"Pay-as-you-go" Processing for Tracing Queries in a P2P Record Exchange System

Fengrong Li[1], Takuya Iida[1], and Yoshiharu Ishikawa[2]

[1] Graduate School of Information Science, Nagoya University
[2] Information Technology Center, Nagoya University
Furo-cho, Chikusa-ku, Nagoya 464-8601, Japan
{lifr,iida}@db.itc.nagoya-u.ac.jp, ishikawa@itc.nagoya-u.ac.jp

Abstract. In recent years, data provenance or lineage tracing which refers to the process of tracing the sources of data, data movement between databases, and annotations written for data has become an issue of acute importance in database research. Our research concerns the data provenance issue in peer-to-peer (P2P) networks where duplicates and modifications of data occur independently in autonomous peers. To ensure reliability among the data exchanged in P2P networks, we have proposed a reliable record exchange framework with tracing facilities based on database technologies [14, 15]. Tracing operations in the system are executed as distributed recursive queries among cooperating peers in a P2P network. The framework is based on the "pay-as-you-go" approach in which the system maintains a minimum amount of information for tracing with a low maintenance cost and the user pays the cost when he or she issues a tracing query to the system. This paper focuses on the query processing strategies used in the system. We propose two alternative query processing strategies and compare their characteristics on the basis of performance.

1 Introduction

Data provenance is a facility or an activity that helps database users to interpret database contents and it enhances the reliability of the data stored[6, 7, 19]. Typical questions addressed by data provenance include, "Why is this data located in the database?" and "Where did this data originate from?" The notion of data provenance is important, especially in the sharing and exchange of scientific data, and practical and theoretical methodologies for describing, querying, and maintaining provenance information have been proposed.

In our research, we focus on the data provenance issue in relation to information exchanges in peer-to-peer networks. *Peer-to-peer* (*P2P*) networks are widely used in various applications such as file exchange, user communication, and content distribution. A P2P network provides a flexible and scalable solution for data exchange, but it also brings with it a critical problem; since the copying and modification of data are performed independently by autonomous peers without specific central server control, it is difficult to determine how data is exchanged among the peers and why the data is located in a particular peer. This results in a lack of reliability in the data exchanged.

In [14, 15], we proposed the concept of a *reliable P2P record exchange system*, where a *record* means a tuple-structured data item that obeys a predefined schema globally shared in a P2P network. An important feature of the P2P record exchange system is that it is based on the database technologies that are utilized to support the notion of *traceability*. Records are exchanged between peers, and peers can modify, store, and delete their records independently. To ensure the reliability of the exchanged data, we assume that each peer maintains its own relational tables for storing record exchange and modification histories to facilitate traceability. To make the tracing process easy, the system provides an abstraction layer which virtually integrates all distributed relations and a datalog-like query language for writing tracing queries in an intuitive manner. The system employs a "pay-as-you-go" approach [12] for tracing; the system performs the minimum tasks required to maintain information for tracing and the user pays the cost when he or she issues a tracing query.

In this paper, we focus on the issue of query processing strategies, which was not fully covered in our earlier papers [14, 15]. We show two alternative query processing strategies and compare their properties. The remainder of this paper is organized as follows. Section 2 describes the fundamental framework of the P2P record exchange system. Section 3 presents our "pay-as-you-go" query processing strategies. Section 4 shows the experimental results for two query processing strategies. Section 5 reviews related work. Finally, Section 6 summarizes the conclusions of the paper and addresses future work.

2 P2P Record Exchange

2.1 Background and Motivation

Rapid progress in many specialized areas such as molecular biology and computational science leads to a vast and constantly increasing amount of data sharing. For example, in the field of molecular biology, there exist a number of huge databases that are exchanged and shared on a network. One of the problems in this situation is that the users (research laboratories and institutes) copy and modify the original data and provide the resulting data to other users. The process may include the activity of *curation* [6], in which the data is corrected and/or annotated based on professional knowledge, new experimental results, and so forth. Due to the copying, modification, and exchange of data performed by autonomous users, it becomes difficult to know the original source of the data and the reason why the data is located in a particular database. This is a problem of *data provenance*.

A *peer-to-peer* (*P2P*) network is a technology for supporting flexible and efficient data sharing among autonomous peers, and many systems and proposals exist [1, 3] but they do not support the notion of data provenance. For reliable data sharing in a P2P network, we want to know, for example, the original creator of the given data and the path of the data in circulation before reaching the current peer. The current P2P data exchange systems, however, do not provide such information to users, which means that the users cannot fully rely on the exchanged data.

