VIEWS AND DATA MINING IN A PARALLEL DATA SERVER

By

Sanhita Sarkar

and

Shyam Sundar Sarkar

IMA Preprint Series # 1475
March 1997

INSTITUTE FOR MATHEMATICS AND ITS APPLICATIONS
UNIVERSITY OF MINNESOTA
514 Vincent Hall
206 Church Street S.E.
Minneapolis, Minnesota 55455
Views and Data Mining in a Parallel Data Server

Sanhita Sarkar
Institute for Mathematics and its Applications (IMA),
University of Minnesota,
514 Vincent Hall, 206 Church Street, S.E.,
Minneapolis, MN 55455.
sarkar@ima.umn.edu

Shyam Sundar Sarkar
Informix Software, Inc.,
4100 Bohannon Drive,
Menlo Park, CA 94025.
sarkar@informix.com

Keywords and Phrases.- Data Mining, Parallel Databases, Materialization of Views in Relational Databases, Performance Analysis, Distributed Databases.

ABSTRACT

This paper describes how data mining can be performed on Views in a parallel data server. Here, we consider a decision tree construction. Tests are performed on a ViewCache generated from the relational operations on the base level tables. Branches are made for the values of the test to form subsets of the ViewCache. The leaves are labeled with the class names. Quinlan’s ID3 algorithm for a proper selection of attributes at a particular level of the decision tree calculates the information gain for all such attributes and considers the one with the maximal gain. This requires multiple View materializations at a single level. So optimal View materialization is a concerning factor. When we are considering a parallel data server, the number of records in a ViewCache, may be skewed to a processor/s and the materialization at any tree-level/s will not exploit efficient parallelism. In this paper, we have developed a technique for normal distribution of “data packets” from an overloaded processor to its neighbouring processors at run-time. This normal distribution is implemented as a replication technique for small fractions of records and ViewCache partitions. At run-time, based on mutual agreements, task load can be shared between processors, to minimize the overall cost of materialization. We have given a bipartite graph representation of a ViewCache and posed the problem of optimal View Materialization as a problem of optimal traversal through the edges of the graph. Data packets for replication are well chosen subgraphs. This leads to a run-time solution for load balancing.

INTRODUCTION

Data mining, the extraction of hidden predictive information from large databases, is a powerful technology with great potential to help companies focus on prospective and proactive information in their data warehouses. One of the most commonly used techniques in data mining is to build a decision tree - a tree-shaped structure representing sets of decisions which can generate rules for the classification of a dataset. It is the smallest tree with greatest predictive power, providing a set of rules that can be applied to a new dataset to predict an outcome [2]. Since data mining deals with very large databases to get predictions, an effective way is to use ViewCaches on top of Relational Databases to exploit all its advantages [7]. A ViewCache is a stored collection of pointers pointing to records of underlying relations needed to materialize a View.

Example 1.1.

A part of a market-basket analysis star schema, (Figure 1) may be taken to explain the formation of

\footnote{To whom correspondence should be addressed.}
ViewCache and how a decision tree can be built on top of it.

Figure 1: A part of a market-basket analysis star schema for a grocery retailer.

Figure 2: The Binary ViewCache created from the relational operations on the star schema. View = Join over Products, Purchases and Store.

We take a “join” between the tables Products, Purchases and Store to create the ViewCache V (Figure 2). Suppose, we want to perform some tests on the View, corresponding to the ViewCache, V. Let us take the test as: Which of the products Food or Non-Food, are mostly purchased in Northern Minnesota and Southern Minnesota? In case of Food products are they Dairy or Non-Dairy products? In case of Non Food products are they Fuel or Non-Fuel products? We then need to formulate the decision tree as follows:

Figure 3: Scheme of a Decision tree for a market-basket analysis.

In Figure 3, a decision tree has been sketched out by selecting the test on the attribute “Type” on the generated ViewCache V and then on the attribute “Category” at the next level.

Figure 4: Result of tests at each level generates a hierarchy of ViewCaches.

Figure 4 shows the formation of a hierarchy of ViewCaches, as a result of tests at each level of the corresponding decision tree in Figure 3. Now, how far
informative is this tree and how is the information gained from a decision tree, relevant to the ViewCache hierarchy? The goal of a data miner is not to build any tree but to build a compact one with as few tests and branches as possible. But there may be for example, 1000 possible choices or decision trees and to select the smallest one, is not an easy job at all. One way in symbolic machine learning is to use an information theoretic measure to determine the “goodness” of a potential test. We now discuss the above questioned relevance between the information measure in a decision tree and that in a ViewCache hierarchy using the approach in symbolic machine learning. As an example, let us calculate the information gained after performing the test on the attribute “Type” for the ViewCache V. Following [6], let \( \text{freq}(C_j,V) \) denote the relative frequency of a class \( C_j \) say, in the ViewCache \( V \), representing the training data set. Here \( j = 1,2 \), where \( C_1 \) denotes the class or the outcome North_MN and \( C_2 \) denotes the class or the outcome South_MN. The average information [2,15] needed to identify the class of a case in \( V \) is \( \text{info}(V) \), given by,

\[
\text{info}(V) = - \sum_{j=1}^{2} \frac{\text{freq}(C_j,V)}{|V|} \times \log_2 \left( \frac{\text{freq}(C_j,V)}{|V|} \right).
\]

