A GRAPH THEORETIC APPROACH FOR PARALLEL VIEW MATERIALIZATION WITH DYNAMIC LOAD BALANCING

By

Sanhita Sarkar

and

Shyam Sundar Sarkar

IMA Preprint Series # 1447

December 1996

INSTITUTE FOR MATHEMATICS AND ITS APPLICATIONS

UNIVERSITY OF MINNESOTA

514 Vincent Hall
206 Church Street S.E.
Minneapolis, Minnesota 55455

A Graph Theoretic Approach for Parallel View Materialization with Dynamic Load Balancing

Sanhita Sarkar[1]
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.

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

ABSTRACT

A *view* or a *subscheme* is an abstract model of a portion of the conceptual database and is generated as a result of a query on the physical Relational Database. A ViewCache is a stored collection of pointers pointing to records of underlying relations needed to materialize a view. In this paper, we concentrate on the ViewCache method of materialization which has been proven to achieve both high performance and efficient incremental updates. First, 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 edges. Next, we have concentrated on an efficient way of parallelizing it. Currently, for shared-nothing MPP architecture, data is partitioned at load time in a shared-nothing way. But a view generated by a query on the Relational Database may be such that it is skewed to a processor/s. So the materialization of this view 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 the ViewCache partitions. At run-time, based on mutual agreements, task load can be shared between processors, to minimize the overall cost. This normal distribution of fractional replicated data will lead to a unique probabilistic run-time solution for load balancing, leading to an optimal cost for view materialization on a parallel data server.

1. INTRODUCTION

The concept of a view that stems from the "closure" of relational algebra is probably the most important asset of the relational model. Views have been used for providing conceptual subsets of the database to different users and as an implementation tool for efficiency, security, etc [Roussopoulos 1991]. One of the principal methods for implementing views is the *ViewCache* method. It *caches* access paths of views and intermediate results in terms of pointers to actual data records. The pointers are materialized as needed.

The *Incremental Access Method* [Roussopoulos 1991] or IAM capitalizes on the View Cache storage organization for performing the update and materialization of the ViewCaches in an interleaved mode using one-pass algorithms. Compared to the other standard techniques for supporting views that requires re-execution of the definition of the view, the IAM offers significant performance advantages. In the ViewCache method, the cost of answering a query against a view for

[1] To whom correspondence should be addressed.

Technical report, Institute for Mathematics and its Applications, University of Minnesota.

the second time, can be significantly lower than the cost of answering it the first time around. For example, if no updates have occurred between the first and a subsequent request, the cost of the latter is reduced to retrieving only the absolutely needed data for the result. We assume that the first-time request (FTR) cost for the ViewCache is approximately the same as that of query reexecution. In processing a subsequent request after an update has occurred, the IAM can be invoked in order to correct its affected pointers before materializing from them. We refer to this update cost as the ViewCache Update cost (VCU) and the cost of materializing the view as the ViewCache Materialize cost (VCM). In addition, we have to take into account for the ViewCache Read cost (VCR) for reading the ViewCache in before materialization, and the ViewCache Write cost (VCW) for writing the modified portion of it back to disk for later requests. A ViewCache is cost effective, if VCR + VCU + VCM +VCW < FTR. We assume that the update of the ViewCache is done first followed by materialization. Performance gains are also higher for multilevel ViewCaches because all the I/O and CPU for handling intermediate results are avoided.

Now, let us briefly discuss the *ViewCache model*. Let DB = $\{R_i\}$ be a set of base relations. A unique tuple identifier (TID) is assigned to each tuple of a relation R_i. The access path for a view consists of a collection of TIDs of records of the underlying base relations or other access paths needed for materializing the view. An efficient storage structure is the *ViewCache index* which inherits properties from both caches and indexes. Like a cache, the system can use it without having to search for the records that correspond to the view. It can follow the TIDs to materialize the view records. Like an index, it stores pointers to the records and is buffered in main memory or read in its entirety. A ViewCache entry is either a single TID or a pair of TIDs, depending on whether the view corresponds to a unary or binary relational operator, respectively. Each entry in a ViewCache has also its own TID. This allows the definition of multiple-level hierarchies of ViewCaches. Materialization of an entry from a ViewCache is done by recursive retrieval of lower ViewCache tuples until base relation tuples are obtained. Views stored as ViewCaches can be referenced by any (SQL) query for selectively retrieving from it or for creating other higher level views.

To discuss about optimally materializing a view, let us consider a binary ViewCache structure. A binary ViewCache is derived from two base relations, say R_i and R_j occupying n_i and n_j pages, respectively. The entries of the ViewCache are now pairs of TIDs $<TID_i, TID_j>$, which point to R_i and R_j respectively. Let PID_i and PID_j be defined as

$$PID_i = pageid(R_i, TID_i),$$
$$PID_j = pageid(R_j, TID_j),$$

where the function *pageid* returns the page number for a tuple. We partition the entries of the ViewCache into groups of pairs having the same ordered pair $<PID_i, PID_j>$. These partitions are grouped and stored into the ViewCache disk pages in such a way that for any pair of ViewCache pages u and v for which u preceeds v,

$$pageid(R_i, TID_s) \le pageid(R_i, TID_k)$$
$$pageid(R_j, TID_t) \le pageid(R_j, TID_l)$$

for any $<TID_s, TID_t> \in u$ and $<TID_k, TID_l> \in v$.

In this paper, we have described these partitions and the corresponding disk pages of R_i and R_j respectively, by the weighted edges and vertices of a graph. Materializing the view, is equivalent to traversing all the edges, fetching the vertices; the disk pages of the operand relations that corresponding to each edge and joining the tuples in them. The materialization cost depends on the traversal sequence through the edges of the graph, because intelligent utilization of the buffers may cause significant reduction in the number of disk page accesses. Materialization of a View with an optimal cost can thus be described as an *optimal graph traversal problem* where one has to find a path that visits all edges of the graph at a minimal cost.

Now, if we want to implement the above graph traversal problem on a parallel data server based on a shared-nothing MPP architecture, we have to explore all possible ways to exploit parallelism in an efficient manner. We have to first ensure optimal partitioning of the edges, the nodes of the graph and the corresponding update logs among the processors. Secondly, we have to ensure optimal task partitioning throughout the parallel server. Here, by "task", we mean optimal traversal of the edges distributed to processors.

Though data partitioning is done initially at load time in a shared-nothing way, the graph representing a view created on the basis of the execution of relational operations underlying a query may be skewed on a particular processor or processors in the parallel data server. 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 neighbouring processors at run time, for replication. We can explain the possibility of such distribution by an example.

Example 1.1.

Let us consider an example using data from a national grocery retailer. The retailer wants to know more about the various buying patterns of customers and which products were commonly purchased at that time. A star schema was selected as the data model. We consider a part of the star schema to explain the distribution of data packets for replication (Figure 1). We consider 3 tables: *Products* of size 83K, *Purchases* of size 170M and *Store* of size 900. There is a foreign-key relationship from *Products* to *Purchases* and also from *Store* to *Purchases*. For uniform distribution over 100 processors, say, in a 2-d network, we shall first distribute at load time, to each node, 830 *Products* tuples or records, the corresponding *Purchases* records carrying foreign keys from those 830 *Products* records, as well as records from the *Store* table whose keys are also inherited by those *Purchases* records. In this specific example, the *Store* table can be distributed to every node because it is small in size. We are thus done with the initial distribution at load time. Now at run time, suppose we take a join between the tables *Products* and *Purchases* in parallel to create the ViewCache on a particular processor (Figure 2).

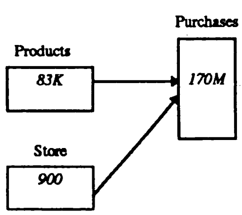
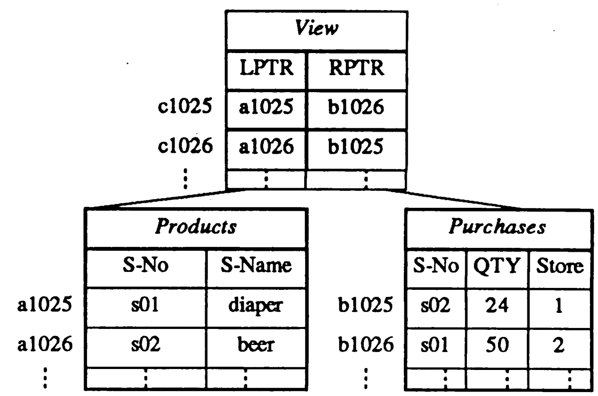


Figure 1: A part of a market analysis star schema for a grocery retailer.

The View as shown in Figure 2, may be skewed to a particular processor. This processor with the ViewCache will then partition it into groups of pairs having the same ordered pair of page-ids. It will then for a few iterations of Monte-Carlo simulation, distribute in packets, a few of these partitions, the corresponding pages of data to its neighbouring processors for replication using the Binormal distribution. This is done so that the I/O cost is not exceptionally high for a single user due to transfer of data. After several usages, the distribution of the partitions will be uniform across the network to ensure efficient parallelism. So the number of iterations of Monte-Carlo simulation, and the data packet size, in each iteration will depend on the number of partitions, the user decides to send.



View = Join *Products Purchases* where *S-No* = *S-No*.

Figure 2: The Binary ViewCache created from the relational operation on the star schema..