With this background, we proposed the concept of a *traceable P2P record exchange system* [14, 15] in which tuple-structured *records* are exchanged in a P2P network[3]. We assume that each peer corresponds to a user and maintains the records owned by the user. Each record has the same structure, which is defined by a predefined schema and is globally shared within the network.

As an example, assume that information about novels is shared among peers in a P2P network. Figure 1 shows an example record set `Novel` owned by some peer that consists of four attributes: `title`, `author`, `language`, and `year`. Other peers also maintain their `Novel` records with the same structure, but their contents are not the same.

title	author	language	year
Pride and Prejudice	Jane Austen	English	1813
Madame Bovary	Gustave Flaubert	French	1857
War and Peace	Leo Tolstoy	Russian	1865

Fig. 1. Example record set `Novel`

For ease of presentation, we simplify the example shown in Fig. 1 and assume that each peer maintains a `Novel` record set that has two attributes `title` and `author`. Figure 2 shows three record sets maintained by peers A to C. Each peer maintains its own records and wishes to incorporate new records from other peers in order to enhance its own record set.

Peer A

title	author
t1	a1
t3	a3

Peer B

title	author
t1	a1
t4	a4

Peer C

title	author
t1	a1

Fig. 2. Record sets in three peers

In our record exchange framework, every peer can act as a provider and a searcher. A peer can find desired records in other peers by issuing a query. In addition, a peer can register the retrieved or created records into the local record management system if it wants, and can modify and delete records in the local system. For example, the record `(t1, a1)` in peer A in Fig. 2 may have been copied from peer B and registered in peer A's local record management system.

A *traceability problem* occurs, for instance, when peer A wishes to ask the question: "Was the novel `t1` actually written by the author `a1`?" Peer A must check the correctness of the record `(t1, a1)` in Fig. 2 and try to find evidence

[3] We use the term "Record exchange" differently from that of *data exchange* [13]; the latter is the problem of taking data that obeys a source schema and creating data under a target schema that reflects the source data as accurately as possible.

supporting its validity. However, finding such evidence from a P2P network is quite difficult. In the next subsection, we introduce our framework of a traceable P2P record exchange system.

2.2 Traceable P2P Record Exchange System

The Design Strategy In our traceable record exchange framework, peers can behave autonomously and exchange information when required. Peers are highly distributed and there is no central server that can answer tracing queries using the complete histories of all the records in the network. To provide a traceability facility in our system, we have adopted the following strategies:

– All the information required for tracing is maintained in distributed peers; each peer maintains its own historical information, which consists of creation (registration), modification, deletion, and exchange histories, related to the peer itself.
– When a tracing query is issued, the query is processed by coordinating related peers in a distributed and recursive manner. Historical information stored in related peers is collected for answering the query.
– To help users write queries, we provide an abstraction layer (the *logical layer*) in which all the data in a P2P network is integrated in global virtual views (in contrast, the underlying layer mentioned above is called the *physical layer*). User queries are written with a modified version of the *datalog* query language [2].

The underlying idea behind this design strategy is the notion of "pay-as-you-go" data integration [12]. Since copies and updates are performed everywhere in a distributed autonomous P2P network, it is quite costly to maintain the historical information in a central server or several hub servers. Instead, each peer in our framework maintains only minimal historical information related to the peer. This means that we need to aggregate the required historical information from the distributed peers when a tracing query is issued from a user; the user should pay the cost when he or she traces information. Although the tracing cost becomes high, the strategy makes sense for the following reasons:

– It is natural to assume that tracing queries are not issued often when compared to the total maintenance operations. In particular, if a peer does not have a requirement for tracing, our approach ensures that the total cost for the peer is relatively small.
– In a P2P network, each peer acts autonomously and a peer may not be interested in all the information in the network. For example, a group of peers may copy records only from the group members. In that case, local historical information alone would be sufficient for tracing.

As an alternative strategy, we may be able to take the approach that every record brings its historical information when it is copied among peers. This approach simplifies tracing for some cases, but 1) historical information could be large for some records, and 2) some types of tracing queries (e.g., "Who copied my records?") cannot be processed. In contrast, our approach is flexible and as such it supports various types of tracing queries.

In the following, we briefly summarize the notions of physical and logical layers.

