Report of article : Querying Data under Access Limitations

Student : Mengzi ZHAO

When we want to obtain result from data source, we can fill a web form to generate a query to access the data source to obtain results, but sometimes, it is possible that the value needed to access a source must be obtained from another source, this is the access limitation. We need to use the recursive query in this case : output of one query is the input of another query and repeat this step to obtain the result. The most important indicator for the query plan is the number of access to the data source, so the recursive query can cause a high number of access, in this article, authors introduce a system Toorjah to solve this problem.

Before starting to study the Toorjah system, there are some base knowledge that we need to know.

At first, we need to know how to obtain the answer of the query with access limitations. I take an example from the article, we have a relation schema (Example 2): $R = \{r_1^{io}(A,C), r_2^{io}(B,C), r_3^{io}(C,B)\}$, the query over R is $q_1(B) \leftarrow r_1(a_1,C), r_2(B,C)$, we have $r_1 = \{< a_1,c_1 >, < a_1,c_3 >\}, r_2 = \{< b_1,c_1 >, < b_2,c_2 >, < b_3,c_3 >\}, r_3 = \{< c_1,b_2 >, < c_2,b_1 >\}$, we can know that the input of the query is a_1, we need to query the relation who contains a_n as the input, we find there are r_1 who contains a_1, we can obtain c_1, c_3, by using the same principle, we use the c_1, c_3 to query the r_3, there is just c_1 in r_3 , so we can obtain the b_2, b_2 can be one of results, then we use b_2 to query r_2, we can get c_2, then we take c_2 to query r_3, we can obtain b_1, b_1 is also a result for the query, then we use b_1 to query r_2, but we find we get another time c_1, so b_1, b_2 is the final result, b_3 is not obtainable.

But we can notice that some access to data source are not necessary, because sometimes the relation unused in the query will not help to find answers, for example (Example 3), there is a relation schema $R = \{r_1^{io}(A,B), r_2^{io}(B,C), r_3^{io}(C,A)\}$ and a query $q(C) \leftarrow r_1(a,B), r_2(B,C)$, we can find r_3 does not appear in the query, so r_3 is not helpful to find the answer, results are extracted by the tuples obtained from r_1 by parameter A binding value a.

To determine the relevance of relations, authors introduce a way to create dependency graph. At first, we need to create constant-free query by eliminating constants in the query. For example, $q(Y) \leftarrow r(a,Y)$, there is constant a in the query, it acts as an artificial relation l_a having just one output whose content is tuple $< a >$, we can replace the query by $q(Y) \leftarrow r(A,Y), l_a$.

We present the parameters which are in the query by the black node and which are not in the query by the white node, each relation represents is called source, if 2 nodes u and v are in the same abstract domain, u is output and v is input, then we can construct arcs between these 2 nodes. For example (Example 4), for the relation schema $R = \{r_1^{io}(A,B), r_2^{io}(B,C), r_3^{io}(C,A)\}$ and query $q(C) \leftarrow r_1(a,B), r_2(B,C)$, we can obtain a dependency graph as below :

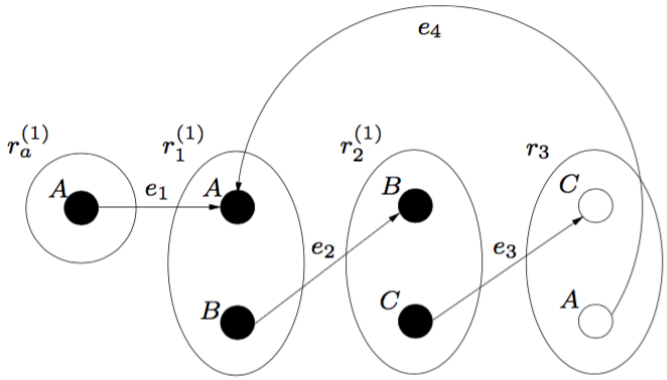

Figure of example 4

Then we need to eliminate this graph to decrease number of access to data source. Before doing this, we need to know some definitions, the strong arc $u \to v$ means : these 2 nodes are black and joined in the query and the source of v does not give random value to other relations in the query, all remain arcs are weak arcs, the strong arc represents that all useful results obtained from relation of v are extracted by using the values offered by u. There are some conditions to do the eliminations : the incoming weak arc can be deleted when there is an incoming strong arc on this node, when the 2 nodes of arcs are white, this means this arc has no relation to this query, and when the output node of an arc is white, this means the output is not useful for this query. (Example 5) Let's execute a running example by using the graph above, at first we need to find the candidate strong arc at first, the candidate strong arc is arcs whose nodes are black and are joined in the query, we can find the arc e_1,e_2 of the graph are candidates strong arc, and we need also find the delete arcs, we can find e_3, e_4 whose nodes are not all black and we can also find that e_1 is strong arc and e_4 is weak arc so e_1 dominates e_4. After deleting e_3, e_4, we find the relation r_3 has no arc connecting to other relations, so we can delete the relation r_3 from the graph. So we obtain a graph as below after elimination :

