Towards Collaborative Query Planning in Multi-party Database Networks

Mingyi Zhao, Peng Liu, Jorge Lobo

Towards Collaborative Query Planning in Multi-party Database Networks

Mingyi Zhao[1], Peng Liu[1] and Jorge Lobo[2]

[1] Pennsylvania State University
{muz127,pliu}@ist.psu.edu
[2] ICREA-Universitat Pompeu Fabra
jorge.lobo@upf.edu

Abstract. Multi-party distributed database networks require secure and decentralized query planning services. In this work, we propose the collaborative query planning (CQP) service that enables multiple parties to jointly plan queries and controls sensitive information disclosure at the same time. We conduct several simulated experiments to evaluate the performance characteristics of our approach compared to other planning schemes, and also study the trade-off between information confidentiality and query plan efficiency. The evaluation shows that when sharing more than 30% of query planning information between coalition parties, the CQP service is able to generate reasonably efficient query plans. We also outline potential improvements of the CQP service at the end.

Keywords: Information Confidentiality, Multi-party Database Network, Optimization

1 Introduction

Many organizations today form coalitions to facilitate information sharing and processing for a common mission. In a typical coalition, multiple parties may connect their database severs with each other to form a federated database and enable querying over the network. Such federation of multiple database servers can be supported by products like IBM GaianDB [1].

Since coalition networks are formed by independent parties with different level of trust among each other, information confidentiality becomes an important concern. Significant effort has been made to protect data confidentiality in mutli-party database networks by enforcing data authorization policies [3, 8, 9, 14]. Previous work has also proposed new query engines that consider query information confidentiality during collaborative query execution [4–6].

A common assumption behind most of previous work is the existence of a fully trusted central query planning server. This query planning server will enforce data authorizations to protect confidentiality while finding the optimal execution plan for a query. Therefore, the planner needs to know query planning information, such as metadata of relations, query information, data authorizations, etc., from all coalition parties. However, this assumption limits the usage of federated databases for most coalition scenarios for mainly two reasons. First, in a coalition network, it is very rare that a party can be fully trusted by all other parties. Thus, building a central authority server which all trust will be difficult. Second, the central planning server becomes a single point of failure in this multi-party database network. This is particularly worrisome in ad hoc and dynamic situations where nodes may enter and leave the network anytime. Some existing systems such as the GaianDB avoid using this central planning design, and only use the basic data shipping plan as the query execution strategy where all required data is sent to the querying node and the query is processed locally, hence losing all the performance benefits from distributed query processing.

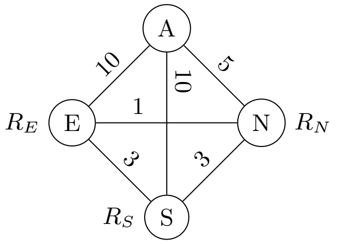
We cannot directly apply existing decentralized query planning frameworks [11–13] to multi-party scenarios, since they are designed for scenarios where servers can trust each other (e.g., all servers belong to a single organization). More specifically, the interaction process between servers in these frameworks could leak sensitive information in multi-party scenarios.

Our first contribution in this work is a new decentralized query planning service, called collaborative query planning (CQP), for multi-party database networks. This service allows coalition parties to collaboratively plan queries while at the same time control information disclosure among collaborating parties. A premise of our work is that coalitions are formed because there is some willingness among the parties to collaborate and hence some level of trust. Therefore, we assume that coalition parties share certain query planning information with other parties in order to facilitate the common mission. Given a query, different coalition parties might be able to do part of the query planning based on the amount of information they know. The querying party can thus assemble these partial results to generate a final plan. Our second contribution is to empirically evaluate the performance characteristics of our approach compared to other planning schemes in different scenarios. We are particularly interested in how the level of information sharing between coalition parties influences the decentralized query planning. That is, we study and measure the trade-off between information confidentiality and query plan efficiency.

2 Background

2.1 Query Planning in Database Networks

We briefly describe key concepts of query planning in database networks based on a simple example. In Figure 1, four companies formed an information sharing coalition on top of a database network. A is a target advertising company who analyzes data from various sources and creates targeted advertisements for Internet users. E is an E-commerce company that maintains a large online sale data in relation R_E. S is a search engine company that stores search log in R_S. Finally, N is an online social network company that owns user profile information as relation R_N. Each party has a server and the topology of this database network is shown in Figure 1a. The number on each link is the communication cost of using that link. Now, A wants to retrieve the data of all customers who have bought products from E and have also used services provided by S and N. So A formulates the query $R_E \bowtie R_S \bowtie R_N$ shown in Figure 1c.



(a) A multi-party database network example.