The Physical Layer In the physical layer, each peer maintains the minimum amount of information that is required to represent its own record set and local tracing information. It consists of four relations. For example, peer A shown in Fig. 2 contains the following relations:

– `Data[Novel]`: It maintains records owned by peer A. Figure 3 shows an example. Every record has its own record id for maintenance purposes. Each record id should be unique in the entire P2P network. Note that there are additional records compared to Fig. 2; they are *deleted* records and are usually hidden from the user. They are maintained for data provenance.

title	author	id
t1	a1	#A001
t2	a2	#A002
t3	a3	#A003

Fig. 3. `Data[Novel]` of peer A

from_id	to_id	time
–	#A002	4/10/08
#A002	#A003	8/10/08

Fig. 4. `Change[Novel]` of peer A

– `Change[Novel]`: used to hold the creation, modification, and deletion histories. Figure 4 shows an example for peer A. Attributes `from_id` and `to_id` express the record ids before/after a modification. Attribute `time` represents the modification timestamp. When the value of the `from_id` attribute is a null value (–), it means that the record has been created at the peer. Similarly, when the value of the `to_id` attribute is a null value, it means that the record has been deleted.

– `From[Novel]`: records which records were copied from other peers. When a record is copied from another peer, attribute `from_peer` contains the peer name and attribute `from_id` has its record id at the original peer. Attribute `time` stores the timestamp information. The first tuple in Fig. 5 shows the record with id `#A001` is a copy of the record with id `#B001` at peer B.

id	from_peer	from_id	time
#A001	B	#B001	3/2/08

Fig. 5. `From[Novel]` of peer A

id	to_peer	to_id	time
#A001	C	#C001	9/16/08

Fig. 6. `To[Novel]` of peer A

– `To[Novel]`: plays an opposite role to that of `From[Novel]` and stores information about which records were sent from peer A to other peers. Fig. 6 shows the `To[Novel]` relation of peer A.

Although `From[Novel]` and `To[Novel]` contain duplicated information, the duplicates are stored in different peers. For example, for the tuple of `From[Novel]` in

Fig. 5, a corresponding tuple `(#B001, A, #A001, 3/2/08)` exists in `To[Novel]` at peer B. When the record is registered at peer A, `From[Novel]` at peer A and `To[Novel]` at peer B are updated cooperatively to preserve the consistency.

In our framework, every peer maintains the four relations in its local record management system which is implemented using an RDBMS. Since historical information is distributed among peers, it is not easy to aggregate related information when a user needs to trace based on his or her requirement. For ease of understanding and writing tracing queries, we provide an abstraction layer called the *logical layer*, which is described next.

The Logical Layer In the logical layer, three relational views are constructed by unifying all the relations in the peers. Relation `Data[Novel]` in Fig. 7 expresses a view that unifies all the `Data[Novel]` relations in peers A to C shown in Fig. 2. The `peer` attribute stores peer names. Relation `Change[Novel]` shown in Fig. 8 is also a global view which unifies all `Change[Novel]` relations in a similar manner.

title	author	peer	id
t1	a1	A	#A011
t2	a2	A	#A002
t3	a3	A	#A003
t1	a1	B	#B001
t2	a2	B	#B002
t1	a1	C	#C001

Fig. 7. View `Data[Novel]`

peer	from_id	to_id	time
A	–	#A002	4/10/08
A	#A002	#A003	8/10/08
B	–	#B001	2/15/08
⋮	⋮	⋮	⋮

Fig. 8. View `Change[Novel]`

`Exchange[Novel]` shown in Fig. 9 unifies all the underlying `From[Novel]` and `To[Novel]` relations in a global view. Attributes `from_peer` and `to_peer` express the origin and the destination of record exchanges, respectively. Attributes `from_id` and `to_id` contain the logical ids of the exchanged records in both peers.

from_peer	to_peer	from_id	to_id	time
B	A	#B001	#A001	3/2/08
A	C	#A001	#C001	9/16/08

Fig. 9. View `Exchange[Novel]`

2.3 Query Specification

When a tracing requirement occurs, we need to aggregate the related historical information stored in the distributed peers. Since recursive processing is required to collect historical information, our framework provides a modified version of *datalog* query language [2]. We now present some examples of tracing queries.

Query 1: Suppose that peer A holds a record with title `t1` and author `a1` and that peer A wants to know which peer originally created the record:

```
BReach(P, I1) :- Data[Novel]('t1', 'a1', 'A', I2),
                 Exchange[Novel](P, 'A', I1, I2, _)
BReach(P1, I1) :- BReach(P2, I2), Exchange[Novel](P1, P2, I1, I2, _)
Origin(P) :- BReach(P, I), NOT Exchange[Novel](_, P, _, I)
Query(P) :- Origin(P)
```

`P` and `I1` are variables and '_' indicates an anonymous variable. Relation `BReach` defined by the first two rules means "Backward Reachable". It recursively traverses the arriving path of tuple `(t1, a1)` until it reaches the origin. The third rule is used for finally determining the originating peer name; it should be reachable from peer A and should not have received the record from any other peer. The last rule represents the final result expected by the user. Note that the query is written using the three views in the logical layer. The user does not need to consider how the actual data is distributed among the peers.