Since \( V \) has been partitioned into \( V_1, V_2 \) by applying the test on the attribute “Type” (originally in the Products table), the weighted average to identify a class of a case after its partitioning, is

\[
\text{info}_{\text{Type}}(V) = \sum_{i=1}^{2} \frac{|V_i|}{|V|} \times \text{info}(V_i).
\]

Therefore, the quantity

\[
\text{gain}(\text{Type}) = \text{info}(V) - \text{info}_{\text{Type}}(V)
\]

measures the information gained by applying the test on the attribute “Type” for the ViewCache \( V \). The best thing is to always choose the test on an attribute with the maximal gain [2,15].

It is always a better idea to provide a probabilistic interpretation of a decision tree rather than a deterministic one, because the given attributes can almost surely not capture all the relevant information for making a classification. So pruning of a decision tree after having generated it, is the most usual approach. Pruning in a decision tree is done by replacing subtrees by leaf nodes. The leaf is then marked with the majority class and a probability is attached to this leaf. The probability is taken as the frequency with which a training case classified by this node to be in the majority class, is really in this class. We will now discuss how the concept of pruning can be applied on a ViewCache hierarchy. For example, in Figure 4, if we prune at the node, representing the ViewCache \( V11 \), we get a probabilistic decision tree (Figure 5) with the case \( V13 \) belonging to the class North_MN, with probability calculated using the formula [15].

\[
\frac{\text{freq}(\text{class1},\text{training subset})}{\text{freq}(\text{class1},\text{training subset}) + \text{freq}(\text{class2},\text{training subset})} = \frac{\text{freq}(\text{North_MN},V11)}{\text{freq}(\text{North_MN},V11) + \text{freq}(\text{South_MN},V11)} = 0.5.
\]

![Figure 5: A pruning step of the decision tree in Figure 4.](image)

We see that the probability depends on the cardinality of the ViewCache at the pruning site. For numerous number of records, the cardinality and the best ViewCache node to prune at, are the most concerning factors. In our example, if we prune at the node representing the ViewCache \( V11 \), the probability of reaching the classes North MN and South MN are both equal to 0.5. Thus we should try some other node to prune at or start with a test on a different attribute altogether. This will require materialization of the original ViewCache as well as the subsequent ViewCaches. Thus to find out the best test and hence the best decision tree, we need to perform multiple View materializations. So optimal View materialization is a concerning factor.

Traditional RDBMSs can only join two tables at a time. If a complex join involves more than two tables, the RDBMS needs to artificially break the query into a
series of pair-wise joins. For example, we need to join data from 3 tables: Purchases, Store and Products (Figure 1). A traditional RDBMS will have to select two tables to join initially, say Purchases and Store and an intermediate result consisting of Purchases joined to Store will be generated. This intermediate result, joined with Products, produces the final result. For $n$-tables, say this process would continue in order to generate the full result. At each step of a pair-wise join, we generate a binary ViewCache containing the record pointers to the base level tables. Join of Purchases and Store generates an intermediate binary ViewCache $V'$ which again joined with Products, generates the final ViewCache $V$.

![Diagram](image)

**Figure 6:** Pair-wise joining inside a RDBMS engine. An n-ary join is a sequence of binary joins.

Thus each step of an n-ary join process involves a pair-wise join with the generation of a binary ViewCache (Figure 6).

We now concentrate on optimally materializing a View for a test at each level of the decision tree. Let us consider a binary ViewCache derived from two base relations, say $R_i$ and $R_j$ occupying $n_i$ and $n_j$ pages, respectively. We partition the entries of the ViewCache into groups of pairs having the same ordered pair of page-ids [8]. So it is feasible to represent the partitions of any binary ViewCache and the corresponding disk pages of $R_i$ and $R_j$ respectively, by the weighted edges and vertices of a bipartite graph. Materialization of the View, is then equivalent to traversing all the edges, fetching the vertices and joining the tuples in the edges. Materialization of a View with an optimal cost can be described as an optimal graph traversal problem where one has to find a path that visits all its edges graph at a minimal cost.

The graph representing a ViewCache generated by the execution of relational operations underlying a query, may be skewed on a particular processor/s of a parallel data server and thus materialization of this View in parallel, will not be efficient. As a solution to this, we have first developed a theory for normal distribution of small portions of the vertices and edges of the graph from a processor to its neighbours, for replication. The percentage of the distribution for replication will however depend on the cardinality of the ViewCache concerned. The replicated edges will form a dynamic domain for task load sharing at run-time. The processor/s with the ViewCache will for a few iterations of Monte-Carlo simulation [3], distribute in packets, a few of the ViewCache partitions, the corresponding pages of data to its neighbouring processors for replication using a Binormal distribution.

