Informatica **22** (1998) Number 2, pp. 127–251

Volume 22 Number 2 May 1998 **ISSN 0350-5596**

Informatica

An International Journal of Computing and Informatics

Special Issue:
Parallel and Distributed Database Systems

1977

The Slovene Society Informatika, Ljubljana, Slovenia

Informatica

An International Journal of Computing and Informatics

Basic info about Informatica and back issues may be FTP'ed from `ftp.arnes.si` in magazines/informatica ID: anonymous PASSWORD: *<your mail address>*
FTP archive may be also accessed with WWW (worldwide web) clients with
URL: `http://www2.ijs.si/~mezi/informatica.html`

Subscription Information Informatica (ISSN 0350-5596) is published four times a year in Spring, Summer, Autumn, and Winter (4 issues per year) by the Slovene Society Informatika, Vožarski pot 12, 1000 Ljubljana, Slovenia.
The subscription rate for 1998 (Volume 22) is
- DEM 50 (US$ 35) for institutions,
- DEM 25 (US$ 17) for individuals, and
- DEM 10 (US$ 7) for students

plus the mail charge DEM 10 (US$ 7).
Claims for missing issues will be honored free of charge within six months after the publication date of the issue.

LaTeX Tech. Support: Borut Žnidar, Kranj, Slovenia.
Lectorship: Fergus F. Smith, AMIDAS d.o.o., Cankarjevo nabrežje 11, Ljubljana, Slovenia.
Printed by Biro M, d.o.o., Žibertova 1, 1000 Ljubljana, Slovenia.

Orders for subscription may be placed by telephone or fax using any major credit card. Please call Mr. R. Murn, Jožef Stefan Institute: Tel (+386) 61 1773 900, Fax (+386) 61 219 385, or use the bank account number 900–27620–5159/4 Ljubljanska banka d.d. Slovenia (LB 50101-678-51841 for domestic subscribers only).

According to the opinion of the Ministry for Informing (number 23/216-92 of March 27, 1992), the scientific journal Informatica is a product of informative matter (point 13 of the tariff number 3), for which the tax of traffic amounts to 5%.

Informatica is published in cooperation with the following societies (and contact persons):
Robotics Society of Slovenia (Jadran Lenarčič)
Slovene Society for Pattern Recognition (Franjo Pernuš)
Slovenian Artificial Intelligence Society; Cognitive Science Society (Matjaž Gams)
Slovenian Society of Mathematicians, Physicists and Astronomers (Bojan Mohar)
Automatic Control Society of Slovenia (Borut Zupančič)
Slovenian Association of Technical and Natural Sciences (Janez Peklenik)

Informatica is surveyed by: AI and Robotic Abstracts, AI References, ACM Computing Surveys, Applied Science & Techn. Index, COMPENDEX*PLUS, Computer ASAP, Computer Literature Index, Cur. Cont. & Comp. & Math. Sear., Current Mathematical Publications, Engineering Index, INSPEC, Mathematical Reviews, MathSci, Sociological Abstracts, Uncover, Zentralblatt für Mathematik, Linguistics and Language Behaviour Abstracts, Cybernetica Newsletter

The issuing of the Informatica journal is financially supported by the Ministry for Science and Technology, Slovenska 50, 1000 Ljubljana, Slovenia.

Post tax payed at post 1102 Ljubljana. Slovenia taxe Percue.

Performance Modeling of Parallel Database Systems

Silvio Salza and Massimiliano Renzetti
Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza", Roma, Italy
E-mail: salza@dis.uniroma1.it

Keywords: parallel architectures, performance evaluation, workload modelling

Edited by: Tadeusz Morzy

Received: October 30, 1996 **Revised:** March 7, 1997 **Accepted:** April 25, 1997

The paper investigates the main issues in performance analysis and tuning of parallel database applications. More specifically we present a modelling methodology that was developed for an important class of parallel relational systems, and is devised for a strict integration with the design procedure. Our approach is meant to provide the designer with a valuable feedback since the early stages of a project, i.e. before the system is even implemented. Developing the model has required to deal with several interesting problems, and has led to some original contributions especially for the bufferpool model and the evaluation of transaction response times.

1 Introduction

Performance on demanding applications has been one of the main targets in designing parallel database systems. After about two decades of evolution, the last generation of scalable parallel DBMS, mostly based on standard hardware and operating systems, can deliver enough processing power to become a competitive alternative to traditional mainframe based database architectures and to distributed systems, especially for specific applications, like massive transaction processing and data warehousing.

As a matter of fact, despite the evident centrality of the performance problem, in the early university-based database machines projects not enough energy was devoted to this topic. This led to a first (and unsuccessful) generation of parallel database systems, where the main concern had been improving the parallel execution of relational operators, and the most common result was getting stuck with the I/O bottleneck. Later the problem was better understood (Cesarini and Salza 1987), and the first successful commercial systems have been based on MIMD architectures with non-shared disks and data replication to attain a high degree of I/O parallelism (Stonebraker 1986). As a further evolution of this trend, the last generation of parallel DBMS is mostly based, as we will discuss later, on *shared-nothing* architectures, with largely independent processing units and private memory and disks.

Dealing with these innovative architectures has considerably changed the process of developing database applications. Designers and DB administrators have in fact to face a series of new problems, mostly connected with the more sophisticated physical data organization (e.g. relation declustering), and, in general, with the more complex execution model. In addition to this the traditional well known problems of relational systems remain as well, i.e. substantial difficulty in following Codd's original (and partially wishful) idea to make the logical design of a relational application largely independent from the details of the physical organization.

The crucial problem, both in parallel and in sequential database systems, is that query execution plans, that directly affect the execution cost and the performance, are generated by the optimizer, and therefore are completely out of the designer's control. Therefore one has to get anyway involved with the physical level, both to detect where the performance problem is, and to set up a solution. Performance tuning may then become an extremely painful process, especially if one has to wait for the final stages of the implementation before being able to trace the problem.

Besides this there is indeed a main difference in designing parallel and sequential database applications. In the sequential case the designer should *also* consider the performance aspects, and he actually does it (very often *a posteriori*) only if he runs into performance problems. For parallel DBMS systems, instead, performance *is* the problem, since it is the most likely reason why a parallel system was selected, and then performance analysis has to be an essential ingredient of the design and configuration process.

In this paper we discuss the main issues of performance-oriented parallel database application design. More specifically we present a modeling methodology that we have developed for an important class of parallel relational database systems, and covers all the phases of the design process: preliminary design, configuration, tuning and capacity planning. The methodology is based on the results of previous work we have done on this subject for traditional sequential DBMS (Salza and Terranova 1989, Salza and Tomasso 1992), and is devised for a strict integration into the design procedure. This is meant to provide both an early feedback on the performance of the application, and a detailed account of the workload and the execution cost, that help focus the problem and give hints for the design improvement.

The approach we propose is quite general, but, of course, in developing the methodology we have referred to a specific parallel RDBMS, namely DB2 *Parallel Edition* (Baru et al. 1995) an IBM product that runs on IBM SP2 parallel architecture , a system with a typical MIMD architecture, i.e. is composed by the interconnection, via a fast interconnection network, of several largely independent systems, each with its own private memory and disks managed by local Unix operating systems (Agerwala et alii 1995).

Connected to the methodology we have also designed and implemented in a prototype version a modelling tool, that was specifically developed for DB2-PE, and that was also used to produce the results that are reported in this paper. The tool, as we shall discuss in more detail in the next sections, accepts as an input a detailed description of the database, of the transaction workload and the configuration of the parallel system, and generates a set of estimates of important performance measures, such as execution costs and response times.

The paper is organized as follows. In the next section we discuss the main architectures that have been proposed in the literature for parallel database systems, and in Section 3 in more detail the *shared-nothing* model. Then in Section 4 we introduce the workload model, i.e. the set of parameters that we use to give a quantitative characterization of the database (*static workload*) and of the set of transactions to be processed (*dynamic workload*). The system configuration parameters are discussed in Section 5. Sections 6 and 7 deal with the evaluation of the transaction execution cost. In particular in Section 7 we present an approximated mathematical model that allows to take into account the influence of database buffering on the I/O load. In Section 8 we discuss how the transaction cost estimates can be used to perform bottleneck analysis, and to guide load balancing strategies. Next, in Section 9, we present a probabilistic model for the computation of transaction response time. Finally conclusions are given in Section 10.

2 Parallel DB architectures

The focus in a parallel database architecture is the interconnection between processing units, main memory banks and mass storage units. In a well known paper Stonebraker has proposed a taxonomy for the interconnection topology that has been since then universally adopted (Stonebraker 1986).

- *Shared-everything*: in this topology (which is also known as *shared-memory*) all processors share both the disks and the main memory, therefore there are no communication costs and no constraints connected to the data allocation scheme. The main advantages are flexibility in implementing load balancing strategies and in general in designing the structure of the parallel DBMS. On the other hand there is a limited scalability, since the memory may easily become a bottleneck, and may seriously affect system performance.

- *Shared-nothing*: in this topology each processor accesses its own private memory and disks, and communicates with other processors only through a fast interconnection structure. In this case, compared to shared-memory, advantages and disadvantages are reversed. There is little or no resource contention, and therefore there are no memory or disk bottlenecks. This actually guarantees an almost complete scalability of the architecture. But the lack of sharing makes the design of the DBMS and of the applications far more complex, and the performance strongly depends on the partitioning of data among the storage units.

- *Shared-disks*: this represents a somehow intermediate solution. Each processor has its own private memory but all disks are shared. It is easier to implement than the *shared-nothing* architecture since a fast interconnection structure is not needed, and has a better flexibility in load balancing. On the other hand the concurrent access to shared mass memory structures requires sophisticated concurrency control protocols, similar to the ones used in distributed DBMS. Moreover the performance of these systems may suffer from a disk I/O bottleneck.

Besides the main models above, several other *hybrid* topologies have been proposed in the literature in the last decade, for instance the *shared-something* architecture where a shared-disk system is formed by the interconnection of several shared-memory subsystems (Valduriez 1993).

As one may easily understand, any methodology for the design and tuning of parallel database applications is strongly dependent on the reference architecture. As a matter of fact most commercial parallel DBMS have a shared-nothing architecture (Carino 1992, Baru et al. 1995), and only a few adopt the shared-disk model (e.g. Oracle); practically only research prototypes have till now used the shared-memory topology (Bitton 1983, Katz et al. 1988, Graefe 1990). Therefore in this paper we will concentrate on the performance modelling of shared-nothing systems.

3 The shared-nothing model

In a shared-nothing architecture data are partitioned among the processors and stored on local disks, and queries are executed according to a *function shipping* philosophy, i.e. to perform database operations where the data reside.

More precisely each relation is *declustered* (i.e. horizontally partitioned) on a subset of processors called *node-group*. Declustering means distributing the tuples of the relation on the nodes by hashing it on a given attribute, which is called the *partition key*.

Relational operations on base relations are executed in parallel by all the nodes of a node-group, typically the one on which one (and possibly both) operands are declustered. Operations performed on intermediate results get their data from other operators by communicating in several different ways:

- *temporary file*: the intermediate result produced by a previous operator is materialized, declustered and stored on local disk;
- *pipeline*: two or more operators can be pipelined; the intermediate results are not written to disk, and the destination operator may start as soon as the first block of data is available;
- *table-queue*: two operators communicate through a FIFO file, managed at the block level.

In the first two cases the source and the destination operators must be on the same node-group, but operators executed by different node-groups have always to communicate through table-queues.

There are several options for the execution of each relational operator, mostly depending on data placement. For instance a join can be performed in four different ways:

- *collocated join*: when the two relations are declustered on the same node-group and the partition key is the join attribute; no data exchange is needed and all the nodes in the node-group may work in parallel.
- *directed join*: if only one relation is partitioned on the join attribute, then the other relation is hashed on the join attribute and distributed between the nodes of the node-group;
- *broadcast join*: if neither relation is partitioned on the join attribute either the inner or the outer table is broadcasted and joined in parallel in each node with the other;
- *repartition join*: both relations are partitioned at execution time and then a collocated join is performed.

It is clear from the above, that the query optimization problem in such an environment becomes considerably harder than in a sequential RDBMS. Much effort has been indeed devoted in the literature to this topic (Chen 1993, Graefe 1993, Lanzelotte et al. 1993, Lu and Tan 1993), though actually only specific aspects have been thoroughly analyzed (e.g. Borla-Salamet et al. 1993, Hong and Stonebraker 1993).

The main point is that in the parallel model the solution space and the execution cost strongly depend on the data placement. Therefore one has to actually face two distinct, but intimately connected, problems: query and data placement optimization. The application designer is in fact confronted with the problem of selecting the best data allocation for a given workload of predefined queries.

The purpose of our modeling methodology is indeed to provide the designer with an easy way to compare different design alternatives, and to have an analytical account of the execution cost that may guide him in the tuning and in the capacity planning activity.

4 The workload model

In our model we assume that all transactions arriving to the parallel DBMS belong to a set of *predefined transaction types*. To give a quantitative description of the application the user must therefore specify the two main components of the workload:

- the *static workload*, composed of the logical schema of the database, of a set of parameters that summarize the physical extension of the relations and the statistical characteristics of the attributes, the specification of the relation declustering, and of the definition of the access structures;
- the *dynamic workload*, composed of the specification of a set of predefined transaction types and of their arrival rates;

4.1 Static workload

Formally we define a *database* as a set of *relations*: $D = \{\mathcal{R}_i, i = 1 \dots N\}$. Each relation is a set of *tuples* $\mathcal{R}_i = \{r_{ij}, j = 1 \dots c_i\}$, where c_i indicates the *cardinality* of the relation.

Each tuple r_{ij} is an ordered set of k_i values, where k_i is said the a-rity of the relation:

$$r_{ij} = \langle r_{ij}[1], \dots, r_{ij}[k_i] \rangle \quad (1)$$

$$r_{ij}[h] \in V_i[h] \quad h = 1 \dots k_i; j = 1 \dots c_i \quad (2)$$

$$\mathcal{R}_i[h] = \{r_{ij}[h], j = 1 \dots c_i\} \quad h = 1 \dots k_i \quad (3)$$

The multisets $\mathcal{R}_i[h]$, containing all the values assumed by a given field in the relation tuples, are called *attributes*, and the corresponding base sets $V_i[h]$ are called *value sets* and contain only distinct values.

```
CUSTOMERS  Card=75000            ITEMS      Card=3000000         SUPPLIERS
Card=5000
  Cust#     Char(4)     75000      Price     Num(8)        500       Supp#
Char(4)    5000
  Name      Char(25)    74950      Qty       Num(8)        100       Name
Char(25)   4995
  Phone     Char(15)    75000      C_date    Date         1825       Addr
Char(40)   5000
  Addr      Char(40)    75000      Status    Char(1)         4       Phone
Char(15)   5000
  Comm      Char(117)      80      Ord#      Char(4)    750000       Comm
Char(101)   30
  Nat#      Char(4)        20      Part#     Char(4)     99800       Nat#
Char(4)     15
                                   Supp# Char(4)      4950

ORDERS     Card=750000            PARTSUPP  Card=400000          PARTS
Card=100000
 Ord#      Char(4)    750000       A_qty     Num(4)       3500       Part#
Char(4)    100000
 Clerk     Char(15)     1000       S_cost    Num(8)      12000       Name
Char(55)    1200
 Price     Num(8)      75000       Price     Num(8)      25000       Part#
Char(4)    100000
 O_date    Date         1825       Supp#     Char(4)      5000       Descr
Char(23)    5000
 Status    Char(1)         4
 Comm   T  Char(79)      120
 Cust#     Char(4)     75000

REGIONS    Card=5                 NATIONS  Card=25
 Reg#      Char(4)         5        Nat#      Char(4)      25
 Name      Char(25)        5        Name      Char(25)     25
 Comm      Char(152)       3        Comm      Char(152)    25
                                    Reg#       Char(4)     25
```

Figure 1: Sample static workload

```
Transaction T1          Rate = .20           Transaction T2       Rate = .15

SELECT PARTS.Name,ITEMS.Qty,ITEMS.Price      SELECT
SUPPLIERS.Name,PARTSUPPS.S_cost,A_qty
FROM CUSTOMERS,ITEMS,ORDERS,PARTS            FROM PARTS, PARTSUPPS, SUPPLIERS
WHERE CUSTOMERS.Cust# = ORDERS.Cust# AND     WHERE PARTSUPPS.Supp# =
SUPPLIERS.Supp# AND
      ITEMS.Ord# = ORDERS.Ord# AND                 PARTS.Part# =
PARTSUPPS.Part# AND
      ITEMS.PART# = PARTS.Part# AND                PARTS.Name = 'CPU'
      CUSTOMERS.Name = 'Smith' AND           ORDER BY PARTSUPPS.S_cost,
SUPPLIERS.Name
      ITEMS.Price > 1000000 [.2]
ORDER BY PARTS.Name


Transaction T3          Rate = .35           Transaction T4      Rate = .30

SELECT SUPPLIERS.Name,CUSTOMERS.Name,
      NATIONS.Name                           SELECT SUPPLIERS.Name
FROM CUSTOMERS,NATIONS,SUPPLIERS             FROM NATIONS,REGIONS,SUPPLIERS
WHERE CUSTOMERS.Nat#=SUPPLIERS.Nat# AND      WHERE
NATIONS.Nat#=SUPPLIERS.Nat# AND
      NATIONS.Nat#=SUPPLIERS.Nat#
NATIONS.Nat#=REGIONS.Nat# AND
                                                  REGIONS.Name='Eur' OR
Name='Afr'
```

Figure 2: Sample dynamic workload

For our purposes, a quantitative description of the database can be given by specifying the cardinalities $c_i, i = 1 \ldots n$ of the relations, and the following parameters for each attribute $\mathcal{R}_i[h]$ in the database:

- $o_i[h] = card(V_i[h])$, called the attribute *originality*, which represents the number of distinct values in $\mathcal{R}_i[h]$;
- $\lambda_i[h]$, called the attribute *extension*, which represents the number of bytes needed to store each element of $\mathcal{R}_i[h]$.
- $\tau_i[h] \in \{char, number, date\}$, called the attribute *type*, which represents its data type.
- $\nu_i[h]$, which represents the expected fraction of *null values* in $\mathcal{R}_i[h]$.

