2000

Cache-based Query Processing for the Boolean Retrieval Model

Jae-heon Cheong
Seoul National University

Sang-goo Lee
Seoul National University

search

Cache-based Query Processing for the Boolean Retrieval Model

Jae-heon Cheong and Sang-goo Lee[*]

Department of Computer Science

Seoul National University

San 56-1 Shillim-dong Kwanak-gu, Seoul, Korea

{cjh, sglee}@cygnus.snu.ac.kr

***Abstract-* We propose a new method of processing general Boolean queries utilizing previous query results stored in a result cache in a mediator architecture. A simple but noble normalization form is developed to describe keyword-based Boolean queries and the content of the result cache. We propose Boolean query processing algorithms based on this form of presentation that utilizes the result cache. We show that the proposed method theoretically guarantees improved performance over the conventional query processing method without using a cache.**

I. INTRODUCTION

The fast and wide spread of the Internet has caused exponential increase in the amount of information that can be obtained from it. An Internet search engine plays the role of a lighthouse in this sea of information. The recent growth in the number of search engines has generated the need for mediator systems. Mediator systems provide users with seamless access to information from diverse and heterogeneous information sources (collections) on the Internet [1].

In general, the collections that a mediator system deals with are geographically distributed over the network. Therefore, if we assume that every collection is configured optimally, the performance of the mediator system is mainly influenced by the data transmission time between the mediator and the individual search engines. In order to reduce the size of the query results being transferred, most contemporary mediator systems makes use of results of previous queries stored in a *result cache* in answering the current query. A result cache can also be used to answer a query partially or completely even when some of the collections are temporarily unavailable. Unfortunately, however, most cache-based query processing methods adopted by current mediator systems are based on a simple query model considering only conjunctive queries.

Cache representation and inference based on the general Boolean model give rise to complicated problems. First of all, it is not feasible for a result cache to be equipped with a keyword index, such as an inverted file, since it cannot be known in advance what keywords will occur in queries in a certain time frame. Thus, if a query contains at least one keyword not already in the result cache, the entire cache must be searched exhaustively for proper sub-results that satisfy the current query. Another complication arises when collections do not support the '*NOT*' operator. To retrieve the the portion of the result for a query Q that is not already in the result cache C (we call this a *miss result*), a new query 'Q *AND NOT* C' (the *miss query*) would be sent to the collections. If, however, some of the collections do not support the '*NOT*' operator, the mediator cannot retrieve those results directly.

In this paper, we present an efficient method that makes use of the result cache to minimize the size of result that needs to be retrieved from the target collections for a keyword-based Boolean query. The main idea of our method is to partition a result cache into several disjoint sub-results. Each sub-result is represented by a sub expression of the original one. By doing this, we can considerably reduce the search space of a result cache and can find query results more efficiently. We propose a new normalized form, a variation of the disjunctive normal form (DNF), to describe queries and the cached results. Also presented are algorithms to process Boolean queries with a result cache. The miss results can be answered effectively even when the target collections do not support the '*NOT*' operator. We show that the proposed scheme is better in terms of performance than schemes that do not use cached results.

Other issues pertaining to cache management, such as replacement strategies and consistency maintenance, are beyond the scope of his paper and are left to future work.

In the next section, we present other works related to the current topic. A basic cache-based query-processing model is presented in section III. In section VI, we propose our new method of cache-based query processing. Performance issues are considered in section V. We conclude the paper in section VI with a summary of our contributions and a brief discussion on future directions.

II. RELATED WORKS

There have been a number of articles on semantic data caching [2, 3, 4, 5, 6]. In [2], the detailed method of rewriting an initial query using a cached query is described. However a single database environment is assumed and the algorithm for finding a query match (a hit query in our context) is incomplete. A query optimization method is proposed in [3], which uses cached queries in a mediator system named HERMES. This system, however, deals with only simple conditions of the form $c_1\theta c_2$ (where θ is any of $<$, $\leq$, or $=$). Cache replacement strategies based on recency and semantic distance of cached queries are considered in [4], while consistency problems of conjunctive queries are dealt with in [5]. The query approximation method presented in [6] also considers only conjunctive queries.

[*] Author's work was supported in part by research fund provided by the Korea Research Foundation, Support for Faculty Research Abroad.

search

Perhaps the work most closely related to semantic caching in the database literature is [7], which deals with materialized views [7]. This work, however, deals only with conjunctive queries in SQL. A scheme for rewriting queries under the Boolean query model considering the capabilities of information sources is presented in [8]. The rewriting method is carried out predicate by predicate.

In [9], a query history based virtual index (QVI) is proposed. It is based on the observation that queries are repeated over time and users. However, only simple keyword queries like 'Retrieve documents that contain 'Database' and 'Warehouse' were considered. We propose to extend this basic idea to the general Boolean retrieval model.

III. PRELIMINARIES

In this section, we present some basic concepts necessary to build our new method. First of all, we adopt the keyword-based Boolean query model, where a query, for instance, "Retrieve documents that contain both keywords 'digital' and 'library'" can be represented by a Boolean expression 'digital AND library'. For notational convenience, we use the symbol '$\wedge$', '$\vee$', and '$\neg$' to denote the Boolean operators '*AND*', '*OR*', and '*NOT*', respectively.

A. Query Model

In a keyword-based Boolean query model, queries are defined recursively as follows:

Definition 1 (Query):

1. A keyword is a query.
2. If A is a query, then $(\neg A)$ is a query.
3. If A and B are queries, then $(A \wedge B)$ and $(A \vee B)$ are also queries.
4. All queries are generated by applying the above rules.

