Abstract

Most previous works on query optimization techniques deal with conjunctive queries only because the queries with disjunctive predicates are complex to optimize. Hence, for disjunctive queries, query optimizers based on these techniques generate plans using rather simple methods such as CNF- and DNF-based optimization. However, the plans generated by these methods perform extremely poorly for certain types of queries. In this paper, we propose a new query optimization method, union-pushdown, for disjunctive queries. This method is composed of four phases, and each phase utilizes some advantageous techniques of CNF- and DNF-based methods. We analyze the performance of the union-pushdown plan against those of conventional plans and show that union-pushdown can be applied to various disjunctive query types without performance degradation.

1. Introduction

Query optimization is a process of generating an “optimal” query execution plan for a given query. A non-procedural query can be transformed into a number of execution plans, and the query optimizer should search for the cheapest among these plans. Hence, the query optimizer is one of the most complex components of a DBMS, and for this reason, is still the subject of numerous research efforts. The traditional focus of query optimization has been on the choice of operator order and operator algorithms [3, 6, 8]. Operator order has been determined mainly by heuristics on selection pushdown [11] and join enumeration [6]. Operator algorithms have been selected by applying a cost estimation formula. In recent years, as the application areas of DBMSs become diverse, the need for extensible DBMS (which can accommodate new data types and functions) increases and, therefore, much research has been done on query optimizer architecture which can be customized for different application environments, (e.g., [1, 10]).

However, most of the works mentioned above have been focused only on conjunctive or select-join-project queries because disjunctive queries (which have “or” operators in their predicates) are not easy to handle and complex to optimize. In most DBMSs, in order to generate plans for disjunctive queries, their predicates are translated into a normal form such as DNF (Disjunctive Normal Form) and CNF (Conjunctive Normal Form)[4, 7, 10]. These optimization techniques are very simple and easy to understand, but they have very serious problems in terms of performance. In a recent paper [9], a plan generation method for disjunctive queries, called bypassing join, was proposed. However, this technique is effectively utilized only when there is an opportunity of avoiding unnecessary join operations, and in other cases, the generated plan is almost the same as the CNF-based plan. Muralikrishna [5] proposes the method that merges two or more disjunctive predicates to minimize the number of selection operations. However, this method focuses only on selection predicates and ignores join predicates.

In this paper, we describe a new query optimization method, union-pushdown, for disjunctive query. The chief goal of our method is to push down the union operator used for processing disjunctive predicates in the query tree. This is a hybrid method of CNF- and DNF-based optimizations, and the union operators to be pushed down are determined according to criteria which we will present. We believe that, in many cases, an early evaluation of the union operator will bring great performance enhancement. Union-pushdown is implemented in the rule- and cost-based query optimizer which we have currently developed, and all phases of our method are represented by a rule language. However, in this paper, we only describe the process of query transformation and omit rule specifications.

The remainder of this paper is organized as follows. In
the next section, we motivate the union-pushdown technique by means of an example and compare this with conventional methods such as CNF- and DNF-based optimization and bypassing join. Then, in Section 3, we describe the transformation process for generating the union-pushdown plan with some examples. Performance study is discussed in Section 4, and we finally present conclusions in Section 5.

2. Motivating Example

To illustrate the motivation of our proposal, we compare the relative cost of union-pushdown plan with those of conventional optimization schemes. We use the following example database, which is used as a running example throughout this paper.

```
faculty(fid, SSN, f_name, dept)
relation of faculty members.
attribute : faculty ID, social security number, faculty name, department.
size : 3000 records, 30 pages.
index : 2 b-tree indices on fid and SSN, respectively.

course(c_name, fid, enrollment)
relation of courses offered.
attribute : course name, faculty ID of instructor, number of students.
size : 10000 records, 10 pages.

society(s_name, president, vice-pres)
relation of academic societies.
attribute : society name, SSN of president, SSN of vice-president.
size : 1000 records, 10 pages.
```

Consider the following query.

“At among the faculties who teach 'program' or 'math', list faculty names and societies in which he/she serves as a president or a vice-president.”

This query is represented in SQL as follows:

```
SELECT F.f_name, S.s_name
FROM faculty F, society S, course C
WHERE
(F.SSN = S.president or F.SSN = S.vice-pres) and
(F.fid = C.fid) and
(C.c_name = 'program' or C.c_name = 'math')
```

For join operation, we shall only consider nested loop join (NJOIN) algorithm and sort-merge join (MJOIN) algorithm. Nested loop join is a unique join algorithm which can evaluate disjunctive predicates. Sort-merge join, on the other hand, is representative of algorithms which cannot evaluate such predicates. For selection operation, we use sequential scan (SSCAN) and indexed scan algorithm (ISCAN). The union operation (UNION), which is used for processing disjunctive queries, is carried out by sorting and merging all the input relations in order to remove duplicate records [3]. However, since the union operation used in the bypassing technique [9] does not require duplicate removal, we represent this special union operator as UNION-.