Moreover to represent the coupling between attributes we must specify for every couple of *union-compatible* attributes the *overlapping factor* :

$$w_{i,h}^{j,k} = \frac{card(V_i[h] \cap V_j[k])}{card(V_i[h])} \tag{4}$$

which gives the percentage of values that are common to both attributes.

This characterization is simple enough to allow reasonably the designer to estimate the parameter values even during early phases of the design, and on the other hand it is sufficient, as we shall discuss later, to compute the execution cost of any relational operation performed directly on the base relations. However, dealing with complex queries requires to get estimates of the parameters for the intermediate relations as well. This can be performed using a methodology that we have proposed in previous papers (Cesarini and Salza 1987, Salza and Tomasso 1992).

Figure 1 shows the description of the static workload for a sample application, a typical customer-supplier-part database, to which we will refer all along the paper. Relation schemata are shown together with cardinalities and attribute originalities.

4.2 Dynamic workload

For the dynamic workload, as mentioned before, we assume that all transactions arriving to the system belong to a fixed set of predefined *transaction types*:

$$\mathcal{Q} = \{\mathcal{T}_i, i = 1, \ldots, m\} \tag{5}$$

For each transaction type $\mathcal{T}_i$ all the details except some of the constants are specified, and the following information is supplied:

- the *relative arrival rate* λ_i, i.e. the fraction of the transaction load that belongs to transaction type $\mathcal{T}_i$;
- a SQL definition of the transaction type, from which the execution plan can be deduced;
- the expected selectivity of each atomic predicate, necessary to compute the size of the intermediate results, and then the execution cost;
- the CPU and I/O overhead of the *application part* of the transaction;

In addition to this one must also specify the *workload intensity*, i.e. the overall rate Λ at which transactions arrive to the system.

Again this information can reasonably be derived by the designer, from the user specification of the application. Perhaps the most delicate part is estimating the selectivities, since these may have a considerable impact on the execution cost.

The transaction set of the sample application is reported in Figure 2. The user estimates of the selectivities for the atomic predicates are printed in square brackets. No selectivities are specified for equality predicates, since, assuming a uniform distribution, they are directly computed by the model from the database parameters.

5 Configuration parameters

The static and dynamic workload descriptions of the previous section, are the *input data* to the design: they come from the problem, and the designer cannot change them. The designer may instead take decisions about the physical design of the database and the configuration of the system, i.e. the number of nodes, the speed of the processing elements, the size of the database buffers, the number of disks and their size and access times.

This information is reported in the *allocation map* which specifies the way relations are declustered, in the *index map* that specifies which access structures are maintained by the DBMS, and in the *node configuration parameters* which specify the configuration of the nodes.

relation	*n-group*	*key*	*skew*	*node*
CUSTOMERS	γ_A	Nat#	.65	n_3
ITEMS	γ_0	Ord#	.30	n_7
NATIONS	γ_2	–	–	–
ORDERS	γ_0	Ord#	.38	n_7
PARTS	γ_B	Part#	–	–
PARTSUPP	γ_B	Part#	.40	n_8
REGIONS	γ_2	–	–	–
SUPPLIERS	γ_A	Supp#	.70	n_3

Figure 3: The allocation map

An allocation map for the sample application of Figg. 4.1 and 2 is shown in Fig. 3. The figure refers to a system of 8 nodes in which we define a set of node-groups:

$$\begin{aligned}\gamma_0 &= \{n_1 \dots n_8\} \\ \gamma_A &= \{n_3, n_4\} \\ \gamma_B &= \{n_5, n_6, n_7, n_8\} \\ \gamma_i &= \{n_i\}\end{aligned} \tag{6}$$

The map specifies for each relation the partition key, the node-group on which the relation is partitioned, and the *data skew.* The latter is a measure of the non-uniformity of the partitioning, and gives the fraction of the tuples that are allocated on the most 'crowded' node, assuming (as it is usually done) that the distribution on the remaining nodes is uniform (Lakshmi and Yu 1990). This is indeed a very important information, since the data skew may considerably affect the performance by umbalancing the load.

relation	*n-group*	*attribute*	*type*
CUSTOMERS	γ_A	Name	B+tree
ITEMS	γ_0	Ord#	B+tree
SUPPLIERS	γ_A	Name	B+tree
ORDERS	γ_0	Ord#	B+tree
PARTSUPP	γ_B	Part#	B+tree
PARTSUPP	γ_B	Supp#	B+tree
PARTS	γ_B	Name	B+tree

Figure 4: The index map

An index map for the example is shown in Fig. 4. Referring to the dynamic workload of Fig. 2, indexes are provided to support most of the joins. In a shared-nothing parallel DBMS indexes are usually local to processing nodes. So when a relation is declustered on a given node-group, actually a separate index is built on each node of the node-group.

A set of values for the configuration parameters of a sample node is shown in Fig. 5. The processing speed is expressed as the relative speed compared to a standard CPU assumed as a reference in the model, moreover a CPU overhead by the OS and other applications is specified as a fraction of the utilization. Other parameters specify the size in blocks of the bufferpool and sortheap areas. The latter may considerably influence the cost of merge-sort operations.

For each disk in addition to the usual parameters a number of segments is also specified. When a relation is declustered the tuples allocated on each node are distributed with a block level round-robin scheme among the segments of all the disks in the node. This gives the designer a way to balance the load between local disks, by changing the relative number of segments.

Though some details of the configuration parameters discussed above actually refer to some peculiarity of DB2-PE/SP2, similar arrangements are found in other shared-nothing architectures, and therefore this does not affect the generality of our approach.

6 Virtual execution costs

In our methodology, estimates of the execution costs can be directly computed from the specification of the static and dynamic workload, and from the configuration parameters.

We consider two components of the execution cost: the *storage cost*, i.e. the secondary memory layout and the storage requirements of the database (tables and indices), and the *transaction execution cost* (CPU, I/O and transmission cost), that can be computed for every transaction type in the set $\mathcal{Q}$.

To provide an accurate specification of the I/O cost, in terms of disk access rates, we consider the tables and the indices as partitioned in *homogeneous data segments.* These are collections of blocks belonging to the same table or index such that all the blocks in the same segment have the same probability to be accessed during the execution of the application, for example the blocks at the same level of a B-tree index. The partition in segments has the purpose to allow, in a later phase of modeling, the computation of the actual I/O cost, taking into account the uneven effect of DBMS buffering on segments with different access rates.

As a first step we characterize the transaction execution cost in terms of *resource demands*, expressed independently from the system configuration and the workload profile (global arrival rate and transaction mix). We call these *virtual costs.* These include the CPU service time estimated on a reference processor, transmission cost, and the number of *logical* disk accesses, i.e. computed without taking into account the buffering.

For each transaction type $\mathcal{T}_i$, virtual costs are computed with reference to the *parallel execution plan*, which represents the sequence of concurrent tasks performed by the parallel systems during the evaluation of the query. The execution tree is actually generated by the *query optimizer* (traditionally the most *mysterious* parts of a RDBMS), according to a complex strategy, mostly driven by the data placement and the availability of indices. To *mimic* that strategy may in fact give a few technical problems, and actually requires intensive checking against the DBMS.

A sample parallel execution plan is shown in Figure 6 for transaction $\mathcal{T}_1$. The leaves represent base relations and the internal nodes parallel operators, each labeled with the node-group on which it is executed. The arcs represent the flow of data between operators, and their label specifies the kind of connection (F: temporary file; TQ: table-queue; PIPE: pipeline).

The execution plan is a tree (or more precisely a DAG, since the same base relation may be an input

CPU - MEMORY

Parameter	*Value*
Relative speed	.20
CPU overhead	.05
Bufferpool area	1000
Sortheap area	1000
Memory segments	16

DISKS

Parameter	$D_{i,A}$	$D_{i,B}$
Disk size (Gbyte)	1	2
Access time (ms)	16	24
Block size (Kbyte)	4	4
Overhead	.1	.15
Disk segments	6	10

Figure 5: Node configuration parameters

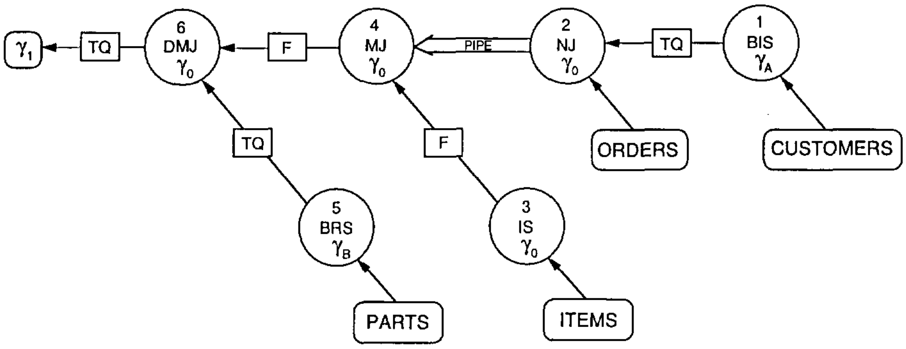


Figure 6: Parallel query execution plan

Resource	$\mathcal{T}_1$	$\mathcal{T}_2$	$\mathcal{T}_3$	$\mathcal{T}_i$	ρ	$\hat{\rho}$
CPU$_1$	4.990 (.998)	4.750 (.950)	.215 (.043)	.015 .003()	13.660	.205
$D_{1,A}$	4302 (4676)	90 (90)	204 (204)	12 (12)	16.400	.246
$D_{1,B}$	7171 (7793)	151 (151)	341 (341)	20 (20)	40.933	.614
CPU$_2$	4.760 (.952)	.035 (.007)	.005 (.001)	.005 (.001)	10.340	.155
$D_{2,A}$	3060 (3569)	33 (33)	1 (1)	1 (1)	32.012	.148
$D_{2,B}$	5101 (5948)	56 (56)	1 (1)	1 (1)	24.670	.370
CPU$_3$	3.810 (.762)	.110 (.022)	1.530 (.306)	.065 (.013)	13.670	.205
$D_{3,A}$	3329 (3571)	78 (93)	1128 (1260)	46 (61)	17.268	.259
$D_{3,B}$	5548 (5951)	131 (156)	1880 (2100)	76 (101)	43.400	.651
CPU$_4$	3.805 (.761)	.070 (.014)	.815 (.163)	.025 (.005)	10.671	.160
$D_{4,A}$	3277 (3570)	51 (59)	588 (671)	19 (27)	14.031	.210
$D_{4,B}$	5464 (5950)	87 (99)	982 (1118)	33 (44)	35.011	.525
CPU$_5$	4.215 (.843)	.110 (.022)	—	—	11.000	.165
$D_{5,A}$	3405 (3777)	54 (62)	—	—	10.000	.150
$D_{5,B}$	5676 (6296)	89 (102)	—	—	44.000	.660
CPU$_6$	4.215 (.843)	.110 (.022)	—	—	11.000	.165
$D_{6,A}$	3405 (3777)	54 (62)	—	—	10.000	.150
$D_{6,B}$	5676 (6296)	89 (102)	—	—	44.000	.660
CPU$_7$	9.345 (1.869)	.110 (.022)	—	—	18.670	.280
$D_{7,A}$	8214 (8588)	57 (62)	—	—	37.000	.555
$D_{7,B}$	13693 (14314)	94 (102)	—	—	66.670	1.000
CPU$_8$	4.215 (.843)	.700 (.140)	—	—	10.335	.155
$D_{8,A}$	3406 (3777)	54 (62)	—	—	11.000	.165
$D_{8,B}$	5676 (6296)	89 (102)	—	—	27.467	.412
NET	1474	41	547	33	.267	.004

Figure 7: Service demands and resource relative utilizations

to several operators). The tree is first visited in postorder to compute the size and all the extensional parameters of the intermediate relations. These are used, as we will see, to compute the I/O and CPU cost of each operator, that directly depends on the extensional characteristics of the operands.

To understand how the computation of the virtual costs is carried out, let us consider the set Ω_i of parallel operators in the execution plan of transaction $\mathcal{T}_i$, and let us call γ_ω the node-group on which operator $\omega \in \Omega_i$ is executed. Each parallel operator actually represents a set of sequential operators $\omega(n_j)$, that we shall call *operator instances*, and that are executed each on a processing node $n_j \in \gamma_\omega$ of the node-group.

Therefore, given the extensional parameters of the operands, that have been computed in the previous step, and once the implementation of the relational operators in the given DBMS is known, the computation of the execution cost of each instance becomes a straightforward task.

Then, if we call $\text{CPU}(\omega, n_j)$ (service time on the reference CPU) and $\text{DISK}(\omega, D_{j,k})$ (disk accesses) the CPU cost and the I/O cost on disk $D_{j,k}$ generated on node n_j by the execution of operator instance $\omega(n_j)$, we may compute the total virtual cost generated on node n_j by each execution of transaction $\mathcal{T}_i$ by adding up on all the operators of its execution plan:

$$\text{CPU}(\mathcal{T}_i, n_j) = \sum_{\omega \in \Omega_i \wedge n_j \in \gamma_\omega} \text{CPU}(\omega, n_j) \quad (7)$$

$$\text{DISK}(\mathcal{T}_i, D_{j,k}) = \sum_{\omega \in \Omega_i \wedge n_j \in \gamma_\omega} \text{DISK}(\omega, D_{j,k}) \quad (8)$$

Virtual costs given by the formulas above are an intrinsic characteristic of *each* transaction type, that is they depend only on the static workload and on the execution plan of that transaction. To get *physical costs*, i.e. the actual load for the resources, we must take into account both the node configuration parameters and the dynamic workload, that is *all* transaction types and their arrival rates.

CPU physical cost can be directly derived by scaling the virtual cost $\text{CPU}(\mathcal{T}_i, n_j)$ by the CPU speed factor σ_i. To get physical disk costs, i.e. the number of accesses actually performed to each disk unit, we must instead consider the effect of buffer caching, which for each node depends on the size of the bufferpool area, and, of course, on the transaction mix.

7 The buffer model

On each node the bufferpool area is managed with a global LRU policy. Therefore the data blocks with high global access rate tend to reside in it. Actually, according to the assumption made in Section 6 all the blocks of the same homogeneous data segment have the same access rate, and then the same probability to be in the buffer. Therefore for a given workload, and hence for a given access pattern of the data blocks, to compute the number of physical accesses, we need to evaluate for each segment the *buffer hit ratio*, that is the equilibrium probability that a block of the segment is found in the buffer.

Models to represent the effect of the buffering on database operations have been proposed by several authors, analyzing both the buffer requirements of relational queries, and the effect of local LRU policies on the number of page fetches (Mackert and Lohman1989, Sacco and Schkolnick 1986, Chou and DeWitt 1985). Unfortunately these results do not apply to our case, and we had to develop a new approach to represent the same buffer being shared by several transactions. The model we propose approximates LRU with Random policy, but, despite this simplification, seems to evaluate quite effectively the hit ratios of the segments, and to be accurate enough for our purposes. Anyway, not taking into account the locality of the references leads to conservative estimates.

More formally, let us refer to a node with buffer of N_b blocks, and to a set of N_s homogeneous data segments, with global access rates f_i, and sizes B_i (in blocks). Let now R_i be the *resident set size* of segment i, i.e. the average number of blocks in the buffer, and let α_i and ω_i be respectively the rates at which the blocks of segment i enter and leave the buffer. In an equilibrium condition the number of resident pages of every segment is constant, and then the two rates α_i and ω_i must equate:

$$\alpha_i = f_i \frac{B_i - R_i}{B_i} \quad (9)$$

Similarly ω_i depends on the resident set size of the segment, and on the global replacement rate:

$$\omega_i = \frac{R_i}{N_b} \sum_{j=1}^{N_s} \omega_j \quad (10)$$

Then substituting the (9) in the (10) and considering that $\alpha_i = \omega_i$ we get:

$$f_i \frac{B_i - R_i}{B_i} = \frac{R_i}{N_b} \sum_{j=1}^{n_s} f_j \frac{B_j - R_j}{B_j} \quad (11)$$

The (11) are a set of nonlinear equations in the unknowns R_i. An approximate solution to the system can then easily be computed with the iterative formulas:

$$\hat{R}_i[0] = N_b \frac{B_i}{\sum_{j=1}^{N_s} B_j} \quad (12)$$

$$\hat{R}_i[k] = \frac{f_i B_i N_b}{f_i N_b + Bi \sum_{j=1}^{N_s} f_j \frac{B_j - \hat{R}_j[k-1]}{B_j}} \qquad (13)$$

¿From a practical point of view, the experience shows that the iteration converges quickly to the solution of the system, and hence we may compute the hit ratios, that can be expressed as the ratio between R_i and B_i.

8 Bottleneck analysis

Transaction execution costs for the sample application are shown in Fig. 7. CPU costs are given in seconds and I/O and transmission costs in block accesses and transfers. Virtual costs are shown in parenthesis. Costs are split by node and by transaction type, and represent the service demand of a single transaction to each resource in the system.

The sum of the service demands from all transaction types $\mathcal{T}_i \in \mathcal{Q}$ to a given resource weighted by their relative arrival rate λ_i is called the relative utilization of the resource (and is the utilization up to a constant factor).

Relative utilizations are then given by:

$$\rho(n_j, \sigma_j) = \sigma_j \sum_{i=1\ldots m} \lambda_i \,\mathrm{CPU}(\mathcal{T}_i, n_j) \qquad (14)$$

$$\rho(D_{j,r}, N_b) = t_{j,r} \sum_{i=1\ldots m} \lambda_i \,\mathrm{DISK}(\mathcal{T}_i, D_{j,r}, N_b) \qquad (15)$$

where σ_j is the CPU speed factor for node j, λ_i the relative arrival rate of $\mathcal{T}_i$, $t_{j,r}$ is the average service time of disk $D_{j,r}$ and $\mathrm{DISK}(\mathcal{T}_i, D_{j,r}, N_b)$ is the number of physical accesses to disk $D_{j,r}$ by transaction $\mathcal{T}_i$ with a bufferpool area of size N_b.