Query 2: This query detects whether peer C copied the record `(t1, a1)` owned by peer B or not:

```
Reach(P, I1) :- Data[Novel]('t1', 'a1', 'B', I2),
                Exchange[Novel]('B', P, I2, I1, _)
Reach(P, I1) :- Reach(P1, I2), Exchange[Novel](P1, P, I2, I1, _)
Query(I) :- Reach('C', I)
```

Relation `Reach` means "Reachable". After the execution, it will contain all of the peer names which copied the target record of peer B. If `Reach` contains peer name C, its corresponding record id is returned. Otherwise, an empty relation is returned to the user and it means that peer C did not copy the record offered by peer B.

Note that Query 1 and 2 perform backward and forward traversals of the provenance information, respectively. Datalog is so flexible that we can specify various queries types. Refer to [14, 15] for details of the various tracing queries available.[[Was this the intended meaning?]]

3 Query Processing

Although we decided to adopt the "pay-as-you-go" approach where the user pays the cost when performing tracing queries, the efficiency of query processing is still quite an important factor. In this paper, we consider applying two major strategies for datalog query execution, the *seminaive method* and the *magic set method*, to our context. We have outlined the query processing approach based on the seminaive method in [14], but the details were not discussed. In the following, we use Query 2 shown above as an example query for purposes of illustration. Query 1 can be easily executed using the seminaive strategy shown below; we omit the details here.

3.1 Evaluation Based on Seminaive Method

The *seminaive method* is based on simple iterative processing, but ensures that no redundant evaluations are performed to process a recursive datalog query. It modifies a given datalog program using *delta* relations, which will contain the difference of tuples between previous and current iterations. Only the new tuples computed in the previous iteration are used as input in the next iteration. The process is repeated until it reaches the *fixpoint* at which no new tuples are produced. For the details of this method, refer to [2]. A datalog query is executed within a single database system in the traditional context of deductive databases. In contrast, a tracing query in our P2P record exchange framework is executed by the cooperation of distributed peers using query forwarding.

Consider that Query 2 is issued at peer B. Since the query is described in datalog using virtual views in the logical layer, it is necessary to transform it while considering the organization of the physical layer. Query 2 is translated into the following *physical layer query*:

```
Reach(P, I1) :- Data[Novel]@'B'('t1', 'a1', I2),
                To[Novel]@'B'(I2, P, I1, _)
Reach(P, I1) :- Reach(P1, I2), To[Novel]@P1(I2, P, I1, _)
Query(I) :- Reach('C', I)
```

The transformation method is not so difficult and is omitted here. A notation such as `Data[Novel]@'B'` indicates a physical relation at a specific peer, in this case it means the relation `Data[Novel]` at peer B. Similarly, `To[Novel]@P` represents the relation `To[Novel]` at peer `P`. Note that `P` is a variable peer name. Resolution from a peer variable to the corresponding exact peer name is resolved at query execution time.

After the transformation, the query is executed using the extension of the seminaive method. The query processing steps based on the seminaive method is summarized in Fig. 10. Query 2 is executed as follows. First, a local query, the first rule for Query 2, is executed on the local record management system of peer B and the result is stored in a delta relation $\Delta_{\mathtt{Reach}}$. If we assume the example shown in Fig. 9, $\Delta_{\mathtt{Reach}}$ will contain a tuple `(A, #A001)`, which is a copy of the record `(t1, a1)` at peer B. Next, the given query is compiled into a *seminaive evaluation program*:

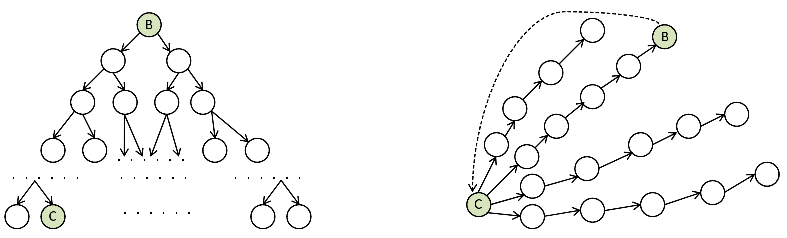
$$
\begin{array}{l}
\mathtt{temp_{Reach}(P,\ I1)\ :\!-\ }\Delta_{\mathtt{Reach}}\mathtt{(P1,\ I2),\ To[Novel]@P1(I2,\ P,\ I1,\ \_)}\\
\mathtt{temp_{Query}(I)\ :\!-\ }\Delta_{\mathtt{Reach}}\mathtt{('C',\ I)}\\
\mathtt{Reach\ :=\ Reach} \cup \Delta_{\mathtt{Reach}}\\
\mathtt{Query\ :=\ Query} \cup \Delta_{\mathtt{Query}}\\
\Delta_{\mathtt{Reach}}\mathtt{\ :=\ temp_{Reach}\ -\ Reach}\\
\Delta_{\mathtt{Query}}\mathtt{\ :=\ temp_{Query}\ -\ Query}
\end{array}
$$

Then peer A tries to execute this program until it reaches the fixpoint. However, the program cannot be executed further since the execution needs to access `To[Novel]` relation at peer A for the computation. Therefore, the program is *forwarded* to peer A with the current temporal results (`Reach`, `Query`, $\Delta_{\mathtt{Reach}}$, and $\Delta_{\mathtt{Query}}$). Next, peer A continues execution based on the given program and data then forwards the query and the partial data (i.e., {`(A, #A001), (C, #C001)`}) to peer C. Since peer C cannot forward the query further, the execution is finished and the result is finally returned to peer B.

```
 1. /* Initial Peer */
 2. Execute local queries to obtain initial results for target relations.
 3. Compile the given query into a seminaive evaluation program.
 4. Initialize each $\Delta$ relation by an empty relation.
 5. Forward the query to related peers with partial evaluation results.
 6. /* Succeeding Peer */
 7. Execute the given program and data using the local database.
 8. If next peers exist, forward the query to them.
 9. Otherwise, continue the evaluation until it reaches the fixpoint.
10. If it reaches the fixpoint, the result is returned to the former peer.
```

Fig. 10. Query evaluation based on the seminaive method

Although this example only forwards twice, the seminaive method generally visits all the peers which copied the record `(t1, a1)` offered by peer B for Query 2. Since records provided by a certain peer are often copied by multiple peers, it should visit a number of peers. If we assume that the target record provided by a peer is copied by n peers and m forwards are performed along every path started from peer B, the process should contain n^m peers in total. This situation is illustrated in Fig. 11(a).



(a) Seminaive (b) Magic set

Fig. 11. Processing Query 2 based on two strategies

3.2 Evaluation Based on Magic Set Method

The *magic set* technique is a well-known strategy for the efficient execution of datalog programs [2]. By modifying a given program, it simulates "selection pushdown" for the top-down evaluation approach within the bottom-up evaluation approach. We describe the query processing approach using Query 2 as an example.

First, we transform Query 2 into the following query according to the magic set rewriting rules. Magic set rewriting is a two-step transformation in which the first phase consists of constructing an adorned rule set that is derived from the original database with respect to the binding pattern of the query, and the second phase is the actual magic set rewriting. Due to space constraints, we refrain from presenting the magic set approach in detail and instead present the translated result of the query.

```
Reach(P, I1) :- magic_Reach(P, I1), Data[Novel]@'B'('t1', 'a1', I2),
                From[Novel]@P(I1, 'B', I2, _)
Reach(P, I1) :- magic_Reach(P, I1), Reach(P1, I2),
                From[Novel]@P(I1, P1, I2, _)
magic_Reach(P1, I2) :- magic_Reach(P, I1), From[Novel]@P(I1, P1, I2, _)
magic_Reach('C', I):-
Query(I) :- Reach('C', I)
```