On the Internet, the target of a query is the set of documents. A document can be defined as a conjunction of keywords that occur in it. A document is an answer to a query if the keywords in the document are such that make the query expression true. A set of documents can be represented as a Boolean query expression such that the result of the expression would be exactly the set of those documents.

Definition 2 (Query Result): For a given query Q, the query result of Q, denoted by $[Q]$, is defined as a set of documents that satisfy Q. The result of a query $\neg Q$ is defined as $U - [Q]$, where U is the universal set of documents. The result of a query $A \wedge B$ and $A \vee B$ is defined as $[A] \cap [B]$ and $[A] \cup [B]$ respectively.

If the results of two different queries are the same, we say that these two queries are equivalent. A query is a sub-query of another query if the results of the former are fully contained in those of the latter.

Example 1: Suppose the set of keywords is $\{t_1, t_2, t_3, t_4, t_5, t_6\}$ and the whole set of documents is $\{ t_1, t_2 \wedge t_3, t_1 \wedge t_3 \wedge t_5, t_4 \wedge t_6, t_5 \wedge t_6\}$. Then, the set of results of a query $(t_1 \vee t_2) \wedge (t_3 \vee t_5)$ is $\{t_2 \wedge t_3, t_1 \wedge t_3 \wedge t_5\}$. ■

To simplify our query model, we assume that all keywords are independent of each other.

B. Query Processing with A Result Cache

A result cache is the collection of previously posed queries and their results.

Definition 3 (Result Cache): Suppose $Q_1, Q_2, \ldots, Q_n$ are queries that have been issued by users up to this point. The current *result cache*, denoted by C, is defined as the disjunction of previous queries $(Q_1 \vee Q_2 \vee \ldots \vee Q_n)$.

By definition of the query result, the result of C, denoted by $[C]$, is as follows:

$$[C] = [Q_1 \vee Q_2 \vee \ldots \vee Q_n]$$
$$= [Q_1] \cup [Q_2] \cup \ldots \cup [Q_n]$$

When a new query is submitted, it can be decomposed into two sub-queries; a hit query and a miss query. While a hit query can be answered from the current result cache, a miss query must be transferred to the appropriate collections to get answers.

Definition 4 (Hit Query, HQ): For a given query Q, if a query Q_C subsumes $(Q \wedge C)$, then Q_C is called a *hit query* for Q and C. Q_C is called the *optimal hit query* (*OHQ*) if Q_C is equivalent to $Q \wedge C$.

Definition 5 (Miss Query, MS): For a given query Q, if a query $Q_{\neg C}$ subsumes $(Q \wedge \neg C)$, $Q_{\neg C}$ is called a *miss query* for Q and C. $Q_{\neg C}$ is called the *optimal miss query* (*OMQ*) if $Q_{\neg C}$ is equivalent to $Q \wedge \neg C$.

Intuitively, if the result of $Q \wedge C$ is not empty and Q is not equivalent to C, then the original query Q can be both a hit query and a miss query for Q and C by itself. If Q is equivalent to C, the optimal hit query for Q and C is equivalent to Q. And if $Q \wedge C$ is empty, the optimal miss query for Q and C is equivalent to Q.

Example 2: Suppose a query Q is $B \wedge (A \vee C)$ and the current cache C is $A \vee (B \wedge C)$. Then, the OHQ is $Q \wedge C = (A \wedge B) \vee (A \wedge C) \vee (B \wedge C)$ and the OMQ is $Q \wedge \neg C = (B \wedge \neg(A \vee C))$. ■

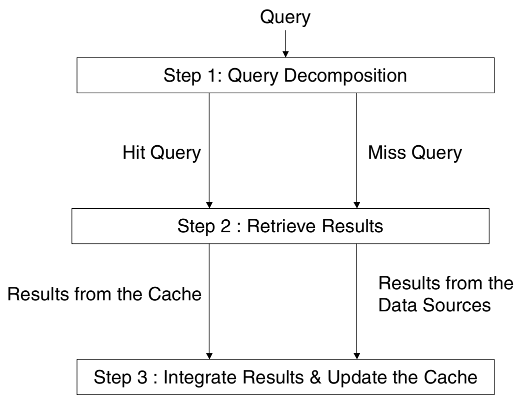
Query
Step 1: Query Decomposition
Hit Query
Miss Query
Step 2 : Retrieve Results
Results from the Cache
Results from the Data Sources
Step 3 : Integrate Results & Update the Cache

Fig. 1. Query Processing Using a Result Cache

The procedure to process queries using a result cache is composed of 3 steps as shown in fig. 1.

search

Step 1 (Query Decomposition): When a query is posed, the mediator decomposes the query into a HQ and a MQ. In general, the initial query can be used as both HQ and MQ.

Step 2 (Retrieve Results): The mediator retrieves cached results from the cache by applying HQ, and un-cached results by sending MQ to the appropriate search engines. If we use an OHQ, we can reduce the time to retrieve data from the cache. The size of the results that need to be transferred from the search engines can be minimized by using an OMQ.

Step 3 (Integration of Results & Cache Update): generates The final result is generated by integrating the results from the cache and those from the data sources. The cache is updated with the results retrieved from the data sources.

III. PROBLEM STATEMENT

In order to realize the above procedure, we should have reasonable and practical solutions to the following problems.

First of all, if we want to minimize the whole computation time, we should decompose the initial query into an OHQ and an OMQ. However it might be costly if the decomposition process is employs general logic inference. In addition, splitting up a query into two parts leads to more complicated queries that might be too expensive to process.