The relative utilization corresponds to the actual resource utilization for unitary overall transaction arrival rate. Thus for a given overall arrival rate of Λ the utilizations are given by:

$$U(n_j, \sigma_j, \Lambda) = \Lambda\, \rho(n_j, \sigma_j) \qquad (16)$$

$$U(D_{j,r}, N_b, \Lambda) = \Lambda\, \rho(D_{j,r}, N_b) \qquad (17)$$

The resource with the largest relative utilization, say ρ_{max}, is called the *system bottleneck*. As we consider the system as an open network of queues, this sets a limit to the maximum overall transaction throughput.

$$\Lambda_{max} = \frac{1}{\rho_{max}} \qquad (18)$$

Relative utilizations ρ for the sample application are also shown in Fig. 7. The last column shows normalized relative utilization, i.e. the resource utilizations that correspond to the saturation throughput Λ_{max}.

Cost analysis gives very valuable feedback to the designer. First it allows to locate the bottleneck (disk $D_{7,B}$ in the example), and to compute the maximum overall transaction arrival rate Λ_{max} (0.015 trans/sec in the example). More in general it shows which resources have problems and which transactions are responsible for them. A more detailed account is also produced by the tool that gives the resource demands of each operator in a query execution plan.

All this information can be utilized by the designer to take the appropriate actions in reconfiguring the system. In the case of Fig. 7, the cost analysis shows a very unbalanced I/O bound situation. Appropriate actions for load balancing could be:

- extend the size of the bufferpool area on the nodes with higher relative disk utilizations;
- add new disks and/or change file allocation on the disks of a given node, by modifying the number of disk segments;
- modify the node-group configuration, and the operator/node-group allocation;

As the designer changes the configuration the new costs and utilizations can be readily recomputed by the model. For instance using the buffer model of Section 7 it is possible to evaluate how the number of physical accesses on the bottleneck disk and its utilization change by increasing the bufferpool area size.

N_b	$\mathcal{T}_1$	$\mathcal{T}_2$	$\rho(D_{7,B}, N_b)$	$\rho(N_b)/\rho(1)$
1000	13693	94	4839	1.000
2000	13067	90	4586	.948
4000	11821	84	4149	.859
8000	9311	76	3720	.769
16000	4385	65	1544	.319

Figure 8: Bufferpool size (in 4k pages) and number of physical accesses for disk $D_{7,B}$.

The results of this analysis are shown in Fig. 8. The figure reports for different bufferpool area sizes the number of physical accesses to $D_{7,B}$ by $\mathcal{T}_1$ and $\mathcal{T}_2$ (the only transaction types accessing $D_{7,B}$), and its relative utilization. The last column in the table gives the decrease in the relative utilization compared to the original value of Figure 5, according to which the values of Fig. 7 have been computed. It is clear from the table that increasing the bufferpool area size to 16000 pages removes the bottleneck. There is indeed no use in a further increase of the bufferpool size, since, at that moment $D_{5,B}$ and $D_{6,B}$ have become the new bottlenecks.

9 Transaction Response Time

¿From the execution costs it is possible to compute the average transaction response time, for a given workload intensity, i.e. for a given overall transaction arrival rate Λ. We have used a hierarchical modeling approach which first takes into account the cross interaction between transactions executed concurrently through slow-down factors computed from resource utilizations, and then considers the interaction between operators.

Actually the main problem is to model both for the parallel execution of individual operators by all the nodes of the node-group, and the concurrent execution of different operators in the same parallel execution plan, which may be overlapped, for instance because they are pipelined or connected by a table-queue.

To do this one must consider two different levels of parallelism:

- *inter-operator parallelism*: the overlapping between the execution of different operators, e.g. because of pipeline or table-queue communication;
- *intra-operator parallelism*: the parallel execution of a given operator by all the nodes of the node-group.

9.1 Net operator execution times

The first step consists in computing how much the execution of each operator is slowed down because of the concurrent execution of other operators on the same node. We shall call this scaled time the *net execution time*, since it refers to the ideal condition in which the operator is not slowed-down by the execution of other operators that supply its input data.

As discussed in Section 6, each parallel operator $\omega \in \Omega_i$ of a transaction execution plan (like the one in Fig. 6) represents a set of operator instances $\omega(n_i)$ each executed on a node $n_i \in \gamma_\omega$ of the operator node-group.

If we model the system as an open product form queueing network, each service center can be solved as an independent M/M/1 queue. Therefore we may take into account the interference between concurrent operators on the same node through resource utilizations (given by (16) and (17)). Then, using well known queueing theory results, the net expected execution time of the operator instance is given by:

$$E[T(\omega,n_i)] = \frac{\sigma_i}{1-U(n_i)}\mathrm{CPU}(\omega,n_i) + \sum_k \frac{t_{i,k}}{1-U(D_{i,k})}\mathrm{DISK}(\omega,D_{i,k}) \qquad (19)$$

In the (19) we have made the assumption that the queueing times on resources of the same node are not overlapped. A more complex formula can be written to account for CPU/disk and disk/disk overlapping.

Net execution times correspond to actual execution times for instances of operators that have base relations as operands (as operators 1, 3 and 5 in Fig. 6), since there may be no further slow-down due to input shortage. In the general case, however, some of the operands are intermediate results produced by other operators, and therefore the execution time may extend because of lack of input data.

9.2 Operator interconnection

Let us start with some definitions. Given an operator instance $\omega(n_i)$ with $n_i \in \gamma_\omega$, we define the following random times:

- $T(\omega,n_i)$: the execution time of the operator when there are no waiting times for input data (the expectation of this time is given by the (19));
- $t_{start}(\omega,n_i)$: the time at which the execution of $\omega(n_i)$ begins;
- $t_{stop}(\omega,n_i)$: the time at which the execution of $\omega(n_i)$ ends;
- $\tau_{first}(\omega,n_i)$: time taken by $\omega(n_i)$ to produce the first block of output, after it has started;
- $\tau_{last}(\omega,n_i)$: time taken by $\omega(n_i)$ to complete after the last block of input has been delivered to ω.

Given an execution plan, *solving it for response times* means to compute $t_{start}(\omega,n_i)$ and $t_{stop}(\omega,n_i)$ for all the instances of its operators. From these the transaction response time can directly be derived, since all leaf operator instances start at transaction start time, and the transaction is completed when the last instance of the root operator ends.

All the times above are indeed random variables, and our final goal is to compute their expected values. Nevertheless it is important to explicitly represent in the model their stochastic nature, since the simplistic assumption that all times are deterministic and equal to their expectation, produces optimistic estimates that may be very misleading. Therefore, according to a consolidated tradition in queueing network models, we make the conservative assumption that all net and elapsed execution times of operator instances (i.e. $T(\omega,n_i)$ and $t_{stop}(\omega,n_i) - t_{start}(\omega,n_i)$) are exponentially distributed.

Similarly we consider an exponential distribution for τ_{first} and τ_{last} as well. Moreover if we call $N(\omega,n_i)$ the total number of output blocks produced by $\omega(n_i)$, and assuming a uniform execution rate, we may express their expected values as:

$$E[\tau_{first}(\omega,n_i)] = \frac{t_{stop}(\omega,n_i) - t_{start}(\omega,n_i)}{N(\omega,n_i)} \qquad (20)$$

$$E[\tau_{last}(\omega, n_i)] = \frac{T(\omega, n_i)}{N(\omega, n_i)} \qquad (21)$$

Now let us consider the simple case of a *consumer* operator α that takes its input from *producer* operator β. The starting and ending times of the instances of α depend on the starting and ending times of the instances of beta, and, of course, on the way the two operators communicate (see Sect. 3).

If the communication is through a temporary file, each instance $\alpha(n_i)$ of α may start only after the corresponding instance $\beta(n_i)$ of β has finished materializing its output. Therefore in this case:

$$t_{start}(\alpha, n_i) = t_{stop}(\beta, n_k) \qquad (22)$$

If the two operators are pipelined, this means that they are executed on the same node-group ($\gamma_\alpha = \gamma_\beta$), and each instance $\alpha(n_i)$ gets locally its input from $\beta(n_i)$ that runs on the same node. Therefore, for our purpose, they may be considered as a single instance with net execution time equal to $T(\alpha, n_i) + T(\beta, n_i)$. The same obviously holds for any number of pipelined operators.

The last case, connection through a table-queue, is the most complex. This is still some kind of pipelining, but the operators are executed on different node-groups, therefore there is no instance-to-instance connection like in the previous case, and the connection is at block level and not at tuple level. Each instance of the consumer operator may start only when the first block is produced by the *fastest* instance of the producer:

$$t_{start}(\alpha, n_i) = \min_{n_k \in \gamma_\beta} (t_{start}(\beta, n_k) + \tau_{first}(\beta, n_k)) \qquad (23)$$

Similarly each instance of the consumer operator may end only after processing the last input block supplied by the *slowest* instance of the producer, and anyway not earlier than its net execution time has passed:

$$\begin{aligned} t_{stop}(\alpha, n_i) = \max(&t_{start}(\alpha, n_i) + T(\omega, n_i), \\ &\max_{n_k \in \gamma_\beta} (t_{stop}(\beta, n_k) + \tau_{last}(\alpha, n_i))) \end{aligned} \qquad (24)$$

The results extend immediately to the case of a consumer operator with two producers, since a constraint in the execution model says that at least one must be a temporary file. Therefore a consumer instance may start only after the temporary file has been completed *and* the first block is on the table queue (if one of the producers actually communicates through it). That means the maximum between the two times has to be selected.

Using the above formulas the execution plan can be solved for the response times, in the sense defined above. This is accomplished by performing a postorder visit of the tree, i.e. starting from leaf operators whose starting time is known, since they have no producer and begin immediately at transaction starting time, and proceeding thereafter to the root operator.

9.3 Stochastic execution times

A last problem to be mentioned is the computation of the maximum and minimum functions in the (20) to (24), i.e. the computation of minimum starting time or maximum completion time of a set of operator instances concurrently executed. Given a set N of independent random variables $r_1 \dots r_N$ we are interested in the distribution of the minimum $r_{min} = \min_1 r_i$ i.e.:

$$\begin{aligned} P[r_{min} \le t] = P[r_1 \le t] \vee \dots \vee P[r_N \le t] \\ P[r_1 \le t] + P[r_2 \le t](1 - P[r_1 \le t]) + \dots \\ +P[r_N \le t](1 - P[r_1 \le t]) \dots (1 - P[r_{n-1} \le t]) \end{aligned} \qquad (25)$$

In the special case of exponentially distributed random variables:

$$P[r_i \le t] = 1 - e^{-\frac{t}{\rho_i}}, i = 1 \dots N \qquad (26)$$

the (25) becomes:

$$P[r_{min} \le t] = 1 - e^{-t \sum_i \frac{1}{\rho_i}} \qquad (27)$$

and for the expectation:

$$E[r_{min}] = \frac{1}{\sum_i \frac{1}{\rho_i}} \qquad (28)$$

With a similar, but slightly more complex approach, we get for the maximum:

$$E[r_{max}] = \sum_{i=1}^{N} (-1)^{i-1} \sum_{\pi_j \in \Pi_{\pi_0, i}} \frac{\pi_j}{\sum_{\pi_k \in \ Pi_{\pi_j, i-1}} \pi_k} \qquad (29)$$

where we denote by π_0 the product $\rho_1 \dots \rho_N$, by π_j the generic product of a subset of the terms of π_0, and with $\Pi_{\pi_j, i}$ the set of all product of i terms selected among the terms of π_j. Moreover we assume conventionally that the sum in the denominator of the (29) is 1 for $i = 1$.

To give an idea of the importance of an explicit representation of stochastic execution time we may point out that if all N random variables r_i have the same expectation ρ, then:

$$E[r_{max}] = N\rho, E[r_{min}] = \frac{\rho}{N} \qquad (30)$$

i.e. when compared to a deterministic distribution the expectations respectively increase and decrease by a factor of N.

Λ	$\mathcal{T}_1$	$\mathcal{T}_2$	$\mathcal{T}_3$	$\mathcal{T}_4$
.005	644.0	4.4	77.7	3.2
.010	1148.9	7.9	101.9	4.1
.014	28260.9	251.4	154.2	6.2

Figure 9: Transaction response times

Transaction response times for the sample application are shown in Fig. 9 for three different overall arrival rates. As one would expect from the bottleneck analysis of Section 8, response times for the arrival rate of .014 are very high since this value is quite near to the maximum rate, and in all cases transaction T1 has a response time considerably worse than the other transactions.

The response time estimates computed by the model are indeed of great help for the designer since they give, even in the early phases of the design, an important feedback on a very crucial performance index. Even if affected by some degree of approximation, such information is indeed very valuable in selecting the proper system configuration and carrying on a capacity planning study.

10 Conclusions

In this paper we have presented a systematic methodology for the performance analysis of parallel database applications. There are at least two good reasons for incorporating the modeling approach in the design procedure. First, performance evaluation is a far more crucial problem in parallel RDBMS than it is for sequential RDBMS, since high performance is the most likely reason why a parallel architecture was selected. Then a parallel database application *has* to be performing. Second with these innovative architectures designers and DB administrators have to deal with a far more complex physical organization and execution model, and this makes in some cases very difficult to predict the quantitative effect of their design decisions. Therefore it is very important to be able to get some feedback in the early stages of the design, even before the application is implemented.

Three kinds of performance measures are produced by the model:

- execution costs: i.e. the resource demands by every transaction to system resources; this may suggest changes in the physical organization (i.e. a different relation partitioning or node-group definition);
- resource utilizations: these allow to locate the bottlenecks and are of crucial help in improving the system configuration (bufferpool area size, disk allocation etc.), and in implementing load balancing strategies;
- transaction response times: these are end-user performance indices, and may be checked against the original design specifications or used in a capacity planning study.

The modelling task has proved to be considerably more complex than in the classical case of sequential RDBMS, and several new specific problems had to be solved. The most interesting parts are the bottleneck analysis and the evaluation of transaction response times that are thoroughly discussed in the paper.

To validate our methodology we have developed a prototype modelling tool, and we have run on it a series of experiments to compare, for several system configurations, the performance estimate computed by the tool, with the measures taken on a running application. For each experiment a stream of transactions with Poisson interarrival times was first generated off line and recorded. This was done referring to the transaction set and the relative arrival rates of Fig. 2, and randomly selecting the constants in the queries. The experiments consisted then in running a set of processes, that connected to the DBMS to simulate user sessions, and passed to it the transactions of the stream. Preliminary results show a good agreement between measures and performance estimates produced by the tool.

Acknowledgments

We would like to thank IBM Italy that has made the experimental part of our study possible by granting us access to the computing facilities of the ECSEC Research Center in Rome, and by supporting part of the work through a fellowship. We are especially indebted to Gianni Punzo that helped us with his precious advice and assistance in understanding the internal structure and the operation of SP2 and DB2-PE.

References

[1] T. Agerwala et alii. SP2 system architecture. *IBM System Journal*, 34(2):152–184, 1995.

[2] C.K. Baru et alii. DB2 Parallel Edition. *IBM System Journal*, 34(2):292–321, 1995.

[3] D. Bitton, H. Boral, D.J. DeWitt. Parallel algorithms for the execution of relational database

operations. *ACM Trans. on Database Syst.*, 8(3): 324–353, 1983.

[4] P. Borla-Salamet, M. Zait, M. Ziane. Parallel query processing in DBS3. In *Proc. of Parallel and Distributed Inforamtion System Conf.*, San Diego, Jan. 1993, pp.93–102.

[5] F. Carino,P. Kostamaa. Exegesis of DBC/1012 and P-90 - industrial supercomputer database machines. In *Proc. of Intl. Conf. on Parallel Architectures and Languages Europe*, pp. 877–892, 1992.

[6] F. Cesarini, S. Salza eds. *Database Machine Performance: Modeling Methodologies and Evaluation Strategies.* Lecture Notes in Computer Science n. 257, Springer-Verlag, Berlin 1987.

[7] M.S. Chen, M.L Lo, C.V. Ravishankar, P.S. Yu. On optimal processor allocation to support pipelined hash joins. In *Proc. of ACM SIGMOD Conf.* 1993, pp. 185–196.

[8] M.S. Chen, H. Hsiao, P.S. Yu. On parallel execution of multiple pipelined hash joins. In *Proc. of ACM SIGMOD Conf.* 1989, pp. 185–196.

[9] H. Chou and D. DeWitt. An evaluation of buffer management strategies for relational database systems. In *Proc. of Eleventh International Conference on Very Large Data Bases*, Stockolm, 1985, pp. 127–141.

[10] G. Graefe. Encapsulation of parallelism in the Volcano query processing system. In *Proc. of ACM SIGMOD Conf.*, 1990, pp. 102–111.

[11] G. Graefe. Query evaluation techniques for large databases. *ACM Computing Surveys*, 25(2):73–170, 1993.

[12] W. Hong, M. Stonebraker. Optimization of parallel query execution plans in XPRS. *Distributed and Parallel Databases*, 1(1):9–32, 1993.

[13] R. Katz,J. Ousterout,D. Patterson,M. Stonebraker. The design of XPRS. In *Proc. of Int. Conf. on Very Large Databases*, 1993, pp. 55–68.

[14] H. Lakshmi and P.S. Yu. Effectiveness of parallel joins. *IEEE Transactions on Knoledge and Data Engineering*, 2(4): 410–424, 1990.

[15] R.S. Lanzelotte, P. Valduriez, M. Zait. On the effectiveness of optimization search strategies for parallel execution spaces. In *Proc. of Int. Conf. on Very Large Databases*, Dublin 1993, pp. 493–504.

[16] H. Lu, K.L. Tan. On resource scheduling on multijoin queries in parallel database systems. *Information Processing Letters*, 48(4): 189–195, 1993.

[17] L. Mackert and G. Lohman. Index scan using a finite lru buffer: A validated I/O model. *ACM Trans. on Database Syst.*, 14: 401–424, 1989.

[18] G. Sacco and Schkolnick. Buffer management in relational database systems. *ACM Trans. on Database Syst.*, 11: 473–498, 1986.

[19] S. Salza, M. Terranova. Evaluating the size of queries on relational databases with non uniform distribution and stochastic dependence. In *Proc. of ACM SIGMOD Conf.*, Portland, 1989, pp. 8–14.

[20] S. Salza, R. Tomasso. A Modeling Tool for the Performance Analysis of Relational Database Applications. In *Computer Performance Evaluation: Modeling Techniques and Tools*, R. Pooley and J. Hillston eds., Edimburgh University Press, pp. 247–259, 1992.

[21] M. Stonebraker. The Case for Shared-Nothing. *Database Engineering*, (9)1, 1986.