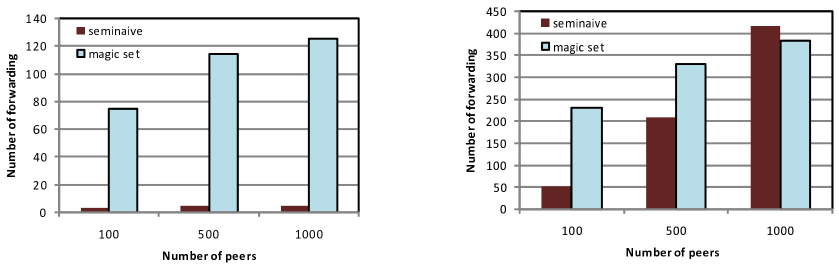
Once a program is modified by the magic set-based rewriting, we can execute the program using the seminaive method. The behavior of the modified program is, however, quite different from the normal seminaive method. In this case, the fourth rule above defines the actual starting point; it first triggers the evaluation of the third rule. This means that we need to evaluate `From[Novel]@'C'` since the tuple obtained by the fourth rule assigns peer name 'C' to variable `P` in the third rule. As the third rule cannot be executed in peer B, peer B forwards the program to peer C. Given the forwarded program, peer C starts execution, but it requires `From[Novel]@'A'` during the evaluation of the third rule; this is due to the fact that peer C copies a record from peer A as shown in Fig. 9. In summary, the additional magic predicate `magic_Reach` requires the following: for each record in peer C, we should traverse the path from peer C to the origin of the record. Note that the forwarding direction is opposite to that in the case of the normal seminaive method. When each forwarding process reaches the origin, the process will return along the same path to peer C. The contents of relation `Reach` are fulfilled during this returning phase. This magic set-based evaluation approach is illustrated in Fig. 11(b). The entry that satisfies the given query corresponds to the path from peer C to peer B; the other paths finally fail.

We roughly estimate the cost of the query. Assume that peer C has l records. For each record in peer C, we need to traverse its path to the source. Since we follow the path towards the ancestor, the path does not contain branches. If we assume that the path length is a constant value on average, then the total fowarding cost would be $O(l)$. This simple analysis shows that the magic set-based strategy would be a promising method for Query 2. In the next section, we check whether this assumption holds by comparing the performance of two strategies.

4 Experimental Results

The purpose of the experiments is to observe the behaviors of two query processing strategies using a simple P2P record exchange model. We assume that Query

2 is given and executed based on the strategies shown in the previous section. The simulation model is summarized as follows. We first create $N = 100$ peers and $M = 500$ records; each record is randomly assigned to one of the peers. We assume that records consist of two classes: "hot" records (20%) and normal records (80%). Hot records are more likely to be exchanged; when a peer wants to get a record from other peer, a hot record is selected with 80% probability. We perform random record exchanges until each peer exchanges $L = 50$ records on average. This means that we perform 5,000 record exchanges for this parameter setting. Figure 12 shows the results. In this figure, we added the experimental results for $N = 500$ and $N = 1000$. Their experimental parameters are same except for N.



(a) Average case (b) Maximum case

Fig. 12. Query forwarding cost for Query 2

Figure 12(a) shows the average numbers of query forwarding. As shown in the figure, the behavior of the magic set approach is worse when compared to that of the seminaive approach. This is not well suited to our analysis in the previous section. The main reason is that the average number of branches is not high in our simulation model. Although Fig. 11(a) illustrates the exponential growth of query forwarding, the actual number does not increase exponentially. In contrast to the seminaive method, the magic set method needs to follow multiple paths starting from the initiating peer (peer C in our example). That is the main factor behind the high query cost.

Figure 12(b) shows the same experimental results, but with the highest numbers of query forwarding shown. As can be seen, the cost of the magic set method improves for $N = 1000$. The reason is that $N = 1000$ represents the case of frequent record exchanges. In this case, a hot record is copied by a large number of peers so that the number of branches of forwarding paths becomes quite large. This makes the performance of the seminaive method worse.

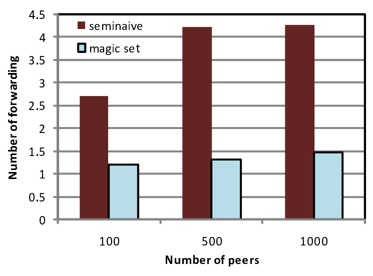
Although the magic set has poor performance for the above experiment, it is quite effective in some situations. See the following Query 3, which is a modified version of Query 2. It checks whether the record `(t1, a1)` in peer C actually came from peer B.

Query 3 Is the record (t1, a1) in peer C a copy of (t1, a1) in peer B?

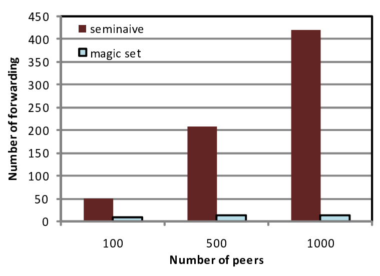
```
Reach(P, I1) :- Data[Novel]('t1', 'a1', 'B', I2),
                Exchange[Novel]('B', P, I2, I1, _)
Reach(P, I1) :- Reach(P1, I2), Exchange[Novel](P1, P, I2, I1, _)
Dup(I) :- Reach('C', I), Data[Novel]('t1', 'a1', 'C', I)
Query(I) :- Dup(I)
```

We can process this query using the magic set-based approach as in Query 2, but the behavior is different. As in Query 2, first the query is sent to peer C and peer C then forwards the query to the ancestor. In contrast to Query 2, there is only one forwarding path because the additional constraint of the third rule is used for determining the start record.