After distribution, we developed the theory for optimal task load partitioning at run time. The dynamic nature of task load sharing will obviously depend on the nature of the distribution of the edges and the nodes between the processors. Each processor maintains its local cost for traversing the edges (sum of I/O cost for reads, CPU cost for joining the tuples and communication cost for sending messages) below a certain specified threshold so that the global cost of graph traversal is minimized [Sarkar, June 1996]. At run time, the processors coordinate with each other asynchronously in a way to minimize the global cost in the whole network. In case of an unbalanced task load for traversal, a processor sends a message asynchronously to another processor containing its partial replicated nodes and edges, so that this other processor may take over its task only if its own cost fraction is less than a specified threshold. The above mentioned global cost as the sum of the local contributions from all processors is thus minimized following a gradient descent method of optimization.

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

2. GRAPH REPRESENTATION

In this section, we will pose the problem of optimal View materialization in terms of a problem of optimal graph traversal. For the representation, we will consider a binary ViewCache because a unary ViewCache will be a trivial case to consider. Let us first consider the following definitions:

A *directed* and *weighted graph* G can be represented as a triplet (V,E,w) consisting of a finite set of vertices V, a finite set of edges E connecting these vertices and a real valued function $w: E \rightarrow \Re$ defined on E. G is a *directed*, implies that for any $u, v \in V$, the edge (u,v) is an ordered pair indicating that there is a connection from u to v. G is *weighted*, implies that it has weights associated with each of its edges.

An *optimal graph traversal problem* refers to traversing along all the weighted edges E of a graph G at most once, such that the total cost for the traversal with respect to the weights, is minimum.

As explained earlier, a binary ViewCache is derived from two base relations R_i and R_j occupying say, n_i and n_j pages respectively, i.e., $i = 1,2,...,n_i$ and $j = 1,2,...,n_j$.

Definition 2.1. A binary ViewCache is a weighted, directed, bipartite graph $G \equiv (V,E,w)$. The set of vertices V refers to the union of sets of vertices representing page sets for the relations R_i and R_j which are pointed to, by the pointers in the ViewCache partitions; the set of edges E refers to the set of ViewCache partitions connecting a vertex representing a page in relation R_i to a vertex representing a page in R_j; and the function w assigns a weight to each edge of G, where the weight represents the number of tuples in the ViewCache partition represented by that edge.

We will consider two types of colored vertices according to the relations R_i or R_j they correspond. 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.

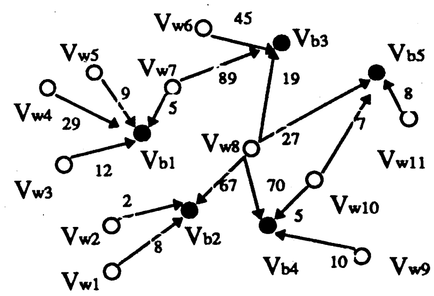


Figure 3: Graphical Representation of a ViewCache showing the black and white vertices and the weights on the directed edges.

A white vertex v_w is considered to be "*linked*" with a black vertex v_b by an edge, if there is at least one tuple in a page of R_i corresponding to v_w that is joinable with another tuple in a page of R_j corresponding to v_b. These edges represent the partitions of the ViewCache. A white vertex v_w can only be *directed* to a black vertex v_b. The *degree* of a vertex is the total number of edges linked to it. The *weight* of an edge from any v_w to a v_b is the number of tuples that have been grouped together to refer to the same v_b.

Definition 2.2. The most current set S of vertices, which are being touched, during traversal of the graph is called the *working set* of vertices.

The size of the working set of course influences the cost for the traversal. The larger the size, the more vertices it may hold for some time, and as a result, the I/O cost to traverse to any other node from a vertex already present in the set, will be only one page read. Else, it will be two pages.

Definition 2.3. Materialization of a View is traversal of the corresponding graph G under the conditions of the working set S.

Posed Problem: Traverse *once* in parallel, all the edges E of the graph $G \equiv (V,E,W)$ with a minimum global cost.

During a white-black traversal in G, the working set, S is always checked to see whether the white or the black vertex already exists in it. If it does, the traversal will incur a cost of value 1, for reading one page. If none of the vertices exist, the cost value will be 2. Thus, for a white-black traversal,

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

where, C_{v_w,v_b} is the I/O cost for one white-black traversal.

2.1. Average I/O Cost for Graph Traversal

Lemma 2.1.

The average I/O cost in traversing all E edges of the graph $G \equiv (V,E,w)$ is approximately of the order of

$$|E|+\frac{1}{8}\times\{b^2-b(3d_w^{av}+3d_b^{av}+5)+6(d_w^{av}+d_b^{av})\}$$

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.

Proof.

Without loss of generality, let us start from the situation that S is already full with b vertices, black and white.

If S consists of one white and (b-1) black vertices, then the probability of choosing the next first edge whose *one* of the vertices is in S, is $\frac{d_w^{av}}{|E|}+\frac{d_b^{av}}{|E|}\times(b-1)$. Subtracting from this term, the probability of both vertices of the chosen edge to be in S, (I/O cost is 0) we get the probability of having *exactly one* of its vertices in S (I/O cost is 1), as

$$\left\{\frac{d_w^{av}}{|E|}+\frac{d_b^{av}}{|E|}\times(b-1)\right\}-\frac{(b-1)}{|E|}$$

For S with two white and (b-2) black vertices, the probability will change to,

$$\left\{\frac{d_w^{av}}{|E|}\times 2+\frac{d_b^{av}}{|E|}\times(b-2)\right\}-\frac{(b-2)\times 2}{|E|}.$$

Considering all such cases the total probability is,

$$\left\{\frac{d_w^{av}}{|E|}\times\sum_{k=1}^{b-1}k+\frac{d_b^{av}}{|E|}\times\sum_{k=1}^{b-1}(b-k)\right\}-\sum_{k=1}^{b-1}\frac{(b-k)\times k}{|E|}$$

$$=\frac{(b-1)(b-2)}{2}\left\{\frac{d_w^{av}+d_b^{av}}{|E|}\right\}-\frac{b(b-1)(b-2)}{2|E|}+\frac{(b-1)b(2b-1)}{6|E|}.$$

The average probability over all cases is,

$$\frac{(b-2)}{2}\left\{\frac{d_w^{av}+d_b^{av}}{|E|}\right\}-\frac{b(b-2)}{2|E|}+\frac{b(2b-1)}{6|E|}$$

$$=\frac{3(b-2)\left(d_w^{av}+d_b^{av}\right)-3b(b-2)+b(2b-1)}{6|E|} \quad [2.1.1].$$

The probability that *none* of the vertices of the chosen edge is in S (I/O cost is 2), is

$$1-\frac{3(b-2)\left(d_w^{av}+d_b^{av}\right)-3b(b-2)+b(2b-1)}{6|E|} \quad [2.1.2].$$

So the *average I/O cost* for traversing the first edge is of the order of

$$\frac{1}{2}\left[2\times\left\{1-\frac{3(b-2)\left(d_w^{av}+d_b^{av}\right)-b^2+5b}{6|E|}\right\}+\left\{\frac{3(b-2)\left(d_w^{av}+d_b^{av}\right)-b^2+5b}{6|E|}\right\}\right]$$

The average I/O cost for the nth edge is thus of the order of

$$1-\frac{3(b-2)\left(d_w^{av}+d_b^{av}\right)-b^2+5b}{12(|E|-n+1)}$$

$$\cong 1-\frac{3(b-2)(d_w^{av}+d_b^{av})-b^2+5b}{12(1+|E|)\left(1-\frac{n}{1+|E|}\right)}$$

$$\approx 1-\frac{\{3(b-2)(d_w^{av}+d_b^{av})-b^2+5b\}\left(1+\frac{n}{1+E}\right)}{12(1+|E|)}$$

taking the predominant terms only, considering $|E|$ large.

So, for all n = 1 to $|E|$, the above term becomes,

$$|E|-\frac{3(b-2)(d_w^{av}+d_b^{av})-b^2+5b}{12(1+|E|)}\times\sum_{n=1}^{|E|}\left(1+\frac{n}{1+|E|}\right)$$

$$\cong|E|-\frac{3(b-2)(d_w^{av}+d_b^{av})-b^2+5b}{12}\times\left\{\frac{|E|}{1+|E|}+\frac{|E|(|E|-1)}{2(1+|E|)^2}\right\}$$

$$\approx|E|-\frac{3(b-2)(d_w^{av}+d_b^{av})-b^2+5b}{12}\times\left\{1-\frac{1}{|E|}+\frac{1}{2}\right\}$$

$$\cong |E| - \frac{3}{2}\times\left\{\frac{3(b-2)(d_w^{av}+d_b^{av})-b^2+5b}{12}\right\}$$

considering $|E|$ large.

Simplification yields the I/O cost as the order of

$$|E| + \frac{1}{8}\times\left\{b^2 - b(3d_w^{av}+3d_b^{av}+5)+6(d_w^{av}+d_b^{av})\right\} \quad [2.1.3].$$

■

2.2. Heuristics for Cost Optimal Graph Traversal

It has been shown by Amir and Roussopoulos (1990) that finding a path to traverse all ViewCache partitions for materialization with an optimal cost, is *NP* complete. Equation 2.1.2 gives the average I/O cost for graph traversal. The worst case involves a high I/O cost for which every edge traversal incurs a cost 2, whereas, the best case involves a I/O cost of 1 for each edge traversal. We will discuss below a strategy that deals with an average case instead of an extreme best or worst case. The algorithm for cost-optimal traversal of the graph *G* following the status of *S*, is as follows:

<u>*Step 1.*</u> Select a white vertex v_w of minimum degree. If more than one, choose the one connected to a black vertex v_b of maximum degree. Insert the chosen vertex in the working set.

