Volume 11 Number 4 July 1996

Executive Editor Jeremy Thompson
Sub Editor Brenda Devers

Special issue: Massively Parallel Computing
Guest Editor: Giacomo R Sechi, CNR, Istituto di Fisica Cosmica, Milan, Italy

Performance comparison of interprocessor communication schemes for a hashing technique on the Connection Machine
Z S KHAN AND E KWATNY 189

Load balancing in a massively parallel semantic database
N RISHIE, A SHAPOSHNIKOV AND S GRAHAM 195

RETRAN: a recurrent paradigm for data-parallel computing
A V SHAFARENKO 201

The Modular Expandable Multiprocessor System MEMSY
M DAL CIN, H HESSENAUER AND W HOHL 211

Problems on routing bounded distance assignments in hypercubes
N BAGHERZADEH AND M DOWD 221

Divide-and-conquer minimal-cut bisectioning of task graphs
S LOR, H SHEN AND P MIAHESHWARI 227

Fast algorithm for data exchange in reconfigurable tree structures
S SRINIVAS AND N N BISWAS 235

Scheduling, timing and intractability in massively parallel systems
G NEMETH 245
Load balancing in a massively parallel semantic database

Naphtali Rishie, Artyom Shaposhnikov and Scott Graham
School of Computer Science, Florida International University, University Park, Miami, FL 33199, USA. Email: rishen@fiu.edu

We are developing a massively parallel semantic database machine. Our basic semantic storage structure ensures balanced load for most parts of the database. The load to the other parts of the database is kept balanced by a heuristic algorithm which repartitions data among processors in our database machine as necessary to produce a more evenly balanced load. We present one inexpensive, dynamic load balancing method together with a fault-tolerant data transfer policy that will be used to transfer the repartitioned data in a way transparent to the users of the database.

Keywords: DBMS, massive parallelism, semantic data models, load balancing, database machine

1. INTRODUCTION

Database management systems are emerging as prime targets for enhancement through parallelism. In order for parallel database machines to be efficient, the processors in the system must have comparable load. A massively parallel database machine which uses thousands of processors will allow for massive throughput of transactions and queries if no processors become a bottleneck. This paper proposes a load balancing method for a massively parallel semantic database.

Much work on load balancing for relational databases and file systems has been done and can be utilized in our research. For example, Sitaram et al.\(^1\) propose several dynamic load balancing policies for multi-server file systems. A dynamic load balancing algorithm for large, shared-nothing, hypercube database computers which makes use of relational join strategies is presented in Hua and Su\(^2\). Lee and Hua\(^3\) present a self-adjusting data distribution scheme which balances computer workload in a multiprocessor database system at a cell level during query processing. A run-time reorganization scheme for rule based processing in large databases is discussed in Stolfo et al.\(^4\).

Our database computer will make use of a shared-nothing architecture. The computational load on each processor of our database computer will vary directly with the demand for data on that processor. Imbalances in the number of data accesses among nodes can be rectified by repartitioning the database, much as imbalances in computational demands in process scheduling can be rectified by moving processes from one machine to another. When a range of facts in our database is moved from one processor’s control to another processor’s control, the load on the first processor will go down. The methods for determining imbalances in our system, and the methods to relieve these imbalances in our system, are very similar to the methods used for computational dynamic load balancing in shared-nothing computers. An adaptive, heuristic method for dynamic load balancing in a message-passing multiprocessor is presented in Xu and Hwang\(^5\). A semi-distributed approach to load balancing in massively parallel multiprocessor systems is presented in Ahmad and Ghafoor\(^6\).

Our massively parallel database machine architecture makes use of a distributed system of many processors, each with its own permanent storage device. This shared-nothing approach requires that any load balancing operations be performed by message passing. The data distribution scheme that is used in our database system allows load balancing to be achieved by
data repartitioning among the nodes of our system.

This paper refines the results reported in Rishie et al. and extends them by adding a fault tolerant data transfer policy for data repartitioning.