After distribution, we developed the theory for optimal task load sharing so that the global cost of graph traversal follows a gradient descent towards minimization [9].

The details of the graph representation of a ViewCache, distribution for replication, dynamic load balancing and cost analysis are given in the subsequent sections.

## 2. GRAPH REPRESENTATION

In this section, we consider a binary ViewCache and its graph representation.

A binary ViewCache can be represented by a weighted, directed, bipartite graph $G = (V, E, w)$. The set of vertices $V$ refers to the union of all page sets for the relations $R_i$ and $R_j$; the set of edges $E$ refers to the set of ViewCache partitions; and the function $w$ assigns to each edge of $G$, a weight representing the number of tuples in each ViewCache partition.

To materialize a View, we need to access the pages corresponding to the relations $R_i$ and $R_j$ and join the tuples in them. The materialization cost depends on the sequence of the pages, because intelligent utilization of the buffers may cause significant reduction in the number of disk page accesses [8]. When we represent a binary ViewCache by a bipartite graph, the materialization becomes equivalent to traversing all the edges, fetching the vertices and joining the tuples in the edges. The most current set $S$ of vertices in the Cache
which are being accessed during traversal of the graph is called the working set of vertices. Intelligent utilization of the buffer for optimality, is then equivalent to efficiently utilizing the vertices in the working set during the graph traversal. The size of the working set influences the cost of the traversal.

Materialization of a View with an optimal cost can then be described as an optimal graph traversal problem as follows.

**Posed Problem:** Traverse once, all the edges \( E \) of the graph \( G = (V,E,w) \) with a minimum global cost.

Let a “white” vertex \( v_w \) correspond to one full page of \( R_i \) and a “black” vertex \( v_b \) to that of \( R_j \).

![Graphical Representation of a ViewCache](image)

**Figure 7:** Graphical Representation of a ViewCache showing the black and white vertices and the weights on the directed edges.

The cost for a white-black traversal in \( G \), is given by,

\[
C_{v_w,v_b} = \begin{cases} 
0, & \text{both } v_w \text{ and } v_b \text{ are in } S; \\
1, & \text{either } v_w \text{ or } v_b \text{ is in } S; \\
2, & \text{neither } v_b \text{ or } v_b \text{ is in } S.
\end{cases} \quad [2.1]
\]

### 2.1. Average I/O Cost for Graph Traversal

We have shown that the average I/O cost in traversing all \( E \) edges of the graph \( G = (V,E,w) \) is approximately of the order of

\[
|E| + \frac{1}{8} \times \left\{ b^2 - b(3d_w^{av} + 3d_b^{av} + S) + 6(d_w^{av} + d_b^{av}) \right\}
\]

where \( d_w^{av} \) and \( d_b^{av} \) are respectively the average degrees of white and black vertices in \( G \), and \( b \) is the size of the working set \( S \), assuming that \( |E| \) is large. Detailed proof is in [12].

### 2.2. Heuristics for Cost Optimal Graph Traversal

In [1], it was shown that finding a path to traverse all ViewCache partitions for materialization with best optimal cost, is \( NP \) complete. We will discuss below a strategy that deals with a heuristic approach.

First, select a white vertex \( v_w \) of minimum degree and insert it in \( S \). Then, start traversing all connected black vertices updating \( S \) with a new black vertex for each traversal until \( S \) is full. If \( S \) is full, do not bring further vertices. Decrease the degree of the white vertex by one, each time the traversal of an edge connected to a black vertex is over. If the degree of the white vertex becomes zero, get rid of it. Then choose a maximum degree black vertex connected to the white vertex and start traversing all connected white vertices until \( S \) is full. If \( S \) is full, do not bring further vertices. Continue till the traversal of all edges of graph \( G \) is complete [12].

Our intention is to free as many white vertices as possible, in every step, so that we save I/O by not fetching them again. By choosing maximum degree black vertices, we are fetching more white vertices, thus increasing the probability of having a very low degree white vertex in \( S \). This lowest degree white vertex is processed for all connected black vertices and is freed if possible. Figure 8 shows the traversal of \( G \) in Figure 7 according to the steps of the above algorithm.

The optimal path of traversal of \( G \) is the sequence \( \{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p\} \).

![Graph traversal following the algorithm](image)

**Figure 8:** Graph traversal following the algorithm.

![The sequence of changes in the working set during the graph traversal in figure 7.](image)

**Figure 9:** The sequence of changes in the working set during the graph traversal in figure 7.
3. DISTRIBUTION FOR REPLICATION

For market basket analysis as explained in Example 1.1, a View at a particular decision tree-level, may be distributed such that one processor corresponds to 2 million records, another processor to 5 million and another to 5000 and so on. Here, we have developed a method of clustering and distribution of data packets from unbalanced processor/s for replication. This distribution at a particular level of the decision tree, depends on the cardinality of the corresponding ViewCaches at that level which is again dependent on the test performed at that level. It is worth distributing in this way because the View will be materialized many times not only for different tests at different levels of the decision tree, but also for different tests at the same level to get a satisfactory information gain.

