Consistent Query Plan Generation in Secure Cooperative Data Access

Meixing Le
meile@cisco.com
Cisco Corp.

Krishna Kant
kkant@temple.edu
Temple Univ.

Sushil Jajodia
jajodia@gmu.edu
Geoerge Mason Univ.

Abstract. In this paper, we consider restricted data sharing between a set of parties that wish to provide some set of online services requiring such data sharing. We assume that each party stores its data in private relational databases, and is given a set of mutually agreed set of authorization rules that may involve joins over relations owned by one or more parties. Although the query planning problem in such an environment is similar to the one for distributed databases, the access restrictions introduce significant additional complexity that we address in this paper. We examine the problem of efficiently enforcing rules and generating query execution plans in this environment. Because of the exponential complexity of optimal query planning, our query planning algorithm is heuristics based but produces excellent, if not optimal, results in most of the practical cases.

Keywords: Rule enforcement, Consistent query planning, Cooperative data access

1 Introduction

Providing rich services to clients with minimal manual intervention or paper documents requires the enterprises involved in the service path to collaborate and share data in an orderly manner. For instance, to enable automated shipping of merchandise and status checking, the e-commerce vendor and shipping company should be able to exchange relevant information, perhaps by enabling queries to retrieve data from each other's databases. Similarly, in order to provide integrated payment and payment status services to the client, the e-commerce vendor needs to share data with the credit card companies or other vendors that specialize in payment processing. There may even be a need for some data sharing between the payment processing and shipping companies so that the issue of payment for shipping can be smoothly handled.

Traditionally, such cross enterprise data access has been implemented in ad hoc ways. In particular, incoming queries may not be allowed to directly access the databases maintained by a company, and instead handled via some intermediate module. This has the advantage of isolation but could be quite inefficient. More significantly, cross-enterprise data access is typically driven by bilateral agreements between the two parties that no other party knows anything about.

While attractive from isolation perspective, such bilateral agreements introduce a high degree of cost, complexity, and inefficiency into the processes. In particular, bilateral agreements may require more data to be exposed to other parties so that it is possible to answer complex queries that require composition of data from multiple parties. Bilateral agreements also rule out possibilities of sharing computation results between parties. For instance, if the e-commerce company needs to get information involving join of data over three parties (e.g., the e-commerce company itself, a warehouse, and a shipping company), under bilateral agreements, we have to bring the relevant data from the other two parties to the e-commerce company first and then do joins. With multiparty interactions enabled, such data may already be available. The purpose of this paper is to explore the general *multi-party collaboration* model and to develop algorithms for safely implementing the authorization rules so that only desired data can be accessed by authorized parties.

We assume that the multi-party data sharing is driven by twin consideration of business need and privacy; therefore, the rules are expected to grant sufficient privileges for answering the agreed upon set of queries but no more. We assume that the collaborating parties generally trust one another and play by the rules. Typically, this would be enforced through legal and financial provisions in the agreements, but there may still be a need to take the "trust-but-verify" approach. The verification issue is beyond the scope of this paper and will be addressed in future work. The purpose of this paper is to focus on efficient mechanisms for executing queries in what amounts to a distributed database with access restrictions. To the best of our knowledge this is the first work of its kind, even though query planning in distributed databases has been considered extensively.

Although the enterprise data may appear in a variety of forms, this paper focuses on the relational model, with *authorization rules* specifying access to certain attributes over individual relations and their meaningful joins (e.g., join over key attributes). For simplicity and schema level treatment, we do not consider tuple selections as part of the rules in this paper. The problem then is to find ways of enforcing the rules and constructing efficient query plans.

Since each party is likely to frame rules from its own perspective, the rules taken together may suffer from inconsistency, unenforceability, and other issues. The consistency problem refers to the fact that if a party is provided access to two relations, say R and S, it is very difficult to prevent it from joining these relations, but the rule may deny access to $R \bowtie S$. Our previous research has addressed this issue [12] and here we simply assume that the rules are *upwards closed*, i.e., access to R and S will automatically enable access to $R \bowtie S$. The enforceability problem can be illustrated as follows: If a party P is given access to $R \bowtie S$ but it and no other party has access to both R and S, it is not possible to actually compute $R \bowtie S$. We have examind this problem in [13]. In some cases, enforceability requires introducing a trusted third party [14] that is given sufficient access rights to perform the operation in question (e.g., $R \bowtie S$ in our example). Third parties bring in their own security risks, and we do not consider them in this paper. We instead focus on generating efficient query plans in an environment without any trusted third parties, and do so in two steps:

1) We examine each authorization rule and check whether the rule can be totally enforced (or implemented) among the collaborating parties. Since this issue has been addressed in [13], we do not focus on this step here.

2) We build a safe and efficient query plan based on the available rule enforcement steps. We discuss the complexity of finding optimal answer in our scenario, and how it differs from classical query processing. We then propose an efficient algorithm that derives query plans based on a greedy heuristic. We prove that our algorithms are both correct and complete, and experimentally show the quality of the results.

The rest of the paper is organized as follows. Following the related work in Section 2, the problem is defined formally in Section 3. Section 4 analyzes the complexity of query planning. Section 5 then describes the algorithm for generating query plans.