Relation	Size
R_E	12,000
R_S	2,000
R_N	5,000
$\pi(R_E)$	1,000
$\pi(R_S)$	1,000

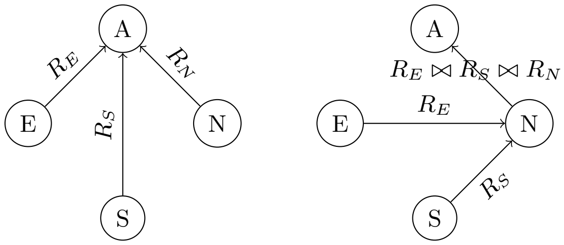
(b) Size of relations.

```
SELECT *
FROM R_E, R_S, R_N
WHERE R_E.uid = R_S.uid
AND R_S.uid = R_N.uid
```

(c) Query

Fig. 1: The setting of a simple multi-party database network.

The query needs to be translated into an executable *query plan*, which is defined as a multi-graph $< V, E >$, where V is the set of vertices in the graph. Each vertex refers to a database server and describes operations to be performed on that server. E is the set of edges. Each edge is represented by (S_i, S_j, D) and it means that server S_i will send data D, which will be the result of the operations executed in S_i, to server S_j. Figure 2 gives two possible query plans for the example query. In Figure 2a, the querier A first retrieve data from remote servers and the process the data locally. In Figure 2b, party N helps A to process the query and then sends the result to A.



(a) The data shipping plan. (b) A query shipping plan.

Fig. 2: Two final plans for the query.

The process of translating a query into a query plan is called *query planning*. Since usually a query can be translated into a large number of possible execution plans, the query planner will aim to find the minimal cost plan using algorithms such as dynamic programming [7, 10]. In this work, we consider a basic cost model, which estimates the cost of a query plan only based on data transmissions.

Relation	Size
$\pi(R_E) \ltimes R_S$	500
$\pi(R_S) \ltimes R_E$	1,000
$R_S \bowtie R_E$	2,000
$R_S \bowtie R_E \bowtie R_N$	5,000

Table 1: Size of join results. π represents the projection operation in the semi-join [10].

Creator	Data Auth.	Query Auth.	Auth. Sharing
A	-	7.$A \xrightarrow{Q'} E$ 8.$A \xrightarrow{Q} S$	$A \xrightarrow{A_7} S$ $A \xrightarrow{A_8'} E$
E	1.$E \xrightarrow{R_E} A$ 2.$E \xrightarrow{R_E} S$	-	$E \xrightarrow{A_1} S$
S	3.$S \xrightarrow{R_S} A$ 4.$S \xrightarrow{R_S} E$	-	$S \xrightarrow{A_3} E$ $S \xrightarrow{A_4} A$
N	5.$N \xrightarrow{R_N} A$ 6.$N \xrightarrow{R_N} S$	-	$N \xrightarrow{A_5} S$

Table 2: Information sharing policies defined in this coalition network. We assign an id for each authorization for easy reference. Q' is $R_E \bowtie R_S$.

We denote the cost of sending a unit data from S_i to S_j as $cost(S_i, S_j)$. Then the cost of a query plan QP is defined as:

$$cost(QP) = \sum_{\forall (S_i, S_j, D) \in QP} |D| \times cost(S_i, S_j) \qquad (1)$$

To estimate the cost of query plan candidates, the query planner also needs to estimate the size of join results based on metadata including the statistics of join operands. We have listed the estimation results of our example in Table 1. The cost model and the join result size estimation show that the cost of the data shipping plan is 165,000, while the cost of the query shipping plan is only 43,000, because the server of N is closer to data sources than the server of A.

2.2 Information Confidentiality Requirements in Multi-party Database Networks

The example shows that *collaborative query execution* can significantly reduce the cost of processing queries. That is, instead of having the querier to retrieve all base relations from remote servers and process the data locally, the querier now delegates sub queries to collaborating parties, which can process the sub queries more efficiently. However, collaborative query execution might disclose sensitive information, such as data tuples [3, 14] and query information [4]. To protect information confidentiality, coalition parties create polices to share selected information with others. The query planner, knowing all information sharing policies, will enforce these policies during the generation of query plans. In this work, we consider the following three types of information confidentiality.

Data Confidentiality. For a party in a multi-party database network, not all data can be shared. Instead, the party wants to selectively share certain data with other parties. A party P_i can authorize another party P_j to access relation R using a *data authorization* $P_i \xrightarrow{R} P_j$. We call P_i the *owner* and P_j the *consumer*. We have created data authorizations for the above example in Table 2. The query planner shall make sure that the final query plan is compliant with all data authorizations. The plan in Figure 2a is legal because A is authorized to access R_E, R_S and R_N. However, the plan in Figure 2b is illegal because N is neither authorized to access R_E nor R_S. In a special case, P_i can only authorize P_j to access metadata of R, including its schema and statistics, in order to let P_j be able to help with planning queries related to R, as we shall discuss later. This metadata sharing policy is defined as $P_i \xrightarrow{M(R)} P_j$.