2. SEMANTIC BINARY DATABASE MODEL

The semantic database models in general, and the Semantic Binary Model SBM (Rishie and others) in particular, represent the information of an application's world as a collection of elementary facts categorizing objects or establishing relationships of various kinds between pairs of objects. The central notion of semantic models is the concept of an abstract object, which is any real world entity that we wish to store information about in the database. The objects are categorized into classes according to their common properties. These classes, called categories, need not be disjoint — that is, one object may belong to several classes. Further, an arbitrary structure of sub-categories and super-categories can be defined. The representation of the objects in the computer is invisible to the user, who perceives the objects as real-world entities, tangible, tangible, such as persons or cars, or intangible, such as observations, meetings, or desires. The database is perceived by its user as a set of facts about objects. These facts are of three types: facts stating that an object belongs to a category: \( x \in C \); facts stating that there is a relationship between objects: \( xRy \); and facts relating objects to data, such as numbers, texts, dates, images, tabulated or analytical functions, etc. \( xRy \). The relationships can be of arbitrary kinds: stating, for example, that there is a many-to-many relationship \( x \) between the category of persons and texts means that one person may have an address, several addresses, or no address at all.

3. STORAGE STRUCTURE

An efficient storage structure for semantic models has been proposed in Rishie. The collection of facts forming the database is represented by a file structure which ensures approximately 1 disk access to retrieve queries of any of the following forms:

1. For a given abstract object \( x \), verify/.
   find what categories the object belongs to.
2. For a given category, find its objects.
3. For a given abstract object \( x \) and relation \( R \), retrieve all/certain \( y \) such that \( xRy \).
4. For a given abstract object \( y \) and relation \( R \), retrieve all/certain abstract objects \( x \) such that \( xRy \).
5. For a given abstract object \( x \), retrieve (in one access) all (or several) of its direct and/or inverse relationships, i.e. relations \( R \) and objects \( y \) such that \( xRy \) or \( yRx \). The relation \( R \) in \( xRy \) may be an attribute, i.e. a relation between abstract objects and concrete objects.
6. For a given relation (attribute) \( R \) and a given concrete object \( y \), find all abstract objects such that \( xRy \).

7. For a given relation (attribute) \( R \) and a given range of concrete objects \( \{ y_1, y_2 \} \), find all objects \( x \) and \( y \) such that \( xRy \) and \( y_1 \leq y \leq y_2 \).

The entire database can be stored in a single file. This file contains all of the facts of the database (\( x \in C \) and \( xRy \)) as well as additional information called inverted facts: \( x_{C} \) and \( x_{Ry} \). The inverted facts ensure that answers to queries of forms 2, 4, 6 and 7 are kept in a contiguous segment of data in the database which allows them to be answered with one disk access. The direct facts \( xC \) and \( xRy \) allow the database to answer queries of forms 1, 3, and 5 with one disk access. The file is maintained as a B-tree. The variation of the B-tree used allows both sequential access according to the lexicographic order of the items comprising the facts and the inverted facts, as well as random access by arbitrary prefixes of such facts and inverted facts. Facts which are close to each other in the lexicographic order reside close to each other in the file. (Notice that technically the B-tree key is the entire fact, it is of varying length and on the average is only several bytes long, which is the average size of the encoded fact \( x_{Ry} \).)

Consider, for example, a database containing information regarding products manufactured by different companies. The following set of facts can be a part of a logical instantaneous database:

1. object1 COMPANY
2. object1 COMPANY-NAME 'IBM'
3. object1 MANUFACTURED object2
4. object1 MANUFACTURED object3
5. object2 PRODUCT
6. object2 COST 3600
7. object2 DESCRIPTION 'Thinkpad'
8. object3 PRODUCT
9. object3 COST 100
10. object3 DESCRIPTION 'TrackPoint'

The additional inverted facts stored in the database are:

1. COMPANY object1
2. COMPANY-NAME 'IBM' object1
3. object2 MANUFACTURED-BY object1
4. object3 MANUFACTURED-BY object1
5. COST 3600 object1
6. COST 100 object3
7. DESCRIPTION 'Thinkpad' object2
8. DESCRIPTION 'TrackPoint' object3
9. PRODUCT object2
10. PRODUCT object3

To answer the elementary query “Find all objects manufactured by object1”, we find all the facts whose prefix is object1_MANUFACTURED. ("_" denotes concatenation.) These entries are clustered together in the sorted order of direct facts.

