asks the DSP for the identifiers of the peers that implement those target schemas. Once the target peers have been identified, the query peer then prepares the query message for sending. Depending on the query semantics, a query can be sent using either a serial or parallel method. When a target peer has received a query message, it can execute the query by using local data sources (if any) and/or forwarding it to other target peers.

4. Query Processing

The main objective of the query processing component of our system is to robustly process queries over a P2P network. The main contribution of this part of our work is a cache-driven query processor which is capable of:

- supporting both partial and complete answers to a data query using a combination of live and cached data thereby allowing queries to be answered even if some of the data sources are unavailable when the query is posed.
- adaptive query processing based on time and freshness constraints.

Before describing how these capabilities are achieved, we define our data caching model. A schema-based cache is a store of results to specific queries posed on a given public schema at a specific time. It is used for fast look-up of already obtained query results without the overhead of re-executing the query over the data sources on the network. Formally, a data cache C of a public schema PS consists of one or more cache entries $c \in C$ of the form $c = \langle PS, Q, R, T \rangle$ where PS is the name of the public schema, Q is a query, R is the result object, and T is the time at which the cache entry is committed. We write $c[PS]$ to refer to the schema entry of c, $c[Q]$ the query entry of c, *etc*.

4.1. Query Answering Based on Cached Data

Although it is likely that the data stored in a data cache differ from the live data of the corresponding data source, it is desirable in many applications to return a result based on cached data rather than no result at all, or to wait a very long time for a result.

To derive a cached result for a query, we analyse each cached entry c_i to match it against a query Q. In principle, entry c_i satisfies Q if $c_i[Q]$ shares with Q a set of sub-goals and the predicates which are associated to these sub-goals do not conflict. We then combine all the matching entries to produce a *maximal* result for the query. However, if there are no matches then we conclude that the query could not be answered from the cache.

The above technique is useful for answering a query over the data cache of a given public schema on a *specific* peer. If we view the network as a whole then there could be several instances of the data associated with the same sub-goal existing on different peers. To obtain the maximal result, as many of these data caches as possible are interrogated, until either no more cached entries can be found or we exhaust a time-out that we place on query evaluation.

Consider the scenario shown in Figure 2 based on the example in the introduction We will assume that all base station A ($\mathsf{BS_A}$)'s data sources have been integrated to form the public schema PS_A and base station B ($\mathsf{BS_B}$)'s data sources have been integrated to form the public schema PS_B. We will further assume that a bidirectional BAV pathway has been defined between PS_A and PS_B. $\mathsf{BS_A}$ and $\mathsf{BS_B}$ are in intermittent contact with the PDAs in their region which periodically store data in a cache on each of the base stations.

A researcher at $\mathsf{BS_B}$ wishes to get an update on the movement of a particular animal population. She poses a query q on her local schema, S_B. This is transformed to a query on PS_B using the pathway $S_B \rightarrow PS_B$. If the data link between $\mathsf{BS_B}$ and $\mathsf{PDA_{B1}}$ is up the query is transformed to a query on $S_{PDA_{B1}}$ using the pathway $PS_B \rightarrow S_{PDA_{B1}}$. This query is then forwarded

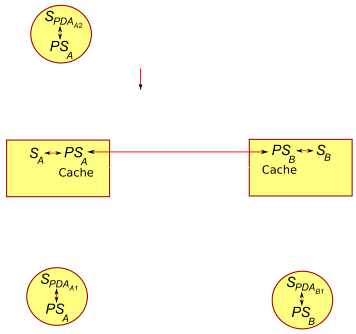

Figure 2. Cache-driven Query Processing

to the PDA for execution. If the data link is down then results from the cache on $\mathsf{BS_B}$ are used. To get results from $\mathsf{BS_A}$, q is transformed into q' expressed over the public schema PS_A using the pathway $PS_B \rightarrow PS_A$. A similar process to that described for $\mathsf{BS_B}$ then occurs at $\mathsf{BS_A}$. If the links to the PDAs are up we can get live data, otherwise cached data is used. The query could be propagated to a number of other base stations in diverse geographical locations, if required. We see in the next section how to control this propagation. Finally, the results are combined to give a maximal result. Assuming that there are some previously cached results, we can answer the query even if the links to all the PDAs are down.

As seen above, a user query is generally propagated along to all the peers that either directly implement the query schema or a schema, which is in the transformation pathway(s) of the public schema. Therefore, if the query schema is popular and there are a large number of peers implementing it then either the result set may be very large or it will take a long time to process the query. It may also be that peers are very far apart or only connected with slow network connections. To reduce the amount of time a query takes to return a result, we introduce a user-configurable **freshness filter** which limits the age of the result entries. To ensure that query processing *always* terminates, we define a **time** factor for each query. This effectively limits the length of the query path. The next section will explain these improvements in greater detail.