Query Confidentiality. A query Q is considered sensitive information because it contains the intent of the querier [4]. The execution of non-data shipping plans can disclose information of the query to other parties. For example, in Figure 2b, the party N who executes the query from A, will also learn the content of the query, which might not be desirable for A. A party P_i can share a sub query $Q' \subseteq Q$ with another party P_j using *query authorization* $P_i \xrightarrow{Q'} P_j$. We list some example query authorizations in Table 2. Specifically, A shares the full query with S but only shares a partial query with E. The planner then guarantees that the final query plan is compliant with query authorizations [5].

Authorization Confidentiality. The data authorizations and query authorizations defined between coalition parties are also sensitive, for they contain information regarding the collaboration relationships between different parties. For instance, in the example of Figure 1, party E authorizes A to access R_E. However, E might not want N to know this policy $E \xrightarrow{R_E} A$ due to business secret concerns. A party P_i can share an authorization with P_j using *authorization sharing policy* $P_i \xrightarrow{A} P_j$, where A is an authorization and P_i is the owner of A. Examples of authorization sharing policies can be found in Table 2.

The query planning server needs to know metadata, queries and authorizations in order to generate query plans. So we call these three types of information *query planning information*. There are other types of information required for query planning, such as the network topology of the database network. In this work, we assume these types of information are known to all parties.

2.3 Discussions

A central query planning service needs to know all query planning information from every coalition party in order to find the optimal plan. However, as we have discussed in Section 1, this assumption usually does not hold for multi-party database networks.

Our Goal. In this work, we propose a new decentralized query planning service that only uses information shared explicitly between parties to generate executable query plans. This query planning service enforces the information sharing policies to ensure that the query planning process only disclose information allowed by policies to collaborators.

Threat Model. We assume a curious-but-honest model. That is, each party will follow the steps in the service but might passively learn information that might be disclosed during query planning and query execution. The rationale behind this threat model is that certain level of trust is the pre-condition of establishing a coalition network. Therefore, parties that maliciously attack other parties should be excluded from the coalition in the first place.

3 Collaborative Query Planning

3.1 Overview

The basic idea of our collaborative query planning (CQP) service is that the querier first delegates sub queries[3] to collaborating parties under query authorizations. Then the collaborating parties generate query plans for the received sub queries, and report their findings back to the querier. Finally, the querier assembles the final query plan. Using this service, the querier can utilize information known to collaborating parties to generate an efficient query plan. Figure 3 provides an overview of the CQP. We assume that each party has a planning server, which plans its owner's queries and offers planning service to other parties. We also assume that before executing any query, all parties have created information sharing policies according to its needs.

The CQP service is similar to previous decentralized query planning frameworks [11–13] that attempt to address the problem of missing query planning information in a distributed database setting. However, the cause of insufficient information in previous scenarios was the difficulty in obtaining planning information, rather than information sharing policies. We will discuss more about the differences between these frameworks and our design in Section 6.

In general, we can consider four query planning frameworks in the potential design space of query planning services for multi-party distributed database: (1) if there is no query planning information sharing at all, then all queries have to be executed using data shipping plans. However, the cost of data shipping plans are usually high; (2) if there is certain level of query planning information sharing but no planning collaboration between coalition parties, then we call it *local planning*, since the querier relies on its own limited knowledge of the whole database network to plan the query; (3) if there is certain level of query planning

[3] For simplicity, the term sub query also refers to the full query.

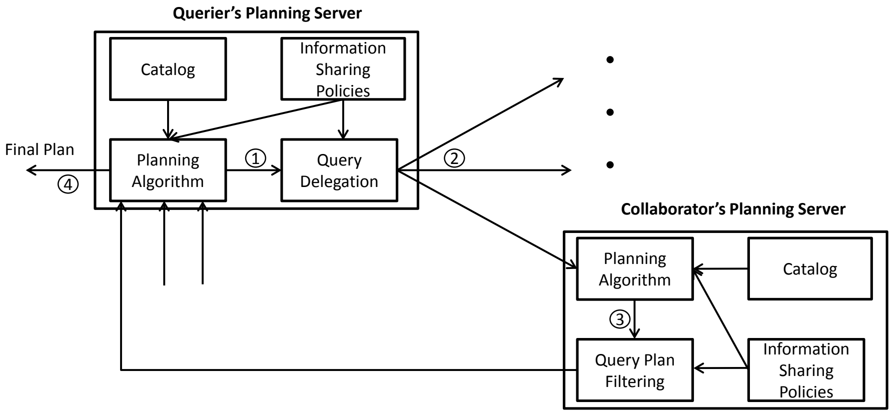

Querier's Planning Server
Catalog
Information Sharing Policies
Final Plan
④
Planning Algorithm
①
Query Delegation
②
Collaborator's Planning Server
Planning Algorithm
Catalog
③
Query Plan Filtering
Information Sharing Policies