To answer the elementary query “Find all products costing between $0 and $800”, we find all the facts whose prefix is in the range \( \text{COST}_0 \) to \( \text{COST}_{800} \). These entries are clustered together in the sorted order of inverted facts.

In the massively parallel version that we are developing, the B-tree is partitioned (on a secondary memory) that is off the other nodes. This disk-processing retrieves information from the other nodes. Pertinent integrity constraints in the order on disk or updated concrete objects.

The queries are transmitted through host interface to the copy of the Partitioned System. Since the whole database is represented by a collection of only a small number of minimally minimal and relatively small datum that is stored on the disk, the database is re-partitioned during the shifting of data with the normal operation.

Most of the philosophy of a semantic bina model is the same. These facts are of the form of objects, which as an abstract object, and since the objects are objects, traffic to each over time. Other objects with an inverted representation between an abstract object is not possible that at some certain attribute or category. The values of a given particular inverted atom can occur in some manner, taining the facts. We will contain this. The second block with an inverted at third subfile contains which are partitioned partitioned evenly.

Vol 11 no. 4 July 19
R and a given range d all objects x and y
single file. This file (xC and xRy) as well d facts: Cx, Rx. The
series of forms 2, 4, 6 nent of data in the
were with one disk allow the database to
with one disk access. variation of the B-tree ordering to the lexico-
g the facts and the
by arbitrary prefixes s which are close to
reside close to each
tically the B-tree
length and on the aver-
s the average size of
containing information different companies.
of a logical instanta-
the B-tree is partitioned into many small fragments, each
residing on a separate storage unit (e.g. a disk or non-volatile
memory) that is associated with a fairly powerful processor.
This disk-processor pair is called a node. Each node can
retrieve information from the disk, perform the necessary
processing on the data and deliver the result to the user, or
to the other nodes. For updates the node verifies all of the
relevant integrity constraints and then stores the updated infor-
many database fragments can be queried
or updated concurrently.
The queries and transactions will enter into the network
through host interfaces. Every host interface maintains a
copy of the Partitioning Map (PM) of the entire database.
Since the whole database is a lexicographically ordered file
represented by a set of B-trees, the map needs to contain
only a small number of facts for each node: the lexicographi-
minimal and maximal facts for each B-tree fragment
that is stored on that node. The map changes only when the
database is re-partitioned. The distribution policy that we
propose in this work provides repartitioning that is rare,
inexpensive, localizable, invisible to the system until all of
the shifting of data is complete, and that does not interfere
with the normal operation of the system.
Most of the physical facts that are in our implementation of
a semantic binary database start with an abstract object.
These facts are ordered by the encoding of the abstract
objects, which assigns a unique quasi-random number to
each abstract object. Since there are so many of these facts,
and since the objects are randomly ordered, we can assume
that traffic to each partition of these facts will be balanced
to range. Other facts in a semantic binary database start
with an inverted category or an inverted attribute (i.e., a rela-
tion between an abstract object and a printable value). It is
possible that at some time there may be a need to access a
certain attribute or category more often than other attributes
or categories. The same may be true for a specific range
of values of a given attribute. Since all facts that refer to a par-
ticular inverted attribute or inverted category are clustered
together, this may cause a higher load on some
processors/disk pairs than on others. Since load imbalances
can occur in some kinds of facts but not others, the file con-
taining the facts will be split into two subfiles. The first sub-
file will contain all the facts that begin with an abstract
object. The second subfile will contain the facts that begin
with an inverted attribute or category. Additionally there is a
third subfile containing long data items: texts, images, etc.,
which are pointed to by facts. Each subfile will be initially
partitioned evenly over all the processor/disk pairs in the
system. The first subfile is already balanced; the second and
third subfiles may become unbalanced and will require a
block placement algorithm that allows the data to be reparti-
tioned. By repartitioning data, we will be able to more evenly
balance the load to each data partition.