Second, if the size of a cache is large, the time required to get answers from the cache can be quite long.

Finally, since it is common that miss queries contain one or more '*NOT*' operators and there might exist some search engines that cannot directly answer those miss queries because they do not support the '*NOT*' operator. To those search engines, the mediator should send a query that subsumes the miss query and does not contain any '*NOT*' operator. Then, irrelevant data should be filtered out. Unfortunately, however, it is expensive and complicated to compute those subsuming queries.

IV. UNFOLDED DISJUNCTIVE NORMAL FORM (UDNF)

A. UDNF

The results of a result cache are partitioned into several sub-results, and each sub result is represented by a minterm. A minterm for a given set of keywords is defined as a conjunction in which every keyword occurs exactly once, either in its positive or negative form [11]. For example, if the set of keywords is $\{A, B, C\}$ then the set of minterms is $\{A\wedge B\wedge C,\ A\wedge B\wedge\neg C,\ A\wedge\neg B\wedge C,\ \neg A\wedge B\wedge C,\ A\wedge\neg B\wedge\neg C,\ \neg A\wedge\neg B\wedge C,\ \neg A\wedge B\wedge\neg C,\ \neg A\wedge\neg B\wedge\neg C\}$.

Any Boolean formula can be equivalently transformed into a disjunction of selected minterms.

Example 3: A formula $A\vee B$ for the set of keywords $\{A, B\}$ is equivalent to $(A\wedge\neg B)\vee(A\wedge B)\vee(\neg A\wedge B)$, which is a disjunction of minterms. ■

Unfolded Disjunctive Normal Form (UDNF), a variation of the DNF, is represented by a disjunction of appropriate minterms, where every conjunct are pairwise disjoint or unsatisfiable. This means that the intersection of the result sets of any two conjuncts is always empty. In example 3, the resulting formula is the UDNF of $A\vee B$.

Definition 6 (UDNF): A formula F is said to be in an *unfolded disjunctive normal form* if and only if F has the form of $F = F_1 \vee F_2 \vee \ldots \vee F_n$, where each of $F_1, \ldots F_n$ is a minterm.

UDNF is used to represent both queries and the cache. Initially, the cache is empty. When the first query is posed, the query is transformed into its UDNF and sent to the data sources for results. The cache is then initialized with this query and its results.

In general, it is well known that normalization of a Boolean expression is very expensive and often leads to an exponential increase in the size of the formula. In this paper, we propose an incremental algorithm that processes the query efficiently with cached results. We show that UDNF leads to a more efficient search of results in the cache without using any special index structures. In addition, we can simply generate a query subsuming the optimal miss query by using UDNF.

B. Generation of UDNF

In this section, we introduce a method to generate a UDNF for a given Boolean expression. Algorithm 1 describes the normal method to generate the UDNF from an initial query. NormGenUDNF(A, B) returns a set of minterms in the UDNF of B under the set of attributes A.

Algorithm 1: Normal Generation of UDNF

```
NormGenUDNF(A, B)
Input A: set of keywords
      B: target Boolean expression
Output  UDNF of B considering A
BEGIN
1.   Convert B to the corresponding postfix form.
2.   U : set of minterms that can be generated by A
3.   for every literal $l_i$ of B
4.       convert $l_i$ to the disjunction of appropriate
         minterms of U
5.   end for
6.   $R \leftarrow$ minterms of the first left literal
6.   for every Boolean operator op of B
7.       lr: minterms of the second argument of op
8.       if op is 'AND' then
9.           $R \leftarrow R \cap lr$
10.      else op is 'OR' then
11.          $R \leftarrow R \cup lr$
12.      end if
13.  end for
14.  return R
END
```

The main drawback of algorithm 1 is that its time complexity is exponential in the number of total keywords, even though the number of keywords of the current Boolean expression is far smaller than the number of all keywords. For this reason, the UDNF of the cache is generated incrementally by considering only the current keywords. IncGenUDNF(CU, a) of algorithm 2 returns a new set of minterms extended by a new keyword a.

search

Algorithm 2: Incremental Generation of UDNF

```
IncGenUDNF($a$, $CU$)
Input   $CU$: current UDNF
        $a$: new keyword
Output  new UDNF of $CU$
BEGIN
1.  for every conjunct $c_i$ of $CU$
2.      $c_i \leftarrow c_i \wedge a$
3.      $CU \leftarrow CU \cup c_i$
4.      $c_i \leftarrow c_i \wedge \neg a$
5.      $CU \leftarrow CU \cup c_i$
6.  end for
7.  return $CU$
END
```

In using algorithm 2 to maintain the current cache, we should split each sub-result of the cache into two whenever a new keyword is added. For example, suppose the UDNF of the current cache is $(A \wedge B) \vee (A \wedge \neg B)$ and C is a new keyword, then the new UDNF of the cache will be $(A \wedge B \wedge C) \vee (A \wedge B \wedge \neg C) \vee (A \wedge \neg B \wedge C) \vee (A \wedge \neg B \wedge \neg C)$. The first sub-result of the cache represented by $(A \wedge B)$ must be split into two other sub-results represented by $(A \wedge B \wedge C)$ and $(A \wedge B \wedge \neg C)$ respectively. To do this, every query object of the sub-result should be checked at least once against the new keyword C.

C. Decomposition of a Query into OHQ and OMQ

In this subsection, we introduce a method to decompose a query into two sub-queries, OHQ and OMQ. We also propose a method to compute a query that subsumes the OMQ and does not contain a '*NOT*' operator.