3.1. Clustering of Data for Distribution

Considering a sub-graph \( G' = (V', E', w') \) in a processor, we now proceed to cluster its vertices and edges to form a data packet and send this data packet to another processor for replication.

A cluster in a graph \( G' \), is a single black vertex along with the connectivities to several white vertices. The maximum size of a cluster is the number of records in a black page.

A data packet is a collection of one or more clusters.

![Figure 10. A cluster in G consisting of a black vertex connected by the edges to several white vertices.](image)

For a cluster, I/O cost is incurred for several white-black traversals related to the same black vertex. Thus, if a few such clusters are sent at each test level, to other processors for replication, the sending processor may save some I/O and processing at each such level.

3.1.1. Algorithm for Cluster Selection

Choose a white vertex \( v_w \) with the minimum degree and look for the black vertex \( v_b \) with the maximum degree linked to it. Eliminate the black vertex \( v_b \) and its associated edges to get a sub-graph. This forms a cluster to send. Subtract the degree of \( v_w \) and that of all the white vertices connected to \( v_b \), by one. Continue similarly in a processor to select a few clusters this way.

3.1.2. Cost Analysis

Let us consider the sub-graph \( G' = (V', E', w') \) to be traversed by a processor. The processor generates several clusters following the steps of algorithm 3.1.1 and sends each of them to other processors for replication.

The set of edges locally owned by a processor after sending away clusters for replication, is the static set of edges, that it has to traverse. A processor may also have due to Binormal distribution, a replicated set of edges from other processors, which it may have to traverse if necessary, during load balancing. The set of all the replicated sets from all processors forms the dynamic domain of the whole network.

We have found that the order of average saving in the I/O cost in the static set of edges for a processor at a decision tree-level, by replication of a cluster consisting of one black vertex and \( 1, 2, \ldots, k \) white vertices, can be given by

\[
\frac{d_w^{av} d_b^{av}}{2} \left( 1 - \frac{\Phi}{|E|} \left( d_w^{av}\right)^2 \right) \left( \frac{1}{d_w^{av} + 1} - \frac{\Phi}{|E|} \left( d_b^{av}\right)^2 \right)
\]

where, \( d_w^{av}, d_b^{av} \) are respectively the average degrees of white and black vertices in \( G' = (V', E', w') \); \( \Phi \) is defined as the summation

\[
\frac{1}{b-1} \sum_{k=1}^{b-1} (b-k)k
\]

\( b \) is the size of the working set \( S \), and \( |E'| \) is assumed to be large. This gives the order of edges contributed to the dynamic domain by replication [12]. Now let us talk about the average I/O saving for sending several clusters at each level of the decision tree.

The average I/O saving in the static set of a processor after replications at the first tree-level, is of the order of
where, \( t^{(1)} \) is the number of iterations of sending the clusters; and \( K, K_1, K_2, K_3 \) are defined as follows,

\[
K = d_b^{av}d_w^{av}; \quad K_1 = \phi \left( d_w^{av} \right)^2; \quad K_2 = \phi \left( d_b^{av} \right)^2;
\]

\[
K_3 = 1 + \frac{1}{d_w^{av}}. \quad \text{This gives the order of edges contributed to the dynamic domain, at the first tree-level [12].}
\]

Now, another important advantage is a cumulative effect of the I/O saving when we reach a certain level of the decision tree starting from the first. We have proved that, the average cumulative I/O saving in the static set for \( m \) levels of the decision tree can be written as the order of

\[
\frac{Kt^{av}m}{2} \left( K_3 - \frac{K_3K_1 + K_2}{|E|} \right)
\]

where, \( t^{av} \) is the average over the number of clusters sent at all \( m \) levels of the decision tree, and

\[
K = d_b^{av}d_w^{av}; \quad K_1 = \phi \left( d_w^{av} \right)^2; \quad K_2 = \phi \left( d_b^{av} \right)^2;
\]

\[
K_3 = 1 + \frac{1}{d_w^{av}} \quad \text{and assuming} \quad |E| \quad \text{large. This gives the order of edges contributed to the dynamic domain for all} \quad m\text{-decision tree-levels [12].}
\]

### 3.2. The Distribution

A data packet from an overloaded processor "migrates" to another processor for replication, following Binomial distribution whose corresponding density function \( g(x,y) \) with mean zero and standard deviation \( \sigma \), is given by,

\[
g(x,y) = \frac{1}{\sqrt{2\pi\sigma}} \exp \left( -\frac{x^2 + y^2}{2\sigma^2} \right)
\]

The graph of such a density function is a bell-shaped curve that is symmetric around the origin (Figure 11). We are using this function to ensure a uniform spread of the data in the \( x \) and \( y \) directions of a 2-d network of processors (extendible to three and higher dimensions). The probability of a data packet to fall within a range along \( x \) and \( y \) axes, can be evaluated by obtaining the area under this density function between the range [3]. Now, repeating a Monte-Carlo simulation by a processor at \((i,j)\), we draw a pair of random deviates in the interval \([0,1]\) from the above Binormal distribution [11,13]. This pair determines the location of the replicated data packet in the network and thus contributes to the edges in the dynamic domain [14]. Here, \( \sigma \) is a processor-specific parameter.

