Towards Securing Query Processing in Distributed Databases

Qiang Zeng  
Penn State University  
Email: quz105@cse.psu.edu

Jorge Lobo  
IBM Watson Research Center  
Email: jlobo@us.ibm.com

Abstract—Distributed databases involving multiple parties, which may want to share their information selectively, requires effective implementations of distributed disclosure policies. While some research based on policies restricting the visibility of table attributes has been done, access control on tuple sets and horizontally data partition have not been considered. We will discuss the research issues brought up by these extensions.

I. INTRODUCTION

Information sharing across different parties is more and more popular for military, scientific and industrial collaborations. Multiple parties, each of which owns and manages a large amount of information, cooperate to form a distributed system for information exchange and distributed computations. Such scenarios range from multinational military tasks; to international scientific cooperation; to ad hoc coalition formation for humanitarian emergency operations. Independent parties may want to share part of their information selectively, and different subsets of data may be granted to different peers. Effective access control policies should be enforced; otherwise, the concerns of confidentiality and privacy may inhibit wide collaboration. Distributed relational databases provide domain independent declarative query interfaces for the efficient exchange of information. Unlike centralized data warehouses and traditional distributed databases, which can assume a trusted mediator for query processing, information sharing across independent parties leads to new challenges.

There are many granularities of access control that we can impose on a database. Our work starts with table-based policies. At the table level, access control has two dimensions: columns (attributes) and rows (records), which partition relations vertically and horizontally, respectively. Previous work [1], which allows participating databases to specify which attributes of their tables are visible for each subject, resolved the former for distributed query processing. However, how to define and enforce disclosure policies which enable each party to selectively share subsets of their table records is not answered.

The aforementioned work [1] also regards query plan generation and access control verification to be separate tasks. Given a query plan candidate, it verifies whether the plan can be safely executed under the access control. Our position, however, is that in practice policy verification should be integrated into query plan generation. Policy verification is interleaved with other parts of query plan generation, so that an invalid query plan candidate can be aborted before it is fully generated.

Our goal in this paper is to describe with examples the problems that need to be addressed in query processing when horizontal access control is introduced and identify potential approaches.

II. MOTIVATION

We describe a military scenario performing distributed query processing. In this scenario the database is distributed in six nodes, as shown in Figure 1. Server 1 is deployed in the mission area containing a table sensorLocation; another server b at the US command center contains a table sensorInventory. The schemas of the two tables are as shown in Figure 1. The numbers in the figure indicate link latency and the bold lines mark the shortest path tree rooted at d.

![Fig. 1. A military scenario for distributed query processing.](image)

For the sake of presentation, we call table sensorLocation A and sensorInventory B, and assume server c contains another table C, which also contains an attribute TaskID. Assume server a is only allowed to see the the set B1 of tuples with specific task IDs in table B (and B2 = B − B1); Server b is only allowed to see the set A1 of tuples with specific locations in A (and A2 = A − A1), and server d is allowed to see all the tuples.

Handling selections and projections is much simpler than join operations, hence, we focus on joins. To list the owners of the sensors in the mission area inventoried by the command center, server d issues the query A Join B. A naive approach to executing the query is to send both A and B to d and perform the join operation at d. However, the two tables may be large, while the result of A ▷◁ B may be small. Guided by the access control, the join can be re-written as

\[ A ▷◁ B = (A1 ▷◁ B) U (A2 ▷◁ B1) U (A2 ▷◁ B2). \]

Then A1 can be sent to b to perform A1 ▷◁ B, B1 can be sent to a to perform A2 ▷◁ B1, and the intermediate results together with A2 and B2 are then sent to d. This approach splits the join operation into three sub-joins and may reduce data transmission, compared to sending the whole tables to the destination.
Another observation is that servers belonging to the same party may be scattered in a large distributed system, so that they can trust each other to delegate the query execution; we call them buddy servers. In Figure 1, we assume server $e$, which is near from both $a$ and $b$, is a buddy server of $d$; consequently, compared to the cost of sending tables to the distant destination, the network transmission can be reduced if the join operation is delegated to $e$, which then sends the result to $d$.