Fig. 3: Overview of the CQP service. The four main steps are labeled.

information sharing and planning collaboration between parties, we have a collaborative query planning framework and our design of CQP is one instance of it; (4) if we increase the level of query planning information sharing to the point that one party knows all query planning information, then we actually have the central query planning framework. We will next present the CQP service, and then compare it with other designs in the spectrum in Section 4.

3.2 The CQP Service

In this section, we explain the details of the CQP service.

1. Initiation. Given a query, the querier first runs a local planning solely based on information it knows and obtains the locally generated optimal plan QP_l, and the data shipping plan QP_d. Given a fixed *delegation threshold* T_c defined by the querying party, if $\frac{cost(QP_l)}{cost(QP_d)} < T_c$, the querier will directly execute QP_l and no further query planning is required. Otherwise, the querier will initiate collaborative query planning in order to obtain a more efficient query plan. $T_c = 0.5$ means that collaborative query planning will only happen if the locally generated plan cannot reduce at least 50% of the cost of the data-shipping plan.

Example. Let's revisit the example in Figure 1. The querier A first runs a local planning. However, it can only generate the data shipping plan QP_d (Figure 2a) due to the lack of planning information. More specifically, A almost knows nothing about authorizations between other parties, so it cannot generate a plan utilizing other parties' servers. Therefore, we have $QP_l = QP_d$. The cost of the data shipping plan QP_d is $12,000 \times 10 + 2,000 \times 10 + 5,000 \times 5 = 165,000$. If we set $T_c = 0.5$, then we have $\frac{cost(QP_l)}{cost(QP_d)} = 1 > T_c$, so A will initiate the CQP to find a cheaper plan.

2. Delegation. The querier next decides which portion of the query will be delegated to which collaborating party. We currently consider a simple process in which the querier tries to maximize the delegation in order to get as much help as possible from others, constrained by its own query authorizations. We will discuss potential improvements of the delegation strategy in Section 5. The pseudo code of the delegation process is shown in Algorithm 1. Given a query Q and information sharing policies defined by the querier, this algorithm generates a set of delegation tasks. A delegation task $(collaborator, subquery)$ asks the *collaborator* to plan the *subquery*. Line 5 checks whether party P is allowed to know sub query Q' based on information sharing policies. Lines 7 - 9 remove delegation tasks for which sub queries are contained by other tasks.

Algorithm 1: Collaborative query planning delegation task generation.

```
  input : Query $Q$ and information sharing policies of the querier
  output: Delegation tasks for $Q$
1 tasks = {};
2 foreach $P$ in parties do
3     for $i$ in 2 to $n$ do
4         foreach subquery $Q'$ with length $i$ do
5             if $allows(P, Q')$ then
6                 tasks.addNew($P$, $Q'$);
7                 foreach $t$ in tasks and $t.Q'' \subset Q'$ do
8                     tasks.remove($t$);
9 return tasks;
```

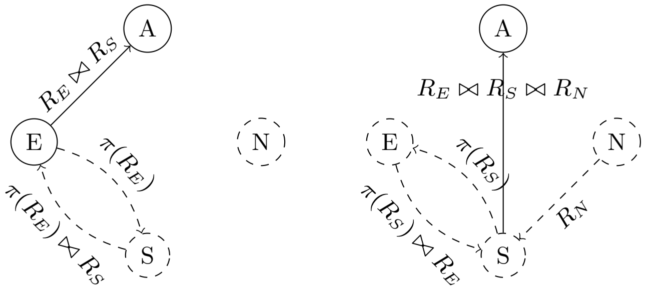
Example. Based on the information sharing policies defined in Table 2, A generates the following two delegations:

- $(E,\ R_E \bowtie R_S)$
- $(S,\ R_E \bowtie R_S \bowtie R_N)$

3. Optimization. Each collaborating party tries to find the optimal plan of delegated sub queries based on query planning information known to that party. This process is based on the classic dynamic programming algorithm [7], with enforcement of information sharing policies [5, 14]. We will not describe the algorithm in this paper. The result is a triple $(collaborator, subquery, plan)$ in which the *plan* is the best plan found by this collaborator, or null if the collaborator fails to find any plan. Note that this query plan has to be filtered by the collaborator before sending it to the querier. This is because the query plan could contain sensitive information of data authorization that is not supposed to be known by the querier. The filtering algorithm is shown in Algorithm 2. In line 3, $L_{DA}(v)$ returns data authorization information associated with the vertex v in the query plan graph. As long as there is one data authorization that the querier is not allowed to see, the algorithm will replace this vertex with its cost and remove all associated edges in Lines 4-5. *v.succ* returns the successor of this vertex in the query plan graph. *v.preds* returns a list of predecessors of the vertex.