This way of distribution of data packets for replication is to ensure the uniformity of the dynamic domain of a processor which again ensures uniform load balancing scenario at run-time.

![Figure 11: The normal distribution of data from a single processor to the neighbouring processors in the 2-d space for replication.](image)

### 4. Run-Time Load Balancing

At run-time, based on agreements between the processors, a subset of tasks of traversal of edges in the dynamic domain may be transferred in a way such that the global cost for traversing all the edges of the total graph \( G \) at a particular time step is minimized. Each processor minimizes its local cost function by keeping the task load below a certain specified threshold and calculates its local Likelihood Ratio in making a decision (equation 4.1.1). The number of static and dynamic edges traversed by a processor are interdependent in making such a probabilistic decision.

![Figure 12: The static and the dynamic portions of the tasks, associated with the processors P1, P2, etc.](image)
4.1. Cost Analysis For Graph Traversal

In this section, we provide a cost model for the dynamic domain of the 2-d network of processors. We are considering the dynamic domain to give a total picture of the dynamic load balancing scenario in order to minimize the overall cost of View materialization, at any decision tree-level.

The probability of the processor at \((i,j)\) becoming active is proportional to the total cost it contributes to the network \([4]\). Here, by saying that a processor is active, we mean that it is actively taking part in receiving an extra load from other processors. By assumption, the cost function is additive under network decomposition. So adding the costs of the components' states should correspond to multiplying the probabilities of the components' states in the network. We consider an exponential function to relate the mapping from addition into multiplication.

Now, a processor at \((i,j)\) has an earlier estimated cost for static edge traversals depending on the weights of the edges and on the vertices whether or not, they are in \(S\) during the traversal. It will activate depending on the probability whether this static estimated cost (sc) at the current step \(t\) is lower than the cost incurred by it for the traversal in the dynamic domain (dc), at the previous time step.

Every processor at \((i,j)\) must thus compute locally:

\[
\text{Likelihood Ratio} = \frac{\text{prob(to activate)}}{\text{prob(not to activate)}} = \exp\{\text{dc} (t - 1) - \text{sc} (t)\} \quad [4.1.1]
\]

If the exponential term is positive, i.e., if the estimated cost for the static set at time \(t\) is lower than the cost incurred for the dynamic domain at time \((t-1)\), the processor at \((i,j)\) will more likely activate in receiving task loads in the dynamic domain.

4.1.1. The Global Cost in the Dynamic Domain

Let us consider a time step \(t\), when a load balancing situation arises. The processors with unbalanced task loads in the dynamic domain, send messages represented by vectors carrying information about task loads to be shared. The \((1 \times nk)\) incoming block message vector from \(k\) processors, denoted by

\[
r(t) = [r_0(t), r_1(t), r_2(t), \ldots, r_{k-1}(t)]',
\]

represents information regarding the source processor-id, its current status, the number of edges from the dynamic domain that it might share with some other processor for traversal, the weights and vertices corresponding to these edges, etc. A \((1 \times nk)\) block feature vector \(f_{i,j}(t)\) for a processor at \((i,j)\) represents features that need to be matched with the \(k\) incoming message vectors. This vector conveys the information whether it has the replicated data packet corresponding to the edge traversals requested by a processor sending the message vector or vice versa, its current status whether or not its task load fraction is less than a threshold and should accept an extra task load or give out some, etc. The matching is done by a processor as follows:

For \(p = 0, ..., (k-1)\),

\[
\text{label}(r_p(t), f_{i,j}(t)) = 0, 1, \text{or } -1.
\]

The list of all task loads \(\{q_{i,j}^p(r_p(t),t)\}\), messaged to a processor at \((i,j)\) via all \(r_p(t), p = 0, ..., (k-1)\), forms the input vector \(q_{i,j}(r(t),t)\) representing the number of edges and vertices in the dynamic domain, a processor needs to share at that particular time step and \(w_{i,j}(q_{i,j}(r(t),t))\) is the vector representing the associated weights. Let the communication for this agreement involve a message of size \(m_{i,j}(r_p(t),t)\).

Let us denote the state of the network by the pair \((r(t),t)\). The global cost function for the network, has as time dependent parameters \(\{f_{i,j}(t)\}\), associated with each processor at \((i,j)\) and we call this the feature base \(F\).
Let the global cost function for the dynamic domain of the network, as a contribution from all \((i,j)\) at time \(t\), be given by

\[
C_F(r(t), t) = \sum_{i,j} \{ C_{\text{comm}}(r(t), f_{ij}(t), m_{ij}(r(t), t)) \\
+ C_{i/o}(r(t), f_{ij}(t), q_{ij}(r(t), t)) \\
+ C_{\text{comp}}(r(t), f_{ij}(t), w_{ij}(q_{ij}(r(t), t))) \}
\]

[4.1.1.1]

where \(C_{\text{comm}}(r(t), f_{ij}(t), m_{ij}(r(t), t))\) is the cost due to agreements with the adjacent processors through communication of messages;

\(C_{i/o}(r(t), f_{ij}(t), q_{ij}(r(t), t))\) is the I/O cost; and

\(C_{\text{comp}}(r(t), f_{ij}(t), w_{ij}(q_{ij}(r(t), t)))\) is the CPU cost for joining the tuples during dynamic edge traversal [5,9]. Details are in [12].