Figure 1–4 show the expected optimal plans for the four alternative optimization methods. In the DNF-based optimization, the predicates are translated into DNF. Then, each conjunct is optimized independently and merged by union operation. The final optimal plan is shown in figure 1. The cost - the number of disk block I/Os - of this plan is about 12,000\(^1\). Most of this cost is due to the final union operation. This operation must eliminate duplicates, which requires sorting of all its sub-results.

The CNF-based plan is shown in Figure 2. In order to generate this plan, predicates of the query are translated into CNF and each disjunct is treated as an atomic predicate. The expected cost of this plan is about 5,600. The major limitation of CNF-based optimization is obvious in this example, that is, for operators with disjunctive predicates, we must choose only those algorithms that are able to evaluate disjunctive predicates\(^2\).

Bypassing technique [9] is designed so that records which are determined to satisfy the condition of the query are not subject to unnecessary join operations and are output to the final result as soon as possible. But, as shown in figure 3, if the given query does not have such a chance, this technique has no effect. In this example, the generated plan and the expected cost are very similar to the CNF-based case.

---

\(^1\) For cost estimation, we use the cost functions defined in System R [8] and assume selectivity factors of join and selection predicates to be 0.001 and 0.01, respectively.

\(^2\) Although, in figure 2, we can employ “scan and probe” join which sequentially scans the relation society and for each tuple, probes the relation faculty using b-tree index on SSN, this algorithm is applicable to only this type of disjunctive predicate. Therefore, in this example, we exclude this algorithm because it cannot be applied to general disjunctive predicates.
Finally, The plan generated by the union-pushdown is shown in figure 4. Unlike the DNF-based plan, the union operation is performed right after the join of faculty and society. Intuitively, it seems that this plan performs worse than DNF-based plan because the union operation increases the size of intermediate results. However, if the number of records processed by the union operator is small relative to the overall number of records handled by a given query, the results of this operation will have little effect on the query performance. In this example, because the selectivity factor of the join predicate is 0.001, we can expect that the number of records to be merged by the union operator is very small relatively. This plan also avoids the overhead of merging many sub-plans in the final phase which was required in the DNF-based plan. Moreover, the sort operation which was required in the sort-merge join in the top-level of the tree is not necessary since the sort order of results in the union operators can be pipelined into the sort-merge join. The expected cost of this plan is 2,400, which is half the cost of the CNF-based plan and of the bypass plan, and only 1/5 the cost of the DNF-based plan.

Note that the early evaluation of union operation does not always guarantee a good performance. In the above query, if the selectivity factors of join and selection predicates are same, the DNF-based plan shows better performance than the union-pushdown. Therefore, criteria for deciding whether a union operator should be pushed down or not are not presented.

3. Query Optimization

We describe in this section the optimization process for generating union-pushdown plan, and as an example, show the process of building the optimal plan of figure 4.

3.1 Query Representation

For internal query representation, we use a LISP-like notation which has been defined in [1]. This notation enables the query to be represented uniformly and independently from any particular database model and query language. We define a set of operators that will be used to describe our method:

- \((\text{SELECT } p \mid t)\): logical operator of selection
- \((\text{JOIN } p \mid t \_\text{list})\): logical operator of join
- \((\text{UNION } t \_\text{list})\): union operator
- \((\text{ISCAN } p \mid t)\): indexed scan algorithm
- \((\text{SSCAN } p \mid t)\): sequential scan algorithm
- \((\text{NJOIN } p \mid t_1 \mid t_2)\): nested loop join algorithm
- \((\text{MJOIN } p \mid t_1 \mid t_2)\): sort-merge join algorithm

where \(p\) is a list of predicates, \(t\_\text{list}\) is a list of relations, \(t\) is a relation and \(t_1, t_2\) are outer and inner relation respectively.

Although this set of operators is not complete for representing all queries, it is sufficient to express the process of the query transformation. The sort operator is assumed to be embedded in MJOIN and UNION.

For simplicity, we express the predicates of the example query shown in Section 2 in the following form:

\[
J_1(F, S) \setminus J_2(F, S) \setminus J(F, C) \setminus (S_1(C) \setminus S_2(C))
\]

where

- \(J_1(F, S)\) is \((F.SSN = S\text{-president}),
- \(J_2(F, S)\) is \((F.SSN = S\text{-vice-pres}),
- \(J(F, C)\) is \((F.fid = C\text{-fid})\),
- \(S_1(C)\) is \((C\text{-name} = \text{‘program’})\) and
- \(S_2(C)\) is \((C\text{-name} = \text{‘math’})\).