Algorithm 2: Query plan filtering.

```
  input  : Query plan $QP$ and authorization sharing policies of the collaborator
  output: Filtered query plan $QP'$
1 $QP' \leftarrow QP$;
2 foreach $v$ in $QP.V$ do
3     if $!allows(querier, L_{DA}(v))$ then
4         v.succ.preds[v] ← cost($v$);
5         $QP$.remove(v.edges);
6 return QP';
```



(a) Plan QP_E generated by E. (b) Plan QP_S generated by S.

Fig. 4: Query plans generated by collaborating parties. Dashed components are not disclosed to the querier A.

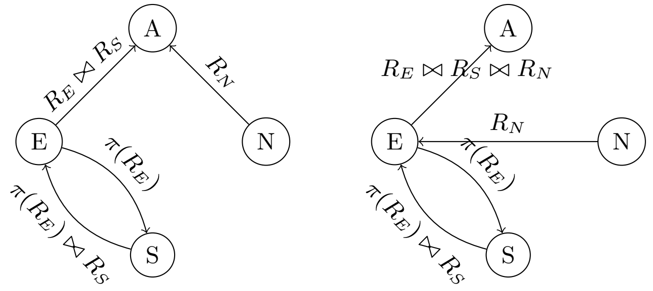
Example. E and S now plan the sub queries delegated to them. For $R_E \bowtie R_S$, E has found a semi-join based query plan QP_E in Figure 4a. For $R_E \bowtie R_S \bowtie R_N$, S has found the plan QP_S shown in Figure 4b. We calculate the cost of these two query plans as follows: $cost(QP_E) = 1000 \times 3 + 500 \times 3 + 2000 \times 10 = 24,500$ and $cost(QP_S) = 1000 \times 3 + 1000 \times 3 + 5000 \times 3 + 3000 \times 10 = 51,000$. Both plans are filtered based on authorization sharing policies in Table 2.

4. Synthesis. The querier synthesizes sub query plans from collaborating parties and creates the final plan by running the query planning algorithm again. Different from the first run, this time some sub queries already have candidate plans available from collaborating parties. These candidate plans might be building blocks for an efficient final plan. The querier might not know the content of these plans due to the filtering in step 3. However, the cost information is enough for the querier to do optimization. It is also possible that none of the collaborating parties have provided a valid plan. In this case, the querier has to use the initial local planning result.

Example. Since A is not allowed to view the content of QP_E and QP_S based on policies, A will only receive the cost of each query plan:

- (E, $R_E \bowtie R_S$, 24,500)
- (S, $R_E \bowtie R_S \bowtie R_N$, 51,000)

The querier A will run the query planning algorithm again and take QP_E as one candidate for executing $R_E \bowtie R_S$, and take QP_S as one candidate for executing the whole query $R_E \bowtie R_S \bowtie R_N$. Figure 5a shows the final optimal plan QP_A. This plan incorporates QP_E and costs 49,500, less than the cost of QP_S. Also, compared with the data-shipping plan QP_d, QP_A saves 70% of the query execution cost.



(a) Final optimal plan QP_A. (b) A cheaper final query plan.

Fig. 5: Two final plans for the query.

There are actually query plans cheaper than QP_A that are also compliant with all policies. For example, the plan in Figure 5b only costs 39,500. However, this plan cannot be generated in CQP because of limited information sharing and limitations of the collaboration process. In a centralized scenario, the central planner with all planning information available can discover it. We will evaluate the performance gap between CQP and central planning at different levels of information sharing in the next Section.

Summary. In this section, we have presented the design of the CQP service. However, this design is only the first step towards secure decentralized query planning for multi-party database network. We will discuss potential improvements in Section 5. Next section, we will evaluate the CQP service's performance by comparing it with the central planning and the local planning.