2 Related Work

The problem of collaborative data access has been considered in the past, and this has inspired our multi-party collaboration approach. In particular, De-Capitani, et.al. [7] consider such a model and discuss an algorithm to check if a query with a given query plan tree can be safely executed. However, this work does not address the problem of how the given rules are implemented and how the query plan trees are generated. The same authors have also proposed a possible architecture for the collaborative data access in [8] but this work does not address query planning. As we shall show shortly, *regular query optimizers cannot be used here since they do not comprehend access restrictions and may fail to generate some possible query plans.*

There are also existing works on distributed query processing under protection requirements [4, 15] which consider a limited access pattern called binding pattern. It is assumed that the accessible data is based on some input data. For instance, a party can provide names and ID's of some individuals, it may be allowed to access their medical records. This is a completely different model from ours. There are also many classical works on query processing in centralized and distributed systems [3, 11, 5], but they do not deal with constraints from the data owners, which differs from our work.

Answering queries that takes advantage of materialized views is another well investigated research direction. Some of these works focus on query optimization [9] which use materialized views to further optimize existing query plans. In our case, we need to generate a query plan from scratch. Some works use views for maintaining physical data independence and for data integration [16]. They assume the scenario where data is organized in different formats and comes from different sources, and accessing data via views may not provide the complete information to answer the queries. Using authorization views for fine-grained access control is discussed in [17], and [19] analyzed the query containment problem under such access control model. Similarly, conjunctive queries are used to evaluate the query equivalence and information containment, and the work [10] presented several theoretical results. Compared to these works, our data model is homogeneous across the parties, and our authorization model not only puts constraints based on relational views but also the interactions among collaborating parties. Consequently, generating a query plan in our scenario is even more complicated. Some results from these works can be complementary to our work and can be used to further optimize the query plans generated by our approach. However, this is out of the scope of this paper.

In addition, there are services such as Sovereign joins [2] to provide third party join services; we can think this as one possible third party model in our scenario. There is also some research [1, 6, 18] about how to secure the data for out-sourced database services. These methods are also useful for enforcing the authorization rules, but we consider the scenario without any involvement of third parties.

3 Problem and Definitions

We consider a group of collaborating parties each with its own relational database but with collectively known key attributes and authorizations that allow for useful joins among the tables. We assume that the join schema is also collectively known, and we only consider select-project-join (SPJ) queries. To enable working at the schema level, selections are treated like projections (i.e., attributes mentioned in selection predicates are assumed to be accessible). We also allow an incoming query to be answered by any party that has the required authorizations. The basic query planning problem is as follows: Given a set of authorization rules $\mathcal{R}$ on n cooperating parties, and a query q authorized by $\mathcal{R}$, find an efficient query execution plan for q that is consistent with the rules $\mathcal{R}$.

3.1 A Running Example

In the following, we use a running e-commerce scenario with four parties: (a) *E-commerce*, denoted as E, is a company that sells products online, (b) *Customer_Service*, denoted C, that provides customer service functions (potentially for more than one Company), (c) *Shipping*, denoted S, provides shipping services (again, potentially to multiple companies), and finally (d) *Warehouse*, denoted W, is the party that provides storage services. To keep the example simple, we assume that each party owns but one table described as follows. In reality, each party may have several tables that are available for collaborative access, in addition to those that are entirely private and thus not relevant for collaborative query processing.

1. E-commerce (<u>order_id</u>, product_id, total) as E
2. Customer_Service (<u>order_id</u>, issue, agent) as C
3. Shipping (<u>order_id</u>, addr, delivery_type) as S
4. Warehouse (<u>product_id</u>, location) as W

The relations are self-explanatory, with underlined attributes indicating the key attributes. In the following, we use *oid* to denote *order_id* for short, *pid* for *product_id*, and *delivery* for *delivery_type*. The possible join schema is given in figure 1. Relations E, C, S can join over their common attribute *oid*; relation E can join with W over the attribute *pid*. The relations are in BCNF, and the only FD (Functional Dependency) in each relation is the underlined key attribute determining the non-key attributes. To keep our discussion simple, we do not consider foreign keys in this paper. Foreign keys are unlikely to be used for linking data across organizational boundaries; nevertheless, our model can be easily extended to consider foreign key constraints.

3.2 Definitions and authorization model

An authorization rule r_t is a triple $[A_t, J_t, P_t]$, where J_t is called the join path of the rule, A_t is the authorized attribute set, and P_t is the party authorized to access the data.

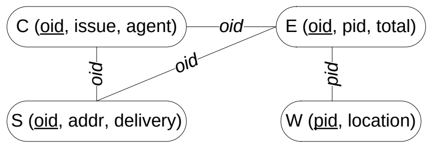

Fig. 1. The given join schema for the example

Definition 1 *A* **join path** J_t *is the result of a series of joins over the relations* $JR_t = \{R_1, R_2...R_n\}$ *with specified equi-join predicates* $(A_{l1}, A_{r1}), (A_{l2}, A_{r2})...$ (A_{ln}, A_{rn}) *among them, where* (A_{li}, A_{ri}) *are the lists of join attributes from two relations. The* **length** *of a join path is the cardinality of* JR_t*.*

The authorized attributes is given by A_t part of the rule r_t, which we assume to include all key attributes as well. Table 1 shows all the rules in our example system. (The last column specifies the party to which the authorization is given.) Since our analysis does not deal with selections directly, all attributes appearing in selection predicates are treated as projection attributes. Thus, a query q can be represented by a pair $[A_q, J_q]$, where A_q is the set of attributes appearing in the Selection and Projection predicates. For instance, the SQL query "**q:** Select $oid, total, addr$ From E Join S On $E.oid = S.oid$ Where $delivery =$ 'ground'" can be represented as the pair $[A_q, J_q]$, where A_q is the set $\{oid, total, addr, delivery\}$; J_q is the join path $E \bowtie_{oid} S$. We say $J_i \cong J_j$ (join path equivalence) if any tuple in J_i appears in J_j and vice versa. Then, a query q is **authorized** if there exists a rule r_t such that $J_t \cong J_q$ and $A_q \subseteq A_t$. In other words, the rule and the authorized query must have the equivalent join paths.