The query plan of figure 4 can be represented as follows:
Therefore, we can conclude that the generated plan will show good performance if the union operator for a given disjunct satisfying one of the above criteria is pushed down. The selected disjuncts based on the above criteria are treated as atomic predicates and we call these disjuncts as semi-atomic predicates. If all disjuncts are selected as semi-atomic predicates, the generated plan will be the same as the CNF-based plan. And if no disjunct is selected, the generated plan will be the same as the DNF-based plan.

In the second phase, the predicates are distributed to the operators of the query. In order to distribute the predicates, we begin by re-normalizing the predicates into DNF. Note that, during the normalization, semi-atomic predicates are treated as atomic predicates. Let the following be the transformed DNF:

\[
p_1 \lor p_2 \lor \cdots \lor p_m
\]

where \(p_i(1 \leq i \leq m)\) consist of atomic or semi-atomic predicates connected with \(\wedge\). (3)

Then, for each conjunct \(p_i\), an order of join and selection operators are determined. If we use selection pushdown heuristics [11], a sub-query for \(p_i\) is translated into as follows:

\[
\begin{align*}
\text{(JOIN } p & \text{ (SELECT } s_1 \text{ } t_1) \\
& \vdots \\
& \text{(SELECT } s_n \text{ } t_n) \\
\end{align*}
\]

where \(p = p_1 \wedge s_1 \ldots \wedge s_n\) and \(s_j(1 \leq j \leq n)\) consist of zero or more, atomic or semi-atomic predicates containing only selection predicates of relation \(t_j\). (4)

For the join operation, if we assume that \(t_j\) and \(t_k\) are chosen to be joined first, we can translate each \(p_i\) as follows:

\[
\begin{align*}
\text{(JOIN } p & \text{ (JOIN } J(t_j, t_k) \text{ } t_{j+1} \\
& \ldots \\
& t_{k-1} \text{ } t_k) \\
\end{align*}
\]

where \(p = p_1 \wedge J(t_j, t_k)\), and \(J(t_j, t_k)\) consist of zero or more, atomic or semi-atomic predicates containing join predicates of \(t_j\) and \(t_k\), or selection predicates of \(t_j\) and \(t_k\). (5)

For the remaining join operations, we can apply the same transformation to distribute the join predicates.

In the third phase, the operators which have semi-atomic predicates are decomposed using union operator so that each operator contains only conjunctive predicates. Since operators which contain predicates are only selection and join, we can define the decomposition as follows:

\[
\begin{align*}
\text{(SELECT } p & \text{ } t) \text{ is decomposed into} \\
\text{(UNION } & \text{ (SELECT } s_1 \text{ } t) \\
& \text{(SELECT } s_2 \text{ } t) \\
\end{align*}
\]

where \(p = s_1 \lor \cdots \lor s_n\), and \(s_j(1 \leq i \leq n)\) is a single conjunct. (6)

\[
\begin{align*}
\text{(JOIN } p & \text{ } t_1 \text{ } t_2) \text{ is decomposed into} \\
\text{(UNION } & \text{ (JOIN } j_1 \text{ } t_1 \text{ } t_2) \\
& \text{(JOIN } j_n \text{ } t_1 \text{ } t_2) \\
\end{align*}
\]

where \(p = j_1 \lor \cdots \lor j_m\), and \(j_i(1 \leq i \leq m)\) is a single conjunct. (7)
The fourth phase translates each operator into an applicable algorithm. Because all disjunctive predicates are decomposed, each operator can be translated into any of the available algorithms without restrictions. However, decomposition does not always result in a good plan. Assume that we have decomposed the operator into several sub-operators. In the case where one of these sub-operators is mapped into an algorithm capable of evaluating disjunctive predicates, the plan will show a worse performance than the CNF-based plan. In this case, therefore, the decomposed operators are re-merged and translated into an algorithm which can evaluate merged predicates. Furthermore, even if we assume that all the decomposed operators are mapped into the efficient algorithms such as indexed scan algorithm, we cannot decide which is the better plan between decomposed and re-merged plans. In this case, we select the optimal plan based on heuristics or cost-based optimization techniques.