<u>*Step 2*</u>. Start traversing to all connected black vertices and update the working set inserting a new black vertex for each traversal but make sure, the black vertex with maximum degree chosen in Step 1 is not replaced and stop when the working set is full. 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 node becomes zero, get rid of it.

<u>*Step 3.*</u> Switch back to the black vertex with maximum degree.

<u>*Step 4.*</u> Eliminate from *G* and working set, the vertices with all edges connected to it, already traversed.

<u>*Step 5.*</u> Follow Steps 2-4, for the white vertices connected to the black vertex of Step 3, till the working set is full.

<u>*Step 6.*</u> If the working set has no black node undone, choose a white vertex of minimum degree from the working set connected to another black vertex of maximum degree and perform a white-black traversal. If there is no such white node in the working set, choose from *G*. Update the working set inserting the new vertex in it.

<u>*Step 7.*</u> Continue with Steps 2-6 till the traversal of all edges of graph *G* is complete.

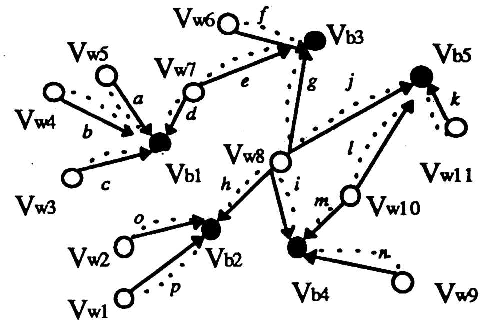


Figure 4: Graph traversal for minimum I/O cost, following the algorithm. The dotted lines labeled with alphabets signify the sequence of traversals through the edges in compliance with the working set.

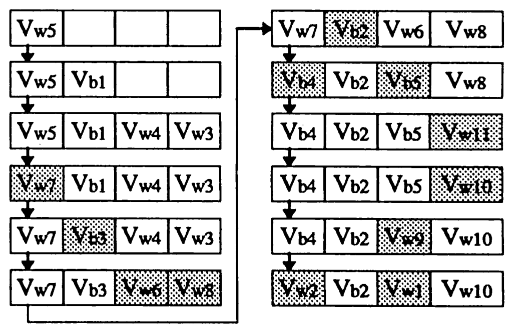


Figure 5: The sequence of changes in the working set during the graph traversal in figure 2.

In Steps 1-5, the number of white vertices connected to a maximum degree black vertex that we are fetching during traversal, does not exceed the working set size. Similarly, the number of black vertices connected to a minimum degree white vertex that we are fetching during traversal, does not exceed the working set size. Our intention is to free as many white vertices as possible, in every step. 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 the working set. This lowest degree white vertex is processed for all connected black nodes when we go back to Step 1, and is freed if possible.

Figure 4 shows the traversal of *G* in Figure 3 according to the steps of algorithm 2.1. The working set is updated at every step of traversal and is shown in Figure 5. First, as in Step 1, the vertex v_{w5} is chosen as the starting vertex and is inserted into the working set.

v_{w4} or v_{w3} can also be chosen as the starting vertex because they each have the minimum degree of 1 connected to the same black vertex v_{b1} with the maximum degree of 4, compared to all other black vertices in G. There are no black vertices other than v_{b1} connected to v_{w5}. So, the edge connecting v_{w5} with v_{b1} is traversed following Step 2 and v_{b1} is inserted into the working set. Step 3 is skipped as it is the same as Step 2 in this case. By Step 4, v_{w5} can be eliminated from the graph and replaced in the working set, but v_{b1} can't be eliminated as there is yet another edge to v_{w7} that needs to be traversed. According to Step 5, Steps 2-4 are followed similarly for the white vertices v_{w4} and v_{w3}. All the white-black joins of v_{w5}, v_{w4}, v_{w3} and v_{w7} to v_{b1} are thus completed and the vertices v_{w5}, v_{w4}, v_{w3} and v_{b1} are eliminated from G and ready to be replaced in the working set. Following Step 6 now, v_{w7} is the only white vertex with minimum degree in the working set and so, a white-black join of v_{w7} and v_{b3}, is performed. Steps 2-6 are similarly done for all other vertices in G. At one point, Step 5 is followed by v_{w8} where, it performs white-black joins with each of the black vertices v_{b2}, v_{b4} and v_{b5}, switches back to v_{b3} and then eliminates v_{b3}, v_{w7} and v_{w6} from G and eventually replaces them from the working set. It leaves intact the undone black vertices v_{b2}, v_{b4} and v_{b5} in the working set. After similar steps through v_{b5} and v_{b4}, a stage is reached when there is no white vertex connected to v_{b4} that is again connected to a new black vertex. So, the only option is to switch to the undone black vertex v_{b2} in the working set and perform the white-black joins from v_{w2} and v_{w1} following similar steps. We are then done with the traversal of all edges of G at a minimum I/O cost. The optimal path of traversal of G is thus the sequence $\{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p\}$ as shown in Figure 4.

3. DISTRIBUTION FOR REPLICATION

In this section we will describe an efficient way to parallelize the optimal traversal of the graph G. For this, we need to distribute the edges and vertices of G uniformly among the processors. For market basket analysis as explained in Example 1.1, the View may be distributed such that one processor corresponds to 2 million records, another processor to 5 million and another to 5000 and so on. In graphical terms, G may be such that most of its edges and vertices are skewed to one processor or processors. To deal with this scenario of unbalanced distribution of edges and vertices of G, we have developed a method of clustering and distribution at run time, some of the edges, vertices and the update log (explained in the next section) from a processor with an unbalanced load, not only to reduce its I/O cost but also for the advantage of dynamic load balancing (to be discussed later). It is worth distributing in this way because the View as in the case of market basket analysis will be materialized many times.

3.1. The Update Log

In this section, we will discuss how the updates taking place in the ViewCache and the physical database in a processor, are conveyed to the other processors carrying replicated portions of its data, so that it may simultaneously perform the necessary updates. The processor with the skewed ViewCach maintains a primary *incremental update log* ∂L which is the differential of the update log corresponding to the record values in the physical database [Roussopoulos, September 1991]. Whenever an insert or delete takes place in any partition in the ViewCache of the original processor, the updated ∂L with the statements of inserts or deletes is sent to the processor which has the replicated partitions that needs to be updated (Figure 6). The processor acting as the receiver, checks the incoming ∂L for the inserts or deletes; updates the records in the physical database corresponding to the relation R_i or R_j and also updates the corresponding tuples in the ViewCache partitions replicated in it.

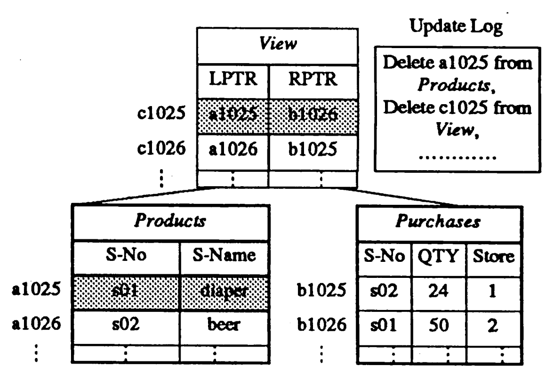


Figure 6: The updation procedure in a processor. The light shaded boxes indicate the partitions in ViewCache and the corresponding records in the tables to be deleted. The update log with the statements for the insertion or deletion.

3.2. Clustering of Data for Distribution

Considering a sub-graph $G' \equiv (V', E', w')$ 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.

A *cluster* in a graph G', is a single black vertex and 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.

The clustering is done in a way to enable the sender to reduce the I/O cost for repeated reads. For a cluster, several white-black joins related to the same black vertex, need to be performed during the traversal of its edges incurring I/O cost. So if such clusters are sent in steps, to other processors for replication, then the unbalanced processor may save some I/O in each such step. Clusters in small numbers, sent in several iterations lead to a cumulative saving of the I/O, instead of a one time large saving where one has to pay for the high cost of distribution of huge amount of data. The algorithm for the formation of clusters in G' is given in the following section.

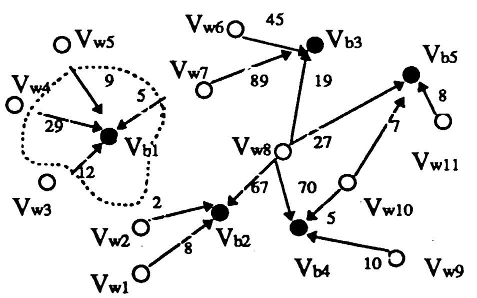


Figure 7. A cluster in G consisting of a black vertex connected by the edges to several white vertices.

3.2.1. Algorithm for Cluster Selection

<u>Step 1.</u> 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.

<u>Step 2.</u> Eliminate the black vertex v_b and its associated edges to get a sub-graph. This forms a cluster to send.

<u>Step 3.</u> Subtract the degrees of v_w and that of all the white vertices connected to v_b, each by one.

<u>Step 4.</u> Again start from v_w.

<u>Step 5.</u> Continue from Step 1 in the remaining sub-graph, till the degree of v_w in Step 1 becomes zero. Then eliminate v_w.

<u>Step 6.</u> Again look for a white vertex with the minimum degree and continue from Step 1.

3.2.2. Cost Analysis

In this section we will analyze the cumulative advantage to successive users with respect to the I/O costs, due to cluster sending. The equations in the ensuing paragraphs refer to a processor in location *(i,j)*, considering a 2-d network of processors.