We first present two theorems.

Lemma 1: Let $\{m_1, m_2, ..., m_n\}$ be the universal set of minterms defined by the set of keywords. Suppose Q is a Boolean expression and is represented by $m_1 \vee m_2 \vee ... \vee m_k$ where $k \leq n$. If Q is true, then only one m_i $(1 \leq i \leq k)$ is true and the other m_is are false.

Proof) Suppose m_i and m_j $(1 \leq i, j \leq k, i \neq j)$ are true at the same time when Q is true. Since m_i and m_j are not identical, there is at least one common keyword that occurs in m_i (m_j) in negated form and in m_j (m_i) in non-negated form. Let a be that common keyword. Since m_i and m_j are true, $m_i \wedge m_j$ should also be true, which requires $a \wedge \neg a$ to be true, which is a contradiction. Consequently, if both m_i and m_j are true, then m_i should be identical to m_j.

Theorem 1: For a given query Q and a cache C, let $M_1 = \{m_1 \vee m_2 \vee \ldots \vee m_n \mid m_i$ is a minterm that occurs in both Q_{UDNF} and $C_{\text{UDNF}}\}$. Then M_1 is logically equivalent to $Q \wedge C$.

Proof)

$M_1 \Rightarrow Q \wedge C$:

Suppose M_1 is true. M_1 is the disjunction of some minterms. Therefore, there exists at least one minterm that occurs in M_1 and is true. Since all minterms of M_1 occurs in Q_{UDNF} and C_{UDNF} simultaneously, $Q_{\text{UDNF}} \wedge C_{\text{UDNF}}$ should be true. And $Q \wedge C$ is true, too.

$M_1 \Leftarrow Q \wedge C$:

Suppose $Q \wedge C$ is true. Since $Q \wedge C$ is equivalent to $Q_{\text{UDNF}} \wedge C_{\text{UDNF}}$, $Q_{\text{UDNF}} \wedge C_{\text{UDNF}}$ should also be true. Suppose Q_{UDNF} is $q_1 \vee q_2 \vee ... \vee q_m$ and C_{UDNF} is $c_1 \vee c_2 \vee ... \vee c_t$ where q_i and c_j are minterms. Then, $Q_{\text{UDNF}} \wedge C_{\text{UDNF}}$ is $(q_1 \wedge c_1) \vee (q_1 \wedge c_2) \vee ... \vee (q_1 \wedge c_t) \vee (q_2 \wedge c_1) \vee ... \vee (q_m \wedge c_t)$. Since $Q_{\text{UDNF}} \wedge C_{\text{UDNF}}$ is true, there exists at least one $(q_i \wedge c_j)(1 \leq i \leq m, 1 \leq j \leq t)$ that is true. If q_i is not identical to c_j, $q_i \wedge c_j$ should be false according to lemma 1. Therefore, q_i should be identical to c_j and would be a member of M_1. By the definition of M_1, M_1 is true.■

Lemma 2: If Q is a Boolean expression, then $\neg Q_{\text{UDNF}}$ is logically equivalent to the disjunction of minterms that does not occur in Q_{UDNF}.

Proof) Let $\{m_1, m_2, ..., m_n\}$ be the universal set of minterms defined by the set of keywords. Suppose Q_{UDNF} is $m_1 \vee m_2 \vee ... \vee m_k$ where $k \leq n$. Then, $\neg Q_{\text{UDNF}} = \neg m_1 \wedge \neg m_2 \wedge ... \wedge \neg m_k$.

$\neg m_1 \wedge \neg m_2 \wedge ... \wedge \neg m_k \Rightarrow m_{k+1} \vee m_{k+2} \vee ... \vee m_n$:

Suppose $\neg m_1 \wedge \neg m_2 \wedge ... \wedge \neg m_k$ is true. Then, $m_1, m_2, ...,$ and m_k should be false at the same time. Since $m_1 \vee m_2 \vee ... \vee m_n$ should be true by definition of a minterm, there exists at least one minterm that is true in $\{m_{k+1}, m_{k+2}, ..., m_n\}$. Therefore, $m_{k+1} \vee m_{k+2} \vee ... \vee m_n$ is true.

$\neg m_1 \wedge \neg m_2 \wedge ... \wedge \neg m_k \Leftarrow m_{k+1} \vee m_{k+2} \vee ... \vee m_n$:

Suppose $m_{k+1} \vee m_{k+2} \vee ... \vee m_n$ is true. Then, there exists at least one m_i $(k+1 \leq i \leq n)$ that is true. Therefore, $m_1 \vee m_2 \vee ... \vee m_k$ should be false and $\neg m_j$ $(1 \leq j \leq k)$ should be true. Consequently, $\neg m_1 \wedge \neg m_2 \wedge ... \wedge \neg m_k$ is true. ■

Theorem 2: For a given query Q and a cache C, let $M_2 = \{m_1 \vee m_2 \vee \ldots \vee m_n \mid m_i$ is a conjunction that occurs in Q_{UDNF} but not in $C_{\text{UDNF}}\}$. Then M_2 is logically equivalent to $Q \wedge \neg C$.

Proof)

$M_2 \Rightarrow Q \wedge \neg C$:

Suppose M_2 is true. Since M_2 is a disjunction of minterms and there exists at least one minterm in M_2 which is true. Since all minterms of M_2 occurs in Q_{UDNF}, Q_{UDNF} is true. Since all minterms of M_2 do not occur in C_{UDNF}, they all occur in $\neg C_{\text{UDNF}}$ according to lemma 1. Therefore, If M_2 is true, then $Q_{\text{UDNF}} \wedge \neg C_{\text{UDNF}}$ is true.