[22] P. Valduriez. The case of shared something. In *Proc. of IEEE Conf. on Data Engineering*, 1993, pp. 460–465.

An Efficient Strategy for Beneficial Semijoins

Ye-In Chang, Bor-Miin Liu and Cheng-Huang Liao
Department of Applied Mathematics
National Sun Yat-Sen University, Kaohsiung, Taiwan, R.O.C.
E-mail: changyi@math.nsysu.edu.tw

Keywords: distributed databases, query optimization, relational databases, semijoins

Edited by: Tadeusz Morzy

Received: October 30, 1996 **Revised:** April 11, 1997 **Accepted:** May 7, 1997

Semijoins have traditionally been applied for reducing the communication cost required for distributed query processing. Semijoins whose execution will reduce the amount of data transmission required to perform a join sequence are termed beneficial semijoins for that join sequence. Beneficial semijoins include conventional profitable semijoins and gainful semijoins that are not profitable themselves but become beneficial due to the inclusion of join reducers. In this paper, based on the values of dynamic cumulative benefit (DCB), we propose an efficient algorithm for finding beneficial semijoins, i.e., to interleave a sequence of join operations with semijoins to reduce the data transmission cost. In this algorithm, a dynamic weighted graph is constructed based on the correlated relationship among semijoins where two semijoins are said to be correlated with each other if the condition for one to be beneficial depends on execution of the other. Then, we compute the dynamic profit for each subgraph, which is recursively constructed. When there are N vertexes in the initial dynamic weighted graph, where each vertex represents a semijoin, our algorithm needs to expand (N + 1) graphs to find a solution in the best case. From our simulation study, we show that our strategy can efficiently find beneficial semijoins and requires a lower data transmission cost than does the profitable semijoin approach.

1 Introduction

In a wide area network, under the assumptions that each site contains one relation, that there is only one copy of each relation, and that the cost of local processing is negligible compared to the transmission cost, a query is usually processed in the following three phases [16]: (1) *the local processing phase,* which involves all local processing such as selections and projections, (2) *the reduction phase,* where a sequence of *semijoins* is used to reduce the size of relations, and (3) *the final processing phase,* in which all the resulting relations are sent to the site where final query processing is performed. Significant research efforts have been focused on the problem of reducing the amount of data transmission required for phases (2) and (3) of distributed query processing [1, 3, 13, 14]. The *semijoin* operation especially has received considerable attention and has been extensively studied in the literature [2, 5, 15, 9]. The *semijoin* operator takes the join of two relations, R and S, and then projects back out on the domains of relation R. For a semijoin to be performed, only the projection of the joining attribute need be sent. If the size of these projections is small relative to the amount by which R and S are reduced, then the preliminary semijoin will be *profitable.*

The first algorithm using semijoins for distributed query processing was implemented in SDD-1 in [3]. This SDD-1 algorithm is based on a *hill-climbing* strategy that produces efficient, but not necessarily optimal, query processing strategies. Theoretical aspects of semijoins were first studied in [2]. *Simple queries* were studied in [13]. Their algorithm for general queries was improved in [1]. It has been proved that a *tree query* can be fully reduced by using semijoins [2], and there has been much research on optimizing semijoin sequences to process certain tree queries, such as *star queries* [6] and *chain queries* [11]. However, the determination of an optimal semijoin sequence to process certain tree queries and general query graphs with cycles has been proved to be NP-hard [12]. Methods based on *dynamic programming* to get an optimal semijoin sequence for *tree queries* and *chain queries* were studied in [10] and [11], respectively.

In addition to semijoins, join operations can also be used as reducers in distributed query processing to further reduce the communication cost [4, 7, 8]. Moreover, the approach of combining join and semijoin operations as reducers can result in more beneficial semijoins due to the inclusion of joins as reducers [7]. (Note that such semijoins are referred to as *gainful semijoins.*) Both *profitable semijoins* and *gainful semijoins* are called *beneficial semijoins.*

In this paper[1], based on the values of the *dynamic cumulative benefit* (*DCB*), we propose an algorithm for finding beneficial semijoins, i.e., to interleave a sequence of given join operations with semijoins to reduce the total data transmission cost. In this algorithm, a *dynamic weighted graph* $G = (V, E)$ is constructed based on the *correlated* relationship among semijoins, where V is the set of semijoins (i.e., each vertex represents a semijoin) associated with its *dynamic cumulative benefit* (DCB) and E is the set of *correlated edges.* Note that two semijoins are said to be *correlated* with each other if the condition for one to be beneficial depends on execution of the other [8]. Based on the values of DCB, there is a *dynamic profit* associated with the whole graph. Then, the graph is transformed to another graph by a *vertex shrinking.* A *vertex shrinking* is done by eliminating a certain vertex and connecting *correlated edges.* This transformation is recursively executed and the related *dynamic profit* is updated at the same time until all vertexes are shrunken or all the values of DCB associated with vertexes are negative. When the algorithm stops, the set of semijoins which provide the maximum total *dynamic profit* is the set of beneficial semijoins. When there are N vertexes in the initial *dynamic weighted graph,* where each vertex represents a semijoin, our algorithm needs to expand (N + 1) graphs to find a solution in the best case. Moreover, since the DCB values of vertexes will be dynamically updated in the process of a *vertex shrinking,* it will speed up the execution and reduce the large space requirement of the proposed recursive algorithm in general cases. From our simulation study, we show that our strategy can efficiently find beneficial semijoins and requires a lower data transmission cost than does the profitable semijoin approach.

The rest of the paper is organized as follows. In Section 2, we give some definitions used in this paper. In Section 3, we present our proposed algorithm. Finally, in Section 4, we give a conclusion.

2 Background

In this section, we describe assumptions and definitions used in the paper, which are mostly adapted from [7, 8] since the concept and properties of *gainful semijoins* and *beneficial semijoins* were proposed and discussed in [7, 8].

2.1 Query Graphs, Joins and Semijoins

Given a query Q with qualification q, we define its corresponding *query graph* $G_Q(V_Q, E_Q)$ as follows:

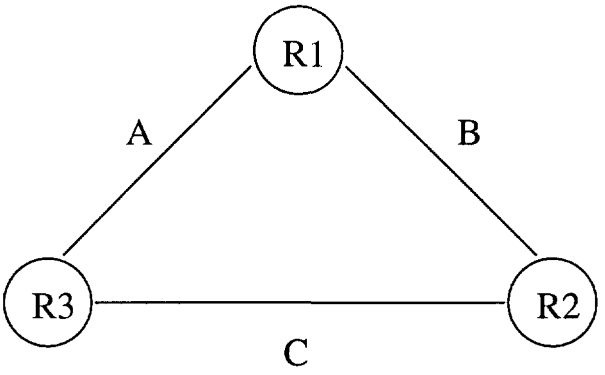


Figure 1: A query graph q_A.

V_Q = the set of all the relation names referenced by q;

$E_Q = \{(i, j) \mid i \neq j$ and some clause of q references both R_i and R_j $\}$.

Figure 1 shows the query graph for the following query, where $R_1.B$ is the target list.

select $R_1.B$
from R_1, R_2, R_3
where $R_1.A = R_3.A$ and $R_1.B = R_2.B$ and $R_2.C = R_3.C$

We assume that we have the following information about the relations.

For each relation R_i, i=1,2,...,m,
$|R_i|$: number of tuples;
w_{R_i}: size (e.g., in bytes) of R_i.

For each attribute A of relation R_i,
$|R_i(A)|$: cardinality;
$\rho_{i,A}$: selectivity;
$w_{R_i(A)}$: size (e.g., in bytes) of the data item in attribute A of relation R_i.

The *cardinality* of attribute A of relation R_i, denoted as $|A|$, is the number of distinct values in attribute A of relation R_i and the *selectivity* $\rho_{i,A}$ of attribute A is defined as the number of different values occurring in the attribute divided by the number of all possible values of the attribute.

A join clause "R_1 *joins* R_2 *on* A" is denoted by $R_1 \overset{A}{\longleftrightarrow} R_2$, where R_1 and R_2 are relations, and attribute A is the joining attribute. Associated with this join are two semijoins: R_1 by R_2 on A, and R_2 by R_1 on A, denoted by $R_2 \xrightarrow{A} R_1$, and $R_1 \xrightarrow{A} R_2$, respectively. $R_1 \xrightarrow{A} R_2$ entails shipping $R_1(A)$, attribute A of R_1, to the site where R_2 resides and joining $R_1(A)$ with R_2. We denote the resulting relation by R_2' (and R_1 is unchanged). After the semijoin $R_i \xrightarrow{A} R_j$ is executed, then the parameters of relation R_j are changed in the following way:

[1] This research was supported by the National Science Council of the Republic of China, Grant No. NSC 84-2213-E-110-023.

$$|R_j| \longleftarrow |R_j| * \rho_{i,A};$$
$$\rho_{j,A} \longleftarrow \rho_{j,A} * \rho_{i,A};$$
$$|R_j(A)| \longleftarrow |R_j(A)| * \rho_{i,A}.$$

2.2 Cost and Benefit of Semijoin Reducers

We assume that the transmission cost is given by cost(n) = $c_0 + c_1 * n$, where n is the amount of data transmitted and c_0 and c_1 are constants. Let the transmission cost be one per data unit transmitted. Consider a semijoin $R_i \xrightarrow{A} R_j$ when R_i and R_j are at different sites. Then, the *cost* of the semijoin is (*size_of* $R_i[A]$), and the *benefit* is (*size_of* R_j *before semijoin* $-$ *size_of* R_j *after semijoin*), where the sizes of the relations are measured in bytes. A semijoin $R_i \xrightarrow{A} R_j$, is called *profitable* if its cost of sending $R_i(A)$, $w_{R_i(A)}|R_i(A)| = w_{R_i(A)}|A|\rho_{i,A}$, is less than its benefit, $w_{R_j}|R_j| - w_{R_j}|R_j|\rho_{i,A} = w_{R_j}|R_j|(1-\rho_{i,A})$, where $w_{R_j}|R_j|$ and $w_{R_j}|R_j|\rho_{i,A}$ are the size of R_j before and after the semijoin, respectively. In addition, we assume that the values of attributes are uniformly distributed over all the tuples in a relation, and that the values of one attribute are independent of those in another attribute. Thus, as in most prior work [1, 2], we assume in this paper that the selectivity and the cardinality of a non-semijoin attribute remain the same after a semijoin operation to simplify our discussion.

2.3 The Effect of Join Operations

To determine the effort of a join operation specified by a query graph, the following theorem was described in [7].

Theorem 1 *Let $G_J = (V_J, E_J)$ be a join query graph. $G_B = (V_B, E_B)$ is a connected subgraph of G_J. Let $R_1, R_2, ..., R_p$ be the relations corresponding to nodes in V_B, let $A_1, A_2, ..., A_q$ be the distinct attributes associated with edges in E_B, and let m_i be the number of different nodes (relations) to which edges with attribute A_i are incident. Suppose R^* is the relation resulting from the join operations between relations in G_B, and that $N_T(G_B)$ is the expected number of tuples in R^*; then*

$$N_T(G_B) = \frac{\prod_{i=1}^{p} |R_i|}{\prod_{i=1}^{q} |A_i|^{m_i - 1}} \quad . \qquad (1)$$

For the query shown in Figure 2, the expected number of tuples in the resulting relation is $(|R_1||R_2||R_3||R_4|)/(|A|^2|B||C||D|)$.

2.4 Join Reducers and Gainful Semijoins

The application of join operations as reducers may mean that more profitable semijoins will be available. Those semijoins which become profitable due to the use of join reducers are termed *gainful semijoins* [7]. Consider the query graph q_A shown in Figure 1 with its profile in Table 1 as an example. It can be verified that the semijoin $R_3 \xrightarrow{A} R_1$ is not profitable since $w_{R_3(A)}|R_3(A)| > w_{R_1}(1-\rho_{3,A})|R_1|$. Note that although this semijoin is not profitable, it is *gainful* if we perform $R_1 \Rightarrow R_2$ and $R_2' \Rightarrow R_3$ after this semijoin operation, where $R_1 \Rightarrow R_2$ means shipping R_1 to the site where R_2 resides and joining R_1 with R_2. It can be obtained that for the total communication costs required,

$$|R_3(A)| + 2|R_1|\rho_{3,A} + 3|R_1 \ join \ R_2|\rho_{3,A} \approx 2190$$

$$2190 < 2|R_1| + 3|R_1 \ join \ R_2| = 2542,$$

meaning that it is advantageous, as far as the cost of data transmission is concerned, to perform $R_3 \xrightarrow{A} R_1$, $R_1' \Rightarrow R_2$ and then $R_2' \Rightarrow R_3$, instead of performing directly $R_1 \Rightarrow R_2$ and $R_2' \Rightarrow R_3$. Thus, it can be seen that whether a semijoin is gainful or not depends on the subsequent join operations.

2.5 A Join Sequence Tree

Every edge in a tree is directed, and all the arrows in edges are away from a single node, which is called the *root* of the tree [7]. Note that a rooted tree can be viewed as a partial order set. We denote $n_i \geq n_j$, if there is a path along the arrows in the tree from n_i to n_j. In such a case, node n_j (n_i) is called an *offspring* (*ancestor*) of n_i (n_j). We use $n_i > n_j$ to mean $n_i \geq n_j$ and $n_i \neq n_j$. Let T_{n_i} denote the subtree formed by n_i and its offspring (ancestor) in a rooted tree T, and let S(T_{n_i}) be the set of nodes in T_{n_i}, i.e., $S(T_{n_i}) = \{n_j \mid n_i \geq n_j, \ n_j \in S(T)\}$. We define the *lowest common ancestor* of two nodes n_i and n_j in a rooted tree, denoted by $n_i \vee n_j$, to be the node that is an ancestor of n_i and n_j and for which none of its offspring is an ancestor of n_i and n_j [7].

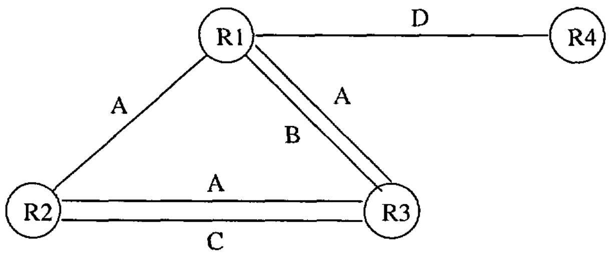


Figure 2: A query graph q_B (adapted from [8]).

Relation R_i	$\|R_i\|$	Size $w_{R_i}\|R_i\|$	Attribute X	$\|R_i(X)\|$	Selectivity $\rho_{i,x}$	W_X
R_1	620	1240	A	400	0.80	1
			B	600	0.60	1
R_2	700	1400	B	580	0.58	1
			C	450	0.75	1
R_3	778	1556	A	360	0.72	1
			C	480	0.80	1

Table 1: Profile for query q_A, where $|A| = 500$, $|B| = 1000$, and $|C| = 600$ (adapted from [7]).

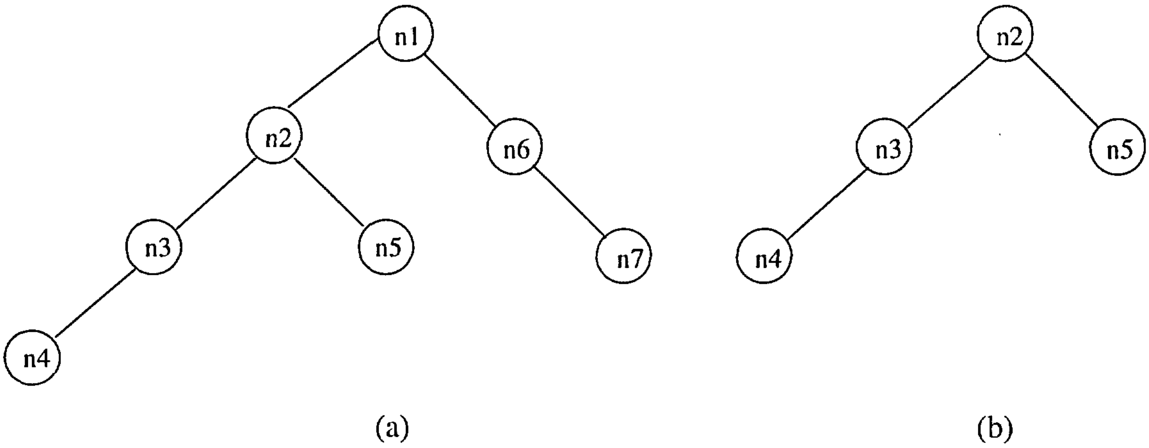


Figure 3: A rooted tree: (a) T; (b) T_{n_2} (adapted from [7]).

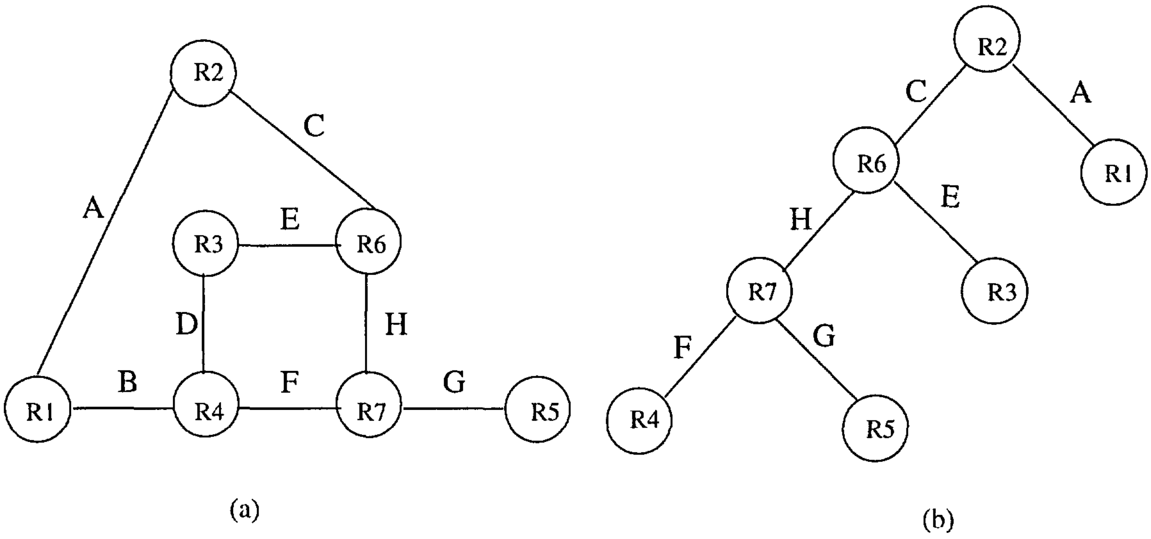


Figure 4: An example: (a) a query graph G_{EX1}; (b) the related join sequence tree (adapted from [7]).