Figure 13 shows the experimental results. The cost for the seminaive method is same as in Query 2. On the other hand, the cost of magic set method for Query 3 is rather small, especially in the case of the maximum number of query forwarding.



(a) Average case



(b) Maximum case

Fig. 13. Query forwarding cost for Query 3

From the experiments above, we can say that the choice of query strategy should depend on the query, the record exchange history, the organization of the P2P network, and so forth. Each of the two query processing strategies, the seminaive method and the magic set method, works well in some situations, but neither is an all round player. For an efficient query evaluation, we need to develop effective cost evaluation and optimization methods.

5 Related Work

The data provenance field is quite wide and covers data warehousing [9,10], uncertain data management [4,20], curated databases [6], and other scientific fields such as bioinformatics [5]. In this area, one of the well-known projects is the *Trio* project at Stanford University which considers both uncertainty and lineage issues [20]. This paper is devoted to the data provenance issue in P2P information exchange where data provenance is important, and there are as yet few proposals for this topic.

There are a variety of research topics regarding P2P databases, such as coping with heterogeneities, query processing, and indexing methods [1]. One related project that deals with our problem is the ORCHESTRA project [11, 13, 17], which aims at the collaborative sharing of evolving data in a P2P network. In contrast to their approach, our research focuses on a simple record exchange scenario and does not consider schema heterogeneity. One of the features of our framework is to employ database technologies as the underlying foundation to support reliable P2P record exchange. Our approach allows users to write various types of tracing queries using datalog, and the queries can be executed in a P2P network.

Another related field is *dataspace management* [12]. This is an emerging new research field in the area of databases and focuses on more flexible information integration over the network in an incremental, "pay-as-you-go" fashion. Many application contexts involving multiple heterogeneous data sources (e.g., personal information management) do not necessarily require full integration of information sources; it may be reasonable to perform information integration dynamically when a user request is issued. Such a concept motivates the "pay-as-you-go" approach to information integration. Since our approach focuses on the integration of historical information stored in distributed peers, the "pay-as-you-go" approach will work well because it does not interfere with the autonomy of the peers and the tracing requests do not occur so often. The approach has an additional benefit in that it allows flexible tracing query representation using the datalog query language.

Finally, we mention the query execution strategies. The seminaive method and the magic set method are well-known query processing strategies for deductive databases [2]. Query processing based on the deductive database approach has not been a hot topic in recent years, but the situation is now changing. As proven in the *declarative networking* project [8, 16, 18], declarative recursive queries are very powerful in writing network-oriented database applications such as sensor data aggregation. In contrast to this approach, our focus is on compact and understandable tracing query specifications. Since our framework shares the requirement of efficient query processing with the declarative networking project approach, it will be possible to extend our query processing method by considering that project 's proposals.

6 Discussion and Conclusions

In this paper, we discussed the query processing methods for our P2P record exchange framework. The approach is based on the concept of "pay-as-you-go" style data integration where we aggregate historical information distributed in autonomous peers when a tracing query is issued from a user. The approach has the benefits of a low maintenance cost of historical information, full autonomy of peers, and flexible query specifications. The datalog-based query specification allows us to write tracing queries in a compact manner. A tracing query written in datalog is evaluated by cooperating peers using query forwarding. The recursive nature of tracing is well suited to the deductive approach.

In this paper, we compared two popular query-processing methods in our context: the seminaive method and the magic set method. The experimental results show that both methods have pros and cons. For example, an appropriate execution strategy depends on the given query, the P2P network organization, the record exchange behaviors, and so forth. Query cost estimation and query optimization are important future research issues to address in order to enhance our framework.

Since this paper focused on the query processing issue, we have omitted other important problems in our framework. Several future research issues are summarized as follows.