Let us consider the sub-graph $G' \equiv (V', E', w')$ to be traversed by a processor *(i,j)*. The processor generates several clusters following the steps of algorithm 3.2.1 and sends each of them to other processors for replication. Now, we will find the order of I/O saving by a processor for sending one cluster consisting of 1 black vertex and say, n_k white vertices.

<u>Lemma 3.2.2.1.</u>

The average saving in the I/O cost in the static set of edges for a processor by replication of a cluster consisting of one black vertex and 1,2,...,k white vertices, is given by the order of

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

[3.2.2.1]

where, d_w^{av}, d_b^{av} *are respectively the average degrees of white and black vertices in* $G' \equiv (V', E', w')$*;* Φ *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. [3.2.2.1] gives the order of edges contributed to the dynamic domain by replication.*

<u>Proof.</u>

A cluster having *k* white vertices connected to a single black vertex, implies that the cluster has *k* edges connected to the black vertex.

Each of the *k* white vertices may be again connected to several other black vertices by edges. So, on sending away the above cluster, the processor will save I/O corresponding to the degree of the black vertex present in the cluster. We will next find the average I/O saving corresponding to all the k white vertices in the cluster.

Let us first consider one white vertex out of k white vertices. The probability of *one* out of all the edges of the first white vertex to remain inside S is

$$\left\{\frac{\frac{1}{b-1}\sum_{k=1}^{b-1}(b-k)k}{|E'|}\right\} d_w^{av} = \frac{\Phi}{|E'|} d_w^{av},$$

by the given definition.

Thus, the probability of all the edges of the first white vertex to remain inside S, is $\frac{\Phi}{|E'|}\left(d_w^{av}\right)^2$.

So, $1-\frac{\Phi}{|E'|}\left(d_w^{av}\right)^2$ is the probability that *none* of the edges of the first white vertex is inside S.

Let us now consider surrounding black vertices connected tothe first white vertex. If all of them are not present in S, we pay for 2 I/Os. Else we pay for 1 I/O, when the first white vertex is brought into S.

The probability that a black vertex is not present in S, is

$$1-\frac{\Phi}{|E'|}\left(d_b^{av}\right)^2$$

Thus, the probability that all black vertices connected to the first white vertex are *not* present in S, is

$$\left(1-\frac{\Phi}{|E'|}\left(d_b^{av}\right)^2\right)^{d_w^{av}}$$

Thus, the probability of the first white vertex and all black vertices connected to it, *not* to be present in S, is

$$\left(1-\frac{\Phi}{|E'|}\left(d_w^{av}\right)^2\right)\left(1-\frac{\Phi}{|E'|}\left(d_b^{av}\right)^2\right)^{d_w^{av}}$$

$= \Psi$, say.

The probability of the first white vertex *not* to be in S and one or more of its black vertices to be present in S, is

$$\left(1-\frac{\Phi}{|E'|}\left(d_w^{av}\right)^2\right)\left(1-\left\{1-\frac{\Phi}{|E'|}\left(d_b^{av}\right)^2\right\}^{d_w^{av}}\right)$$

$= \omega$, say.

Now, the average I/O saving for the first white vertex in the cluster is of the order of

$$\frac{1}{2}[2\times\psi+\omega]$$

which after simplification yields,

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

Considering all the white vertices, $1,2,\ldots k$ in the cluster, the average I/O saving for sending the cluster away is of the order of

$$\frac{d_w^{av}d_b^{av}}{2}\left(1-\frac{\Phi}{|E'|}\left(d_w^{av}\right)^2\right)\left(\frac{1}{d_w^{av}}+1-\frac{\Phi}{|E'|}\left(d_b^{av}\right)^2\right). \quad \blacksquare$$

Lemma 3.2.2.2.

The average I/O saving in the static set of a processor after replications by the first user, is of the order of

$$\frac{Kl^{(1)}}{2}\left\{K_3-\frac{(K_3K_1+K_2)}{|E'|}\right\}$$

where, $l^{(1)}$ *is the number of iterations of sending the clusters; and* K, K_1, K_2, K_3 *are defined as follows,*

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

$K_3 = 1+\frac{1}{d_w^{av}}$. *This gives the order of edges contributed to the dynamic domain by the first user.*

Proof.

Let us consider the case for the first user where $l^{(1)}$ is the total number of iterations, a processor sends a cluster. Let these clusters contain $n_1^{(1)}$ white vertices in the first iteration, $n_2^{(1)}$ white vertices in the second iteration, and so on, till it reaches the $l^{(1)}$ th iteration when it sends a cluster of $n_{l^{(1)}}^{(1)}$ white vertices.

By Lemma 3.2.2.1 and using the definitions above, the average saving in the I/O cost for sending away clusters with $n_1^{(1)}$, $n_2^{(1)}$, ..., $n_{l^{(1)}}^{(1)}$ white vertices can be written as the order of,

$$\frac{K}{2}\left\{K_3 l^{(1)} - (K_3K_1+K_2)\left[\frac{1}{|E|}+\frac{1}{|E|-n_1^{(1)}} + \cdot\cdot + \frac{1}{|E|-(n_1^{(1)}+\cdot\cdot+n_{l^{(1)}-1}^{(1)})}\right] + K_1K_2\left[\frac{1}{|E|^2}+\frac{1}{\left(|E|-n_1^{(1)}\right)^2} + \cdot\cdot + \frac{1}{\left(|E|-(n_1^{(1)}+\cdot\cdot+n_{l^{(1)}-1}^{(1)})\right)^2}\right]\right\}$$

$$= \frac{K}{2}\left\{K_3 l^{(1)} - (K_3K_1+K_2)\left[\frac{1}{|E|}+\frac{1}{|E|\left(1-\frac{n_1^{(1)}}{|E|}\right)} + \cdot\cdot + \frac{1}{|E|\left(1-\frac{(n_1^{(1)}+\cdot\cdot+n_{l^{(1)}-1}^{(1)})}{|E|}\right)}\right] + K_1K_2\left[\frac{1}{|E|^2}+\frac{1}{|E|^2\left(1-\frac{n_1^{(1)}}{|E|}\right)^2}+\frac{1}{|E|^2\left(1-\frac{n_1^{(1)}+\cdot\cdot+n_{l^{(1)}-1}^{(1)}}{|E|}\right)^2}\right]\right\}$$

$$= \frac{K}{2}\left\{K_3 l^{(1)} - (K_3K_1+K_2)\left[\frac{1}{|E|}+\frac{\left(1+\frac{n_1^{(1)}}{|E|}\right)}{|E|} + \cdot\cdot + \frac{\left(1-\frac{(n_1^{(1)}+\cdot\cdot+n_{l^{(1)}-1}^{(1)})}{|E|}\right)}{|E|}\right] + K_1K_2\left[\frac{1}{|E|^2}+\frac{\left(1-\frac{n_1^{(1)}}{|E|}\right)^2}{|E|^2} + \cdot\cdot + \frac{\left(1-\frac{n_1^{(1)}+\cdot\cdot+n_{l^{(1)}-1}^{(1)}}{|E|}\right)^2}{|E|^2}\right]\right\}$$

which after neglecting square and cubic terms in the denominators of the binomial expansions involving $|E|$ and further simplifying yields,

$$\frac{Kl^{(1)}}{2}\left\{K_3 - \frac{(K_3K_1+K_2)}{|E|}\right\}. \qquad [3.2.2.2]$$

[3.2.2.2] denotes the order of the average I/O saving by the first user. ■

Corollary 3.2.2.3. *The average cumulative I/O saving in the static set for m users can be written as the order of*

$$\frac{Kl^{av}m}{2}\left\{K_3 - \frac{(K_3K_1+K_2)}{|E|}\right\}$$

where, l^{av} is the average over the number of clusters sent by all m users, assuming $|E|$ large. This gives the order of edges contributed to the dynamic domain by m users.

Proof.

Considering l^{av} to be the average number of clusters and u_i^{av} to be the average number of white vertices in the clusters, sent by the ith user, $i = 1,2,\ldots m$; the average I/O saving by the first user, using Lemma 3.2.2.2, can be written as,

$$\frac{Kl^{av}}{2}\left\{K_3 - \frac{(K_3K_1+K_2)}{|E|}\right\}.$$

For the second user, the average I/O saving will be,

$$\frac{Kl^{av}}{2}\left\{K_3 - \frac{(K_3K_1+K_2)}{|E|-u_1^{av}}\right\}$$

Similarly, for the mth. user, the average cumulative I/O saving will be,

$$\frac{Kl^{av}}{2}\left\{K_3 - (K_3K_1+K_2)\frac{1}{|E|-u_1^{av}} + \cdot\cdot + \frac{1}{|E|-(u_1^{av}+\cdot\cdot+u_{m-1}^{av})}\right\}$$

where we can again neglect the terms involving $|E|$ in the denominators of the binomial expansions to get the average cumulative I/O saving by all m users, as

$$\frac{Kl^{av}m}{2}\left\{K_3 - \frac{(K_3K_1+K_2)}{|E|}\right\}.$$ ■

3.3. The Distribution

Let us again consider the *first* user. The number of edges as a variable, that a processor (*i,j*) has after the $k^{(1)}$ th send, is given by:

$$x'(i,j) = E' - (n_1^{(1)} + n_2^{(1)} +..+ n_{k^{(1)}}^{(1)}) \qquad [3.3.1]$$

considering (V', E', w') is the sub-graph in processor (*i,j*) and that it has sent $(n_1^{(1)} + n_2^{(1)} +..+ n_{k^{(1)}}^{(1)})$ edges connecting white vertices along with one black vertex in each of $k^{(1)}$ iterations. This is the *static set* of edges owned by the processor (*i,j*). It has also the replicated data packets from other nodes, on which it will have to operate dynamically when task sharing is necessary. Thus the total number of edges a processor (*i,j*) has after $k^{(1)}$ iteration of its sends, is

$$x(i,j) = x'(i,j) + \sum_{p,q} D(p,q) \qquad [3.3.2]$$

where, $D(p,q)$ is the number of edges in the random data packets arriving due to Binormal distribution to the node (*i,j*) from other nodes for replication also termed as the *dynamic set* of edges owned by the processor.