For example, for a rooted tree T shown in Figure 3-(a), T_{n_2} is given in Figure 3-(b), and $S(T_{n_2}) = \{n_2,\ n_3,\ n_4,\ n_5\}$. Also, $n_4 \vee n_5 = n_2$ and $n_5 \vee n_7 = n_1$ in Figure 3-(a). In addition, when $n_i \geq n_j$ in a rooted tree T, we use $P(n_i,\ n_j)$ to denote the set of nodes that are on the path from n_i to n_j excluding n_i, i.e., $P(n_i,\ n_j) = \{n_k \mid n_i > n_k \geq n_j \textit{ and } i \neq k,\ \forall n_k \in S(T)\}$. In the rooted tree shown in Figure 3-(a), $P(n_2,\ n_4) = \{n_3,\ n_4\}$ and $P(n_1,\ n_5) = \{n_2,\ n_5\}$.

A join sequence tree is obtained from a join sequence [7]. Once a join sequence is determined, it can be mapped into its corresponding join sequence tree, which is defined as follows [7].

A join sequence tree is a rooted tree where each node denotes a relation and each edge implies a join between the two relations to which the edge is incident. The tree represents a sequence of join operations which are performed in such a way that each relation in a node is sent to its parent node in the tree for a join operation in the bottom-up sense.

Given a query shown in Figure 4-(a) and a join sequence $R_4 \Rightarrow R_7$, $R_5 \Rightarrow R_7$, $R_7 \Rightarrow R_6$, $R_3 \Rightarrow R_6$, $R_1 \Rightarrow R_2$, $R_6 \Rightarrow R_2$, the corresponding join sequence tree is shown in Figure 4-(b). Recall that T_{R_i} is the subtree formed by R_i and its offspring in the join sequence tree, and that $S(T_{R_i})$ is the set of nodes in T_{R_i}. The *weight* of a relation R_i in the join sequence tree, denoted by $W(R_i)$, is defined as the size of the relation resulting from joining all the relations in $S(T_{Ri})$ (and is computed by Equation 1 as described in Section 2.3). For the join sequence tree shown in Figure 4-(b), $W(R_7) = w_{R_7'}|R_7'|$ and $W(R_6) = w_{R_6'}|R_6'|$, where R_7' is the relation resulting from joining R_4, R_5, and R_7, and R_6' is the one resulting from joining R_3, R_4, R_5, R_6, and R_7. For convenience, the weight of the root of a join sequence tree, which corresponds to the final site, is defined to be zero. Also, to facilitate our study on the effect of semijoin operations, we define the configuration of a query, $J_Q(SMJ)$, to be the structure of the query and its profile associated after the set of semijoins SMJ has been performed. When it is necessary, we use $W(R_i, J_Q(SMJ))$, instead of $W(R_i)$, to mean the weight of R_i after the semijoins in SMJ are performed.

2.6 Properties of Beneficial Semijoins

A relation is said to be *reducible* by a semijoin SJ_i if the size of the relation in the join sequence tree is affected by the execution of the semijoin. Then, the set of reducible relations of a semijoin under a join sequence tree can be determined by the following theorems [7]:

Theorem 2 *Given a join sequence tree T, the set of reducible relations of a semijoin $R_i \longrightarrow R_j$, denoted by $Rd(R_i \longrightarrow R_j)$, is $P(R_i \vee R_j, R_j)$.*

For example, suppose Figure 4-(b) is the join sequence tree derived from Figure 4-(a); then, $Rd(R_1 \longrightarrow R_4) = \{R_4, R_6, R_7\}$, $Rd(R_4 \longrightarrow R_3) = \{R_3\}$ and $Rd(R_2 \longrightarrow R_3) = \{R_3, R_6\}$.

Theorem 3 *A semijoin SJ_k, $R_i \xrightarrow{A} R_j$ in the configuration $J_Q(SMJ)$ is beneficial if and only if $w_{R_i(A)}|R_i(A)| \leq (1 - \rho_{i,A}) \sum_{R_p \in Rd(SJ_k)} W(R_p, J_Q(SMJ))$.*

Corollary 1 *Suppose that R_i and R_j are two relations in a join sequence tree T and $R_i \geq R_j$. Then, $R_j \longrightarrow R_i$ is not a beneficial semijoin for T.*

Two semijoins are said to be *correlated* with each other if the condition for one to be beneficial depends on execution of the other. Thus, using **Theorem 3**, we can determine by the following corollary [7] whether two semijoins are correlated with each other in a join sequence tree.

Corollary 2 *In a join sequence tree, two semijoins, SJ_i and SJ_k, are correlated with each other if and only if $Rd(SJ_i) \cap Rd(SJ_k) \neq \emptyset$.*

3 The Algorithm for Beneficial Semijoins

Given a join sequence, we can map the join reducer sequence into the corresponding join sequence tree. According to **Theorem 2**, we can derive reducible relations from the join sequence tree. Based on the relationship among these reducible relations, we propose a new algorithm for interleaving a sequence of join operations with semijoins, i.e., for locating beneficial semijoins. The proposed algorithm is based on a value, called the *dynamic cumulative benefit*, which will be defined later.

Let SM_T be the set of possible semijoins in the given join sequence tree T except those semijoins $R_j \longrightarrow R_i$ which occur between two relations R_i and R_j in the join sequence tree T and $R_i \geq R_j$ (i.e., R_i is an ancestor of R_j). (Note that, based on Corollary 1, we do not want to include these non-beneficial semijoins in SM_T.) Let SMJ be the set of beneficial semijoins identified so far. Initially, SMJ is an empty set. We define the *dynamic cumulative benefit* of a semijoin SJ_i ($R_k \xrightarrow{A} R_j$), denoted by $DCB(SJ_i)$, as the amount of reduction minus the cost of semijoin SJ_i if this semijoin is applied to the execution of a given join sequence and the profile of the semijoin is the one resulting from the semijoin executions SJ_k, for $SJ_k \in SMJ$. That is, $DCB(SJ_i) = DCB(R_k \xrightarrow{A} R_j) = (1 -$

Relation R_i	$\|R_i\|$	Size of relation	Attribute X	$\|R_i(X)\|$	Selectivity	W_X
R_1	1150	2300	A	420	0.70	1
			B	325	0.65	1
R_2	1200	2400	A	300	0.50	1
			C	385	0.55	1
R_3	850	1700	D	585	0.65	1
			E	550	0.55	1
R_4	1100	3300	B	300	0.60	1
			D	405	0.45	1
			F	770	0.55	1
R_5	900	900	G	585	0.45	1
R_6	900	2700	C	490	0.70	1
			E	400	0.40	1
			H	525	0.50	1
R_7	1000	3000	F	630	0.45	1
			G	650	0.50	1
			H	630	0.60	1

Table 2: Profile for query graph G_{EX1}, where $|A| = 600$, $|B| = 500$, $|C| = 700$, $|D| = 900$, $|E| = 1000$, $|F| = 1400$, $|G| = 1300$, and $|H| = 1050$.

Semijoin SJ_i	in SM_T	$DCB(SJ_i)$
$R_1 \to R_2$	N	-
$R_1 \to R_4$	Y	2753
$R_2 \to R_1$	Y	850
$R_2 \to R_6$	Y	863
$R_3 \to R_4$	Y	1522
$R_3 \to R_6$	N	-
$R_4 \to R_1$	Y	620
$R_4 \to R_3$	Y	530
$R_4 \to R_7$	N	-
$R_5 \to R_7$	N	-
$R_6 \to R_2$	N	-
$R_6 \to R_3$	Y	620
$R_6 \to R_7$	Y	835
$R_7 \to R_4$	Y	1185
$R_7 \to R_5$	Y	-200
$R_7 \to R_6$	N	-

Table 3: SM_T for query graph G_{EX1}.

No.	**Semijoin (SJ_i)**	**Reducible relations**
1	$R_1 \longrightarrow R_4$	$\{ R_4, R_6, R_7 \}$
2	$R_2 \longrightarrow R_6$	$\{ R_6 \}$
3	$R_3 \longrightarrow R_4$	$\{ R_4, R_7 \}$
4	$R_6 \longrightarrow R_7$	$\{ R_7 \}$
5	$R_7 \longrightarrow R_4$	$\{ R_4 \}$

(a)

No.	**Semijoin (SJ_i)**	**Reducible relations**
1	$R_6 \longrightarrow R_3$	$\{ R_3 \}$
2	$R_4 \longrightarrow R_3$	$\{ R_3 \}$

(b)

No.	**Semijoin (SJ_i)**	**Reducible relations**
1	$R_4 \longrightarrow R_1$	$\{ R_1 \}$
2	$R_2 \longrightarrow R_1$	$\{ R_1 \}$

(c)

Table 4: Four groups in SM_T: (a) GP_1; (b) GP_2; (c) GP_3.

$\rho_{k,A}) \sum_{R_p \in Rd(SJ_i)} W(R_p, J_Q(SMJ)) - Cost_i$. Given the query graph shown in Figure 4-(a) with its profile shown in Table 2, Table 3 shows SM_T and the value of DCB for each semijoin.

For all semijoins in SM_T, the sets of reducible relations of semijoins can be further classified according to whether the semijoin is correlated with some other semijoins or not based on **Corollary 2**. We assign those semijoins which are correlated to the same group. For example, given a query graph shown in Figure 4-(a), Figure 4-(b) shows the related join sequence tree, and Tables 4-(a), (b), and (c) shows three groups of semijoins which belong to SM_T and are correlated in the same group. Moreover, some semijoins in SM_T may not be correlated with some other semijoins. If such a semijoin exists and it is beneficial (i.e., $DCB > 0$), we add such a semijoin into SMJ.

For those semijoins which are in the same group, we want to find a good combination of beneficial semijoins such that we can obtain the largest profit and then add them into SMJ. For each group GP_i of semijoins SJ_k with $DCB(SJ_k) > 0$, we construct a

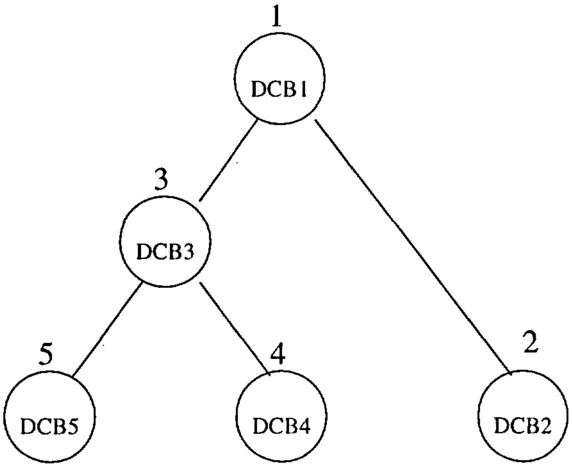


Figure 5: A dynamic weighted graph for GP_1.

dynamic weighted graph. Given a GP_i, a *dynamic weighted graph* $G = (V_{GP_i}, E_{GP_i})$ is constructed based on the *correlated* relationship among the semijoins, where V_{GP_i} is the set of semijoins with positive values of DCB (i.e., each vertex represents a semijoin) associated with its *dynamic cumulative benefit* (DCB), and E_{GP_i} is the set of *correlated edges. A correlated edge* exists between two vertexes v_i (denoting SJ_i) and v_j (denoting SJ_j) such that the intersection of the sets of reducible relations of these two semijoins represented by v_i and v_j is not empty. As an example, Figure 5 shows the dynamic weighted graph for group GP_1 shown in Table 4-(a).

Given the dynamic weighted graph G, G_p is a graph constructed from G by eliminating vertex p and connecting correlated edges (p, x) and (p, y), where the intersection of the sets of reducible relations of these two vertexes x and y (representing two semijoins) is not empty. The step of constructing G_p from G is called a *vertex shrinking.* Figure 6 shows two examples of vertex shrinking for the dynamic weighted graph shown in Figure 5. After the process of *vertex shrinking*, the DCB values of those vertexes which are correlated with the shrunken vertex in the group are also updated. (Note that the DCB values of the vertexes will be the same or be smaller in each update.) We define the *dynamic profit* (denoted as $DP(G)$) of the most profitable set of semijoins for a given dynamic weighted graph G as follows:

$$DP(G) = \max_{p \in G}\{DP(G_p) + DCB_p\},$$

where we always consider positive values of DCB only. (Note that when the DCB value of a vertex is negative, we will give up the vertex shrinking for this vertex in this G_p.) For group GP_1, according to the above formula, we illustrate the process of finding the largest profit $DP(G)$ from Figure 7-(a) to Figure 7-(m).

$DP(G) = \max\ \{DP(G_1) + 2753,\ DP(G_2) + 863,\ DP(G_3) + 1522,$
$DP(G_4) + 835,\ DP(G_5) + 1185\};$

$DP(G_2) = \max\ \{DP(G_{2_1}) + 2317,\ DP(G_{2_3}) + 1522,\ DP(G_{2_4}) + 835,$
$DP(G_{2_5}) + 1185\};$

$DP(G_{2_1}) = \max\ \{DP(G_{2_{1_3}}) + 785,\ DP(G_{2_{1_4}}) + 359,\ DP(G_{2_{1_5}}) + 550\};$

$DP(G_{2_{1_3}}) = 137;\ DP(G_{2_{1_4}}) = 550;\ DP(G_{2_{1_5}}) = 32.$

Thus

$DP(G_{2_1}) = 922$, and $DP(G_2) = 3239$.

Consequently,

$DP(G) = \max\ \{\ 3229,\ 3239,\ 3051,\ 2960,\ 2081\ \} = 3239.$

That is,

$$\begin{aligned} DP(G) &= DP(G_2) + DCB_2 \\ &= DP(G_{2_1}) + DCB_1 + DCB_2 \\ &= DP(G_{2_{1_3}}) + DCB_3 + DCB_1 + DCB_2 \\ &= DCB_5 + DCB_3 + DCB_1 + DCB_2 \end{aligned}$$

Therefore, for group GP_1 shown in Table 4-(a), the most profitable set of semijoins (i.e., SMJ_1) is $\{\ R_1 \rightarrow R_4,\ R_2 \rightarrow R_6,\ R_3 \rightarrow R_4,\ R_7 \rightarrow R_4\ \}$ $(= \{SJ_1, SJ_2, SJ_3, SJ_5\})$. Similarly, for the groups shown in Table 4-(b), Table 4-(c), the most profitable sets of semijoins are $\{\ R_6 \rightarrow R_3\ \}$ $(= SMJ_2)$, $\{\ R_2 \rightarrow R_1\ \}$ $(= SMJ_3)$, respectively. Therefore, $SMJ = SMJ_1 \cup SMJ_2 \cup SMJ_3$. Finally, we interleave those semijoins in SMJ in the given join sequence such that every semijoin $R_i \rightarrow R_j$ precedes every related join $R_j \Rightarrow R_k$.

Note that when the dynamic weighted graph does not have any correlated edge or when the values of DCB of all the vertexes are negative, we stop the execution of $DP(G)$. Moreover, when a vertex shrinking is executed, those related DCB of semijoins will be updated.

Let's consider the following three cases to compare our strategy (Case 3) with other strategies for distributed joins, where we use the query graph shown in Figure 4 with its profile shown in Table 2.

- **Case 1:** Using profitable semijoins.
  Execute profitable semijoins $R_2 \rightarrow R_1$, $R_4 \rightarrow R_1$, $R_4 \rightarrow R_3$, $R_6 \rightarrow R_3$, $R_1 \rightarrow R_4$, $R_3 \rightarrow R_4$, $R_7 \rightarrow R_4$, $R_2 \rightarrow R_6$, $R_6 \rightarrow R_7$ based on the strategy proposed in [3]. Then, send relations R_1, R_3, R_4, R_5, R_6, R_7 to the site containing R_2. The total transmission cost is 300 + 300 + 405 + 400 + 325 + 585 + 630 + 385 + 525 + 690 + 306 + 626 + 900 + 1485 + 1500 = 9362.

- **Case 2:** Applying join operations as reducers without semijoins [7, 8].
  Execute the join sequence $R_4 \Rightarrow R_7$, $R_5 \Rightarrow R_7$, $R_7' \Rightarrow R_6$, $R_3 \Rightarrow R_6$, $R_6' \Rightarrow R_2$, $R_1 \Rightarrow R_2$. The

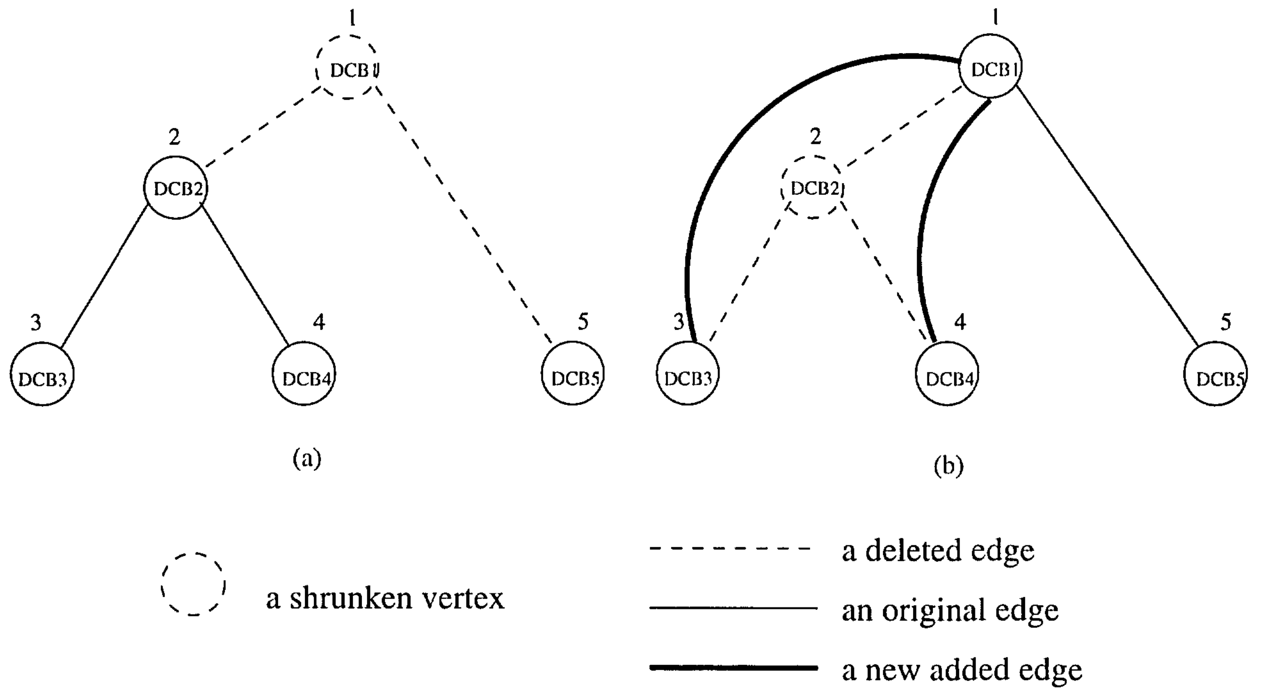


Figure 6: A vertex shrinking: (1) vertex 1; (2) vertex 2.

total transmission cost is 3300 + 900 + 2720 + 1700 + 2772 + 2300 = 13692.

