A Cache Management System for a Distributed Deductive Database

Larry Williams¹, Martin Maskarinec¹, and Kathleen Neumann¹
¹School of Computer Sciences, Western Illinois University, Macomb, IL

Abstract - This paper will illustrate what a cache management system for a deductive database will do when a query is executed, or sent through the presented calculation plan. It will also illustrate what the Cache Management System should do when the cache is full. Once the query goes through the calculation plan, it then passes the results to the Cache Management System. The Cache Management System will then decide if the query should be cached or not. The Cache Management System will also have to decide what should be removed from the cache if the cache is full. This paper will also illustrate the how to calculate the recalculation cost.

Keywords: Intelligent Database, Deductive Database, Cache Algorithms

1. Introduction

According to Ullman, et. al [1], a deductive database is a conventional database containing facts, a knowledge base containing rules, and an inference engine which allows the derivation of information implied by facts and rules. The knowledge base is expressed in a subset of first-order logic.

The results sets for a deductive database can be extremely expensive. This is because each node may need to get information from many levels to produce its result. Thus, a system is needed to cache the results rather than recalculating them each time.

Conventional caching algorithms as mentioned in [2] do not have to take into consideration the dependency cost if the system removes that result from the cache. For this reason a deductive database system cannot use this conventional method of caching the results. The conventional rules do not apply especially when the system removes items from the cache. This paper will describe how to check the cache to see if the results exist. If the results exist, the system will return them. If not, the system will execute the query on the stored relations and decide if the system should cache the results or not. If the system needs to cache the results and the cache is full, the system will also need to decide which of the results should stay in the cache and which should be removed. In a conventional cache management system, the system does not have to consider the cost to recreate that result or the recalculation cost for the result. The recalculation cost is the cost that is incurred when a result is removed. The system should remove results with a low recalculation cost. This paper will discuss what to do when the results that are set to be removed from the cache have a high recalculation cost. In this case, the system will remove results from the cache that are less time intensive to recreate.

2. Execution Cycle

When the system runs a query it follows the Execution Cycle that is illustrated in Figure 1. The Execution Cycle starts by checking if it is the first query executed. If so it caches the whole result including the nodes on the bottom of the tree. If it is not the first time running a query the system sends a probe query to check the cache keys to see if it is cached. If it is, the system increments the hit counter for that node, and stores the date and time that the node was last hit. The system then increments the hit counter for the cache. Next, the results of the query are returned.

If the node is not cached, ghost data of the cache miss is stored. The ghost data is the cache key, the date and time of the last miss, and a counter that is incremented every time it is missed. The cache miss counter is also incremented at this time. Next, the system checks the node to make sure it is not a leaf node. If it is a leaf node, the system calculates the value and returns the results. If the node is not a leaf node then the system checks if the result of the node can be recalculated. If the system cannot calculate the result, it traverses the tree to the next node and goes through this process again.

3. Traversing a Predicate Graph

The order that the system uses to traverse the predicate graph is a Depth First Search. Figure 2 presents an example of a predicate graph. The order of traversal for the predicate graph shown in Figure 2, assuming there are no results cached, is: a1,b1,p1,c1,d1,d2,c2,d3,d4,p2,c3,d5,d6,c4,d7,d8,l1,m1,n1,o1,o2,a2,o3,o4,m2,n3,o5,o6,n4,o7,o8. If the node can be calculated, the system performs the calculation, and returns the results. After the system returns the results, the Cache Management System determines whether they should be cached. If the system cannot calculate the node directly, it must traverse the tree to nodes that are either cached or are leaf nodes. A result cannot be
calculated for a node until it has all the results for all the children of that node.

Figure 1. The Calculation Cycle

Figure 2. A Sample Predicate Graph
4. The Cache Management System

When the system is initiated, the block size and size of the cache are determined. The number of blocks available in the cache is calculated by dividing the available space by the block size. This is the total number of cache blocks that can be assigned to all cached nodes.

Figure 3 shows the Cache Management System (CMS) process. The results from the Execution Cycle are sent to the CMS. The truth table in Figure 4 illustrates the CMS in Figure 3. If the results fit in the cache then the CMS stores the results there. If they do not fit, the system checks to see if the results’ miss count reaches the cache miss threshold. If it does then the CMS makes room for the results. If the initial miss threshold is not reached, a more relaxed threshold is used. If this relaxed threshold is not met, then the result is not cached. If the relaxed threshold is met, the new result must have a high cost to recalculate in order to be cached. If the cost is low, the result is not cached.