4 Evaluation

The goal of the evaluation is threefold. First, we want to compare the CQP with the central planning, which always returns the minimum cost plan. The query planning under the CQP will only perform equal or worse than the central planning, and the goal of our evaluation is to measure this difference quantitatively. The second goal is to compare the CQP with local planning in order to see the benefit of introducing collaboration into query planning. The third goal is to study the trade-off between information sharing and query planning effectiveness under the CQP. Sharing more information can definitively give more chances of finding an efficient plan, and we want to quantitatively measure this trade-off.

Parameters	**Values**	**Explanation**
N_P	9	Number of parties.
N_S	7	Number of database servers each party has. There are $N_P \times N_S = 63$ servers in the simulated database network.
N_R	15	Number of relations each party has. There are $N_P \times N_R = 135$ relations in total.
N_A	$[3, 10]$	Number of attributes a relation has.
$N_t(R)$	$[100, 50000]$	Number of tuples in relation R.
$C(s_i, s_j)$	$[1, 50]$	The communication cost between server s_i and s_j. We used the Erdös-Renyi model to connect two servers at a probability of 0.1. Then the communication cost between any two servers is the cost of the shortest path between them.
T_c	0	The CQP initiation threshold described in Section 3.2. $T_c = 0$ forces CQP to be initiated for every query.
$authProb$	0.5	Data authorization generation probability, which is the probability that a party is authorized to view a relation.
ρ_{DA}	[0,1]	Data authorization sharing probability.
ρ_Q	[0,1]	Query sharing probability.
ρ_M	[0,1]	Metadata sharing probability.

Table 3: Simulation parameters.

4.1 Settings

We have created a multi-party database network simulator whose parameters are listed in Table 3.[4] The simulator will randomly generate information sharing policies between coalition parties based on probability values ρ_{DA}, ρ_Q and ρ_M shown in the table. The process of generating those polices is described in Algorithm 3. Basically, the algorithm goes through each pair of ($info$, *party*) and shares $info$ to *party* with the corresponding probability.

Algorithm 3: Information Sharing Policy Generation

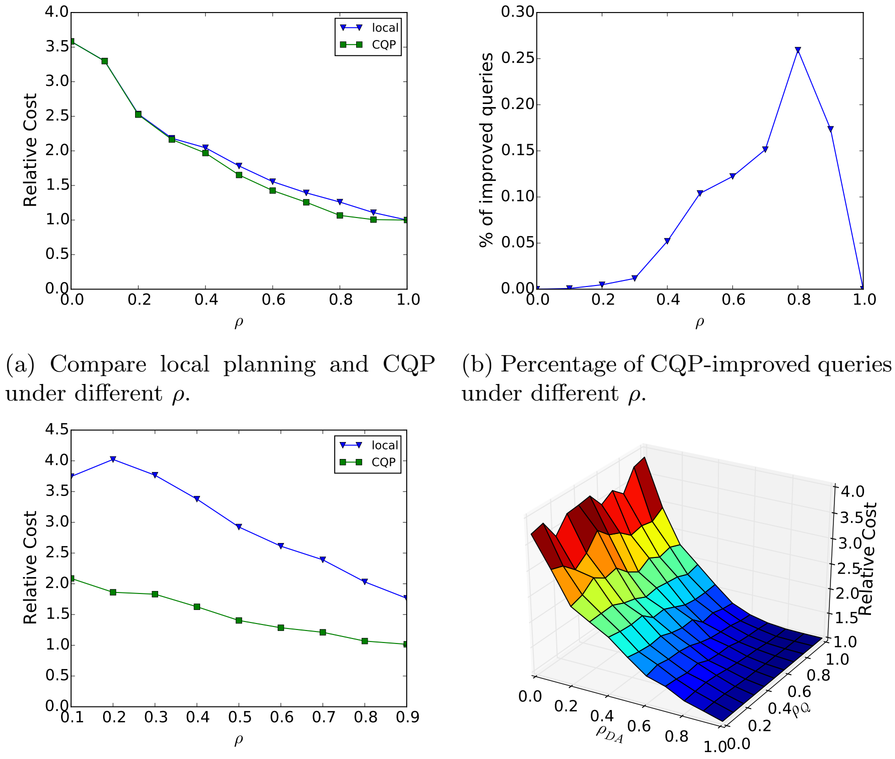
```
input  : Settings of the database network $\mathcal{N}$
output: A set of information sharing policies
policies = {};
infoSet = genPlanInfoSet($\mathcal{N}$);
foreach $P_j$ in parties do
    foreach info in infoSet do
        $P_i$ = owner($info$) ;
        rand = random($P_j$);
        $\rho$ = getShareProb(getType($info$)) ;
        if rand < $\rho$ then
            policies.addNew($P_i \xrightarrow{info} P_j$) ;
return polices;
```

4.2 Results

We conduct 11 query planning experiments with different information sharing probability values $\rho = \rho_{DA} = \rho_Q = \rho_M$ from 0.0, 0.1, ... 1.0. In each experiment, we use CQP, central planning and local planning to plan 10,000 simulated queries. We use the average query cost of central planning as the reference to normalize the average query plan cost for CQP and local query planning. So the cost values shown below are the relative costs compared to the central planning.