- **Case 3:** Interleaving a join sequence with semijoins.
  Execute $R_1 \to R_4, R_3 \to R_4, R_7 \to R_4, R_4' \Rightarrow R_7$, $R_5 \Rightarrow R_7$, $R_7' \Rightarrow R_6$, $R_6 \to R_3$, $R_3' \Rightarrow R_6$, $R_2 \to R_6$, $R_6' \Rightarrow R_2$, $R_2 \to R_1$, $R_1' \Rightarrow R_2$. The total transmission cost is 325 + 585 + 630 + 627 + 900 + 1150 + 400 + 680 + 385 + 259 + 300 + 1150 = 7391.

(Note that in cases 2 and 3, we use the join sequences derived from the given join sequence tree as shown in Figure 4-(b).) From the above comparison, we show that the data transmission cost by using our strategy can be less than that by using profitable semijoins only or by using join reducers only. In general, in our algorithm, when there are N vertexes in the initial *dynamic weighted graph*, where each vertex represents a semijoin, our strategy needs to expand $1 + N! * (1/(N-1)! + 1/(N-2)! + 1/(N-3)! + \cdots + 1/2! + 1/1!)$ graphs to find the solution in the worst case and (N + 1) graphs in the best case. Since in the worst cast, there is no vertex with a negative value of DCB, it results in (1 + N + (N * (N - 1)) + (N * (N - 1) * (N - 2)) + .. + N!) graphs, where there is one initial graph in level 0 and there are (N! / (N - i)!) graphs in the ith recursive execution, $1 \leq i \leq$ N. While in the best case, after executing a vertex shrinking for each vertex in the initial graph, each resulting graph contains vertexes with all negative values. Therefore, (1 + N) graphs are created. (Note that only those semijoins which have positive values of DCB will be included as a vertex in the initial graph.) For the initial state shown in Figure 4 with its profile shown in Table 2, we need to expand 75 graphs to find the solution where 206 graphs are needed in the worst case with some other profile.

Table 6 shows a comparison of the data transmission cost using five different profiles, where Profile C is the one shown in Table 2. For Profiles A and B, we increase the selectivity in all the attributes of Profile C to 0.2 and 0.1, respectively. For Profiles C and D, we decrease the selectivity in all the attributes of Profile C to 0.1 and 0.2, respectively. From the above comparison, we can see that the data transmission cost using our strategy can be less than that using profitable semijoins only or using join reducers only. Table 7 shows a comparison of the number of states created in one case of our algorithm, which was chosen randomly, the worst case of our algorithm and exhaustive search. From this table, we can see that our strategy is very efficient.

In [7], Chen et al. have proposed a heuristic algorithm to interleave a sequence of join operation with semijoins. In their algorithm, they define the *cumulative benefit* of a semijoin SJ_i ($R_k \xrightarrow{A} R_j$), denoted by $CB(SJ_i)$, as the amount of reduction if it is applied individually prior to the execution of a given join sequence, i.e., $CB(SJ_i) = (1 - \rho_{k,A}) \sum_{R_p \in Rd(SJ_i)} W(R_p, J_Q(\emptyset))$. In this algorithm, they use the *cumulative benefit* (CB) as a heuristic to determine the set of semijoins to be interleaved into a given join reducer sequence. However, in their al-

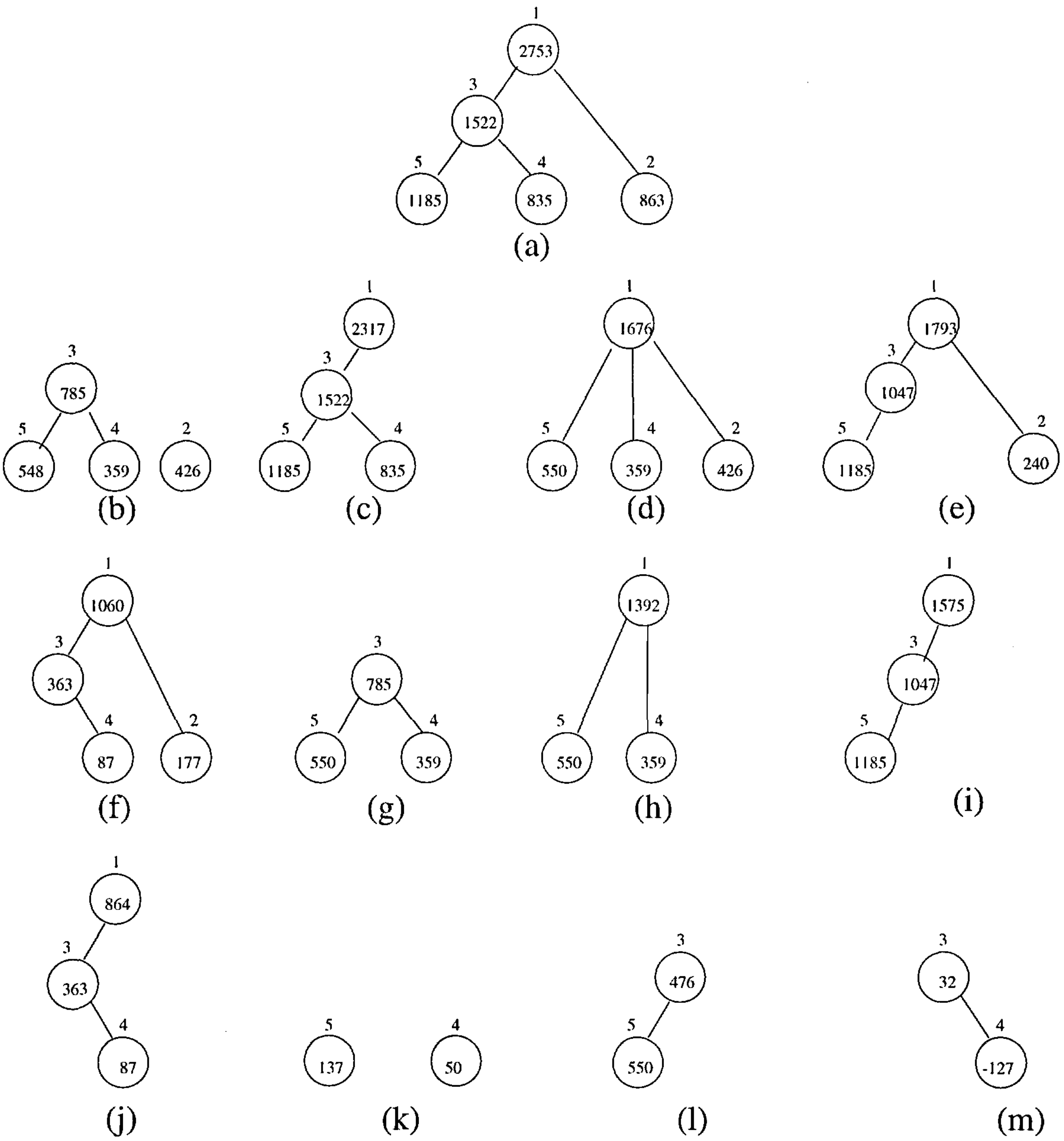


Figure 7: A dynamic weighted graph and its subgraphs: (a) G; (b) G_1; (c) G_2; (d) G_3; (e) G_4; (f) G_5; (g) G_{2_1}; (h) G_{2_3}; (i) G_{2_4}; (j) G_{2_5} (k) $G_{2_{1_3}}$; (l) $G_{2_{1_4}}$; (m) $G_{2_{1_5}}$.

gorithm, CB depends on a static profile of semijoins; while in our approach, DCB depends on a dynamic profile of semijoins. Based on those dynamic values of DCB, we can have fewer computation steps than Chen et al.'s algorithm [7] if there exists vertexes with negative values of DCB. Moreover, our algorithm can find better solution than their algorithm. For example, for the same query graph shown in Figure 4 with its profile shown in Table 2, their algorithm will execute $R_1 \rightarrow R_4$, $R_3 \rightarrow R_4$, $R'_4 \Rightarrow R_7$, $R_5 \Rightarrow R_7$, $R'_7 \Rightarrow R_6$, $R_6 \rightarrow R_3$, $R'_3 \Rightarrow R_6$, $R_2 \rightarrow R_6$, $R'_6 \Rightarrow R_2$, $R_2 \rightarrow R_1$, $R'_1 \Rightarrow R_2$. The total transmission cost is 325 + 585 + 1395 + 900 + 1150 + 400 + 680 + 385 + 259 + 300 + 1150 = 7529, which is larger than our solution.

In [4], based on the heuristic values of DCB, we have proposed a variant of the A^* algorithm to find beneficial semijoins. In the variant of the A^* algorithm, which is a well-known heuristic search method, the search is controlled by a heuristic function $f(x)(= g(x) + h(x))$. The state x chosen for expansion is the one which has the *largest* value of $f(x)$ among all the generated states that have not been expanded so far. In this strategy, a state is defined as a set of correlated semijoins; i.e., it is represented by a dynamic weighted graph. When a state x is expanded, all the states that can be reached from state x by a vertex shrinking are generated. (Note that vertex shrinking can be executed only when the vertex has a positive value of DCB.) Among those leaf nodes x which have not been expanded thus far, the one with the largest $f(x)$ value will be chosen for future expansion. This procedure is repeated until the goal state is reached. The initial state is the given dynamic weighted graph, and the goal state is a dynamic weighted graph which does not have any correlated edge or in which the values of DCB of all the vertexes are negative. Let $SMJ_k(x)$ be the set of beneficial semijoins applied to the initial state to reach a state x in GP_k, and let $C(SMJ_k(x))$ be the total DCB values for the semijoin operation in $SMJ_k(x)$. Then, at a state x, we let $g(x) = C(SMJ_k(x))$. Let $h(x)$ be the largest profit of future semijoins obtained in reaching the goal state from x. Let $DW(G)$ be the *dynamic weight* of the set of semijoins for a given *dynamic weighted graph* G, which is defined as follows:

$$DW(G) = \{DW(G_p) + DCB_p\},$$

where $p \in G$, DCB_p is the largest value among all the current DCB_i in G, and where we always consider positive values of DCB_p only. Then, at a state x, we let $h(x) = DW(x)$. For the same query graph shown in Figure 4 with its profile shown in Table 2, the set of beneficial semijoins found by this heuristic strategy [4] is the same as Chen et al.'s [7]. As compared to the performance of our previous proposed heuristic strategy which is also based on the values of DCB, although the heuristic strategy will expand fewer states than the new proposed one, our new proposed strategy can find better solution than the heuristic one.

4 Conclusion

In this paper, based on the values of *dynamic cumulative benefit* (DCB), we have proposed an algorithm for finding beneficial semijoins, i.e., to interleave a sequence of join operations with semijoins to reduce the data transmission cost. In this algorithm, a *dynamic weighted graph* is constructed based on the *correlated* relationship among semijoins. When there are N vertexes in the initial *dynamic weighted graph*, where each vertex represents a semijoin, our algorithm needs to expand (N + 1) graphs to find a solution in the best case. Moreover, since the DCB values of vertexes will be dynamically updated in the process of a *vertex shrinking*, it will speed up the execution and reduce the large space requirement of the proposed recursive algorithm in general cases. ,From our simulation study, we have shown that our strategy can efficiently find beneficial semijoins and require a lower data transmission cost than does the profitable semijoin approach. In addition, this interleaving a join sequence with semijoins approach can reduce the communication cost further by taking advantage of the removability of *pure join attributes*, where *pure join attributes* are those which are used in join predicates but not part of the output attributes.

References

[1] Peter M. G. Apers, Alan R. Hevner and S. Bing Yao, "Optimization Algorithms for Distributed Queries," *IEEE Trans. on Software Eng.*, Vol. SE-9, No. 1, pp. 57-68, Jan. 1983.

[2] Philip A. Bernstein and Dah-Ming W. Chiu, "Using Semi-Joins to Solve Relational Queries," *Journal of ACM*, Vol. 28, No. 1, pp. 25-40, Jan. 1981.

[3] Philip A. Bernstein, Nathan Goodman, Eugene Wong, Christopher L. Reeve and James B. Rothnie, "Query Processing in a System for Distributed Databases (SDD-1)," *ACM Trans. on Database Syst.*, Vol. 6, No. 4, pp. 602-625, Dec. 1981.

[4] Ye-In Chang, Bor-Miin Liu and Cheng-Huang Liao, "A Dynamic-Cumulative-Benefit-based Algorithm for Beneficial Semijoins," *Proceedings of National Science Council: Part A – Physical Science and Engineering*, Vol. 20, No. 5, pp. 507-518, Sept. 1996.

[5] Arbee L. P. Chen and Victor O. K. Li, "Improvement Algorithm for Semijoin Query Processing Programs in Distributed Database Systems,"

—	Profile A	Profile B	Profile C	Profile D	Profile E
Case 1	13891	11519	9362	7383	5327
Case 2	13692	13692	13692	13692	13692
Case 3	12005	9210	7391	4604	3510

Table 5: A comparison.

the number of vertexes in the initial state (N)	3	4	5	6
the states created in one case of our strategy	4	18	75	614
the states created in the worst case of our strategy	10	41	206	1237

Table 6: The number of states.

IEEE Trans. on Computer, Vol. C-33. No. 11, pp. 959-967, Nov. 1984.

[6] Arbee L. P. Chen and Victor O. K. Li, "An Optimal Algorithm for Processing Distributed Star Queries," *IEEE Trans. on Software Eng.,* Vol. SE-11, No. 10, pp. 1097-1107, Oct. 1985.

[7] Ming-Syan Chen and Philip S. Yu, "Interleaving a Join Sequence with Semijoins in Distributed Query Processing,"*IEEE Trans. on Parallel and Distributed Syst.,* Vol. 3, No. 5, pp. 661-621, Sept. 1992.

[8] Ming-Syan Chen and Philip S. Yu, "Combining Join and Semi-Join Operations for Distributed Query Processing," *IEEE Trans. on Knowledge and Data Eng.,* Vol. 5, No. 3, pp. 534-542, June 1993.

[9] C. Wang and Ming-Syan Chen, " On the Complexity of Distributed Query Optimization," *IEEE Trans. on Knowledge and Data Engineering,* Vol. 8, No. 4, pp. 650-662, Aug. 1996.

[10] D. -M. Chiu and Y. -C. Ho, "A Methodology for Interpreting Tree Queries into Optimal Semijoin Expressions," *Proc. of 1980 ACM-SIGMOD Int. Conf. on Management of Data,* pp. 169-178, 1980.

[11] D. -M. Chiu, P. A. Bernstein and Y. -C. Ho, "Optimizing Chain Queries in a Distributed Database System," *SIAM J. Comput.,* Vol. 13, pp. 116-134, Feb. 1984.

[12] A. Hevner, "Query Optimization in Distributed Database Systems," Ph.D. Dissertation, Univ. Minnesita, 1979.

[13] Alan R. Hevner and S. Bing Yao, "Query Processing in Distributed Database Systems," *IEEE Trans. on Software Eng.,* Vol. SE-5, No. 3, pp. 177-187, May 1979.

[14] Sakti Pramanik and David Ittner, "Optimizing Join Queries in Distributed Databases," *IEEE Trans. on Software Eng.,* Vol. 14, No. 9,pp. 1319-1326, Sept. 1988.

[15] Hyuck Yoo and Stephane Lafortune, "An Intelligent Search Method for Query Optimization by Semijoins," *IEEE Trans. on Knowledge and Data Eng.,* Vol. 1, No. 2, pp. 226-237, June 1989.

[16] C. T. Yu and C. C. Chang, "Distributed Query Processing," *ACM Computing Surveys,* Vol. 16, No. 4, pp. 388-433, Dec. 1984.

Parallel Processing of Temporal Joins

Thomas Zurek
University of Edinburgh
United Kingdom

Keywords: temporal join, parallel join, temporal databases, partitioning

Edited by: Silvio Salza
Received: October 30, 1996 **Revised:** March 31, 1997 **Accepted:** April 25, 1997

In this paper we present a framework for parallel temporal joins. The temporal join is a key operator for temporal processing. Efficient implementations are required in order to make temporal database features attractive and applicable for the many applications that are amenable.

We focus on the temporal intersection as the supertype of temporal joins. A basic algorithm is presented that is based on partitioning the temporal data over its interval timestamps. It consists of a partition, a join and a merge stage. The partition stage has to replicate tuples that intersect with more than one partition range. Any sequential join technique can be used in the join stage. Two possible optimisations for reducing the overhead imposed by replication are discussed.

The algorithm and its optimisations are evaluated on top of a performance model. We describe the model and give details in the appendix. The evaluation shows that both optimisations together decrease the basic costs significantly. Furthermore we can give an idea of the quantitative impact of replication overhead in parallel temporal join processing. A modest workload caused a share of around 70% of the total costs; higher values can be expected in reality.