Both join splitting and query delegation are optimizations pushing the query processing from the query issuer to nodes on the network. They provide additional approaches to the query plan optimization, but can also increase the complexity of query plan generation.

### III. Policies

A distributed database is composed of servers managed by a number of parties. When a query is submitted from some server, it is reasonable and practical to assume that the query is on behalf of the party that manages the server. Therefore, policy definitions in our work are based on parties. Each party defines policies independently at their servers in terms of the hosted tables.

With the aim of horizontal access control, policies are defined as a set of SQL selection statements of the form

\[
\text{Grant view to party-x select * from table-y where constraints}
\]

The semantics is that party-x is allowed to access tuples in table-y that fulfill the constraints which are disjunctions and conjunctions of conditions of the form attribute op value with $op \in \{=, \neq, <, >\}$. This is very similar to the definition of views, and we might be able to explore this similarity during query plan generation.

### IV. Challenge

Incorporating horizontal access control query plan optimizations raises a number of challenges:

**Exponential sub-join operations (C1):** The join splitting approach can potentially reduce communication cost; however, it might result in an exponential number of sub-joins. Consider the query $A \Join B \Join C$ where the three sub-join results due to $A \Join B$ need to be joined with $C$; each of the initial three sub-joins is split to three new sub-joins.

**Non-nested join splitting (C2):** While $A \Join B$ can be executed at intermediate nodes and thus favors further splits of $(A \Join B) \Join C$, $A \Join B \Join 2$ is supposed to be executed at the destination, so that the join splitting for $(A \Join B) \Join C$ may not improve communication cost when compared to sending $C$ to the destination directly. Besides, it partially offsets the benefits from splitting $(A \Join B) \Join C$, which intends to avoid the whole table transmission of $C$ towards the destination.

**Buddy-server choice (C3):** It might be beneficial to exploit buddy-servers for the optimization of join query plans. However, how to pick nearby buddy-servers and arrange them into query execution is challenging, considering the number of possible choices may be large.

**Policy-guided query plan generation (C4):** One of our goals is to investigate how policies can be incorporated into plan selection. Query plan generation and selection involve complex optimizations, thus where and how to put access control verification in place require comprehensive considerations of the whole process.

**Policy transmission (C5):** Consider $(A \Join B) \Join C$ which needs $b$ to sends the results of $A \Join B$ to $c$. Server $b$ needs to consider the policies of $a$ as well as $b$ that affect which tuples in $A \Join B$ are visible from $c$. Thus, we need to work out the potential transmission of policies.

### V. Approaches

The concerns of C1 and C2 can be well resolved by leveraging buddy-servers to consolidate intermediate sub-join results. If buddy-servers exist in the join plan, they can play the role of consolidating sub-join results in the query tree rooted at the buddy-servers. Otherwise, buddy-servers can be arranged as part of query execution plan or servers can communicate and agree on some consolidation point on the fly.

To the end of leveraging buddy-servers (C3), a viable approach is to consider them as part of query refinement after a query plan is selected. For example, in Figure 1, the query plan $A \Join B \Join C$ can be refined as $(A \Join B)_{\text{Consolidate at c}} \Join C$. The arrangement exploits the benefits of buddy-servers without complicating query plan generation.

To integrate access control into traditional query plan generation (C4), there is a gap between a query plan and its security profile. As a result, a query plan cannot be automatically checked against policies. To fill the gap, we would transform the policies to predicates, so that first-order logic could serve as the “glue” and “intermediate language” between query plans and policies. This might allow us to re-use standard query rewriting optimization techniques while ensuring security. In this part we plan to collaborate with Task 5.3 in TA6 which directly addresses plan optimizations.

Policies can be transmitted by push or pull (C5). A server can push policy slices to other servers involved in the plan in advance; or a server can pull (request) needed policies in a lazy fashion. Alternatively, we can learn from the ideas in the link-state routing protocol [2] by regarding policies as link-states: policies are flooded upon being updated and servers exchange knowledge about policies with their neighbours, so that the cost of policy transmission is only related to the frequency of policy changes.

### VI. Final Remarks

A complementary problem, which is not examined in the paper but is part of our goals, is the privacy of query intentions [3]. We will be exploring the problems and approaches identified in the paper in the following two years.

### References