#	Authorized attribute set	Auth. Join Path	To
1	{pid, location}	W	P_W
2	{oid, pid}	E	P_W
3	{oid, pid, location}	$E \bowtie_{pid} W$	P_W
4	{oid, pid, total}	E	P_E
5	{oid, pid, total, issue}	$E \bowtie_{oid} C$	P_E
6	{oid, pid, total, issue, addr}	$S \bowtie_{oid} E \bowtie_{oid} C$	P_E
7	{oid, pid, location, total, addr}	$S \bowtie_{oid} E \bowtie_{pid} W$	P_E
8	{oid, pid, issue, agent, total, addr, delivery}	$S \bowtie_{oid} E \bowtie_{oid} C \bowtie_{pid} W$	P_E
9	{oid, addr, delivery}	S	P_S
10	{oid, pid, total}	E	P_S
11	{oid, pid, total, addr, delivery}	$E \bowtie_{oid} S$	P_S
12	{oid, pid, total, location}	$E \bowtie_{pid} W$	P_S
13	{oid, location, pid, total, addr, delivery}	$S \bowtie_{oid} E \bowtie_{pid} W$	P_S
14	{oid, pid}	E	P_C
15	{oid, issue, agent}	C	P_C
16	{oid, pid, issue, agent}	$E \bowtie_{oid} C$	P_C
17	{oid, pid, issue, agent, total, addr, location}	$S \bowtie_{oid} C \bowtie_{oid} E \bowtie_{pid} W$	P_C

Table 1. Auth. rules for running Example

To answer a query that is authorized by the rules, we still need a **query execution plan** (or "query plan" for short) where each of the steps corresponds to an authorized and realizable operation. In our model, the query execution plan pl can also be represented with a triple $[A_{pl}, J_{pl}, P_{pl}]$ just like a rule. Here, the join path not only for local joins but also counts the data transmitted between the parties as we will discuss later. For this plan to be valid, it is necessary that $J_{pl} \cong J_q$ and $A_q \subseteq A_{pl}$. We introduce the notion of consistent query plan next, and only consistent plans are considered safe to answer the queries.

The desired query plan can be represented hierarchically where at each level, a number of sub-plans are combined to get the next higher level plan. The access plans for basic relations owned by the parties form the bottom level in this structure. For instance, there is a query plan to retrieve all the information of rule r_3 in Table 1, and such a plan contains a join over two subplans based on rules r_1 and r_2 respectively. The subplan for r_1 is to access table W on P_W. The subplan for r_2 is an access plan reading table S at P_S, and another operation transmitting the data from P_S to P_W. The example plan authorized by r_3 has the $J_{pl} = E \bowtie_{pid} W$, and $A_{pl} = \{oid, pid, location\}$. We say a rule r_t **authorizes** ($\succeq$) a plan pl, if $J_{pl} \cong J_t$, $P_{pl} = P_t$, and $A_{pl} \subseteq A_t$.

Definition 2 *An operation in a query plan is* **consistent** *with the given rules $\mathcal{R}$, if for the operation, there exist rules that authorize access to the input tuples of the operation and to the resulting output tuples.*

For the three types of operations in our scenario, we give the corresponding conditions for consistent operation.

1. For a projection (π) to be consistent with the rule set $\mathcal{R}$, there must be a rule r_p that authorizes ($\succeq$) the input information.
2. Join ($\bowtie$) is a binary operation where two input subplans pl_{i1} and pl_{i2} produce the resulting plan $pl_o = pl_{i1} \bowtie pl_{i2}$. For a join operation to be consistent with $\mathcal{R}$, all the three plans need to be authorized by rules. Since join is performed at a single party, and rules are upwards closed, if the input plans are authorized by rules, the join operation is consistent.
3. Data transmission ($\rightarrow$) involves an input plan pl_i on a party P_i and an output plan pl_o for a party $P_o \neq P_i$. If there are rules $r_i, r_o \in R$ with equivalent

join paths (i.e., $J_i \cong J_o$), and $r_i \succeq pl_i, r_o \succeq pl_o$, then the data transmission operation is consistent with $\mathcal{R}$.[1]

For our example, rule r_8 authorizes P_E to get information on the join path $(S \bowtie E \bowtie C \bowtie W)$. Also note that although the attribute set of rule r_{11} is contained in that of rule r_8, there is no rule for P_E to get these attributes on the join path of $(E \bowtie S)$. Therefore, party S, the owner of rule r_{11} cannot send these attributed to P_E.

Definition 3 *A query execution plan pl is* **consistent** *with the rules $\mathcal{R}$, if for each step of the operation in the plan is consistent with the given rule set $\mathcal{R}$.*

3.3 Inadequacy of Classical Query Planning

Generating a consistent plan that answers an authorized query in our scenario is much more complex than the well studied problem of query planning for distributed databases (without any access restrictions). We illustrate this by an example. Suppose that there are two collaborating parties P_R and P_S with database schemas $R(\underline{A}, B, C)$, and $S(\underline{A}, D, E)$ respectively (A is the key attribute for both relations). The party P_R has an authorization rule $r_R = \{A, B, C, D\}, R \bowtie S$ (in addition to access to its own data). The party P_S has two authorization rules: $r_{S1} = \{A, B\}, R$ and $r_{S2} = \{A, B, C, D, E\}, R \bowtie S$. Let us now consider how to generate a consistent plan to answer a query for $\{A, B, C, D, E\}$ over the join path of $R \bowtie S$.