$M_2 \Leftarrow Q \wedge \neg C$:

Suppose $Q \wedge \neg C$ is true. Since $Q \wedge \neg C$ is equivalent to $Q_{\text{UDNF}} \wedge \neg C_{\text{UDNF}}$, $Q_{\text{UDNF}} \wedge \neg C_{\text{UDNF}}$ is true also. Suppose Q_{UDNF} is $q_1 \vee q_2 \vee ... \vee q_m$ and $\neg C_{\text{UDNF}}$ is $c_1 \vee c_2 \vee ... \vee c_t$ where q_i and c_j are minterms. Then, $Q_{\text{UDNF}} \wedge \neg C_{\text{UDNF}}$ is $(q_1 \wedge c_1) \vee (q_1 \wedge c_2) \vee ... \vee (q_1 \wedge c_t) \vee (q_2 \wedge c_1) \vee ... \vee (q_m \wedge c_t)$. Since $Q_{\text{UDNF}} \wedge \neg C_{\text{UDNF}}$ is true, there exists at least one $(q_i \wedge c_j)$ $(1 \leq i \leq m, 1 \leq j \leq t)$ that is true. Suppose there exists at least one keyword that occurs in q_i in negated (or non-negated) form and in c_j in non-negated (or negated, resp.) form. Since q_i and c_j are minterms, $q_i \wedge c_j$ should be false. Therefore, q_i and c_j may not contain a common keyword negated in q_i and not negated in c_I, or visa versa. Therefore, q_i and c_j must be identical. Since c_j occurs in $\neg C_{\text{UDNF}}$, c_j may not occur in C_{UDNF} according to lemma 2. By the definition of M_2, M_2 should be true.■

search

Algorithm 3: Generation of OHQ and OMQ

```
GenOHMQ($C$,$Q$)
Input C: UDNF of the current cache
      Q: current query
Output  OHQ and OMQ
BEGIN
1.  for every keyword $a$ of $Q$ that does not occur in $C$
2.      $C \leftarrow$ IncGenUDNF($a$, $C$)
3.  end for
4.  $Q \leftarrow$ NormGenUDNF(all keywords of $Q$, $Q$)
5.  for every keyword $a$ of $C$ that does not occur in $Q$
6.      Q $\leftarrow$ IncGenUDNF($a$, $Q$)
7.  end for
8.  $OMQ \leftarrow Q$
9.  for every conjunct $q$ of $Q$
10.     for every conjunct $c$ of $C$
11.         if $c$ implies $q$ then
12.             $OHQ \leftarrow OHQ \cup c$
13              $OMQ \leftarrow OHQ - c$
14.         end if
15.     end for
16. end for
17. return $OHQ$ and $OMQ$
END
```

According to theorems 1 and 2, it is clear that the OHQ and the OMQ can be found by a simple conjunct matching procedure. Since results in the cache are partitioned and each partition is represented by a unique minterm, we can simply retrieve the results of OHQ from the cache. In algorithm 3, we show the method for decomposing a query into OHQ and OMQ.

D. 'NOT' Operator

The following theorem makes it possible to retrieve the results of the OMQ without using negations. This is useful when the target data source does not support the '*NOT*' operator.

Lemma 3: Suppose $Q = m_i \vee m_j$ $(i \neq j)$ where m_i and m_j are minterms defined on the set of keywords. Let n_i (p_i) be the sub-conjunct of negative (positive, resp.) literals of m_i and $\boldsymbol{A}(n_i)$ be the set of atoms in n_i. If $\boldsymbol{A}(n_i) \not\subset \boldsymbol{A}(n_j)$ and $\boldsymbol{A}(n_j) \not\subset \boldsymbol{A}(n_i)$, then Q is logically equivalent to $(p_i \vee p_j) \wedge (n_i \vee n_j)$.

Proof) Since $m_i = p_i \wedge n_i$ and $m_j = p_j \wedge n_j$, $m_i \vee m_j = (p_i \wedge n_i) \vee (p_j \wedge n_j)$. And $(p_i \vee p_j) \wedge (n_i \vee n_j) = (p_i \wedge n_i) \vee (p_j \wedge n_j) \vee (p_i \wedge n_j) \vee (p_j \wedge n_i)$.

For $p_i \wedge n_j$ (or $p_j \vee n_i$) , there exist two cases.

(1) $\boldsymbol{A}(n_i) \cap \boldsymbol{A}(n_j) = \varnothing$

In this case, there should exist at least one keyword that occurs in p_i in non-negated form and in n_j in negated form. Consequently, $p_i \wedge n_j$ is always false.

(2) $\boldsymbol{A}(n_i) \cap \boldsymbol{A}(n_j) \neq \varnothing$

Since both $\boldsymbol{A}(n_i) - \boldsymbol{A}(n_j)$ and $\boldsymbol{A}(n_j) - \boldsymbol{A}(n_i)$ are not empty, there exists at least one keyword that occurs in p_i in non-negated form and in n_j in negated form. Therefore, $p_i \wedge n_j$ is false like in the first case.