- Full specification of complete query processing strategies: Although this paper showed the query processing strategies and their experimental evaluations, we need to enhance the strategies to handle more complex tracing queries. This will make the effectiveness and limitation of the declarative language-based approach clearer.
- Enhancement of the query language for practical tracing: Considering the practical requirements of tracing, we need to incorporate additional features and constructs into our language. This means that the query processing strategies should also be enhanced.
- Efficient coupling with DBMSs: In implementing our framework, we assume that a local record management system in each peer is implemented using a conventional RDBMS. We would like to effectively use the powerful and robust DBMS functionalities based on the tight coupling of the record management system and the underlying RDBMS.
- Efficient implementation using replication and caching: Data replication and caching are popular techniques for efficient query processing. In our case, the data used for tracing queries was historical information which can be replicated among peers. This will reduce the query processing cost, but the storage and maintenance cost will increase. We need to consider how to handle the trade-off considering the total cost reduction.
- Fault-tolerance: In this paper we omitted the issue of fault tolerance; however, fault tolerance is important when supporting P2P networks in which failure occurs frequently. Replication will effectively resolve the problem, but we need to consider it in detail.
- Prototype system implementation and experiments: We are currently developing a prototype system of our P2P record exchange framework. We also started to construct a P2P network simulator that can be used for simulating our prototype system in a virtual P2P network. These developments will play a positive role in the improvement of our fundamental framework.

Acknowledgments

This research was partially supported by a Grant-in-Aid for Scientific Research (19300027) from the Japan Society for the Promotion of Science (JSPS).

References

1. K. Aberer and P. Cudre-Mauroux. Semantic overlay networks. In *VLDB*, 2005. (tutorial notes).
2. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley, 1995.
3. S. Androutsellis-Theotokis and D. Spinellis. A survey of peer-to-peer content distribution technologies. *ACM Computing Surveys*, 36(4):335–371, 2004.
4. O. Benjelloun, A. Das Sarma, A. Halevy, and J. Widom. ULDBs: Databases with uncertainty and lineage. In *Proc. VLDB*, pp. 953–964, 2006.
5. D. Bhagwat, L. Chiticariu, W.-C. Tan, and G. Vijayvargiya. An annotation management system for relational databases. In *Proc. VLDB*, pp. 900–911, 2004.
6. P. Buneman, J. Cheney, W.-C. Tan, and S. Vansummeren. Curated databases. In *Proc. ACM PODS*, pp. 1–12, 2008.
7. P. Buneman and W.-C. Tan. Provenance in databases (tutorial). In *Proc. ACM SIGMOD*, pp. 1171–1173, 2007.
8. T. Condie, D. Chu, J. M. Hellerstein, and P. Maniatis. Evita raced: Metacompilation for declarative networks. In *VLDB*, pp. 1153–1165, 2008.
9. Y. Cui and J. Widom. Lineage tracing for general data warehouse transformations. In *Proc. VLDB*, pp. 471–480, 2001.
10. Y. Cui, J. Widom, and J. L. Wiener. Tracing the lineage of view data in a warehousing environment. *ACM TODS*, 25(2):179–227, 2000.
11. T. J. Green, G. Karvounarakis, N. E. Taylor, O. Biton, Z. G. Ives, and V. Tannen. ORCHESTRA: Facilitating collaborative data sharing. In *Proc. ACM SIGMOD*, pp. 1131–1133, 2007.
12. A. Halevy, M. Franklin, and D. Maier. Principles of dataspace systems. In *Proc. ACM PODS*, pp. 1–9, 2006.
13. Z. Ives, N. Khandelwal, A. Kapur, and M. Cakir. ORCHESTRA: Rapid, collaborative sharing of dynamic data. In *Proc. CIDR*, pp. 107–118, 2005.
14. F. Li, T. Iida, and Y. Ishikawa. Traceable P2P record exchange: A database-oriented approach. *Frontiers of Computer Science in China*, 2(3):257–267, 2008.
15. F. Li and Y. Ishikawa. Traceable P2P record exchange based on database technologies. In *Proc. APWeb*, Vol. 4976 of *LNCS*, pp. 475–486, 2008.
16. B. T. Loo, T. Condie, M. Garofalakis, D. E. Gay, J. M. Hellerstein, P. Maniatis, R. Ramakrishnan, T. Roscoe, and I. Stoica. Declarative networking: Language, execution and optimization. In *Proc. SIGMOD*, pp. 97–108, 2006.
17. ORCHESTRA: Managing the collaborative sharing of evolving data. http://www.csi.upenn.edu/~zives/orchestra/.
18. P2: Declarative networking. http://p2.berkeley.intel-research.net/.
19. W.-C. Tan. Research problems in data provenance. *IEEE Data Engineering Bulletin*, 27(4):45–52, 2004.
20. J. Widom. Trio: A system for integrated management of data, accuracy, and lineage. In *Proc. CIDR*, pp. 262–276, 2005.