It is important to mention here, that by sending away clusters for replication, a processor (*i,j*) sends away the weights along with the edges, i.e., it sends away

$$\Psi(i,j) = \sum_{p=1}^{k^{(1)}} \sum_{l=1}^{n_p^{(1)}} w'(l)^{(1)} \qquad [3.3.3]$$

weights for each of the edges in all the $n_{k^{(1)}}^{(1)}$ clusters.

As we explained earlier, a variable *m* typifies the size of a data packet which is at maximum one page. A data packet "*migrates*" to a specific node for replication, following Binormal distribution whose corresponding density function *g(x,y)* with mean zero and standard deviation σ, 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 8). We are using this function to ensure a uniform spread of the data in the x and y directions of 2-d network of processors (extensible 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 [Hiller and Liberian, 1974]. We then generate two uniform random numbers in the interval [0,1] from this Binormal distribution by using a standard routine [William, 1994]. The generated pair of normally distributed independent random numbers, will give the random coordinates of the processor to which the data packet has been distributed for replication. In this way, we carry out several steps of Monte-Carlo simulation for each node on packets of data, and draw a pair of random deviates from a Binormal distribution with mean zero and standard deviation (S.D) σ. This pair of random deviates as discussed above, determines the location of the replicated data packet in the network of nodes. Here, σ is a processor-specific parameter.

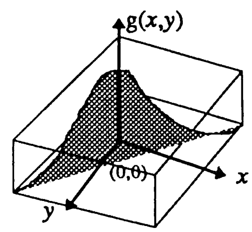


Figure 8: The normal distribution of data from a single node to the neighbouring nodes in the 2-d space for replication.

The total number of iterations for sending data packets for the first user as discussed earlier, is $k^{(1)}$. Now, repeating a Monte-Carlo simulation $k^{(1)}$ times for the processor (*i,j*), we draw a pair of random deviates from a Binormal distribution with mean zero and SD σ. The parameter σ is node-specific, and describes its characteristics related to the number of vertices it holds. The pair of random deviates determines the location of each of the $k^{(1)}$ data packets in the 2-d network, and thus contributes to $D(p,q)$. 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.

4. Run-Time Load Balancing

Run-time load balancing being very expensive in current parallel data servers, is avoided. However, the replication technique described above can solve this problem. After the normal distribution of the clusters is over, a processor has its set of edges and vertices for traversal as well as portions of replicated edges and vertices from other processors, for traversal if necessary.

Definition 4.1. The set of edges owned by a processor after sending away clusters for replication, is the *static set* of edges, that it has to traverse. The set of edges replicated from other processors due to Binormal distribution, is the *dynamic set* of edges, a processor has to traverse if necessary, during load balancing.

The static set is local to a processor, whereas, the dynamic set is transferable to other processors during task load sharing. The dynamic sets from all processors form the *dynamic domain* of the whole network. In our subsequent discussions, we will be concerned about the cost analysis of the dynamic domain of the network.

At run time, based on agreements between the processors, task loads 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. By "task load" here, we mean a subset of the dynamic set of edges, a processor needs to traverse at an optimal cost. Each processor minimizes its local cost function by keeping the task load below a certain specified threshold. It calculates the difference between the cost it incurred on its dynamic set in the previous time step with the estimated cost on its static set at the current step in making a probabilistic decision (equation 4.1.1). Thus the task loads in the static and dynamic sets of a processor, are interdependent in making a probabilistic decision. With such cooperation of all the processors in the network, the global cost function follows a gradient descent towards minimization [Smolensky and Riley, 1984].

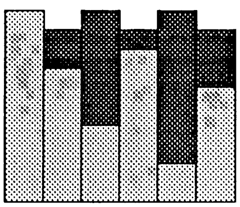


Figure 9: The static sets and contributions to the dynamic domain, by processors P1,...,P6 at a time step. The light shaded portions form the static sets local to a processor and the deep shaded ones are the portions of the dynamic domain, shared by processors for load balancing. P1 has no dynamic contribution at the above time step, whereas, P5 has a maximum dynamic contribution due to its low static load.

Whenever a processor faces a situation of task load imbalance, it *asynchronously* sends a message to other processors. All processors check for any pending messages for receipt. The processors having *the* replicated data packet, corresponding to the edges, the sender needs to traverse, will check the contents of the message for the satisfaction of all the prescribed hypotheses regarding its own load, etc. It will check against a threshold, the task load fraction in terms of the number of edges it will have to traverse. If less than the threshold, it will agree to traverse some or all of the replicated edges and will send an asynchronous message back regarding such a mutual agreement. This mutual agreement follows all the conditions for the local minimization of cost tending to the global minimization.

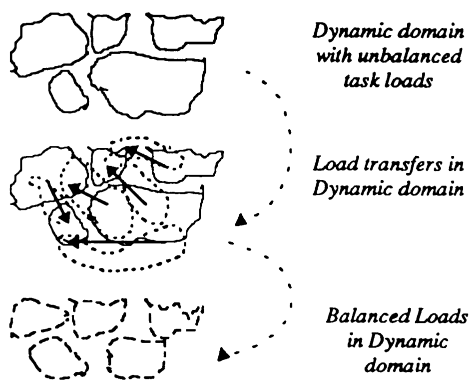


Figure 10: Run-time load transfer between processors with unbalanced dynamic domains in the network.

4.1. Cost Analysis for Traversal in the Dynamic Domain

In this section, we provide a cost model for the dynamic domain of the network and how such a situation is handled with an optimal cost. This model will be different if we consider only the static set of edges, a processor has to traverse. The edges in the static set are fixed for a processor and cannot be shared. 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.

Dynamics are defined as changes over time. Since the total traversal of the edges of G can be discretized into several time steps and the number of steps taken for completion of the traversal, changes with load transfer in the dynamic domain of the network, we describe a discrete, dynamical model for the local and global cost incurred in the network at a particular time step. The task load transfer is based on mutual agreements of the processors at run time. The decisions are made in a way such that the overall cost of the traversal of edges decreases with time. In the following paragraphs, we will present a formal framework for such decision making.

The probability of the processor (i,j) to become active is proportional to the total cost it contributes to the network [Hopfield, 1982]. Here, by saying that a processor is *active*, we mean that it is actively taking part in receiving an extra load from some other processor. A processor will probabilistically decide on such task load sharing, by comparing the estimated cost of its static load at the current step, with the cost, it incurred in the previous time step for the dynamic domain. In the following lines, we describe how such a decision is made. By assumption, the cost function is additive under network decomposition. In such a network what is required of the probability assigned to a state of the network, is nothing but the product of the probabilities assigned to the states of the component networks. Thus adding the costs of the components' states should correspond to multiplying the probabilities of the components' states. It is a mathematical fact that the only continuous functions f that map addition into multiplication, i.e., $f(x+y) = f(x)f(y)$ are the exponential functions $f(x) = a^x$ for some positive number a. Equivalently, these functions can be written $f(x) = e^x$.

In the present case, the proportionality between the probability of the processor (i,j) to become active and the total cost it contributes locally to the network at time t, can be written as,

$$\text{prob} \propto \exp(-cost)$$

The negative sign indicates that lower the cost of its state in the network, higher is the probability of (i,j) to activate in receiving task load from other processors.

Now, a processor at (i,j) has an earlier estimated cost for edge traversals in the static set depending on the weights of the edges and on the vertices whether or not, they are in the working set 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 (i,j) must thus compute locally,

Likelihood_Ratio

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

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 probability for the processor (i,j) to activate will be greater, i.e., (i,j) will more likely activate in receiving task loads in the dynamic domain.

In the following paragraphs, we develop the theory for the local and global cost functions in the dynamic domain.

4.1.1. The Global Cost for the Dynamic Domain

The *global cost* is the *sum total* of the I/O cost, the CPU cost and the parallel communication cost incurred by the processors, for traversal of the edges in the dynamic domain of the network.

Now, we will give a mathematical formulation of the dynamic load balancing scenario and decision making in terms of the agreements between incoming message vectors for task sharing and feature vectors of the processors. 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 in the form of vectors requesting for task loads to be shared. These vectors are mapped onto an observation space.

An *observation space* is a N-dimensional vector space where any N-dimensional vector mapped onto it, is considered to be point.

A block *message vector* $\mathbf{r}(t)$ represented as a $(1 \times nk)$ block vector having k vector blocks with n elements each, corresponds to an incoming message vector mapped onto the observation space at time step t. These k sub-blocks are the message vectors from k processors asking for load sharing. Each element in these k sub-blocks represents information regarding its 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 on these edges, the vertices corresponding to the edges, etc. Thus $\mathbf{r}$ can be represented as,

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

where, $r_0(t), r_1(t), r_2(t), \ldots, r_{k-1}(t)$ are the k message vector blocks.