In classical query planning, we will generate a query plan tree and try to assign the appropriate operations to different parties. There is no constraint of data access in classical case. Therefore, either party P_R or P_S can retrieve the other relation and do the join to answer the query. From performance considerations, semi-joins [11] are usually used in the distributed query processing. However, in our case, even a semi-join is not enough to generate the consistent query plan for the query. It is clear that neither P_R and P_S can obtain the desired result with just one join. If we use the semi-join method, the only possibility is that P_R sends $\{A\}$ to P_S; P_S does the join and ships $\{A, D\}, R \bowtie S$ back to P_R, which then computes $\{A, B, C, D\}, R \bowtie S$ by doing another join. This, in turn is passed back to party P_S, which then obtains the desired result. In contrast, if we use regular join, then party P_S can have at best the attributes $\{A, B, D, E\}, R \bowtie S$ through one join operation.

To generate the consistent plan for answering the query, it is required that we do the semi-join first, and party P_R again sends the $\{A, B, C\}, R \bowtie S$ to party P_S. Another join operation at party P_S could then give the required query results. Figure 2 illustrates the situation. Each box is a rule, and the authorization rule that authorizes the query is in dashed box. The numbers on the arrows indicate the ordered steps for the consistent query plan. It is clear that generating

[1] If P_i is sending information with attributes not in A_o, P_i should do a projection operation $\pi_{A_o}(pl_i)$ first.

a consistent query plan under the data access constraints can be lot more complicated than for distributed query planning. In the following section, we show the complication of query processing in cooperative data access environment.

4 Complexity of query planning

From performance perspective, we always want consistent and optimal query plans with minimal costs. Unfortunately, finding the optimal query plan is *NP-hard* in such a cooperative data access scenario.

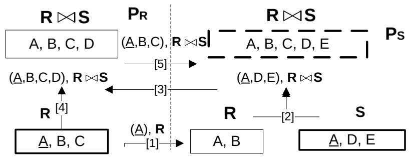

Fig. 2. Illustration of Query Planning

Theorem 1 *Finding the optimal query plan to answer an authorized query is NP-hard.*

Proof. The optimization of set covering problem is known to be NP-hard. In the set covering problem, there is a set of elements $U = \{A_1, A_2, ..., A_n\}$, and there is also a set of subsets $S = \{S_1, S_2, ...S_m\}$ where S_i is a set of elements from U and is assigned a cost. The task is to find a subset of S, say C, that has minimal total cost and covers all the attributes in set U. We can convert this set covering problem into the cooperative query planning problem and thereby prove that the optimal query planning problem is also NP-hard.

Consider 2 basic relations R and S which can join together over a key attribute A_0, distinct from the element set U that we will also use as attributes in our construction. We assign all the attributes in U to relation R, which will have the schema $\{\underline{A_0}, A_1, A_2, ..., A_n\}$. For each S_i in S, we make an authorization rule $\{\underline{A_0}, S_i\}$ on relation S. Thus, for $m+1$ parties, $P_0, P_1 \ldots P_n$ have the following authorization rules:

1. Party P_0 owns R and has a rule r_0 that authorizes the desired query for retrieving the entire set U over the join path $J = R \bowtie (\bowtie_{i=1}^{n} S_i)$. Note that P_0 cannot unilaterally obtain the join path J.
2. Each of the other parties P_i, $i = 1 \ldots n$, has a rule r_i on the relation S with attributes $S_i \bigcup \{A_0\}$.

Note that P_0 cannot locally do the join $R \bowtie S$, but other parties can enforce their rules r_i locally, and their costs are known. Therefore, for P_0 to answer the query, it needs a plan bringing attributes from other parties and merging them at P_0 (multi-way join on attribute A_0) to answer the query. The optimal plan needs to choose the rules with minimal costs, and the union of their attribute sets must cover the query attribute set. If the optimal query plan can be found in polynomial time, the set covering problem also has a polynomial solution, which proves the assertion. □

4.1 Query plan cost model

It is reasonable to assume that the number of tuples in the relations are known. Assuming we have the historical statistic information of the tables, so we can estimate the join results accurately. The notion of *join selectivity* [11], a number between 0 and 1.0 provides an estimate for the size of the joined relation. We assumed that the join selectivity between the relations are known so that the number of tuples in a join path can be estimated.

The cost of a query plan mainly includes two parts: 1) cost of the join operations, and 2) cost of data transmission among the parties. We assume joins are done by nested loop and indices on join attributes are available. Let $Size()$ denote the number of tuples in the relation, and $Pages()$ is the number of pages in the relation. Consider two relations R and S, of which R is the smaller one, i.e., $Size(R) < Size(S)$. Let α denote the output cost of generating each tuple in the results, and let $P_{(X,Y)}$ denote the known join selectivity. Let β denote the per I/O cost. Assuming the cost of finding matching tuples in S is 1. Then the cost of a join operation between R and S can be estimated as:

$\alpha(Size(R) * Size(S) * P_{(R,S)}) + \beta(Pages(R) + Size(R) * 1)$

The costs of data transmission is only decided by the size of the data being shipped. Let γ denote the per tuple cost for data transmission. Then the cost of moving $R \bowtie S$ from a party to another is given by: $\gamma(Size(R) * Size(S) * P_{(R,S)})$

It is worth noticing that our algorithm does not tie to any specified cost model, this is one easy cost model that we can adopt.