4. REQUEST EXECUTION SCHEME
We employ a deferred update scheme for transaction pro-
processing. This means that transactions are not physically per-
formed until they are committed, but are accumulated by the
database management system as they are run. Upon comple-
tion of the transaction the DBMS checks its integrity and
then physically performs the update. A completed transac-
tion is composed of a set of facts to be deleted from the
database, a set of facts to be inserted into the database, and
additional information needed to verify that there is no inter-
fERENCE BETWEEN TRANSACTIONS OF CONCURRENT PROGRAMS. IN OUR
parallel database, each node is responsible for a portion of the
database. When an accumulated transaction is performed,
the sets of facts to be inserted into, and deleted from,
the database must be broken down into subsets that can be sent
to the processors which are responsible for the relevant
ranges of data.
Each host in the system will have a copy of the Partition-
ing Map (PM). The Partitioning Map is a small semantic
database containing information about data distribution in
the system. Figure 1 is a semantic schema of the partitioning
map.
The partitioning map contains a set of ranges and their
lexicographical bounds — the low-bound and the high-bound
values. When a query or transaction arrives, the host will
identify its lexicographical bounds. The host will then use
the partitioning map to determine a set of ranges that needs
to be retrieved or updated and hence the nodes which will be
involved in the current transaction or query.
The partitioning map will be replicated among hosts.
However, this does not imply that we need a global data
structure; the algorithm described below allows updates of
the partitioning database to be performed gradually, without
locking and interrupting all hosts.
A range can be obtained from the node pointed to by the
location reference in the partitioning database. This node
should either have the range or a reference to another node
which contains the range.
To perform load balancing we will need to move ranges
from one node to another. A moved range will be accessible
via an indirect reference that is left at its previous location.
Such an indirect access slows down the system, especially
when the range is frequently accessed by users. To allow a
direct access to the moved range we need to update the loca-
tion reference in the partitioning database. We will not per-
form this update simultaneously for all the host interfaces.
The update will be performed when a host executes the first
query or transaction that refers to the range that was trans-
ferred. The node that actually holds the range will send the
results to the host along with a request to update the parti-
tioning map. This means that the first transaction will have
to travel a little further than all subsequent transactions.
The second and future queries or transactions made through
this host will be executed directly by the node pointed to by the
location reference.
The data structure at each node which supports indirect
referencing will be exactly the same as the partitioning map
described above. We will call this data structure a local parti-
tioning map.
Each range of facts will be represented as a separate B-
tree structure which will reside on the node pointed to by the
partitioning map. Consider a case where a range has been
moved several times from one node to another. We may have
multiple indirect references to the actual location of the
range. These indirect references will be changed to direct
references as described above,
5. DATA TRANSFER POLICY

In order to ensure that the database remains consistent throughout a load balancing data transfer, load balancing actions are executed as transactions initiated by the system. A large range of facts is transferred by executing a series of small system transactions that transfer small portions of data from one partition to another. The system transactions are subject to the same logging and recovery actions as regular, user initiated, transactions. Apart from the data transfer, each small load balancing transaction also includes the data necessary for updating the partitioning map. To ensure that the partitioning map remains consistent, the partitioning map update is executed using a 2-phase commit protocol.

6. LOAD BALANCING POLICY

When idle, the host interfaces will send data and work load statistics recently accumulated from the nodes to a Global Performance Analyzer (GPA). The host interfaces accumulate this data as the results of queries and transactions flow through them back to the user. The GPA is a process that analyzes the statistical information obtained and generates preferable directions of data transfer for each node.