The number of edges in the dynamic domain, traversed at every time step by the processors in the network, is given by the order of

\[
\sum_p \{ \log_e(\text{Likelihood } \_ \text{Ratio}_p) \\
- \log_e(\text{Likelihood } \_ \text{Ratio}_{\text{min}}) \}
\]

where, \(\text{Likelihood } \_ \text{Ratio}_{\text{min}}\) is the minimum over all the \(\text{Likelihood } \_ \text{Ratio}_p\) in the network of \(P\) processors [12].

4.2. The Threshold for Decision Making

A processor at \((i,j)\) calculates the fraction of the total weighted cost incurred on the basis of agreements of \(f_{ij}^p(t)\) with \(r_p(t)\), with respect to an estimated cost for each computational step. This weighted cost fraction should be less than a threshold for minimization. So, we may write,

\[
\text{critical-fraction} = v = \frac{\sum_{p=0}^{k-1} \text{label}(r_p(t), f_{ij}^p(t))q_{ij}(r_p(t), t)}{\text{estimated cost}}
\]

[4.2.1]

where \(v\) is a threshold. Each processor decides to receive or send based on agreements, the critical fraction of the task load [10]. It orients its own feature vector \(f_{ij}(t)\) with the information whether it can send or receive the task loads.

We may express the best threshold range (Figure 14) in terms of the number of processors, \(P\) for an optimal dynamic load balancing situation [10] as,

\[
1 > v > 1 - \frac{2}{P}
\]

[4.2.2]

Figure 14: The best threshold as a function of the number of processors, \(P\). As \(P\) increases, the threshold value limits to 1.

The complexity of the overall cost for View Materialization in parallel is of the order of

\[
\begin{cases}
T_{\max} + \frac{H}{P} \left( \Gamma - \delta \sum_p (T_{\max} - T_p) \right), \\
\text{if } \Gamma > \delta \sum_p (T_{\max} - T_p), \\
T_{\max}, & \text{otherwise}
\end{cases}
\]

where, \(T_{\max}\) is the maximum static cost associated with a processor out of \(P\) processors in the network, \(\Gamma\) is the total number of edges in the dynamic domain of the network, \(T_p\) is the static cost associated with each processor and \(\mu, \delta\) are constants of proportionality [12].

5. THE IMPLEMENTATION

This section presents some of the simulation results using data replication by normal distribution and dynamic load balancing in order to minimize the global cost. The implementation has been carried out on a
parallel computer CM-5 (Connection Machine) which simulates a parallel data server. The normal distribution as a replication technique as well as the asynchronous communication between processors, for dynamic load balancing have been implemented using a program written in C++. The simulation was targeted towards experimenting with the dynamic domain of a graph. We studied variations of threshold and its impact on load balancing and also studied the gradient descent in the global cost (Figures 14, 15 and 16).

In our implementation, we are considering a range of data to lie in a 2-d space. This 2-d space has then been subdivided into small 2-d partitions and the computations in a particular data partition, are entirely local to that partition. The data partitions are then evenly loaded onto \( P \) processors arranged in a checkerboard fashion. A ViewCache is generated arbitrarily using pointer pairs relating data records in all the processors. A specific test is then performed on this ViewCache. During this test, a Monte-Carlo simulation of the normal distribution of several data packets is carried out by each processor at run-time in order to replicate a portion of its data (depending on the cardinality of the ViewCache it owns). After many such replications for several tests at each level, we studied the behaviour of View Materialization.

![Figure 16: Local contributions (costs) along the 2-d network of \( P = 64 \) processors for three different thresholds.](image)

6. CONCLUSIONS

In this research, we concentrated on optimal View materialization in a parallel environment for data mining, using a graph theoretic approach. Since the current problem lies in the huge amount of data movement involved in dynamic load balancing for each decision tree-level, we proposed a technique for fractional data replication from a ViewCache at run-time, using normal distribution function. This generates a dynamic domain of the ViewCache where an optimal, decision-based, load balancing mechanism is applied to speed up the materialization. The dynamic network component is interleaved with the static, processor-specific task of materialization at each tree-level. We have simulated a parallel data server only to get an initial idea of the possible results on the dynamic domain. As a future direction, we plan to experiment more on the interleaved executions of static and dynamic domains.