A block *feature vector* $\mathbf{f}_{ij}(t)$ for a processor (i,j) is also represented as a $(1\times nk)$ block vector having k vector blocks of n elements each. These k sub-blocks represent features that need to be matched with the k message vectors in the observation space. Each element in these k sub-blocks convey 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 list of all task loads $\{q^p_{i,j}(r_p(t),t)\}$, messaged to (i,j) via all $r_p(t)$, $p = 0,\ldots,(k\text{-}1)$, in the observation space, comprises the input vector $\mathbf{q}_{i,j}(\mathbf{r}(t),t)$. It represents the number of edges and vertices in the dynamic domain, a processor needs to share at that particular time step.

A decision rule is applied by the processor (i,j), by associating its own features with the elements of the incoming message vectors $r_0(t), r_1(t), r_2(t), \ldots, r_{k-1}(t)$. Each $f^p_{i,j}(t)$ is compared with each of the message vectors $r_p(t)$, $p = 0,\ldots,(k\text{-}1)$. It checks whether the incoming message in each of the message vector matches the features in its own feature vector or not regarding send or receive of task loads. It labels 1,-1 or 0 to the association of its feature vector with the message vectors. "0" implies no match at all, "1" implies that there is a match and a task load is possibly to be received from the source associated with the message vector. Similarly, "-1" implies that there is a possibility of sending a task load to the receiver specified by the message vector.

Thus, for $p = 0,\ldots,(k\text{-}1)$,

$$label(r_p(t), f^p_{i,j}(t)) = 0,\ 1,\ \text{or}\ -1.$$

An agreement occurs between a processor (i,j) and another processor in the network, if $f^p_{i,j}(t)$ matches any $r_p(t)$, $p = 0,\ldots,(k\text{-}1)$. The processor with which this agreement takes place, is the one specified in $r_p(t)$ and the communication for this agreement involves a message of size m, say. It is denoted by $m^p_{i,j}(r_p(t),t)$. *It is important to mention here, that the communications will only deal with messages regarding the task load transfer and no true data transfer will actually take place.* These communications are random, depending on the run-time situation of the system. The load sharing scenario and the random message vectors r_p with load sharing information, will all be consistent with the initial Binormal distribution of data. Thus, we may say that the run-time sharing of tasks will follow the trend of the Binormal distribution.

Let us denote the state of the network by the pair $(\mathbf{r}(t),t)$ consisting of the incoming block message vector $\mathbf{r}$ mapped onto the mathematically so-called observation space, and the time t. The cost function assigns a real number to each state. The cost function has as time dependent parameters the set of the feature vectors $\{\mathbf{f}_{i,j}(t)\}$ associated with each (i,j) and we call this the feature base $\mathbf{F}$.

The cost function considered should be additive under the decomposition of the system, i.e., if the network is partitioned into two unconnected networks, the total cost of the network will be the sum of the costs of the two sub-networks.

Let the global cost function for the dynamic domain to be covered by the network, as a contribution from all (i,j) at time t, be given by

$$\begin{aligned} C_{\mathbf{F}}(\mathbf{r}(t),t) &= \sum_{i,j} \{C_{comm}(\mathbf{r}(t),\mathbf{f}_{i,j}(t),\mathbf{m}_{i,j}(\mathbf{r}(t),t)) \\ &+ C_{I/O}(\mathbf{r}(t),\mathbf{f}_{i,j}(t),\mathbf{q}_{i,j}(\mathbf{r}(t),t)) \\ &+ C_{comp}(\mathbf{r}(t),\mathbf{f}_{i,j}(t),\mathbf{w}_{i,j}(\mathbf{q}_{i,j}(\mathbf{r}(t),t)))\} \end{aligned} \qquad [4.1.1.1]$$

where $C_{comm}(\mathbf{r}(t),\mathbf{f}_{i,j}(t),\mathbf{m}_{i,j}(\mathbf{r}(t),t))$ is the cost contributed by the processor (i,j) due to agreements with the adjacent processors through communication of messages of size m. $C_{I/O}(\mathbf{r}(t),\mathbf{f}_{i,j}(t),\mathbf{q}_{i,j}(\mathbf{r}(t),t))$ is the I/O cost and $C_{comp}(\mathbf{r}(t),\mathbf{f}_{i,j}(t),\mathbf{w}_{i,j}(\mathbf{q}_{i,j}(\mathbf{r}(t),t)))$ is the computational cost incurred due to the changing task load of edge traversals during load balancing. Here, computational cost refers to the cost of joining the tuples specified by the weight of an edge.

4.1.2. Calculation of the Cost due to Agreements through Communication for the Dynamic Domain

Now the total contribution to the communication cost by (i,j) due to agreements is given by,

$$C_{comm}(\mathbf{r}(t),\mathbf{f}_{i,j}(t),\mathbf{m}_{i,j}(\mathbf{r}(t),t)) = \alpha \sum_{p=0}^{k-1} \left| label(r_p(t), f^p_{i,j}(t)) \right| m^p_{i,j}(r_p(t),t)) \qquad [4.1.2.1]$$

where m is the message size communicated during every task load transfer and α is the constant of proportionality. The "mod" value implies that cost will always be additive, whether the label is 1 or -1 for a receive or send respectively.

4.1.3. Calculation of the I/O Cost due to Agreements for the Dynamic Domain

The *net* input of task load in the dynamic domain, to processor (i,j) is given by

$$I_v(\mathbf{r}(t),\mathbf{f}_{i,j}(t),\mathbf{q}_{i,j}(\mathbf{r}(t),t))$$

$$=\sum_{p=0}^{k-1} label(r_p(t), f_{i,j}^p(t)) q_{i,j}^p(r_p(t),t)$$

due to agreements of $r_p(t)$ and $f_{i,j}^p(t)$ on transferring the critical fraction of task loads, $p = 0,\ldots,(k-1)$.

The I/O cost $C_{I/O}(\mathbf{r}(t),\mathbf{f}_{i,j}(t),\mathbf{q}_{i,j}(\mathbf{r}(t),t))$ will be of the order similar to [2.2.1].

So, provided $I_v(\cdot)$ is positive, we may write,

$$C_{I/O}(\mathbf{r}(t),\mathbf{f}_{i,j}(t),\mathbf{q}_{i,j}(\mathbf{r}(t),t))$$

$$= O(I_v(\mathbf{r}(t),\mathbf{f}(i,j),\mathbf{q}(i,j,\mathbf{r}(t),t))) \qquad [4.1.3.1].$$

4.1.4. Calculation of the Computational Cost for the Dynamic Domain

The computational cost for traversing an edge is proportional to the weight on the edge. The weights are different for different edges. Since the number of edges in the dynamic domain that a processor has to traverse, varies according to the agreements at run time during load balancing, the computational cost will also vary according to the weights on the edges. Now, the net weight on varying edges due to agreements can be written as,

$$W'(\mathbf{r}(t),\mathbf{f}_{i,j}(t),\mathbf{q}_{i,j}(\mathbf{r}(t),t))$$

$$=\sum_{p=0}^{k-1} label(r_p(t), f_{i,j}^p(t)) w'(q_{i,j}^p)$$

due to agreements of $r_p(t)$ and $f_{i,j}^p(t)$ on transferring the critical fraction of task loads, $p = 0,\ldots,(k-1)$.

The computational cost is proportional to the weight of computation and so, we may write,

$$C_{comp}(\mathbf{r}(t),\mathbf{f}_{i,j}(t),\mathbf{q}_{i,j}(\mathbf{r}(t),t))$$

$$= \eta W'(\mathbf{r}(t),\mathbf{f}_{i,j}(t),\mathbf{q}_{i,j}(\mathbf{r}(t),t)) \qquad [4.1.4.1]$$

where, η is a constant of proportionality.

The three cost terms in [4.1.2.1], [4.1.3.1] and [4.1.4.1] from all (i,j) contribute to the global cost $C_{\mathbf{F}}(\mathbf{r}(t),t)$ in [4.1.1.1]. $C_{\mathbf{F}}(\mathbf{r}(t),t)$ will be minimized if the total contribution to the communication, computation and I/O cost from each processor (i,j) is minimized.

Lemma 4.1.

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

$$\sum_{P} \{Log_e(Likelihood\_Ratio_P)$$

$$- Log_e(Likelihood\_Ratio_{\min})\}$$

where, $Likelihood\_Ratio_{\min}$ is the minimum over all the $Likelihood\_Ratio_P$ in the network of P processors.

Proof. From the definition of $Likelihood\_Ratio_P$, it is clear that more the $Likelihood\_Ratio_P$, more is the probability of a processor to be active (which implies traversal of some dynamic edges). Mutual agreements between processors take place based on their values of Likelihood_Ratio and other quantities. So, it is natural that a processor with $Likelihood\_Ratio_{\min}$ will not take part in dynamic edge traversal. The sum over the differences between the logarithms of the two ratios above, relates directly to the cost involved in load sharing. This implies the number of dynamic edges for traversal at a particular time step which proves the above lemma. ■

4.2. The Threshold for Decision Making

A processor calculates the fraction of the total weighted cost incurred with respect to an estimated cost for each computational step. This cost is incurred on the basis of *agreements* of $f_p(i,j)$ with $r_p(t)$, as discussed earlier. A processor (i,j) thus calculates the weighted cost fraction as, wtd-cost-fraction =

$$= \frac{\left| \sum_{p=0}^{k-1} label(r_p(t), f_{i,j}^{P}(t)) q_{i,j}^{P}(r_p(t), t) \right|}{\textit{estimated}\ \text{cost}}$$

We use the "mod" value of the sum of all weighted agreements, to ensure that the cost fraction is always positive. This fraction should be less than a threshold for minimization. So, we update the fraction as,

critical-fraction

$$= \nu - \frac{\left| \sum_{p=0}^{k-1} label(r_p(t), f_{i,j}^{P}(t)) q_{i,j}^{P}(r_p(t), t) \right|}{\textit{estimated}\ \text{cost}} \qquad [4.2.1]$$

where ν is a threshold. Each processor decides to receive or send based on agreements, the critical fraction of the task load [Sarkar, June 1996]. It orients its own feature vector $\mathbf{f}_{i,j}(t)$ with the information whether it can send or receive the task loads.