4.2 Enumerating All Query Plans

Unlike classical query planning, we face a number of hurdles, as illustrated next. To generate a consistent plan for a query, we first need a plan that enforces the query join path. This can be further joined with other plans to get all the requested attributes. Obviously, in order to consider a join path of length n, one needs to consider all top level join subpaths of with lengths k and $n - k$ for suitable values of k. Unfortunately, this is insufficient. Since a longer join path will generally produce relations with fewer tuples, it is often desirable to consider joins of overlapping relations in cooperative data access environment. For instance, generating a join path of $R \bowtie S \bowtie T$ may be better done as $(R \bowtie S) \bowtie (S \bowtie T)$ instead of, say, $(R \bowtie S) \bowtie (T)$. It all depends on the authorization rules setting in the environment as well as the sizes of relations and costs of operations.

An added difficulty is that we can't just pick the subpaths based on the join cost – we also need to pay attention to the attributes we are able to access by doing the join. For instance, if the goal is to answer $\{A, B, C, D\}$ on join path $R \bowtie S \bowtie T$, we may have two ways of getting it: (a) A subplan pl_1 that yields that attribute set $\{A, B\}$, and (b) A higher cost subplan pl_2 that yields the attribute set $\{A, C, D\}$. Since we need more work to get missing attributes, at this stage we can't even pick one of these, and instead must keep both. Thus, in general, we need to maintain many "partial" plans. For each such partial plan, we then need to consider the problem of retrieving the missing attributes. This, in turn, requires checking all possible combinations of relevant rules, followed by a recursive procedure to find enforcement plan for the chosen relevant rules. It is clear that the exhaustive enumerate to find the globally optimal answer can be extremely expensive.

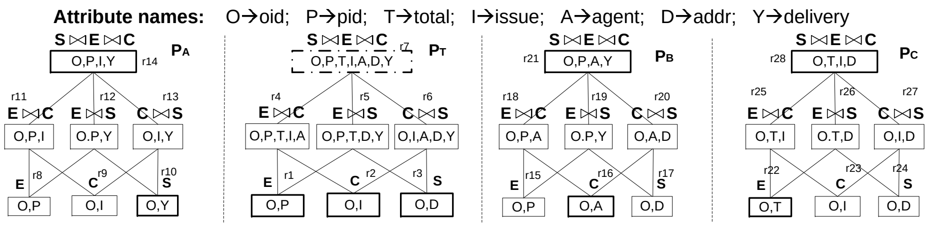

Fig. 3. A simple worst case example

We illustrate the complexity of exhaustive enumeration via the case of join path length of 3 for our running example. In figure 3, there are four parties P_A, P_T, P_B, P_C and they all have rules on equivalent join paths. The attribute names are simplified to save space. In the example, an incoming query asks for all the attributes $\{O, P, T, I, A, D, Y\}$ on $(S \bowtie E \bowtie C)$ and only r_7 (dashed box) can authorize the query.

Although there are many ways to enforce the query join path $S \bowtie E \bowtie C$, none of them can totally enforce all the query attributes. The possible ways to enforce the join path locally on P_t is $3{*}(1{+}2) = 9$ (each join path of length 2 can join with other two rules of length 2 and one rule of length 1). Considering other 3 parties, we have $(3 * 4 * (1 + 2 * 4)) * 4 = 432$ (considering join across parties) different ways of enforcing the join path, and these plans result in 10 different missing attribute sets (although there are many enforcement plans, many of their enforced attribute sets are overlapped). For each of them, we need to check the ways to get missing attributes. For example, if the missing attribute set is $\{total, agent, delivery\}$. Then, there are 12 relevant rules having the missing attributes, and the possible combinations to consider are 2^{12}-1.

5 Consistent query planning

Due to the difficulties in enumerating all possible ways of answering a query, we consider using a greedy algorithm.

5.1 Greedy Query planning algorithm

To find an efficient consistent query plan, we always choose the optimal query plan to enforce the join path first, and then apply greedy set covering mechanism on the missing attributes (the attributes cannot be enforced with the join path enforcement plan) to find required relevant rules (rules authorize subplans for the complete query plan). The optimal enforcement plan for a join path on a specified party can be pre-determined by extending the rule enforcement checking algorithm in a dynamic programming way [13]. As discussed, the selected plan usually results in a missing attribute set. To get these attributes, we explore the graph structure to decompose the target rule r_t (the rule authorizes the incoming query q) into a set of relevant rules that can provide these attributes. We record the required operations among these rules, and then recursively find ways to enforce these rules to generate a query plan.

As the join path enforcement plan enforces J_t, it can be extended to get missing attributes that appear in the relevant rules of basic relations on all **J_t-cooperative parties** (cooperative parities which have authorization rules on join path J_t). This can be done through semi-join operations. In such cases, the party P_t can send only the join attributes to its J_t-cooperative party, and the receiving party does a local join to get these attributes and send it back. P_t then performs another join to add these attributes to the query plan. The remaining missing attributes can always be found in the relevant rules on J_t-cooperative parties. However, these relevant rules are defined on join paths instead of basic relations. Similar to the above case, the missing attributes carried by these relevant rules can be brought to the final plan by performing semi-join operations.