From (1) and (2), every $p_i \wedge n_j$ (or $p_j \wedge n_i$) should be false. Therefore, $(p_i \vee p_j) \wedge (n_i \vee n_j)$ is equivalent to $(p_i \wedge n_i) \vee (p_j \wedge n_j)$. Since m_i is $(p_i \wedge n_i)$ and m_j is $(p_j \wedge n_j)$, Q is equivalent to $(p_i \vee p_j) \wedge (n_i \vee n_j)$. ■

Let $\boldsymbol{N}(m_i)$ be the number of literals that occur in m_i. Then, we can state the following lemma.

Lemma 4: Suppose $Q = m_i \vee m_j$ $(i \neq j)$ where m_i and m_j are minterms defined on the same set of keywords. If $\boldsymbol{A}(n_i) \supset \boldsymbol{A}(n_j)$ and $\boldsymbol{N}(n_i) - \boldsymbol{N}(n_j) = 1$, then Q is logically equivalent to $(p_i \vee p_j) \wedge (n_i \vee n_j)$.

Proof) $(p_i \vee p_j) \wedge (n_i \vee n_j) = m_i \vee m_j \vee (p_i \wedge n_j) \vee (p_j \wedge n_i)$. Since $\boldsymbol{A}(n_i) \supset \boldsymbol{A}(n_j)$, $p_j \wedge n_i$ is false but $p_i \wedge n_j$ is not. Since $\boldsymbol{N}(n_i) - \boldsymbol{N}(n_j) = 1$, the only keyword that does not occur in $p_i \wedge n_j$ occurs in m_i in negated form and in m_j in non-negated form. Let that keyword be a. Then, $p_i \wedge n_j$ is logically equivalent to $(p_i \wedge n_j \wedge a) \vee (p_i \wedge n_j \wedge \neg a)$. Since $(p_i \wedge n_j \wedge a)$ and $(p_i \wedge n_j \wedge \neg a)$ are identical with m_i and m_j, respectively, Q is logically equivalent to $(p_i \vee p_j) \wedge (n_i \vee n_j)$. ■

The following theorem presents a necessary condition for computing a subsuming query of an OMQ.

Theorem 3: Let Q_{UDNF} be the UDNF of a given query Q. Formally, Q_{UDNF} can be defined as follows: $Q_{\text{UDNF}} = m_1 \vee m_2 \vee \ldots \vee m_n$ where m_i is a minterm defined under a set of keywords. For any two minterms m_i and m_j $(i \neq j)$, if either $\boldsymbol{A}(n_i) \not\subset \boldsymbol{A}(n_j)$ and $\boldsymbol{A}(n_j) \not\subset \boldsymbol{A}(n_i)$ or $\boldsymbol{A}(n_i) \supset \boldsymbol{A}(n_j)$ and $\boldsymbol{N}(n_i) - \boldsymbol{N}(n_j) = 1$, then Q is logically equivalent to $(p_1 \vee p_2 \vee \ldots \vee p_n) \wedge (n_1 \vee n_2 \vee \ldots \vee n_n)$.

Proof) $Q = (p_1 \wedge n_1) \vee (p_2 \wedge n_2) \vee \ldots \vee (p_n \wedge n_n)$. And

$(p_1 \vee p_2 \vee \ldots \vee p_n) \wedge (n_1 \vee n_2 \vee \ldots \vee n_n) = (p_1 \wedge n_1) \vee (p_2 \wedge n_2) \vee \ldots \vee (p_n \wedge n_n) \vee (p_2 \wedge n_1) \vee \ldots \vee (p_2 \wedge n_n) \vee \ldots \vee (p_n \wedge n_n)$.

For those $i \neq j$ such that $\boldsymbol{A}(n_i) \not\subset \boldsymbol{A}(n_j)$ and $\boldsymbol{A}(n_j) \not\subset \boldsymbol{A}(n_i)$, $p_i \wedge n_j$ and $p_j \wedge n_i$ are false according to lemma 1. And for $i \neq j$ such that $\boldsymbol{A}(n_i) \supset \boldsymbol{A}(n_j)$ and $\boldsymbol{N}(n_i) - \boldsymbol{N}(n_j) = 1$, $p_j \wedge n_i$ is false and $p_i \wedge n_j$ is logically equivalent to $p_i \wedge n_i$ according to lemma 2. Consequently, Q is logically equivalent to $(p_1 \vee p_2 \vee \ldots \vee p_n) \wedge (n_1 \vee n_2 \vee \ldots \vee n_n)$. ■

If an OMQ satisfies the conditions mentioned in theorem 3, a new query that subsumes the OMQ and does not contain any 'NOT' operator can be generated easily. An OMQ $(a \wedge \neg b \wedge c) \vee (\neg a \wedge b \wedge c) \vee (a \wedge b \wedge \neg c)$ is logically equivalent to $((a \wedge c) \vee (b \wedge c) \vee (a \wedge b)) \wedge (\neg a \vee \neg b \vee \neg c)$. The first part, $((a \wedge c) \vee (b \wedge c) \vee (a \wedge b))$, is sent to the data collections and the second part, $(\neg a \vee \neg b \vee \neg c)$, is used to filter out the extraneous results in the integration stage.

A query that contains a minterm that does not satisfy the conditions of theorem 3 is processed differently. For example, suppose an OMQ is $(a \wedge \neg b \wedge c \wedge \neg d) \vee (\neg a \wedge b \wedge c \wedge d) \vee (a \wedge b \wedge \neg c \wedge d) \vee (\neg a \wedge b \wedge \neg c \wedge \neg d) \vee (a \wedge \neg b \wedge c \wedge d)$. This OMQ as a whole does not satisfy the conditions of theorem 3. We partition this OMQ into two sub-queries. The first sub-query is composed of the first four minterms and the remainder is the second sub-query. Then, we can apply theorem 3 to each sub-query separately (see fig. 2).