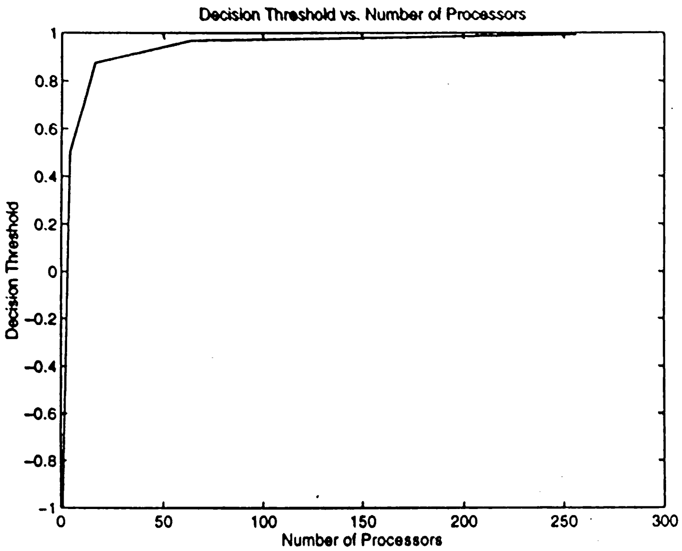


Figure 11: The best threshold as a function of the number of processors, *P*. As *P* increases, the threshold value limits to 1.

We may express the best threshold range (Figure 11) in terms of the number of processors, *P*. The theory has been developed in the thesis [Sarkar, June 1996] and the best threshold range for an optimal dynamic load balancing situation, for a network of *P* processors should lie within the range,

$$1 > \nu > 1 - \frac{2}{P} \qquad [4.2.2].$$

Theorem 4.3.

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

$$\begin{cases} T_{\max} + \frac{\mu}{P}\left\{\Gamma - \delta \sum_{P} (T_{\max} - T_P)\right\}, \\ \quad \textit{if}\ \Gamma > \delta \sum_{P} (T_{\max} - T_P), \\ T_{\max}, \qquad \textit{otherwise} \end{cases}$$

where, $T_{\max}$ is the maximum static cost associated with a processor out of P processors in the network, Γ is the total number of edges in the dynamic domain of the network, T_P is the static cost associated with each processor and μ, δ are constants of proportionality.

Proof.

Since the static and dynamic parallel executions are interleaved with each other, we will have to consider their effects on the total parallel time complexity. A processor traversing the maximum number of static edges will contribute to the maximum time complexity of the order of $T_{\max}$. Meantime, the traversal of all the edges in the dynamic domain may not be complete. The difference $(T_{\max} - T_P)$ for each processor is proportional to the number of edges in the dynamic domain that have been *more likely* traversed by it, in terms of agreements. This follows similar logical reasoning as in Lemma 4.1. This difference summed over all processors *P*, gives the total number of edges in the dynamic domain more likely traversed by all *P*. Since Γ is the total number of edges that exists in the dynamic domain of the network, the quantity $\left\{\Gamma - \delta \sum_{P} (T_{\max} - T_P)\right\}$ represents the dynamic task load that still remains (if there still remains at all) to be shared by all processors after they are done with their static sets and some portions of the dynamic domain. δ is the constant of proportionality for the number of edges traversed in parallel, within a given interval of time. This quantity divided by *P*, gives the average number of edges in the dynamic domain that need to traversed by a single processor. Thus $\frac{\mu}{P}\left\{\Gamma - \delta \sum_{P} (T_{\max} - T_P)\right\}$ is the time taken in parallel to traverse the remaining dynamic edges, where μ is again a constant of proportionality. Thus the total parallel run-time complexity to traverse the static and dynamic edges of the graph in a interleaved pattern, is given by the expression in the above theorem. ■

5. THE IMPLEMENTATION

This section presents some of the simulation results using the 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 the nodes for dynamic load balancing have been implemented on CM-5 using a program written in C++. The simulation was targeted towards experimenting with dynamic domain of a graph. We studied variations of threshold and its impact on decision making and hence load balancing and also studied the gradient descent in global cost.

In our implementation, we considered 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. We are now using a 2-d array of P processors consisting of $\sqrt{P}$ processors along one dimension and again $\sqrt{P}$ processors along the orthogonal dimension. Let $d0$ and $d1$ be the coordinates of a processor respectively in the X and Y dimensions. The value of $d0$ is specified as CMMD_self_address()/$\sqrt{P}$ and the value of $d1$ is specified as CMMD_self_address() % P, where the call CMMD_self_address() returns the processor identifier for the processor passed as an argument. The data partitions are then evenly loaded onto these P processors arranged in the above mentioned checkerboard fashion.

A Monte-Carlo simulation of the normal distribution of several data packets is then carried out by each processor in order to replicate the data, it owns.

Figure 13 shows that the best minima of the local costs for the processors arranged in a 2-d network are got using the threshold value within the range as described in equation 4.2.2. The processors carry on with the traversal of its static edges interleaved with the traversal of edges in the dynamic domain on the basis of agreements during load balancing and come out of the iterative loop with a minimized global cost after all the time steps are over (Figure 12).

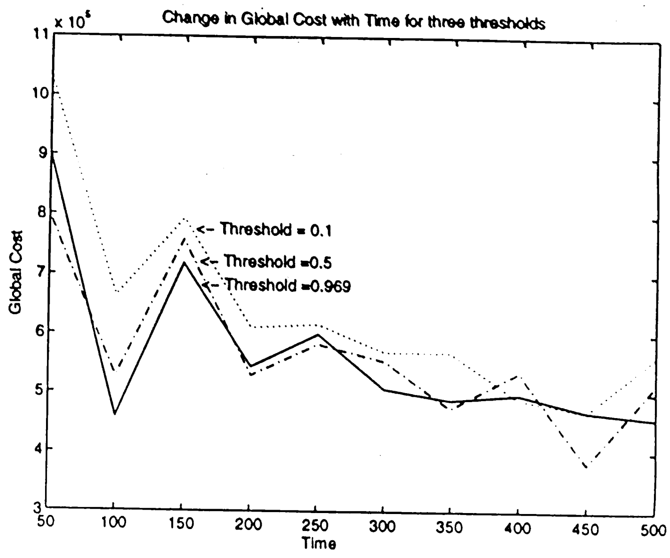


Figure 12: Change in the global cost with time for three different thresholds with $P = 64$, where P is the number of processors in the 2-d space.

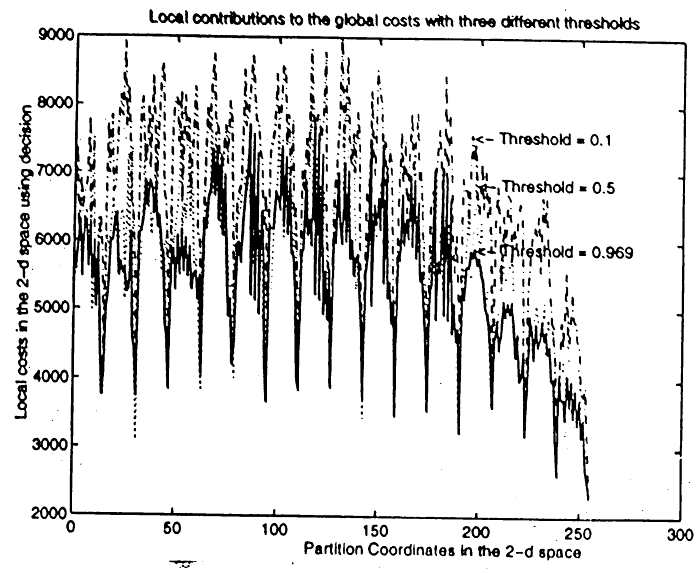


Figure 13: Local contributions (costs) along the 2-d network of processors for three different thresholds. Here, $P = 64$, where P is the number of processors in the 2-d space.

6. CONCLUSIONS

In this research, we concentrated on View materialization in a parallel environment. Since the current problem lies in the huge amount of data movement involved in dynamic load balancing, we proposed a technique for fractional data replication 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. 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. ACKNOWLEDGEMENTS

Minnesota Supercomputer Institute, for the machine time allocations on CM-5 in the primary stage;

Army High Performance Computing Research Center, Minnesota, for funding the CM-5 hours needed for the project for the rest of the period;

Institute for Mathematics and its Applications (IMA), University of Minnesota, for supporting this research with all their resources while hosting the High Performance Computing Program 1996-1997;

8. REFERENCES

Amir, A., and Roussopoulos, N. 1990. Optimal view caching. Information Systems, 15,2. 169-171.

Chacko, George K. 1971. Applied Statistics in Decision Making. New York: American Elsevier Publishing Company, Inc.

Feller, W. 1950, 1957. An Introduction to Probability Theory and Its Applications. Volume 1. New York: John Wiley and Sons, Inc.