The result is shown in Figure 6a, which leads to several observations: (**O1**) a small amount of information sharing can significantly reduce the query plan cost of both CQP and local planning. We can see that when ρ is increased from 0 (no sharing at all) to 0.3, the relative cost drops from 3.5 to 2.2. (**O2**) when the level of information sharing is low ($\rho \in [0, 0.3]$), the CQP and local planning have the same performance. This is because collaborating parties in the CQP do not have enough information to offer help. (**O3**) when there is enough information sharing between parties ($\rho \in [0.4, 0.8]$), CQP will outperform local planning in a

[4] For simplicity, we ignore the sharing of query authorizations. That means whenever P_i shares a sub query with another party P_j, P_i shares all query authorizations of that sub query with P_j as well.



(a) Compare local planning and CQP under different ρ.

(b) Percentage of CQP-improved queries under different ρ.

(c) Compare local planning and CQP for CQP-improved queries.

(d) Relation between ρ_{DA}, ρ_Q and the relative cost.

Fig. 6: Evaluation results.

narrow margin, since the collaborating parties can now utilize their knowledge to help the querier. The narrow margin between CQP and local planning can possibly be explained by two reasons. First, the tested queries are generated from uniform distributions. Therefore, the improvements for certain queries are diluted by other queries. We will investigate this hypothesis further in the next evaluation. The second reason is that the current CQP has certain limitations, so the potential of collaborative query planning has not been fully exploited yet. We will discuss potential improvements in Section 5. (**O4**) as coalition parties further share more information by increasing ρ from 0.8 to 1.0, the gain of CQP is negligible. The cost curves of CQP and local planning converge to the central planning (relative cost is 1) when $\rho = 1$ (all information is shared).

To examine the hypothesis raised in O3, we show the percentages of queries for which CQP outperforms local planning in Figure 6b. We found that (**O5**) as we increase the level of information sharing, the planning result more queries will be improved by CQP. However, after certain threshold ($\rho = 0.8$ in our evaluation), the advantage of CQP over local planning quickly drops to none, which is consistent with O4. We further compare the relative cost of CQP and local planning for CQP-improved queries only in Figure 6c. The result shows that (**O6**) CQP outperforms local planning for certain queries by a large margin. This supports the hypothesis raised in O3. We will further discuss the implication of this observation in Section 5.

Next, we examine the impact of different information sharing parameters. We record the relative cost of CQP under different data authorization sharing probabilities (ρ_{DA}) and query information sharing probabilities (ρ_Q) for 10,000 simulated queries, and show the result in Figure 6d[5]. We see that (**O7**) although increasing ρ_{DA} alone can reduce the cost of query plans (through local planning), the cost reduction is faster if coalition parties share both data authorization information and query information, and utilize CQP.

5 Discussion

Our CQP service is only a first step towards a secure decentralized query planning system for multi-party database networks, and there are potential improvements for the service.

Query Delegation. The current query delegation component (Algorithm 1) in CQP can be improved when we consider it as a decision making process. Basically, the querier needs to estimate the potential gain (e.g. whether the collaborator has enough information, the size of the data, etc.) and lost (e.g. the risk of disclosing a query information to a collaborator) of a potential delegation, and then make a decision based on the trade-off between them. Existing risk-based access control methods could be useful in this scenario [2].

Synthesis. It is also possible that a collaborator fails to find the query plan for the delegated sub query. In our current implementation, the collaborator will return a null plan. However, it is possible that the collaborator obtains some partial results which might still be helpful for the querier. It would be good if the querier can synthesize these partial results as well.

Information Sharing. Our evaluation only considers different levels of information sharing. However, based on observation O5, it seems that certain pieces of query planning information are more important for the queries than others, particularly when the queries are not generated uniformly. In other words, the planning service of CQP can perform better under the same level of information sharing, if information shared is aligned with the query workload. Therefore, an important future work is to create a decision making mechanism to guide coalition parties sharing information in such a way that the risk of information disclosure is low while the performance gain of CQP is high.

[5] We set the metadata sharing probability $\rho_M = 1$ for simplicity.