7. REFERENCES


15. Wüthrich, B. 1995. Knowledge Discovery in Databases. Lecture Notes. Hong Kong University of Science and Technology.
1381 P. Morin & R.D. Spies, Convergent spectral approximations for the thermomechanical processes in shape memory alloys
1382 D.N. Arnold & X. Liu, Interior estimates for a low order finite element method for the Reissner-Mindlin plate model
1383 D.N. Arnold & R.S. Falk, Analysis of a linear-linear finite element for the Reissner-Mindlin plate model
1384 D.N. Arnold, R.S. Falk & R. Winther, Preconditioning in $H(\text{div})$ and applications
1385 M. Lavrentiev, Nonlinear parabolic problems possessing solutions with unbounded gradients
1386 O.P. Bruno & P. Laurence, Existence of three-dimensional toroidal MHD equilibria with nonconstant pressure
1387 O.P. Bruno, F. Reitich, & P.H. Leo, The overall elastic energy of polycrystalline martensitic solids
1388 M. Fila & H.A. Levine, On critical exponents for a semilinear parabolic system coupled in an equation and a boundary condition
1390 J.M. Berg & H.G. Kwatny, Unfolding the zero structure of a linear control system
1391 A. Sei, High order finite-difference approximations of the wave equation with absorbing boundary conditions: A stability analysis
1392 A.V. Coward & Y.Y. Renardy, Small amplitude oscillatory forcing on two-layer plane channel flow
1393 V.A. Pliss & G.R. Sell, Approximation dynamics and the stability of invariant sets
1394 J.G. Cao & P. Robin, A new computational model for heterojunction resonant tunneling diode
1395 C. Liu, Inverse obstacle problem: Local uniqueness for rougher obstacles and the identification of a ball
1396 K.A. Pericak-Spector & S.J. Spector, Dynamic cavitation with shocks in nonlinear elasticity
1397 G. Avalos & I. Lasiecka, Exponential stability of a thermoelastic system without mechanical dissipation II: The case of simply supported boundary conditions
1398 B. Brighi & M. Chipot, Approximation of infima in the calculus of variations
1399 G. Avalos, Concerning the well-posedness of a nonlinearly coupled semilinear wave and beam-like equation
1400 R. Lipton, Variational methods, bounds and size effects for composites with highly conducting interface
1401 B.T. Hayes & P.G. LeFloch, Non-classical shock waves in scalar conservation laws
1402 K.T. Joseph & P.G. LeFloch, Boundary layers in weak solutions to hyperbolic conservation laws
1403 Y. Diao, C. Ernst, & E.J.J. Van Rensburg, Energies of knots
1404 Xiaofeng Ren, Multi-layer local minimum solutions of the bistable equation in an infinite tube
1405 Vlastimil Pták, Krylov sequences and orthogonal polynomials
1406 T. Aktosun, M. Klaus, & C. van der Mee, Factorization of scattering matrices due to aprioting of potentials in one-dimensional Schrödinger-type equations
1408 D.N. Arnold, R.S. Falk, & R. Winther, Preconditioning discrete approximations of the Reissner-Mindlin plate model
1409 M.A. Kouritzin, On exact filters for continuous signals with discrete observations
1410 R. Lipton, The second Stekloff eigenvalue and energy dissipation inequalities for functionals with surface energy
1411 R. Lipton, The second Stekloff eigenvalue of an inclusion and new size effects for composites with imperfect interface
1412 W. Littman & B. Liu, The regularity and singularity of solutions of certain elliptic problems on polygonal domains
1413 C.R. Collins, Spurious oscillations are not fatal in computing microstructures
1414 M.A. Horn, Sharp trace regularity for the solutions of the equations of dynamic elasticity
1415 A. Friedman, B. Hu & Y. Liu, A boundary value problem for the Poisson equation with multi-scale oscillating boundary
1416 P. Bauman, D. Phillips & Q. Tang, Stable nucleation for the Ginzburg-Landau system with an applied magnetic field
1417 J.M. Berg, A strain profile for robust control of microstructure using dynamic recrystallization
1418 P. Klouček, Toward the computational modeling of nonequilibrium thermodynamics of the Martensitic transformations
1419 S. Chawla & S.M. Lenhart, Application of optimal control theory to in Situ bioremediation
1420 B. Li & M. Luskin, Nonforming finite element approximation of crystalline microstructure
1421 H. Kang & J.K. Seo, Inverse conductivity problem with one measurement: Uniqueness of balls in $\mathbb{R}^3$
1422 Avner Friedman & Robert Gulliver, Organizers, Mathematical modeling for instructors, July 29 – August 16, 1996
1423 G. Friescecke, Pair correlations and exchange phenomena in the free electron gas
1424 Y.A. Li & P.J. Olver, Convergence of solitary-wave solutions in a perturbed Bi-Hamiltonian dynamical system
1425 C. Huang, On boundary regularity of vortex patches for 3D incompressible euler systems
1426 C. Huang, A free boundary problem with nonlinear jump and kinetics on the free boundaries
X. Chen, C. Huang & J. Zhao, A nonlinear parabolic equation modeling surfactant diffusion
A. Friedman & B. Hu, Optimal control of chemical vapor deposition reactor