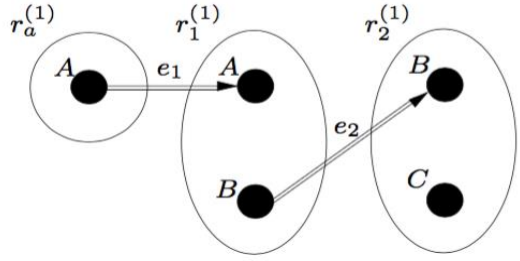

Figure of example 5

Then authors define a definition « minimality » of query plans to find the minimal execute plan for the query. There are $\forall - minimality$ and $\subset - minimality$, for each instance D (a given

database) and every query plan π' of query q, $Acc(D,\pi) \subseteq Acc(D,\pi')$, but sometimes there is no $\forall - minimality$ on the given query. For example (Example 6), we have a query $q(X) \leftarrow r_1(X), r_2(Y)$ and a relation schema $R = \{r_1^{io}(A), r_2^{io}(B)\}$, we can query r_1 or r_2 at first, we suppose that there is an instance D that $r_2 = \emptyset$, in this case the π is not $\forall - minimal$, because we do not need to access any plan then we can get $q^D = \emptyset$, symmetrically, the result is the same, so in this case there is no $\forall - minimal$ plan, so then authors define $\subset - minimality$ which is a less strong notion and by using this notion, there is always a minimal plan, we assume that there are π, π' 2 query plans, we have $\pi \subseteq \pi'$ when $Acc(D,\pi') \subseteq Acc(D,\pi)$ for every instance D and $Acc(D',\pi') \subseteq Acc(D',\pi)$ for an instance D'. If there is no $\pi'' \subseteq \pi$ we can say π is $\subset - minimal$. If a $\subset - minimal$ plan of a query is minimal, a $\forall - minimal$ plan exists. To construct a $\subset - minimal$ plan, we need to start from a minimal conjunctive query whose body atoms is not a subset of another query's body atoms, we can express the $\subset - minimal$ plan by Datalog program and then it is evaluated.by Datalog semantics to guarantee access minimality. When there are more than one source referred to the optimized d-graph, we need to determine a order to access the different sources with these rules : for the weak arc $u \rightarrow v$ and strong arc $u \rightarrow v$ $src(u) \preccurlyeq src(v)$, if the source is traversed by a cyclic d-path, it should have the same order as the other sources in the cycle, the other sources outside the cycle have different order, these steps can be used on the relations corresponding to the sources. The number of possible orders influences the $\forall - minimality$, because when there is just one order, $\forall - minimal$ plan can exist, if not, the $\forall - minimal$ plan is lost whatever the order is. After we set an order and sources with different orders, we can determine a number $pos(s)$ for every source s, $pos(s_i) < pos(s_j)$ if $s_i \prec s_j$.

Now we can construct the $\subset - minimal$ plan, at first, we rewrite the query by replacing the relation in the query by another atom having the same parameters but having a different relation name. For every predicate of the query, we use a new predicate who works as a sort of cache to store the tuples that we extract from the predicate. The form of cache relation has form as below : $\hat{r}(I_1, \dots, I_n, O_1, \dots, O_m) \leftarrow r(I_1, \dots, I_n, O_1, \dots, O_m)$, $s_1^{\hat{r}}(I_1), \dots, s_1^{\hat{r}}(I_2)$, $I_1, \dots, I_n$ are input arguments, $O_1, \dots, O_m$ are output arguments, $s_1^{\hat{r}}(I_n)$ are new predicate name corresponding input argument. At last, the program add a fact for every artificial relation to eliminate the constants from the query. Let us look at an example (Example 7), we use the optimized graph of example 4, there is only one possible order : $\widehat{r_q} \prec \widehat{r_1} \prec \widehat{r_2}$, we obtain the result as below :

$$
\begin{aligned}
q(X) &\leftarrow \hat{r}_a^{(1)}(A), \hat{r}_1^{(1)}(A,B), \hat{r}_2^{(1)}(B,C) \\
r_a^{(1)}(A) &\leftarrow r_a(A) \\
\hat{r}_1^{(1)}(A,B) &\leftarrow r_1(A,B), s^A(A) \\
\hat{r}_2^{(1)}(B,C) &\leftarrow r_2(B,C), s^B(B) \\
s^A(A) &\leftarrow \hat{r}_a^{(1)}(A) \\
s^B(B) &\leftarrow \hat{r}_1^{(1)}(A,B) \\
r_a(a) &\leftarrow
\end{aligned}
$$

The s^A and s^B represents the incoming strong arc, and we can notice that the r_3 is not present in the code because as we study before, the r_3 is irrelevant for this query. The execution strategy is fast-failing to avoid the redundant access and it guarantee to calculate the same answer as the fixpoint semantics for the datalog program, when it knows the answer is empty and then it stops immediately to avoid other access, so this makes us obtain a minimal query plan.

Now we can commence to study the result of the Toorjah system who use the access minimization approach that we study before. There are some other aspects that this system considers in practice, the cache database who stores results of a query, access tables who stores access tuples to access relations constructed by using the minimal query plan and wrappers who wrap data sources to make the query process in a good way. Toorjah tries to avvess in parallel sources to improve query answering, it extracts answers as early as possible when the access tuple is generated from the CDG. The table as below contains the average time of Toorjah system comparing to the naïve approach.

atoms	naive	opt.
2	9,310 ms	684 ms
3	12,161 ms	1,732 ms
4	10,198 ms	959 ms
5	14,879 ms	1,134 ms
6	15,474 ms	1,247 ms

We can notice that the result of experiments shows that answer the query is costly but it is worthy comparing to the execution time that it saves.

There are some shortcomings in this paper, at the beginning of this paper, authors indicate that the number of access is the most important indicator to measure if the query processing is good or not, but there are several different indicators who can influence the query time, for example, the size of the data in the data source, there are many data, the effectiveness of server can also influence the result because when there are some problems it can increase the time to query etc.

The main contribution of this paper introduce an approach to that permits to query the source under the access limitations, create the dependency graph to represent the query processing and the approach to eliminate the graph to decrease number of times of the access to data source, then authors presents the minimality approach to construct the minimal query plan, authors use these approaches to realize a Toorjah system that can optimize the query processing, even if it is costly to find the answer but comparing the time that this system save, it is worthy to create this system.