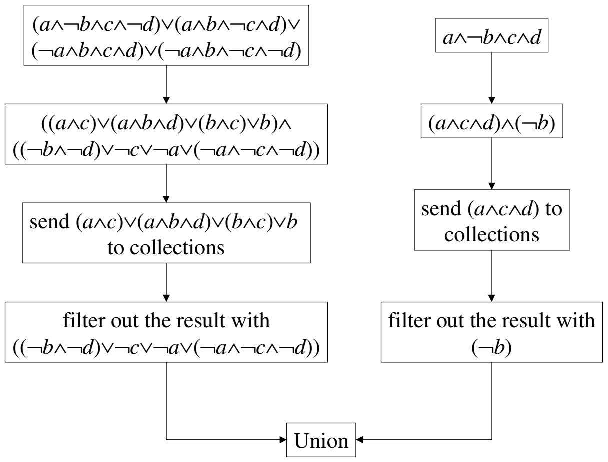
$(a\wedge\neg b\wedge c\wedge\neg d)\vee(a\wedge b\wedge\neg c\wedge d)\vee(\neg a\wedge b\wedge c\wedge d)\vee(\neg a\wedge b\wedge\neg c\wedge\neg d)$
$((a\wedge c)\vee(a\wedge b\wedge d)\vee(b\wedge c)\vee b)\wedge((\neg b\wedge\neg d)\vee\neg c\vee\neg a\vee(\neg a\wedge\neg c\wedge\neg d))$
send $(a\wedge c)\vee(a\wedge b\wedge d)\vee(b\wedge c)\vee b$ to collections
filter out the result with $((\neg b\wedge\neg d)\vee\neg c\vee\neg a\vee(\neg a\wedge\neg c\wedge\neg d))$
$a\wedge\neg b\wedge c\wedge d$
$(a\wedge c\wedge d)\wedge(\neg b)$
send $(a\wedge c\wedge d)$ to collections
filter out the result with $(\neg b)$
Union

Fig. 2. Partitioning an OMQ

There can be more than one way to partition an OMQ. In the above example, another partition, namely $\{(a \wedge \neg b \wedge c \wedge \neg d)\} \vee \{(\neg a \wedge b \wedge c \wedge d) \vee (a \wedge b \wedge \neg c \wedge d) \vee (\neg a \wedge b \wedge \neg c \wedge \neg d) \vee (a \wedge \neg b \wedge c \wedge d)\}$, is possible.

V. Performance

In this section, we analyze the performance of our method. First, we compute the lower bound of the hit ratio of the cache under the reasonable assumption of [10].

Lemma 3: Suppose the probability that a keyword denoted by t being answered from the cache is p. For two independent keyword t_1 and t_2, the probability that the result of $t_1 \wedge t_2$ and $t_1 \vee t_2$ may be answered from the cache is $2p-p^2$ and p^2 respectively.

Proof) If we were to get the result of $t_1 \wedge t_2$ from the cache, either t_1 or t_2 should be answerable from the cache. Therefore, the probability of $t_1 \wedge t_2$ is $2p-p^2$.

In the case of $t_1 \vee t_2$, both of t_1 and t_2 should be answerable from the cache. So, the probability of $t_1 \vee t_2$ should be p^2. ■

Theorem 4: Suppose the probability that a keyword being answered from the cache is p. The probability that a Boolean query Q being answered from the cache, denoted by P(Q), is also equal to p regardless of the number of keywords in Q.

Proof (by induction on n, the number of keywords in Q)

i) $n = 1$

$P(Q) = P(t) = p$

ii) Suppose $P_{n=k}$ is equal to p.

The query of $n = k+1$, $Q_{n=k+1}$, can be generated by adding a new keyword t with either '$\wedge$' or '$\vee$' to the query of $n = k$, $Q_{n=k}$.

$P(Q_{n=k} \wedge t) = P(Q_{n=k}) + P(t) - P(Q_{n=k})P(t) = 2p-p^2$

$P(Q_{n=k} \vee t) = P(Q_{n=k})P(t) = p^2$

$P_{n=k+1} = (P(Q_{n=k} \wedge t) + P(Q_{n=k} \vee t)) / 2 = p$, assuming the two cases are equally likely. ■

This can be verified for $n = 2$ and $n = 3$. For $n = 2$, from Lemma 3,

$$P_{n=2} = (2p-p^2 + p^2) / 2 = p.$$

For $n = 3$, there can exist four types of Boolean expressions such as $t_1\wedge t_2\wedge t_3$, $t_1\vee t_2\vee t_3$, $(t_1\wedge t_2)\vee t_3$, $(t_1\vee t_2)\wedge t_3$, since ordering is meaningless. According to lemma 3,

$P(t_1\wedge t_2\wedge t_3) = 3p-3p^2+p^3$
$P(t_1\vee t_2\vee t_3) = p^3$
$P((t_1\wedge t_2)\vee t_3) = P(t_1\wedge t_2) \times P(t_3) = 2p^2-p^3$
$P((t_1\vee t_2)\wedge t_3) = P((t_1\vee t_2) + P(t_3) - P((t_1\vee t_2)P(t_3) = p+p^2-p^3$

If we assume all the above types of query might occur equally frequently, then the resulting probability $P_{n=3}$ is:

$$P_{n=3} = ((3p-3p^2+p^3) + (p^3) + (2p^2-p^3) + (p+p^2-p^3)) / 4 = p$$

Next, we show that our method is efficient. For the proof of efficiency of our method, we use the following notation:

tD is the average time to decompose a query.
tC is the average time to retrieve data from cache.
tS is the average time to retrieve data from a search engine.
h is the hit ratio of the current cache ($0 \le h \le 1$).
dC is the average number of data objects in a conjunct of the cache.
nQ is the number of distinct keywords in a query.
mQ is the number of distinct keywords that is in a query but not in the cache.
nC is the number of distinct keywords in the cache.
mC is the number of distinct keywords that is in the cache but not in a query.