A. Friedman & B. Hu, A non-stationary multi-scale oscillating free boundary for the Laplace and heat equations
X. Chen, Existence, uniqueness, and asymptotic stability of traveling waves in nonlocal evolution equations
J. Yong, Finding adapted solutions of forward-backward stochastic differential equations – Methods of continuation
J. Yong, Linear forward-backward stochastic differential equations
D.A. Dawson & M.A. Kouritzin, Invariance principles for parabolic equations with random coefficients
R. Lipton, Energy minimizing configurations for mixtures of two imperfectly bonded conductors
D.C. Dobson & F. Santosa, Nondestructive evaluation of plates using Eddy current methods
W. Littman & B. Liu, On the spectral properties and stabilization of acoustic flow
S. Sarkar & S. Sundar Sarkar, Normal distribution as a method for data replication in a parallel data server
S. Sarkar & S. Sundar Sarkar, Parallel view materialization with dynamic load balancing: A graph theoretic approach
S. Sarkar & S. Sundar Sarkar, Internet and relational databases in a multi-tier client/server model
J. Liang & S. Subramaniam, Numerical computing of molecular electrostatics through boundary integral equations
J. Wu, Inviscid limits and regularity estimates for the solutions of the 2-D dissipative quasi-geostrophic equations
P. Constantin & J. Wu, Statistical solutions of the Navier-Stokes equations on the phase space of vorticity and the inviscid limits
M.A. Kouritzin, Stochastic processes and perturbation problems defined by parabolic equations with a small parameter
M.A. Kouritzin, Approximations for singularly perturbed parabolic equations of arbitrary order
A. Novick-Cohen, Triple junction motion for Allen-Cahn/Cahn-Hilliard systems
P. Klouček, Approximations of the laminated microstructures
S. Sarkar & S.S. Sarkar, A graph theoretic approach for parallel view materialization with dynamic load balancing
S. Chawla, A minimax problem for parabolic systems with competitive interactions
B. Luong & F. Santosa, Quantitative imaging of corrosion in plates by Eddy current methods
R. Jordan & B. Turkington, Ideal magnetofluid turbulence in two dimensions
M. Fels & P.J. Olver, Moving coframes. I. A practical algorithm
S.Y. Malassy, On the Schwarz alternating method for eigenvalue problems
R. Lipton, Design of particle reinforced heat conducting composites with interfacial thermal barriers
J. Berg, A. Yezzi, & A. Tannenbaum, Phase transitions, curve evolution, and the control of semiconductor manufacturing processes
G. Avalos & I. Lasiecka, Uniform decay rates of solutions to a structural acoustics model with nonlinear dissipation
M. Nitsche, Siemens/IMA technical report
L. Wang, J.A. Cox, & A. Friedman, Model analysis of homogeneous optical waveguides by boundary integral method
C.P. Fung & S. Lototsky, Nonlinear filtering: Separation of parameters and observations using Galerkin approximation and Wiener chaos decomposition
S. Northshield, Several proofs of Ihara’s theorem
T. Aktosun, M. Klaus & C. van der Mee, Wave scattering in one dimension with absorption
F. Santosa, M. Vogelius, & J.-M. Xu, An effective nonlinear boundary condition for a corroding surface. Identification of the damage based on electrostatic data
J. Wu, Well-posedness of a semilinear heat equation with weak initial data
J. Wu, Quasi-geostrophic type equations with weak initial data
J. Ma & J. Yong, Approximate solvability of forward-backward stochastic differential equations
T.-P. Tsai, On Leray’s self-similar solutions of the Navier-Stokes equations satisfying local energy estimates
M.K. Gobbert, T.P. Merchant, L.J. Borucki, & T.S. Cale, A multiscale simulator for low pressure chemical vapor deposition
M.C. Tesi, E.J. Janse van Rensburg, E. Orlandini, & S.G. Whittington, Torsion of polygons in $\mathbb{Z}^3$
M. Grinfeld & A. Novick-Cohen, The viscous Cahn-Hilliard equation: Morse decomposition and structure of the global attractor
M. Mascagni, Polynomial versus matrix methods for leap-ahead in shift-register type pseudorandom number generators
M. Mascagni, Parallel linear congruential generators with prime moduli
B. Li & M. Luskin, Approximation of a Martensitic laminate with varying volume fractions
D. Yang, Stabilized schemes for mixed finite element methods with applications to elasticity and compressible flow problems
P.J. Olver & V.V. Sokolov, Integrable evolution equations on associative algebras
A. Bondarenko, Singular structure of the fundamental solution of the transport equation
S. Sarkar & S.S. Sarkar, Views and data mining in a parallel data server
P.E. Bigneislen & M. Cheney, Models for an anesthesia breathing circuit
R. Lipton, Influence of interfacial surface conduction on the DC electrical conductivity of particle reinforced composites
B. Cockburn & C.-W. Shu, The local discontinuous Galerkin method for time-dependent convection-diffusion systems