1 Introduction

Recent years have seen increasing research efforts on temporal databases. Temporal data models, temporal query languages and temporal index structures have been the focus of a lot of papers. Only a few proposals have come up discussing algorithms for temporal operations although it is often cited that temporal-specific algorithms are required for performance reasons.

The temporal join is one of the key temporal relational operators. Intuitively, it is used whenever we want to retrieve data that is valid at the same time. Efficient implementations are required in order to make temporal database features attractive and applicable for the many applications that are amenable to temporal data processing, especially in data warehousing and data mining. The performance of temporal join processing, however, suffers from the higher size of temporal relations (because tuples are *logically* deleted rather than *physically* removed) and the high selectivity factor [12] of temporal join conditions.

Various authors argue that the semantics of time can be used for optimisations. A popular example is the 'append-only' characteristic of transaction time applications: when new tuples are inserted they are appended to the collection of existing ones. This results in a natural sort order on the transaction time attribute. The sort order itself can then be exploited in several ways.

Parallelism is another possibility of improving temporal processing performance. It has already been successfully incorporated in traditional database technology and helped to overcome certain performance deficits. However, it has been widely neglected in the context of temporal databases. To our knowledge, there has been only one paper that discusses parallel temporal issues [7]. Its optimisations, however, are bound to special cases of temporal joins and there is no quantitative evaluation and no considerations on parallel architectural issues.

In this paper, we want to overcome some of these deficits and present a framework for parallel temporal join processing. We thereby concentrate on features that are temporal-specific; considerations on data skew or optimisations based on non-temporal parts of the join condition are left out as these have been discussed already by other papers and in more detail. Section 2 discusses types of temporal joins and sequential algorithms that were proposed in the past. Section 3 presents an example for parallel temporal join processing as opposed to conventional parallel equi-join. In section 4, a basic parallel temporal join algorithm is given. This algorithm is improved through two optimisations. Section 5 evaluates these results quantitatively. We modeled the performance of the three algorithms on top of a general-purpose hardware architecture. This makes the results useful for a wide range of parallel environments. Details of the performance model are given in the appendix. Finally, the paper is concluded in section 6.

2 Temporal Joins

2.1 Types of Temporal Joins

Temporal joins combine (at least) two temporal relations using a temporal join condition over the two timestamps. The latter are usually represented as intervals. Temporal join conditions therefore consist of expressions that define relationships between timestamps, i.e. the time intervals. [1] identified seven possible relationships[1] between two intervals. These are shown in table 1.

We adopt the following notational conventions: an interval x consists of a start point, denoted by $x.t_s$, and an end point, denoted by $x.t_e$. When speaking of a timestamped tuple r we also refer to the respective interval boundaries by $r.t_s$ and $r.t_e$. Intervals are represented by its start and end point being enclosed in brackets: [or] means that the respective boundary is included whereas (and) mean that the respective boundary is excluded:

$$\begin{aligned} [x.t_s, x.t_e] &= \{t : x.t_s \leq t \leq x.t_e\} \\ (x.t_s, x.t_e) &= \{t : x.t_s < t < x.t_e\} \\ [x.t_s, x.t_e) &= \{t : x.t_s \leq t < x.t_e\} \end{aligned}$$

Temporal joins can be classified according to the relationship that its join condition is based on: there are temporal contain-, meet-, overlap-, ... joins. Obviously a temporal join condition may contain any arbitrary combination of these relationships. We can consider the corresponding join to be of any type that is involved. In reality it is likely that there is only one.

The literature has mainly concentrated on a 'supertype' of the joins that arise from table 1, namely the temporal *intersection* that includes the relationships *equals, overlaps, during, starts* and *finishes*. We will concentrate on intersection joins as all the other temporal joins can be considered as intersection joins with additional constraints. Furthermore we can draw a clear line between the optimisations that are possible for the temporal intersection join and those that are specific to the other temporal joins.

Usually the tuples that satisfy the join condition are concatenated. In the case of temporal joins this concatenation is not trivial as the value of the timestamp for the resulting tuple has to be defined. This definition depends on the type of the temporal join; assuming temporal intersection the resulting timestamp is defined to be the intersection of the individual timestamps of the participating tuples. For example in the case of the two tuples r and s the resulting timestamp is

$$[\max\{r.t_s, s.t_s\}, \min\{r.t_e, s.t_e\}] \qquad (1)$$

2.2 Algorithms for Temporal Joins

The four principal algorithmic join techniques are brute force nested-loop, sort-merge, hash-joins and index based joins. Most join conditions involve an equality predicate. Such joins are called *equi-joins* and a lot of effort has been spent on algorithms that exploit the low selectivity factor imposed by the equality predicate.

Basic nested-loops for equi-joins usually perform badly due to the lack of any preceding partitioning of the data. The sort-merge approach is based on implicit partitioning given by a sort order on a join attribute. This allows to reduce the number of unnecessary comparisons. The hash join approach requires explicit partitioning prior to its joining stage whereas index based joins use precomputed partitioning [9].

In the literature, several algorithms based on these approaches have also been discussed in the context of temporal joining. Sort orderings on the temporal attributes – these are either achieved through explicit sorting or implied by the 'append-only' characteristic of transaction-time relations – allow various optimisations. Discussions can be found in [3], [6], [4], [13].

In the case of equi-joins, explicit partitioning is the basis for very efficient joining. Applying those techniques to temporal intersection of interval data, however, has the problem that intervals cannot be reduced to a discrete value that allows the grouping of the tuples whose timestamps intersect in one single partition. One way to get around this problem is to group tuples by either one of their temporal interval boundaries (start or end point) or a combination of these (e.g. in [8]). Tuples must then be either replicated and put into those partition fragments that hold other tuples that possibly join, i.e. have intersecting timestamps (as in [7], [14] and [16]) or each partition fragment has to be joined with various others that hold possibly joining tuples (as in [8]). The algorithm proposed in [14] processes fragments sequentially and keeps tuples that have to be present in the following fragment in a cache. This makes it an inherently sequential algorithm.

Replication cannot be avoided in the case of parallel temporal join processing. Such an approach is discussed in [7]. The so called *asymmetry* property of relations – that appears e.g. in contain- or overlap-joins – can be used for reducing the number of tuples that have to be replicated. Asymmetry, however, does not occur in the more general intersection join. Therefore, most optimisations that are suggested by Leung and Muntz cannot be applied in the case of the intersection join.

3 An Example

Before digging deeper into the parallel temporal join processing we want to illustrate the difference between

[1] Actually there are 13 if one takes the six possible reversed relationships into account.

Relationship	Example	Condition
x equals y	xxxx yyyy	$x.t_s = y.t_s \wedge x.t_e = y.t_e$
x before y	xxxx yyyy	$x.t_e < y.t_s$
x meets y	xxxxyyyy	$x.t_e = y.t_s$
x overlaps y	xxxx yyyy	$x.t_s < y.t_s \wedge x.t_e > y.t_s \wedge x.t_e < y.t_e$
x during y	xxx yyyyyyy	$x.t_s > y.t_s \wedge x.t_e < y.t_e$
x starts y	xxx yyyyyy	$x.t_s = y.t_s \wedge x.t_e < y.t_e$
x finishes y	xxx yyyyyy	$x.t_s > y.t_s \wedge x.t_e = y.t_e$

Addtional constraints are: $x.t_s \leq x.t_e \wedge y.t_s \leq y.t_e$

Table 1: The possible relationships between two intervals [1]

conventional parallel equi-join processing and parallel temporal join processing by an example. For that purpose we use two simple temporal relations of figures 1 and 2: one holds information on cities and times of performances the play "Hamlet", the other provides similar information on performances of the play "Faust". For simplicity, times are represented as integers.

City	Start	End
Aachen	3	8
Berlin	2	10
Dresden	4	9
Hamburg	1	8
Munich	6	10
Vienna	1	10

Figure 1: Relation **Hamlet**.

City	Start	End
Bern	1	4
Dresden	1	4
Linz	7	9
Munich	2	5
Salzburg	1	3
Zürich	7	10

Figure 2: Relation **Faust**.

Assuming that the city names are unique we can get all cities in which both plays are performed by computing an equi-join

$$\text{Hamlet} \bowtie_C \text{Faust}$$

with the join condition $C \equiv$ *Hamlet.City = Faust.City*. To compute this join in parallel we can partition the two tables by using the values of the city attributes. Figure 3 shows an example for partitioning the tables into three fragments respectively and thus into three smaller and independent joins. Please note that the partitioning process produced *disjoint* fragments respectively. Each of the three joins can be processed on different nodes in parallel.

Hamlet			Faust		
Aachen	3	8	Bern	1	4
Berlin	2	10	Dresden	1	4
Dresden	4	9			

Cities beginning with A – F

Hamlet			Faust		
Hamburg	1	8	Linz	7	9
Munich	6	10	Munich	2	5

Cities beginning with G – M

Hamlet			Faust		
Vienna	1	10	Salzburg	1	3
			Zürich	7	10

Cities beginning with N – Z

Figure 3: Example of processing an equi-join in parallel.

If we want to know the period during which both plays are performed (respectless of the location) then we need a temporal intersection join between the two relations. The join condition in this case is $C \equiv$ *TIMESTAMP(Hamlet) intersects TIMESTAMP(Faust)*. Similar to the equi-join above, the temporal join can be processed in parallel. However, this time the tables have to be partitioned over the interval timestamps. Figure 4 shows an example of partitioning the tables into three fragments respectively. Please note that in this case the fragments are *not disjoint* and tuples have to be replicated. This causes an overhead, not only because of the effort spent on the replication itself but also because of the additional work imposed on the joining of the fragments.

Hamlet			Faust		
Aachen	3	8	Bern	1	4
Berlin	2	10	Dresden	1	4
Hamburg	1	8	Munich	2	5
Vienna	1	10	Salzburg	1	3

Timestamps intersecting with [1,3]

Hamlet			Faust		
Aachen	3	8	Bern	1	4
Berlin	2	10	Dresden	1	4
Dresden	4	9	Munich	2	5
Hamburg	1	8			
Vienna	1	10			

Timestamps intersecting with [4,5]

Hamlet			Faust		
Aachen	3	8	Linz	7	9
Berlin	2	10	Zürich	7	10
Dresden	4	9			
Hamburg	1	8			
Munich	6	10			
Vienna	1	10			

Timestamps intersecting with [6,10]

Figure 4: Example of processing a temporal join in parallel

4 Parallel Temporal Joins

In this section, we first describe the general structure of parallel temporal joins. We focus on the temporal intersection join. None the less, what we state can also be applied to the more specific cases such as contain- or overlap-joins that allow more specific optimisations due to their increased selectivity.

After an introduction of the basic architectural and notational issues in sections 4.1 and 4.2, a basic parallel intersection join is presented in section 4.3. Finally, two essential optimisations of the basic algorithm are discussed in section 4.4.

4.1 The Basic Structure

We assume a hybrid parallel architecture as it was described in [5]. This architecture has two levels: the inner or node-level is based on a shared-memory approach, whereas the outer level adopts shared-nothing. Put in another way: it is a shared-nothing combination of SMP nodes. This type of architecture has proved to be the most general one and a recent survey of commercial parallel database systems [10] showed that most parallel database systems were optimised for running on this type of hardware. Figure ?? illustrates the architecture[2].

The stages of a parallel (temporal) join of two (temporal) relations R and S are the following:

1. Partition R into fragments $R_1, \dots, R_m$; partition S into fragments $S_1, \dots, S_m$.

2. Perform local temporal joins $R_1 \bowtie S_1, \dots, R_m \bowtie S_m$

3. Merge the local results to form a global result.

For simplicity, we will use the symbol $\bowtie$ throughout the paper to represent a temporal join assuming that there is a temporal intersection condition associated with it. Section 4.3 discusses stages 1 and 2 in more detail because it is there where the differences between a parallel temporal join and a traditional parallel join arise.

It is assumed that R and S are physically partitioned over the nodes. However, we assume that the partitions of R and S for the join have to be created dynamically, i.e. the R_k and S_k do not correspond to the fragments of R and S that exist on the disks for the following reasons:

- It is unlikely that R and S will be partitioned over the *timestamp attribute* using the *same* partition points for *both* relations.

- It is even more unlikely that those partitions are colocated[3].

4.2 Preliminaries

Let T be the time span covered by the tuples of R and S. For the purpose of this paper we assume T to be an interval over a discrete domain, e.g. an interval of integers $[t_{\min}, \dots, t_{\max}]$. A temporal m-way partition P of T is a set $\{p_0, p_1, \dots, p_{m-1}, p_m\}$ of $m+1$ partition points with $p_0 = t_{\min}, p_m = t_{\max} + 1$ and $p_k < p_{k+1}$, $p_k \in T$ for $k = 0, \dots, m-1$. P divides T into m partition intervals

$$[p_{k-1}, p_k) \;=\; \{t \in T \,|\, p_{k-1} \leq t < p_k\}$$

for $k = 1, \dots, m$. The following function determines the number of the fragment, time range respectively, that a time point $t \in T$ belongs to with respect to partition P

$$\mathit{fragment}_P(t) \;\equiv\; k \;\text{ iff }\; t \in [p_{k-1}, p_k)$$

The parallel hardware has N nodes each with n processors. The nodes are numbered from 1 to N and the processors from 1 to nN. For convenience we additionally use an M-way partition Q of T with $M = (m-1)\ \mathit{div}\ n + 1$. Q is a subset of P and

[2] For redundancy reasons the disks are usually not only connected to one SMP node but to several. For the purpose of this paper we do without this feature.

[3] In the sense of colocated joins as discussed in [2].

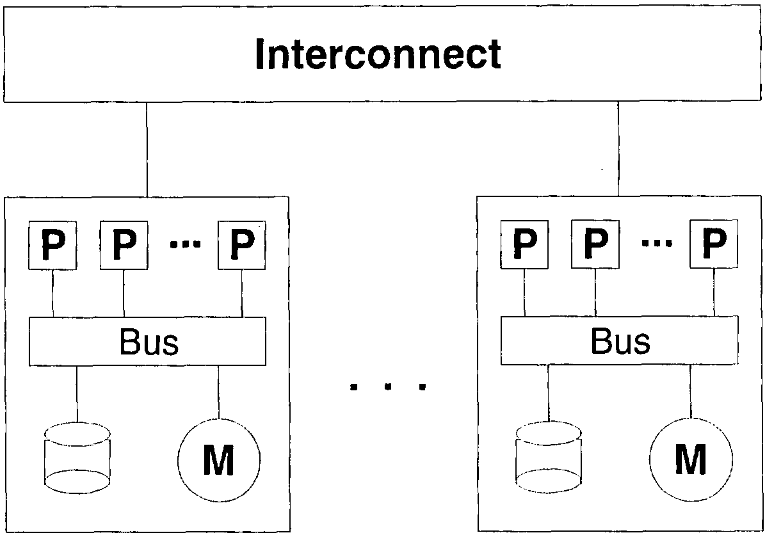


Figure 5: Hybrid parallel architecture

holds every n-th partition point of P, i.e. the partition points that coincide with the points used as node boundaries. In the simple case, i.e. $m = nN$, we have $M = N$. Q is helpful when describing the algorithm in the next section

$$Q = \{q_0, q_1, \ldots, q_{M-1}, q_M\}$$

with $q_k = p_{nk}$. Similarly there is a function $\mathit{fragment}_Q$ for Q.

4.3 The Algorithm

The algorithm adopts the structure presented in section 4.1. We concentrate on stages 1 and 2 only; stage 3 either leaves the result partitioned for further processing or saves the tuples on disk.

We assume that an optimiser has decided on a suitable partition P for the relations R and S. This is in fact a very delicate choice [16] because the partition will have to cope with load balancing problems caused by data skew *and* – as a temporal data specific feature – will also influence the rate by which tuples are replicated. For the scope of this paper we assume that a suitable partition has been decided on; in the experiments in section 5 we assume an ideal uniform distribution of the data which makes this choice simple.

At the start, relations R and S are physically partitioned into N fragments, each of which residing on the disks at the corresponding node. Stage 1 converts these initial partitions into the partitions $R_1, \ldots, R_m, S_1, \ldots, S_m$ that are required for the join. As a result, the secondary storage of processor k will hold $R_k, S_k, R_{k+n}, S_{k+n}, \ldots, R_{k+a_k \cdot n}, S_{k+a_k \cdot n}$ with a_k defined such that $k + a_k \cdot n \leq m$. In the simplest case, $m = nN$, this means that all $a_k = 1$, thus R_k and S_k reside at processor k. The way to do this on the two level architecture might seem confusing at the first sight as various conversions of node, processor, buffer and fragment indices are necessary.

Stage 1

Range-partitioning for R, i.e. creation of the fragments $R_1, \ldots, R_m$ (assume $m = nN$ when you read this for the first time) in stage 1 is done in the following way; S is partitioned similarly:

(a) Each node reads its fragment of R; then each processor processes the n-th part of this fragment.

(b) Each processor k has m hash buffers – one per partition fragment – and $a_k + 1$ output buffers numbered $k, k+n, \ldots, k + a_k \cdot n$.

It hashes its tuples to the hash buffers in the following way: a tuple with timestamp $[t_s, t_e]$ is put into

(i) the hash buffer that corresponds to the first fragment in which $[t_s, t_e]$ falls, i.e. the hash buffer number $\mathit{fragment}_P(t_s)$,

(ii) the hash buffers that correspond to the first fragments in each node in which the tuple has to reside, i.e. buffers with numbers $(k-1)n+1$ where

$$\mathit{fragment}_Q(t_s) < k \leq \mathit{fragment}_Q(t_e)$$

Remark: Step (i) puts the tuple in the fragment that covers the range in which the timestamp's start point t_s falls; step (ii) puts the tuples in the first fragments on nodes (other than that covered by step (i)) that hold ranges with which $[t_s, t_e]$ intersects.

When a hash buffer l is full it is transmitted to the output buffer $(l-1)$ *div* $nN+1$ of processor $(l-1)$ *mod* $nN+1$.