First of all, we assume that *dC* is greater than 2. This assumption is practical enough since it is common that the number of results for a query is large.

Lemma 4: $tC > tD$

Proof) Since *tD* is proportional to the complexity of algorithm 3, we can say the following.

$$tD = 2^{nC+mQ} + 2^{nQ} + 2^{nC+mQ+nQ+mC}$$

$$tC = dC \cdot 2^{nC+mQ} + 2^{nC+mQ+nQ+mC}$$

Therefore, $tC - tD = 2^{nC+mQ}(dC-1) - 2^{nQ+mC}\left(1+\frac{1}{2^{mC}}\right)$

Since $nC + mQ = nQ + mC$ and $dC > 2$,

$$tC - tD > 0$$

$$\therefore tC > tD \quad ■$$

Theorem 5: If both $\frac{tS}{tC}$ and h are large enough, then $tS - (tD + h \cdot tC + (1-h) \cdot tS) > 0$

Proof) Let $\frac{tS}{tC}$ be k. Then,

search

$$\begin{aligned}&tS-\left(tD+h\cdot tC+\left(1-h\right)\cdot tS\right)\\&=h\cdot tS-h\cdot tC-tD\\&=h(tS-tC)-tD\\&=h(k\cdot tC-tC)-tD(\Theta\ tS=k\cdot tC)\\&=tC\cdot h\cdot(k-1)-tD\end{aligned}$$

According to lemma 4, $tC > tD$. So, in order to make $tC\cdot h\cdot(k-1)-tD$ positive, $h\cdot\left(k-1\right)$ should be greater than or equals to 1. Therefore, it should be that $k > 1 + 1/h$. In addition, we can state that h is greater than or equal to p, the probability that a query with any number of keywords might be answered from the cache, due to theorem 4. Since it has been reported that on the average 88% of the total queries uses terms that have been used already [10], we can assume that p is about 0.88. Therefore, the hit ratio of the cache should be at least 0.8. Since $1 + 1/h = 1 + 1/0.8 = 2.25$, there should exist a k greater than 2.25 which is practically feasible. ■

In theorem 5, $\left(tD+h\cdot tC+\left(1-h\right)\cdot tS\right)$ is the time to compute the result of a query when the result cache is used and tS is the time when we do not use the cache. Therefore, our method guarantees better performance than a general solution that does not use cached results.

VI. Conclusion and Future Works

In this paper we studied the management of a result cache under the Boolean retrieval model in a mediator context. We proposed a method based on propositional logic. We can apply our method to a keyword-based Boolean retrieval model.

We showed that our method is more efficient in terms of performance than solutions that do not make use of cached results. Unfortunately, however, the size of a representation for a result cache increases exponentially as query history grows. We are revising a solution to this problem by means of the partitioned representation. We are currently extending the scheme so that it can be applied to an attribute-based query model. In addition, we are particularly interested in the storage model and access methods for the result cache. Furthermore, empirical studies based on the specific cost model are being pursued.

References

[1] G. Wiederhold, Mediators in the Architecture of Future Information Systems, *IEEE Computer*, Mar. 1992, pp. 38-49.

[2] C. M. Chen and N. Roussopoulos, The Implementation and Performance Evaluation of the ADMS Query Optimizer: Integrating Query Result Caching and Matching. *EDBT 1994*, pp. 323-336.

[3] S. Adali, K. S. Candan, Y. Papakonstantinou, and V. S. Shubrahmannian, Query Caching and Optimization in Distributed Mediator Systems, *proceedings of ACM SIGMOD International Conference on Management of Data*, Montreal, Canada, June 1996, pp. 137-148.

[4] S. Dar M. J. Franklin, B. T. Jonsson, D. Srivastava, and M. Tan, Semantic Data Caching and Replacement, *proceedings of the 22nd VLDB conference* , 1996, pp. 330-341.

[5] A. M. Keller and J. Basu, A Predicate-based Caching Scheme for Client-Server Database Architectures, *the VLDB journal*, Vol. 5, No. 1, 1996, pp. 35-47.

[6] D. Miranker, M. Taylor, and A. Padmanaban, A Tractable Query Cache By Approximation, *Technical Report*, MCC, 1998.

[7] S. Abiteboul and O. M. Duschka, Complexity of Answering Queries using Materialized View, *proceedings of ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, Seattle, WA, June 1998.

[8] K. C. Chang, H. Garcia-Molina, and A. Paepcke, Boolean Query Mapping Across Heterogeneous Information Sources, *IEEE Transactions on Knowledge & Data Engineering*, Vol. 8, No. 4, 1996, pp. 515-521.

[9] Dong-gyu. Kim and Sang-goo. Lee, QVI: Query-based Virtual Index for Distributed Information Retrieval System, *proceedings of ISCA 13th International Conference CATA*, 1998, pp.152-155.

[11] M. M. Mano, *Digital Logic and Computer Design*, Prentice-Hall, Englewook Cliffs, NJ, 1979.