Hiller, Fredrick.S and G.J. Lieberman. 1974. Operations Research.: San Francisco, California: Holden-Day, Inc.

Hopfield, J.J. 1982. Neural networks and physical systems with emergent collective computational abilities. Proceedings of the National Academy of Sciences, USA. 79. 2554-2558.

Kumar, V., A. Grama, A. Gupta, and G. Karypis. 1994. Introduction to Parallel Computing: Design and Analysis of Algorithms. California: Benjamin/Cummings Publishing Company, Inc.

Lehmann, E.L. 1959. Testing Statistical Hypotheses. New York: John Wiley and Sons, Inc.

McClelland, J.L., D.E. Rumelhart and the PDP Research Group. 1986. Parallel Distributed Processing. Volume 2. Cambridge: The MIT Press.

Neyman, J., and E.S. Pearson. 1933. On the Problem of the Most Efficient Tests of Statistical Hypotheses. Phil. Trans. Roy. Soc. London. A231. 289.

Roussopoulos, N. June 1982. View indexing in relational databases. ACM Transactions on Database Systems. 7,2. 258-290.

Roussopoulos, N. September 1991. An Incremental Access Method for ViewCache: Concept, Algorithms, and Cost Analysis. ACM Transactions on Database Systems. 16,3. 535-563.

Rumelhart, D.E., J.L. McClelland and the PDP Research Group. 1988. Parallel Distributed Processing. Volume 1. Cambridge: The MIT Press.

Sarkar, S. June 1996. Parallel Implementation of a spatial-temporal control system with decision based communication. Ph.D thesis, University of Minnesota.

Sarkar, S., and S.S. Sarkar. November 1996. Normal Distribution as a method for Data Replication in a Parallel Data Server. Technical Report, High Performance Computing Series, Institute of Mathematics and its Applications, University of Minnesota.

Sarkar, S., and S.S. Sarkar. 1996. Data Replication in a Read-Only Parallel Data Server for Dynamic Load Balancing. High Performance Computing (HPC), 1997, Atlanta, Georgia. Paper accepted.

Sarkar, S., and S.S. Sarkar. 1996. Dynamic Load Balancing in a Parallel Data Server. International Conference on Distributed Computing Systems (ICDCS), 1997, Baltimore, Maryland. Paper submitted.

Smolensky, P. 1984. The mathematical role of self-consistency in parallel computation. Proceedings of the Sixth Annual Conference of the Cognitive Science Society.

Smolensky, P., and M.S. Riley. 1984. Harmony theory: Problem solving, parallel cognitive models, and thermal physics. Tech. Rep. No. 8404. La Jolla: University of California, San Diego, Institute for Cognitive Science.

Thinking Machines Corporation. August 1992. The Connection Machine System: CMMD Reference Manual.

Ullman, Jeffrey.D. 1982. Principles of Database Systems. Computer Science Press.

William, H. 1994. Numerical Recipes in C: the art of scientific computing. 1994. Cambridge University Press.

Recent IMA Preprints

#	Author/s	Title

1354 **X. Chen**, Lorenz equations, Part III: Existence of hyperbolic sets
1355 **R. Abeyaratne, C. Chu & R.D. James**, Kinetics of materials with wiggly energies: Theory and application to the evolution of twinning microstructures in a Cu-Al-Ni shape memory alloy
1356 **C. Liu**, The Helmholtz equation on Lipschitz domains
1357 **G. Avalos & I. Lasiecka**, Exponential stability of a thermoelastic system without mechanical dissipation
1358 **R. Lipton**, Heat conduction in fine scale mixtures with interfacial contact resistance
1359 **V. Odisharia & J. Peradze**, Solvability of a nonlinear problem of Kirchhoff shell
1360 **P.J. Olver, G. Sapiro & A. Tannenbaum**, Affine invariant edge maps and active contours
1361 **R.D. James**, Hysteresis in phase transformations
1362 **A. Sei & W. Symes**, A note on consistency and adjointness for numerical schemes
1363 **A. Friedman & B. Hu**, Head-media interaction in magnetic recording
1364 **A. Friedman & J.J.L. Velázquez**, Time-dependent coating flows in a strip, part I: The linearized problem
1365 **X. Ren & M. Winter**, Young measures in a nonlocal phase transition problem
1366 **K. Bhattacharya & R.V. Kohn**, Elastic energy minimization and the recoverable strains of polycrystalline shape-memory materials
1367 **G.A. Chechkin**, Operator pencil and homogenization in the problem of vibration of fluid in a vessel with a fine net on the surface
1368 **M. Carme Calderer & B. Mukherjee**, On Poiseuille flow of liquid crystals
1369 **M.A. Pinsky & M.E. Taylor**, Pointwise Fourier inversion: A wave equation approach
1370 **D. Brandon & R.C. Rogers**, Order parameter models of elastic bars and precursor oscillations
1371 **H.A. Levine & B.D. Sleeman**, A system of reaction diffusion equations arising in the theory of reinforced random walks
1372 **B. Cockburn & P.-A. Gremaud**, A priori error estimates for numerical methods for scalar conservation laws. Part II: Flux-splitting monotone schemes on irregular Cartesian grids
1373 **B. Li & M. Luskin**, Finite element analysis of microstructure for the cubic to tetragonal transformation
1374 **M. Luskin**, On the computation of crystalline microstructure
1375 **J.P. Matos**, On gradient young measures supported on a point and a well
1376 **M. Nitsche**, Scaling properties of vortex ring formation at a circular tube opening
1377 **J.L. Bona & Y.A. Li**, Decay and analyticity of solitary waves
1378 **V. Isakov**, On uniqueness in a lateral cauchy problem with multiple characteristics
1379 **M.A. Kouritzin**, Averaging for fundamental solutions of parabolic equations
1380 **T. Aktosun, M. Klaus & C. van der Mee**, Integral equation methods for the inverse problem with discontinuous wavespeed
1381 **P. Morin & R.D. Spies**, Convergent spectral approximations for the thermomechanical processes in shape memory allows
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 $H(\mathrm{div})$ 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 polycrystallin martensitic solids
1388 **M. Fila & H.A. Levine**, On critical exponents for a semilinear parabolic system coupled in an equation and a boundary condition
1389 **J.M. Berg, W.G. Frazier, A. Chaudhary, & S.S. Banda**, Optimal open-loop ram velocity profiles for isothermal forging: A variational approach
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. Roblin**, 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 aprtitioning of potentials in one-dimensional Schrödinger-type equations
1407 **C.-S. Man & R. Paroni**, On the separation of stress-induced and texture-induced birefringence in acoustoelasticity
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**, Nonconforming 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. Friesecke**, 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 I. Compactons and Peakons II. Complex Analytic Behavior III. Convergence to Non-Analytic Solutions
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
1427 **X. Chen, C. Huang & J. Zhao**, A nonlinear parabolic equation modeling surfactant diffusion
1428 **A. Friedman & B. Hu**, Optimal control of chemical vapor deposition reactor
1429 **A. Friedman & B. Hu**, A non-stationary multi-scale oscillating free boundary for the Laplace and heat equations
1430 **X. Chen**, Existence, uniqueness, and asymptotic stability of traveling waves in nonlocal evolution equations
1431 **J. Yong**, Finding adapted solutions of forward-backward stochastic differential equations – Methods of continuation
1432 **J. Yong**, Linear forward-backward stochastic differential equations
1433 **D.A. Dawson & M.A. Kouritzin**, Invariance principles for parabolic equations with random coefficients
1434 **R. Lipton**, Energy minimizing configurations for mixtures of two imperfectly bonded conductors
1435 **D.C. Dobson & F. Santosa**, Nondestructive evaluation of plates using Eddy current methods
1436 **W. Littman & B. Liu**, On the spectral properties and stabilization of acoustic flow
1437 **S. Sarkar & S. Sundar Sarkar**, Normal distribution as a method for data replication in a parallel data server
1438 **S. Sarkar & S. Sundar Sarkar**, Parallel view materialization with dynamic load balancing: A graph theoretic approach
1439 **S. Sarkar & S. Sundar Sarkar**, Internet and relational databases in a multi-tier client/server model
1440 **J. Liang & S. Subramaniam**, Numerical computing of molecular electrostatics through boundary integral equations
1441 **J. Wu**, Inviscid limits and regularity estimates for the solutions of the 2-D dissipative quasi-geostrophic equations
1442 **P. Constantin & J. Wu**, Statistical solutions of the Navier-Stokes equations on the phase space of vorticity and the inviscid limits
1443 **M.A. Kouritzin**, Stochastic processes and perturbation problems defined by parabolic equations with a small parameter
1444 **M.A. Kouritzin**, Approximations for singularly perturbed parabolic equations of arbitrary order
1445 **A. Novick-Cohen** Triple junction motion for Allen-Cahn/Cahn-Hilliard systems
1446 **P. Klouček**, Approximations of the laminated microstructures
1447 **S. Sarkar & S.S. Sarkar**, A graph theoretic approach for parallel view materialization with dynamic load balancing
1448 **S. Chawla**, A minmax problem for parabolic systems with competitive interactions
1449 **B. Luong & F. Santosa**, Quantitative imaging of corrosion in plates by Eddy current methods
1450 **R. Jordan & B. Turkington**, Ideal magnetofluid turbulence in two dimensions
1451 **M. Fels & P.J. Olver**, Moving coframes. I. A practical algorithm
1452 **S.Y. Maliassov**, On the Schwarz alternating method for eigenvalue problems