For example, consider the following which represents the example query shown in Section 2.

\[(\text{JOIN } P \land F \land S)\]

where \(P = (J(F, S) \lor J_{2}(F, S)) \land J(F, C) \land (S_{1}(C) \lor S_{2}(C))\) (8)

Assume that \((J(F, S) \lor J_{2}(F, S))\) and \((S_{1}(C) \lor S_{2}(C))\) are selected as semi-atomic predicates. Then, after the second phase, (8) is transformed as follows:

\[(\text{JOIN } J(F, C) (\text{JOIN } J_{1}(F, S) \lor J_{2}(F, S) (\text{SELECT } F (\text{SELECT } S))\}) (\text{SELECT } S_{1}(C) \lor S_{2}(C) C)\) (9)

In the third phase, (9) is decomposed as follows:

\[(\text{JOIN } J_{1}(F, C) (\text{UNION } J_{2}(F, S) (\text{SELECT } F (\text{SELECT } S)) (\text{SELECT } S_{1}(C) C) (\text{SELECT } S_{2}(C) C)\})\) (10)

In the final phase, each operator is translated into an applicable algorithm and re-merged if necessary. The final optimal plan selected is shown in (1). In this plan, the join operator of faculty and society are divided into two sub-joins because the union of two sort-merge joins is expected to show better performance than a single nested loop join. On the other hand, the selection operators of course are re-merged into one sequential scan algorithm because course does not have any index and as a result the two selection operators should be translated into sequential scan algorithms.

The union operation is accomplished by sorting and merging all input relations. But, in contrast to the sort-merge join, the input relations of the union operator can be sorted in any order since it needs not evaluate any predicate. Therefore, if the input relation of some operator is an output of union operation and the operator requires the input relation to be sorted in a particular order, the union operator can sort its input relations in such order, thus, eliminating a sort operation.

4. Performance Evaluation

In this section, we present our study on the performance of the union-pushdown and compare it with the other optimization techniques. Performance is evaluated based on the cost estimation formula used in the System R [8]. We measure performance with respect to linear join query[6], where all relations are connected consecutively. In this query, the predicates are assumed to have the following form when translated into CNF:

\(S(t_{1}) \land \ldots \land S(t_{n}) \land J_{1}(t_{1}, t_{2}) \land \ldots \land J_{n}(t_{n-1}, t_{n})\)

where \(S(t_{i}) (1 \leq i \leq n)\) is a disjunct containing only selection predicates of relation \(t_{i}\) and \(J(t_{i}, t_{i+1}) (1 \leq j \leq n-1)\) is a disjunct containing only join predicates of relations \(t_{i}, t_{i+1}\), and all disjuncts have two predicates connected with \(\lor\).

We have examined the expected cost of query execution for various selectivity factors and number of relations. We assume that each relation has 1,000 records and is stored in 10 disk blocks. Figure 5 shows the expected cost of linear join query as the selectivity factors of the selection predicate vary from 0.01 to 0.5. Number of relations joined was three for every case. The selectivity factor of join predicates is assumed to be \(1/10\) of the selectivity factor of the selection predicates. As shown in this figure, union-pushdown shows the best performance throughout the entire range of selectivity factors. The reasons for such a result have already been illustrated in the motivating example of Section 2. When the selectivity factor is less than 0.15, the DNF-based plan shows better performance than the CNF-based plan. However, the cost of DNF-based plan increases exponentially as the selectivity factor increases because each sub-plan produces many records to be examined in order to eliminate duplicates in final union operator. Bypass join plan shows almost the same performance as CNF-based plan. The bypassing technique is profitable only when there is an opportunity for avoiding unnecessary join, but in this case, there is no such chance and this plan performs almost the same as CNF-based plan.

Figure 6 shows the expected cost as the number of relations increases from 2 to 10. The selectivity factors of selection and join predicates are fixed at 0.1 and 0.01, respectively. Again, union-pushdown shows better performance than others. But as the number of relations increases, re-merging decomposed sub-operators will be beneficial because the number of intermediate union
operators increases more rapidly than the number of relations. If all of the decomposed sub-operators are re-merged, this plan will be same as the CNF-based plan. When the number of relations are small, the DNF-based plan shows better performance over the CNF-based plan. But as the number of relations increases, the cost of this plan increases exponentially because the number of sub-plans quadruples as each single relation is added to the query.

A performance study for linear join query has been carried out between union-pushdown and conventional methods. In this experiment, union-pushdown had the best performance throughout the entire range of selectivity factors and the number of relations. Therefore, we conclude that union-pushdown can be applied to various disjunctive query types without performance degradation.

5. Conclusion

In conventional methods, performance is heavily affected by the type of disjunctive query. To solve this problem, we have designed the union-pushdown incorporating the advantageous techniques of both CNF and DNF-based optimizations. The primary goal of our method is to push down the union operator in the query tree. And, in order to select union operators to be pushed down, we presented the criteria which is helpful for deciding whether it is profitable to push down the union operator. Although we describe the criteria in an abstract manner, we believe that these will serve as a foundation of detailed algorithms for selecting the union operators to be pushed down. The details of criteria and these proof are left to future works.

A performance study for linear join query has been carried out between union-pushdown and conventional methods. In this experiment, union-pushdown had the best performance throughout the entire range of selectivity factors and the number of relations. Therefore, we conclude that union-pushdown can be applied to various disjunctive query types without performance degradation.

References