The next step is to determine these **relevant rules** (rules can provide missing attributes and the join paths include a subset of relations of J_t). Here, we always pick the relevant rule that covers the most attributes in the missing attribute set until all the missing attributes are covered by the picked rules. This is a greedy approach, and is similar in spirit to the approximate algorithms used for the set covering problem. The relevant rules effectively allow us to decompose the rule (i.e., express in terms of) rules with smaller join paths. The missing attributes are also reduced in the process by considering the rules involving basic relations. During the decomposition, the algorithm associates the set of attributes with the decomposed rule that are the missing attributes expected to be delivered by this rule. We record the operations between the existing plan and these decomposed ones. If they are on the same party, a join operation between them is recorded. Otherwise, a semi-join operation is recorded. Since each decomposed rule can be further decomposed, the algorithm uses a queue to process the rules until all the rules are on basic relations. This decomposition process gives the hierarchal relationships among rules that indicate how required attributes can be added to the final plan. After this step, the query plan is going to use all the attributes that available locally (all the picked relevant rules on the same party P_t), and it removes these duplicate attributes (non-key attributes) from remote parties (via projections).

The decomposition process gives a set of rules, but we also need the subplans to enforce the join paths of these rules so as to generate a complete plan. To achieve that, we inspect the join paths of these decomposed rules from bottom-up. We use another priority queue to keep all the join paths from the decomposed relevant rules, and the shortest join path is always processed first. This allows the use of results from the enforcement plans of sub join paths as much as possible. The algorithm uses the best enforcement plan for each join path as discussed. When an enforcement plan of a join path is retrieved, the algorithm combines previously recorded operations to generate the subplan for the decomposed rule on such join path. Finally, the algorithm finds the plans for each join paths in the queue, and generates the final query plan with a series of ordered operations starting from the basic relations. The entire process is summarized in Algorithm 4.

5.2 Properties of the Algorithm

```
Require: The structure of rule set $\mathcal{R}$, Incoming query $q$
Ensure: Generate a plan answering $q$.
 1: if There is a rule $r_t$, $J_t \cong J_q$ and $A_q \subseteq A_t$ then
 2:     Missing attribute set $A_m \leftarrow A_q$
 3:     Initialize queue $Q$, and priority queue $P$
 4:     Enqueue $r_t$ to $Q$ with $A_m$
 5:     while Queue $Q$ is not empty do
 6:         Dequeue rule $r_t$ and the associated $A_m$
 7:         for Each $J_t$-cooperative party do
 8:             Finds the attribute set $A_b$ from basic relations
 9:             $A_m \leftarrow A_m \setminus A_b$
10:             Record connections between $r_b$ and $r_t$
11:         while $A_m \neq \emptyset$ do
12:             for Each relevant rule $r_s$ on $P_{co}$ do
13:                 Find the rule with max $A_m \bigcap A_s$
14:             Enqueue the rule $r_s$ with $\pi(A_m)$
15:             Enqueue the join path $J_s$ to priority queue $P$
16:             Record connections between $r_s$ and $r_t$
17:             $A_m = A_m \setminus A_s$
18:     while The priority queue $P$ is not empty do
19:         Dequeue the rule $r_s$ with join path $J_s$
20:         Add the path to enforce $J_s$ to plan
21:         for Each $J_s$-cooperative party do
22:             if The party has recorded $A_b$ on $J_s$ then
23:                 Add ($\bowtie$ / $\rightarrow$) operations between $r_b$ and $r_s$
24:         for Each decomposed rule $r_d$ from $r_s$ do
25:             Add ($\bowtie$ / $\rightarrow$) operations between $r_d$ and $r_s$
26: else
27:     The query $q$ cannot be answered
```

Fig. 4. Query Planning Algorithm

In this section we show that the query planning algorithm is correct.

Theorem 2 *A query plan generated by Query Planning Algorithm is consistent with the set of rules $\mathcal{R}$.*

The proof is omitted as it's straightforward according to our definition of query plan consistency.

5.3 Preliminary performance evaluation of the algorithm

Finding a globally optimal query plan is not only NP-hard for the situation we are considering, it is also extremely difficult to systematically and efficiently enumerate all possible cases with large number of parties and rules. In view of this and the space limitations of the paper, we only illustrate comparative performance for the following three situations, all relating to our running e-commerce example.
Here we assume that the selected join path enforcement plan carries the maximal attributes along with it. Since we do not have any assumptions on the sizes of the relations and join selectivities between them, we cannot calculate the exact costs of the plans to compare the them. For simplicity, we use the number of joins as the metric to evaluate the efficiency of the plans. This would be a good representation of actual cost if all the relations have roughly the same size.

Example 1 Consider the situation in figure 3 and given the same query discussed before which only rule r_7 can authorize, the *optimal* plan should be as follows: join the three rules on basic relations which are r_1, r_2, r_3 at P_t to enforce the join path of $S \bowtie E \bowtie C$ with attribute set $\{oid, pid, issue, addr\}$. Then P_t sends the *oid* on the join path of $S \bowtie E \bowtie C$ to other three parties P_A, P_B, P_C, and does semi-joins with each of the party to obtain the missing attributes $\{total, agent,$
$delivery\}$ one from each party. Finally, P_t does a local join with this information got from remote parties and such a plan answering the query. The related rules for the consistent query plan are marked using bold boxes in the figure. In this case study, our greedy algorithm generates the same optimal plan. The optimal way to enforce join path $S \bowtie E \bowtie C$ is the local enforcement at P_t, and our plan also gets the missing attributes via semi-join operations. Note that manually finding the optimal plan is easy only under the assumption that all the relations are of the same size.