4.2. Constrained Queries

4.2.1. Timed Query. In the previous scenario, if the researcher had an unlimited amount of time, she could wait until the links to all the PDAs were up and get live results. In reality however, there will generally be an upper bound on the **total processing time** of a query and when this is exceeded the user must make do with a cached result.

The total processing time $T(Q)$ is the sum of two time factors. The first factor is the message time $T_m(Q)$ which is accumulated as a query is forwarded along its execution path on the network. The second time factor is node time $T_n(Q)$ which is the time it takes to process a query at each node. In general, it is only practical to estimate the time $T(Q)$ for each query because network conditions tend to vary over time. $T(Q)$ can be estimated using the timing values recorded from past transactions. This estimation can be used to guide the user when specifying the upper bound on the query processing time of a query. We call a query, which carries this upper bound a *timed* query.

To process a timed query, a peer P, first checks the time bound against its local clock. If the bound has been exceeded then P will not proceed further and will send a *time-exceeded* response back to the source. However, if there is time left then P will attempt to process the query by deciding which part of the query is to be executed locally and which part will be forwarded to another peer on the network. To execute a P2P sub-query, P will keep a *liveness* timer so that if no results are returned before the timer is expired it will resort to using the data cache to find an answer for the query.

4.2.2. Freshness-constrained Query. A freshness constraint concerns the age of a query result in the cache. A freshness constraint is defined as a time interval Δ_t from 'now' (relative to the query time) during which certain data is valid. We can determine how fresh a cached entry c is from the value of its time stamp $c[T]$. If a data source also supports time-stamping then the freshness constraint can also be used to restrict the amount of live data we would like to query from that source.

To process a freshness-constrained query, a peer node applies a *freshness filter* to its data scanning operations to retrieve only those tuples which are at most Δ_t old.

5. Conclusions

We have shown how we can set up a robust P2P data exchange network where peers can define, publish and integrate public schemas. We showed that this network can then be used to ask queries over the public schemas of the relevant peers and thereby exchange data. We developed a cache-driven query processing component to overcome query anomalies which may arise from query execution failures or result unsatisfiability.

We have implemented a prototype of the system described above by extending the AUTOMED software platform (http://www.doc.ic.ac.uk/automed). So far, our testing has been limited to laboratory experiments, but we have successfully shown how our system can be used to integrate and exchange data between peers with heterogeneous data sources and overcome peer and network failure by using the cache. We hope to run larger scale experiments in the future.

References

[1] A. Y. Halevy, Z. G. Ives, D. Suciu, and I. Tatarinov, "Schema mediation in peer data management systems," in *Proc. ICDE'03*. IEEE, 2003.

[2] D. Calvanese, E. Damagio, G. De Giacomo, M. Lenzerini, and R. Rosati, "Semantic data integration in P2P systems," in *Proc. DBISP2P*, 2003.

[3] C. Comito, S. Patarin, and D. Talia, "A semantic overlay network for p2p schema-based data integration," in *ISCC*, 2006, pp. 88–94.

[4] P.J. McBrien and A. Poulovassilis, "Defining peer-to-peer data integration using both as view rules," in *Proc. DBISP2P, at VLDB'03*, 2003, pp. 91–107.

[5] M. Boyd, S. Kittivoravitkul, C. Lazanitis, P. McBrien, and N. Rizopoulos, "AutoMed: A BAV data integration system for heterogeneous data sources," in *Proc. CAiSE'04*, ser. LNCS, vol. 3084. Springer, 2004, pp. 82–97.

[6] A. P. Sheth and J. A. Larson, "Federated database systems for managing distributed, heterogeneous, and autonomous databases," *ACM Comput. Surv.*, vol. 22, no. 3, pp. 183–236, 1990.

[7] A. Poulovassilis and P. McBrien, "A general formal framework for schema transformation," *Data and Knowledge Engineering*, vol. 28, no. 1, pp. 47–71, 1998. [Online]. Available: http://www.elsevier.nl/

[8] M. Boyd and P. McBrien, "Comparing and transforming between data models via an intermediate hypergraph data model," *Journal on Data Semantics*, vol. IV, pp. 69–109, 2005.

[9] G. Wiederhold, "Mediators in the architecture of future information systems," *IEEE Computer*, vol. 25, no. 3, pp. 38–49, March 1992.

[10] P. McBrien and A. Poulovassilis, "Data integration by bi-directional schema transformation rules," in *Proc. ICDE'03*. IEEE, 2003, pp. 227–238.