Figure 3. The Cache Management System Flow Chart

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>Yes</td>
<td>X</td>
<td>X</td>
<td>Make Room</td>
</tr>
<tr>
<td>No</td>
<td>No</td>
<td>No</td>
<td>X</td>
<td>Do Nothing</td>
</tr>
<tr>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Do Nothing</td>
</tr>
<tr>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Make Room</td>
</tr>
<tr>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Cache Results</td>
</tr>
<tr>
<td>Yes</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>Cache Results</td>
</tr>
</tbody>
</table>

Figure 4. The Cache Management System Truth Table

In the next sections, the methodologies for calculating the needed data will be discussed.

5. Miss Percentages

As is seen in Figure 1, the hit and miss counts for all nodes are maintained during the Execution Cycle. The Miss Percentage of any node $i$ is:

$$\text{Miss Percentage}_i = \frac{\#\text{misses}_i}{\#\text{misses}_i + \#\text{hits}_i}$$

6. Making Room

If the CMS decides a result set needs to be cached, but there are not enough available blocks in the cache, some existing elements must be removed from the cache. This is done by determining a normalized “recalculation cost” for all nodes in the cache. The nodes of lowest recalculation cost are removed until enough blocks are freed up to accommodate the new result set.
7. Recalculation Cost

The recalculation cost is based on normalizing the cost of calculation (discussed in the next section), the number of parents of the node (or the “node dependency”), and the miss percentage. The calculation cost is normalized to a number between 1 and 50. The node dependency and the miss percentage are normalized to a number between 1 and 5.

The recalculation cost uses these normalized values to produce a number on the same scale for all nodes. The actual formulae used are:

\[
\text{Norm Cost} = 1 + (\text{Cost} - 1) \times \frac{(50 - 1)}{(\max \_\text{cost} - 1)}
\]

\[
\text{Norm Parent} = 1 + (\text{Parent Count} - 1) \times \frac{(5 - 1)}{(\max \_\text{parent} - 1)}
\]

\[
\text{Norm Miss} = 1 + (\text{Miss Count} - 1) \times \frac{(5 - 1)}{(\max \_\text{node miss} - 1)}
\]

Recalculation cost = (Norm Parent + Norm Miss) * Norm Cost;

The miss count is maintained in the execution cycle and the parent count is easily determined by examining the predicate graph. The next section describes how the cost of a node is calculated.

8. Cost Calculation for a Node

The cost of calculating a node is based on one of three possibilities: 1) the node is cached, 2) the node is a leaf node, or 3) the node is neither a leaf node nor cached. If the node is cached, it may be retrieved from the cache at minimal cost (we assign zero to this cost). If it is not cached, but is a leaf node, then the cost is simply the cost of retrieving the data. If it is not a leaf node, then all children of this node must be found and their costs added to the total cost (note that this may require many levels of recursive calls). The algorithm used to make this calculation is:

```c
Cost_Calculation(Input Node)
{
    if Input Node is Cached
        return 0;
    if Input Node is a leaf Node
        return b_{InputNode} ;
    in all other cases:
        Total Cost = Cost_Calculation(Child[0]);
        PreviousResult = Child[0];
        for(cnt =1; cnt< Child Count; cnt++)
        {
            nodeCost = Cost_Calculation(Child[cnt]);
            joinCost = PreviousResult + PreviousResult * b_{Child[cnt]};
            PreviousResult = PreviousResult joined to Child[cnt];
            Total Cost = Total Cost + nodeCost + joinCost;
        }
    Return Total Cost;
}
```

9. Future Work

A simulation of the cache management system is currently in development and will be used to fine-tune the miss and recalculation cost thresholds. This will then be fully integrated into a deductive database management system.

10. Conclusion

A cache management system has been proposed to allow a deductive database to manage the cache of the result sets of its nodes as efficiently as possible. A heuristic has been proposed to determine when a result set should be cached, and, if room is not available, how to determine what should be removed in order to make room for this result set.

11. Bibliography