Example 2 Consider a query with the join path $S \bowtie C \bowtie E \bowtie W$, and an attribute set that includes every attribute of rule r_8 except *delivery*. Figure 5 illustrates the rules corresponding to our running example. Unrelated rules are removed, and rules on the graph are applied in the generated query plan. For such a query, our algorithm first finds the optimal way to enforce the join path, which can be represented as

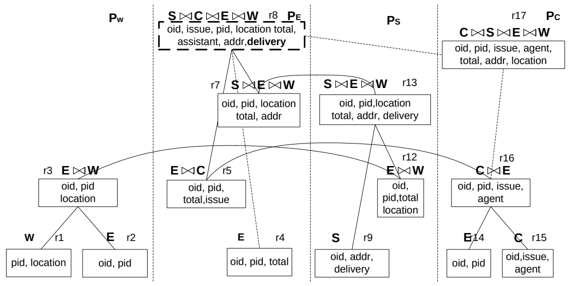

Fig. 5. Simplified relevance graph

$$[((r_1 \bowtie r_2 \rightarrow P_S) \bowtie r_9) \rightarrow P_E] \bowtie [r_{14} \bowtie r_{15} \rightarrow P_E] \qquad (1)$$

This plan results in a missing attribute set $\{total, agent\}$. Next, the algorithm adds a local join with rule r_4 to retrieve *total*, and a semi-join with rule r_{16} to obtain the attribute *agent* because P_E and P_C are J_8-cooperative parties (J_8 is equivalent to J_{17}). Here, r_{16} is enforced during the join path enforcement. In figure 5, the solid lined between rules indicates the steps for enforcing the query join path, and the dashed lines are the operations for retrieving missing attributes. The dashed box shows the rule r_8 which authorizes the query. In fact, there are only two ways to enforce the query join path in this example. The other way is to perform $r_9 \bowtie r_{10}$ first and then join with r_{12} at party P_S. By doing that, the plan can carry the attribute *total* and only has *agent* as missing attribute. However, if we compare the two plans, the difference is that our plan gets the attribute *total* via a join among relation E and join path $S \bowtie C \bowtie E \bowtie W$, and the latter plan performs the join among E and S at party P_S. Since the longer join path usually has fewer tuples, the former plan is better. As for the missing attribute *agent*, it can only be retrieved from party P_S, and getting it from r_{16} is better than r_{15}. Therefore, the query plan generated by our algorithm is again the optimal plan.

Example 3 Here we consider a situation where the algorithm does not produce an optimal plan. Consider a query which is the same as rule r_4. As shown in figure 6, the bold boxes are used in enforcing the query join path $S \bowtie E \bowtie C$ by our algorithm, which is

$$[(r_8 \bowtie r_9 \rightarrow P_T) \bowtie r_1] \qquad (2)$$

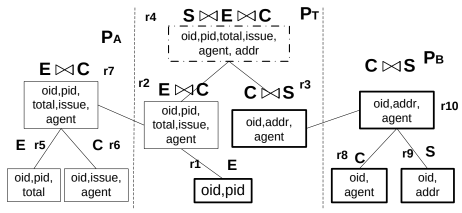

Fig. 6. A simple non-optimal example

The other way to enforce it is to enforce rule r_7 at party P_A first, and send the results to party P_T to enforce R_2 and join with R_3. That is

$$[(r_8 \bowtie r_9 \rightarrow P_T) \bowtie (r_5 \bowtie r_6 \rightarrow P_T)] \qquad (3)$$

As the latter plan requires one more join and data transmission operation, our plan to enforce the query join path appears better. However, the latter plan has no missing attribute, and our plan needs to enforce rule r_7 again to retrieve attributes $\{total, issue\}$ which includes more operations. Therefore, our plan is not optimal in this case. However, compared to the optimal plan, our generated plan only has one extra step involving r_1 joining with r_3, which means that the cost difference between the two plans is likely not significant.

Due to the space limitations, we have listed only 3 detailed case studies here. In addition, we have evaluated other example queries based on our running example given in Table 1. Table 2 lists seven other examples. The second column shows the queries in "attribute set, join path" format, and the third column shows the consistent query plans generated by our algorithm. The symbols π, $\bowtie$ and $\rightarrow$ correspond to the projection, join and data transmission operations. The last column shows whether the generated query plan is optimal, and it turns out that the plan is indeed optimal in all seven cases. This is typical of the behavior we have seen so far, although because of the complexities generating optimal query plans we have so far been unable to generate and test cases in large numbers. We, however, believe that the algorithm does produce optimal or near optimal solution in nearly all practical situations.

Complexity of the algorithm Assuming N_q rules are locally relevant to the query q, the number of relevant rules on J_t-cooperative parties is N_r, and C is a constant to record operations. The overall worst case complexity of our greedy algorithm is $O(N_q * N_r^2 * C)$, which is $O(N^3)$(N is the total number of rules).