6 Related Work

Most existing distributed database systems have a central query planner which uses dynamic programming [7, 10] to find the optimal query plan. A few pieces of work have also built decentralized query planning services [11–13], whose common motivation is that in a distributed scenario, accurate information of remote sites is hard to obtain, so query planning cannot be done by a central planner. Papadimos and Maier have proposed the Mutant Query Plan (MQP) [11], in which a query plan is sent to different servers and each server will try to plan and process a part of it. Farnan et al. proposed the harden MQP to encrypt certain parts of a query plan in order to protect query information [6]. This is similar to the query plan filtering step in our CQP service. Mariposa [13] and the query trading framework [12] apply ideas from Economy to the problem of decentralized query planning. In the query trading framework, the querier makes sub query planning requests to collaborating parties, who then do query planning and make offers to the querier. The querier combines different offers into the final plan, and pay for it. The querier-collaborator interaction process in the CQP is similar to and also simpler than the query trading framework, which also supports iterative negotiation, nested delegation, etc. However, CQP protects information confidentiality during a collaborative query planning process.

Another line of related work focuses on protecting the confidentiality of sensitive information in multi-party database network. Vimercati et al. proposed a new data authorization that enables authorizing the join result of several tables to a party based on the concept of join path [3]. Qiang et al. applied pairwise authorizations to enforce horizontal access control of relation data between coalition parties [14]. Farnan et al. studied the query information protection problem in distributed database network, and then extended the SQL syntax to enable query privacy constraints for distributed query execution [4].

7 Conclusion

In this work, we present CQP, a decentralized query planning service that allows multiple parties to jointly plan queries based on limited information shared between them. Our evaluation shows that CQP performs better than local planning when 30% to 90% information are shared between parties. We have also measured the trade-off between information sharing and query planning effectiveness. Potential enhancements of CQP have been discussed in Section 5.

Acknowledgment

We would like to thank Dr. Alessandra Russo and the anonymous reviewers for their valuable comments and suggestions. This research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence and was accomplished under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the author(s) and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

References

1. G. Bent, P. Dantressangle, D. Vyvyan, A. Mowshowitz, and V. Mitsou. A dynamic distributed federated database. In *Second Annual Conference of ITA, Imperial College, London*, 2008.
2. P.-C. Cheng, P. Rohatgi, C. Keser, P. A. Karger, G. M. Wagner, and A. S. Reninger. Fuzzy multi-level security: An experiment on quantified risk-adaptive access control. In *Security and Privacy, 2007. SP'07. IEEE Symposium on*, pages 222–230. IEEE, 2007.
3. S. De Capitani 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.
4. N. L. Farnan, A. J. Lee, P. K. Chrysanthis, and T. Yu. Don't reveal my intension: Protecting user privacy using declarative preferences during distributed query processing. In *ESORICS 2011*, 2011.
5. N. L. Farnan, A. J. Lee, P. K. Chrysanthis, and T. Yu. Paqo: Preference-aware query optimization for decentralized database systems. In *IEEE 30th International Conference on Data Engineering (ICDE)*, 2014.
6. N. L. Farnan, A. J. Lee, and T. Yu. Investigating privacy-aware distributed query evaluation. In *Proceedings of the 9th annual ACM Workshop on Privacy in the Electronic Society*, pages 43–52. ACM, 2010.
7. D. Kossmann. The state of the art in distributed query processing. *ACM Computing Surveys (CSUR)*, 32(4):422–469, 2000.
8. M. Le, K. Kant, and S. Jajodia. Rule enforcement with third parties in secure cooperative data access. In *Data and Applications Security and Privacy XXVII*, pages 282–288. Springer, 2013.
9. M. Le, K. Kant, and S. Jajodia. Consistent query plan generation in secure cooperative data access. In *Data and Applications Security and Privacy XXVIII*, pages 227–242. Springer, 2014.
10. M. T. Özsu and P. Valduriez. *Principles of distributed database systems.* Springer, 2011.
11. V. Papadimos and D. Maier. Distributed queries without distributed state. In *Proc. of WebDB 2002*, pages 95–100, 2002.
12. F. Pentaris and Y. Ioannidis. Query optimization in distributed networks of autonomous database systems. *ACM Transactions on Database Systems (TODS)*, 31(2):537–583, 2006.
13. M. Stonebraker, R. Devine, M. Kornacker, W. Litwin, A. Pfeffer, A. Sah, and C. Staelin. An economic paradigm for query processing and data migration in mariposa. In *Parallel and Distributed Information Systems, 1994., Proceedings of the Third International Conference on*, pages 58–67. IEEE, 1994.
14. Q. Zeng, M. Zhao, P. Liu, P. Yadav, S. Calo, and J. Lobo. Enforcement of autonomous authorizations in collaborative distributed query evaluation. *IEEE Transactions on Knowledge and Data Engineering*, 2014.