The statistics report will contain only the changes since the previous report:

- Changes in data partitioning
- Number of accesses for each range
- Free space on each node

The GPA will use a heuristic search algorithm which uses a choice function to select a small number of possible data movements for the system. The final state will be estimated by a static evaluation function S. The GPA will select the data movement with the lowest value of the resulting static evaluation S.

The choice function should comply with the following strategies:

1. Whenever possible load balancing should be achieved by joining ranges together. Joining ranges will result in faster query execution and smaller partitioning maps.
2. A criterion for determining preferable destinations for a range transfer is the desire to move a range to a destination node which contains the lexicographically closest range to the transferred range. In other words, it is desirable to locate lexicographically close ranges on the same node whenever possible.
3. If a range has an exceptionally high number of access or requires an exceptionally large amount of storage—split the range into several parts and transfer them to other nodes.

Each node will be characterized by two parameters:

1. The amount of free disk space on the node, \( F \)
2. The percentage of idle time \( I \). In other words the \( I \) is: \( I = \frac{IdleT}{T} \), where \( T \) is a given time interval and \( Idle \) is the node’s idle time during the time \( T \).

The resulting state will be estimated by the following parameters:

1. \( A \) – the total amount of data that will be necessary to transfer in the system
2. \( D_f \) – the mean square deviation of \( F \)
3. \( D_I \) – the mean square deviation of \( I \)
4. \( M \) – total number of ranges in the system

The static evaluation function can be represented as:

\[ S = C_1 \times A + C_2 \times D_f + C_3 \times D_I + C_4 \times M, \]

where \( C_1, C_2, C_3 \) and \( C_4 \) are constants.

7. CONCLUSION

Our load balancing algorithm will provide our massively parallel semantic database machine with a method to repartition data to evenly distribute work among its processors. The algorithm has very little overhead, as its statistics are accumulated during the normal processing of transactions. The load balancing is accomplished by repartitioning parts of the database over the nodes of the database machine. The repartitioning will be transparent to the users and will not adversely affect the performance of the system. Our fault-tolerant data transfer policy will ensure that the database and its partitioning maps remain consistent during repartitioning.

We are currently developing a prototype parallel semantic database on a network of workstations. We will evaluate our load balancing algorithm on this prototype system and experiment with ways to optimize our heuristic search algorithm.

ACKNOWLEDGEMENTS

This research was supported in part by NASA (under grant

\[ \text{computer systems science & engineering} \]
LOAD BALANCING IN A MASSIVELY PARALLEL SEMANTIC DATABASE


REFERENCES


4 Stolfo, S. Dewan, H, Ohsie, D and Hernandez, M 'A Parallel and Distributed Environment for Database Rule Processing: Open Problems and Future Directions', in Emerging Trends in Database and Knowledge-Base

NAGW-4080), BMDO&ARO (under grant DAAH04-0024), NATO (under grant HTECH-LG-931449), NSF (under grant CDA-9313624 for CATE Lab), and the State of Florida.

Joining ranges together. Joining ranges that contain smaller and smaller data sets that are moved to other nodes will eventually result in the lexicographically smaller of the two ranges to be the transferred range. In other words, in this sense, the ranges are lexicographically ordered and whenever possible, small ranges are transferred whenever possible.

We potentially high number of processors and a continually large amount of traffic between several parts and transputer. We use the following parameters:

- The fraction of F, the current traffic on the node, F
- The total amount of I, the traffic level of the system during the time T

This traffic is represented as:

\[ F \times I \times T \]

We provide our massively parallel computer with a method to repartition data among its processors. The database and all of its statistics are accumulated over an interval of 10 seconds. We partitioning parts of the database among the processors. The repartitioning will be done whenever and will not depend on the system. Our fault-tolerant approach to the database and all of its statistics during repartitioning. The prototype parallel semantic database. We will evaluate our parallel semantic database prototype system and its heuristics for load balancing.