#	**Example Query**	**Gerenated Query Plan**	**Optimal?**
1	{oid, pid, location}, $E \bowtie W$	$r_1 \bowtie r_2$ on P_W	Yes
2	{oid, pid, total, issue}, $E \bowtie C$	$(\pi(oid)r_{14} \bowtie \pi(oid, issue)r_{15} \rightarrow P_E) \bowtie r_4$	Yes
3	{oid, pid, total, addr}, $E \bowtie S$	$\pi(oid, addr)r_9 \bowtie \pi(oid, pid, total)r_{10}$	Yes
4	{oid, pid, total, delivery}, $S \bowtie E \bowtie W$	$(\pi(pid)r_1 \bowtie r_2 \rightarrow P_S) \bowtie \pi(oid, delivery)r_9 \bowtie \pi(oid, total)r_{10}$	Yes
5	{oid, pid, total, addr}, $S \bowtie E \bowtie W$	$((\pi(pid)r_1 \bowtie r_2 \rightarrow P_S) \bowtie \pi(oid, addr)r_9 \rightarrow P_E) \bowtie \pi(oid, total)r_4$	Yes
6	{oid, pid, total, addr}, $S \bowtie C \bowtie E \bowtie W$	$((\pi(pid)r_1 \bowtie r_2 \rightarrow P_S) \bowtie \pi(oid, addr)r_9 \rightarrow P_E) \bowtie (\pi(oid)r_{14} \bowtie \pi(oid, issue)r_{15} \rightarrow P_E)\pi(oid, total)r_4$	Yes
7	{oid, pid, agent}, $S \bowtie C \bowtie E \bowtie W$	$((\pi(pid)r_1 \bowtie r_2 \rightarrow P_S) \bowtie \pi(oid)r_9 \rightarrow P_E) \bowtie (\pi(oid)r_{14} \bowtie \pi(oid, agent)r_{15} \rightarrow P_E) \rightarrow P_C$	Yes

Table 2. Illustration of quality of some generated query plans

6 Conclusions and future work

In previous research work, a flexible data authorization model has been proposed to meet the security requirements for collaborative computing among different data owners in a collaborative environment. A regular query optimizer cannot give consistent query plans under the constraints of these rules. In this paper, we propose an algorithm to generate corresponding efficient consistent query plans for answerable queries.

For the future work, we will study the problem of making the unenforceable rules to be enforceable. We can consider using a trusted third party to enforce the rules, and we may also augment the given set of rules. Trusted third parties can be also used to improve the consistent query planning. To evaluate of our approaches comprehensively, we will study the cooperative relationships among enterprises in various real world scenarios, and test our mechanism under these cases. In addition, we will investigate the problem where data are horizontally fragmented and distributed among different parties, which adds selection to the picture. In fact, extension of the model to include limited forms of selection is one area that we wish to pursue in the future. We also plan to extend our model to more general applications that involve non-numeric data (e.g., textual or image data) where the regular join operation may be not be the most interesting operation. Finally, we wish to examine the issue of verifying whether the collaborative parties are really following the rules as advertised or may be behaving in undesirable ways.

References

1. G. Aggarwal, M. Bawa, P. Ganesan, H. Garcia-Molina, K. Kenthapadi, R. Motwani, U. Srivastava, D. Thomas, and Y. Xu. Two can keep A secret: A distributed architecture for secure database services. In *CIDR*, pages 186–199, 2005.

2. R. Agrawal, D. Asonov, M. Kantarcioglu, and Y. Li. Sovereign joins. In *Proceedings of the 22nd International Conference on Data Engineering, ICDE 2006, 3-8 April 2006, Atlanta, GA, USA*, page 26. IEEE Computer Society, 2006.
3. P. A. Bernstein, N. Goodman, E. Wong, C. L. Reeve, and J. B. Rothnie, Jr. Query processing in a system for distributed databases (SDD-1). *ACM Transactions on Database Systems*, 6(4):602–625, Dec. 1981.
4. A. Calì and D. Martinenghi. Querying data under access limitations. In *Proceedings of the 24th International Conference on Data Engineering, ICDE 2008, April 7-12, 2008, Cancún, México*, pages 50–59. IEEE, 2008.
5. S. Chaudhuri. An overview of query optimization in relational systems. In *Proceedings of the 7th ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 34–43, 1998.
6. V. Ciriani, S. D. C. di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Keep a few: Outsourcing data while maintaining confidentiality. In *ESORICS*, volume 5789 of *Lecture Notes in Computer Science*, pages 440–455, 2009.
7. S. De Capitani di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Controlled information sharing in collaborative distributed query processing. In *ICDCS 2008*, Beijing, China, June 2008.
8. S. D. C. di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Authorization enforcement in distributed query evaluation. *Journal of Computer Security*, 19(4):751–794, 2011.
9. J. Goldstein and P. Larson. Optimizing queries using materialized views: a practical, scalable solution. In *Proceedings of the 2001 ACM SIGMOD international conference on Management of data*, pages 331–342.
10. A. Y. Halevy. Answering queries using views: A survey. *VLDB Journal*, 10(4):270–294, 2001.
11. D. Kossmann. The state of the art in distributed query processing. *ACM Computer Survey*, 32(4):422–469, 2000.
12. M. Le, K. Kant, and S. Jajodia. Access rule consistency in cooperative data access environment. In *8th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom2012)*, pages 11 –20, oct. 2012.
13. M. Le, K. Kant, and S. Jajodia. Consistency and enforcement of access rules in cooperative data sharing environment. In *Computers and Security*, Nov. 2013.
14. M. Le, K. Kant, and S. Jajodia. Rule enforcement with third parties in secure cooperative data access. In *27th Data and Applications Security and Privacy, DBSec 2013*, July 2013.
15. C. Li. Computing complete answers to queries in the presence of limited access patterns. *VLDB Journal*, 12(3):211–227, 2003.
16. R. Pottinger and A. Y. Halevy. Minicon: A scalable algorithm for answering queries using views. *VLDB J*, 10(2-3):182–198, 2001.
17. S. Rizvi, A. Mendelzon, S. Sudarshan, and P. Roy. Extending query rewriting techniques for fine-grained access control. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, SIGMOD '04, 2004.
18. R. Sion. Query execution assurance for outsourced databases. In *VLDB*, pages 601–612. ACM, 2005.
19. Z. Zhang and A. Mendelzon. In *Database Theory - ICDT 2005*, volume 3363, pages 259–273. 2005.