(c) A further replication step is performed when tuples (each with some time interval $[t_s, t_e]$) arrive, say at output buffer $k+i \cdot n$ at processor k:

```
if ($k + i \cdot n$) is not the final fragment for
  $[t_s, t_e]$ then
    send tuple to the all output buffers $l$ on
    the same node with $l \leq fragment_P(t_e)$
    (i.e. the number of the final fragment
    for $[t_s, t_e]$)
```

Remark: Alternatively, an index structure can be built (for each processor) that refers to those tuples in shared-memory that fall into the respective fragment. In the case of large tuples this is certainly faster than replicating the tuples in main memory. As we will see from the results of section 5, the costs imposed through this stage, even when using the slower 'copying', comprise only a very minor share of the overall costs. Therefore and due to space restrictions we only describe the simple copying strategy.

(d) When an output buffer is full then its tuples are flushed to disk.

The significant difference, in comparison to partitioning for a traditional parallel join, is the replication of tuples in steps (b).(ii) and (c). We chose a two-level replication:

- (b).(ii) replicates the tuples over the interconnect and positions the tuples on all nodes that hold ranges that intersect with the tuples' timestamps; this step can be seen as an *inter-node replication.*
- (c) replicates the tuples within the nodes and sends them to all processors that cope with a range that intersects with the tuples' timestamps. This *intra-node replication* is faster because it can be done via shared-memory rather than via communication over the interconnection network. If this step was incorporated into step (b).(ii) the advantage of fast communication via main memory would be lost.

Stage 2

We now focus on stage 2 of the algorithm. Actually, any sequential temporal join algorithm can be used for performing the local joins $R_1 \bowtie S_1, \ldots, R_{nN} \bowtie S_{nN}$. We adopt a nested-loop approach for the following reason:

The performance drawback of nested-loop in the case of a traditional equi-join is mainly due to the fact that the algorithm compares the two portions of tuples completely in order to compute the join result. Sort-merge equi-joins can decrease this overlap of the two relations nearly to a minimum as they take advantage of the relations being sorted on a join attribute. Hash-based equi-joins only do the necessary comparisons due to disjoint partitioning [9].

In the case of a temporal intersection join, however, the degrees of overlap of sort-merge and hash joins have to be variable and are – as we argue – very close to a complete overlap: the R_k and S_k originate in a range-partition of R and S; thus they hold tuples whose timestamps intersect very likely. Therefore the degree of necessary overlap between the R_k and S_k is likely to be high. Therefore the advantages of sort-merge and hash are reduced. Furthermore they would require R_k and S_k to be sorted[4] or hashed beforehand which imposes an overhead that additionally reduces their advantages.

Stage 2 of the basic algorithm then works in the following way: each processor k copes with 'its' fragments R_k and S_k of R and S respectively. Without loss of generality we assume $|R_k| \leq |S_k|$ in the following. This only implies that – for efficiency reasons – R_k will be the outer and S_k the inner relation in the nested-loop computation of $R_k \bowtie S_k$.

We assume that the join condition is a temporal intersection and some boolean expression $C(r, s)$. The latter is supposed to be non-temporal and therefore amenable to the same optimisations that may be applied to non-temporal join evaluation. For performance modelling purposes we later assume that $C(r, s)$ evaluates to *true* so that we can neglect any implications given by this part of the join condition and concentrate on the essential temporal aspects. Stage 2 then looks like this:

```
for each tuple $r$ in $R_k$ do
  for each tuple $s$ in $S_k$ do
    if ($[r.t_s, r.t_e]$ intersects $[s.t_s, s.t_e]$) and
      no_previous_join($r, s, k$) and $C(r, s)$ then
        time-concatenate $r$ and $s$
        place result in output buffer $X_k$
        if ($X_k$ is full) then flush to disk
```

To avoid the situation in which the global result contains duplicates that are a consequence of tuple replication, tuples r and s are only joined if at least one of them appears in no preceding fragment R_l or S_l with $l < k$, i.e. if at least one of their timestamps has its start point in the current range. This is determined by the boolean function *no_previous_join(r, s, k)* that can be defined as

[4] In a message-passing environment it is difficult to preserve a sort order on R_k and S_k in the case that relations R and S are already sorted. At least it would slow the communication down and would impose an overhead in this way even though the sorting of R_k and S_k was avoided.

$$\textit{no\_previous\_join}(r,s,k) \equiv (\textit{fragment}_P(r.t_s)=k) \text{ or } (\textit{fragment}_P(s.t_s)=k)$$

When two tuples satisfy the join condition they are concatenated. As we mentioned in section 2, the tuple that results from joining two tuples r and s holds the timestamp defined by (1). This process is called the *time-concatenation* of r and s.

4.4 Optimisations

In this section, we want to point to two possible optimisations of the algorithm presented in the previous section.

Optimisation 1

The function *no_previous_join(r, s, k)* was used to avoid replicated tuples being joined unnecessarily and causing duplicates in the result. Nevertheless these tuples are processed by the algorithm. We can avoid this by splitting up the fragments R_k and S_k into two components respectively:

- the set of tuples that have their timestamp start points in the range $[p_{k-1}, p_k)$ – these are called the *primary tuples* and are put into the sets R'_k and S'_k, respectively,
- the set of tuples that fall into the fragment because of replication (i.e. their timestamp start point is not in the range $[p_{k-1}, p_k)$) – these are denoted as the *replicated tuples* and are put into R''_k and S''_k.

Formally, we can describe the benefit of this procedure as follows: stage 2 in section 4.3 does all the processing for computing the local join

$$R_k \bowtie S_k \tag{2}$$

but is prevented from putting some tuples to the results through the *no_previous_join(r, s, k)* condition. With $R_k = R'_k \cup R''_k$ and $S_k = S'_k \cup S''_k$ (2) evaluates to

$$R'_k \bowtie S'_k \ \cup\ R'_k \bowtie S''_k \ \cup\ R''_k \bowtie S'_k \ \cup\ R''_k \bowtie S''_k \tag{3}$$

which represents four individual joins. The latter one ($R''_k \bowtie S''_k$) defines exactly the set of tuples that is excluded from the result by the *no_previous_join(r, s, k)* condition. It is actually unnecessary and can be skipped. Computation can be reduced to the first three joins. Section 5 will show the benefit of this measure.

As a prerequisite for this optimisation we have to keep primary and replicated tuples separated during the partitioning stage 1: each processor had one hash buffer per processor that kept primary and replicated tuples. Now we use two: a primary hash buffer and a replication hash buffer. Similarly, each processor had one output buffer for collecting tuples of its fragment before they were flushed to disk. Now there are primary and replication output buffers. Steps (b) and (c) of stage 1 are modified like this:

(b) Each processor hashes its tuples to the buffers in the following way: a tuple with timestamp $[t_s, t_e]$ is put into

  (i) the primary hash buffer $k = \textit{fragment}_P(t_s)$

  (ii) the replication hash buffers with numbers $(k-1)n+1$ where

$$\textit{fragment}_Q(t_s) < k \leq \textit{fragment}_Q(t_e)$$

When a primary hash buffer is full it is transmitted to the primary output buffer of the corresponding processor.
When a replication hash buffer is full it is transmitted to the replication output buffer of the corresponding processor.

(c) A further replication step is performed when tuples (each with some time interval $[t_s, t_e]$) arrive at the replication output of processor (say k):

```
**if** $(k < \textit{fragment}_P(t_e))$ **then**
  **for** $l = k+1$ **to**
    $\min\{n \cdot \textit{node}(k), \textit{fragment}_P(t_e)\}$
      send tuple to the replication
      output buffer of processor $l$
```

Optimisation 2

A second optimisation is based on two observations:

- The three remaining joins of (3) can be computed in the following orders (amongst others):

$$R'_k \bowtie S''_k\ ,\ R'_k \bowtie S'_k\ ,\ R''_k \bowtie S'_k \tag{4}$$

or

$$R''_k \bowtie S'_k\ ,\ R'_k \bowtie S'_k\ ,\ R'_k \bowtie S''_k \tag{5}$$

- Until now we assumed that the number m of fragments matches the number nN of processors. Alternatively, we can choose m in a way such that R'_k and/or S'_k are small enough to fit into main memory of the node. This is possible because the sizes of R'_k and S'_k can be nearly *arbitrarily* cut down by increasing m; remember that they only hold tuples that have their timestamp start point in the range $[p_{k-1}, p_k)$. Thus each tuple of R (S respectively) belongs only to one R'_k (S'_k resp.) because the ranges are disjoint. Thus R (S resp.) is partitioned into more and smaller fragments by increasing m.

Assuming that the joins are computed in the order as in (4) or (5) and keeping R'_k and S'_k in main memory we can avoid unnecessary accesses to secondary storage: R'_k is loaded once into main memory for computing the first join in (4) and is then kept for computing the second join, and finally the third join is computed with S'_k in main memory. The procedure for (5) works accordingly.

Optimal join ordering and avoiding I/O accesses by optimally using main memory is no special feature of temporal join processing. However, there are two significant issues about this second optimisation:

- The original join (2) is decomposed into three subjoins, one of them being $R'_k \bowtie S'_k$. R'_k and S'_k have predictable sizes as each tuple of R and S appears in only one of these fragments respectively. In contrast, the sizes of R''_k and S''_k, and therefore also of R_k and S_k, are rather difficult to predict because the rate of tuple replication is difficult to estimate. Furthermore, the negative effects of data skew are higher than for R'_k and S'_k: it is not only the data skew on the timestamp start point values, but also skew on the timestamp interval lengths, that influence the sizes of R''_k and S''_k.
- The optimisation is based on the patterns (4) and (5) that occur *regularly*, namely in each of the local joins of the parallel temporal join execution. It is therefore an integral part of the algorithm and not a feature that an optimiser might exploit whenever a qualifying situation is detected.

5 Evaluation

In this section, we give a quantitative analysis of the parallel temporal joining techniques described in section 4:

- the basic algorithm described in 4.3 (join A),
- the basic algorithm plus optimisation 1 (join B),
- the basic algorithm plus optimisations 1 and 2 (join C).

Section 5.1 describes the evaluation model; section 5.2 gives the analysis of the algorithms.

5.1 Evaluation Model

The performance of the algorithms was modeled in a similar way to the parallel hash join in [5]. Due to space restrictions in this paper we only focus on the major issues of the model. The model is described in the appendix (tables 2 – 7). A complete description can be found in [15].
The major issues are:

- The total response time C_{total} of the algorithms depends on the times C_{part}, C_{join} spent in stages 1 and 2. In reality there might be an overlap between these two stages; thus

  $$\max\{C_{part}, C_{join}\} \leq C_{total} \leq C_{part} + C_{join}$$

  In our model, however, we assume that there is no overlap (e.g. enforced through a barrier type synchronisation). Thus we use the upper bound $C_{total} = C_{part} + C_{join}$. The stages (a), (b) etc. within stages 1 and 2 are treated accordingly.
- Within each stage the overlap between the I/O, communication, CPU and memory access phases is perfect. This can be nearly achieved by separate I/O and communication processors. Thus

  $$\begin{aligned} C_{part} &= \max\{C_{p\_io}, C_{p\_comm}, C_{p\_CPU}, C_{p\_mem}\} \\ C_{join} &= \max\{C_{j\_io}, C_{j\_CPU}, C_{j\_mem}\} \end{aligned}$$

  Put the other way: in tables 2 – 5 the maximum of each row is taken.

We assume a uniform distribution of the data over time, i.e. tuple lifespans do not vary much from their average values τ_R, τ_S and the timestamp start points are uniformly distributed over the relation lifespans T_R, T_S. This assumption is certainly ideal and in reality temporal data skew has to be considered. It allows, however, the avoidance of any (possibly unrealistic or application specific) assumptions about the nature of temporal data skew. Furthermore, the model can be kept simple.

The timespan T, covered by tuples from R and S, is divided into m equally sized ranges, i.e. two partition points p_k and p_{k+1} are on equal distances for $0 \leq k < m$. If $|T|$ is the length of T then $\frac{|T|}{m}$ is the length of a fragment range. If τ is the average length of a tuple timestamp interval then this timestamp occupies a share of

$$\frac{\tau}{\frac{|T|}{m}} \tag{6}$$

of a fragment's range. As interval start points are distributed uniformly over the fragment range, there will be some tuples whose timestamps start near the end of the range. This means that those tuples overlap the range borders (i.e. the partition points p_i). To get the average number δ of fragment ranges that a tuple timestamp spans we have to add 1 to (6). Therefore δ is given by $\delta = \frac{\tau}{|T|} \cdot m + 1$.

The parameter λ in table 5 is the number of times a processor or node has to perform the stage in case that $m > nN$. In the experiments we chose m to be a multiple of nN such that R'_k and S'_k fit into main memory (see section 4.4). Thus

$$\lambda = \left\lceil \frac{m}{n \cdot N} \right\rceil \tag{7}$$

5.2 Analysis

The performance model was used to run several experiments. First of all, we compared the performance of the three joins with respect to the workload of table 6, the architectural parameters of table 7 and varying the number N of nodes. Figure ?? shows the result. As it can be expected join C performs better than join B which itself is better than join A. Interesting facts, however, are the quantitative effects of the optimisations that were discussed in section 4.4:

- Optimisation 1 (join A → join B) improves performance by between 20% ($N = 10$) and 65% ($N = 50$).
- Optimisation 2 (join B → join C) decreases costs by 90%.
- Optimisations 1 and 2 (join A → join C) give a composite improvement of around 95%.

The experiments showed that every algorithm's costs are dominated by the costs for stage 2; partitioning costs and therefore communication costs can be neglected. The costs of joins A and B mainly comprise I/O costs whereas join C's costs consist of CPU and memory access costs. Increasing N implies a higher I/O bandwidth, more memory and more CPU power. This explains the ideal scaleup in figure ??.

Figure ?? shows the split of costs for join C. Graphs for the other two joins look the same; only the respective cost values on the vertical axis are higher but the ratio between the partial cost components is the same. The *replication join costs* are the costs for performing the joins that are due to tuple replication, i.e. the joins involving R''_k and S''_k in (4) and (5). The *primary join costs* are the costs for performing the join $R'_k \bowtie S'_k$.

The overhead costs imposed by tuple replication have a share of 65% to 75% of the total costs. This suggests that any optimisations that reduce tuple replication should translate nicely into a total cost reduction.

Up to now, the workload of table 6 did not require m to exceed nN in order to reduce the size of the R'_k and S'_k such that they fit in main memory (see optimisation 2). In other words: it is $\lambda = 1$ in figures ?? and ??.

In figure ?? the costs of join C are shown for the tuple size r being varied. The graphs break off at around 1050 and 1678 bytes. The first breakpoint is caused by the fact that memory costs overtake CPU costs. The second one is due to the choice of m being a multiple of nN for simplicity. Passing the '1678-bytes-point' λ changes from 1 to 2 (see equation (7)). The crease suggests that this choice is not optimal but not bad either. The share of the replication overhead remains in the 65% to 75% margin throughout the experiment[5].

6 Summary

In this paper, we discussed the parallel processing of temporal joins. We focused mainly on temporal intersection as other types of temporal joins can be considered as special cases to temporal intersection.

Parallelising temporal joins is not trivial. The significant difference with respect to traditional equi-joins is based on the fact that timestamps are usually represented as intervals. The intersection conditions consists of non-equality predicates. Data partitioning over time intervals is therefore not straightforward: tuples have to be replicated and to be put into several fragments of a range-partition of the temporal relations. This causes considerable overhead.

We showed that this overhead can be reduced significantly if one divides a partition fragment into *primary* and *replicated* tuples. This allows to avoid that replicated tuples of one relation are joined with replicated tuples of another relation as this is not necessary (optimisation 1). Furthermore this division enables us to choose a certain number m of fragments such that subfragments that contain the primary tuples fit into main memory. This allows to reduce I/O accesses significantly (optimisation 2).

Finally we gave a performance model for three parallel joins. This was based on a general-purpose parallel hardware architecture. This makes results generally useful. The joins were evaluated on top of this model using a certain workload. Both optimisations reduced costs by around 95%. A further conclusion was that the overhead caused by tuple replication made up around 70% of the total costs. Research efforts on reduction of tuple replication should therefore translate nicely into performance improvement.

An open issue is the choice of an appropriate partition for the temporal relations. This choice is very delicate as it has to cope with data skew and also determines the rate of replication. We plan to investigate this problem in the near future.

References

[1] J. Allen. Maintaining Knowledge about Temporal Intervals. *Communications of the ACM*, 26(11):832–843, Nov. 1983.

[2] C. Baru, G. Fecteau, A. Goyal, H. Hsiao, A. Jhingran, S. Padmanabhan, G. Copeland, and W. Wilson. DB2 Parallel Edition. *IBM Systems Journal*, 34(2):292–322, 1995.

[3] H. Gunadhi and A. Segev. A Framework for Query Optimization in Temporal Databases. In

[5] This margin, of course, depends on the workload, in particular on the lengths of tuple timestamps. We consider the average timestamp / relation lifespan ratio $r/|T|$ of 2% in our example as low. Higher ratios can perfectly be expected. These would increase the 70% margin.

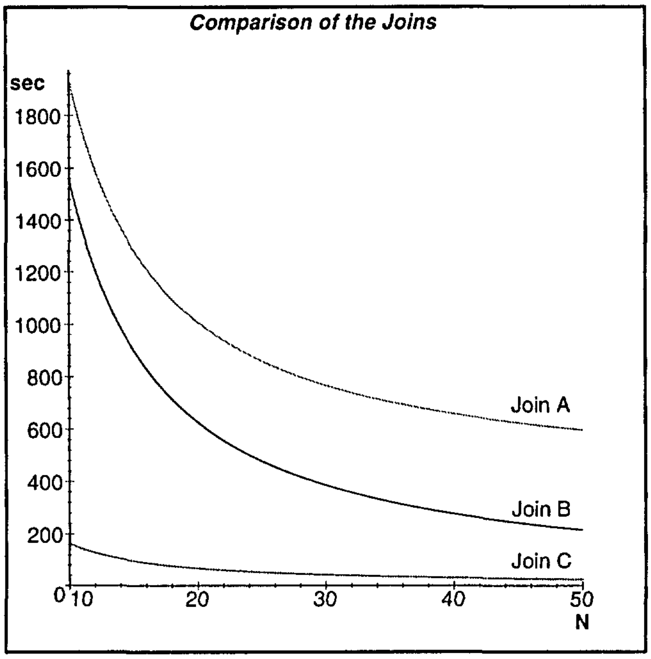


Figure 6: Performances of the three join algorithms

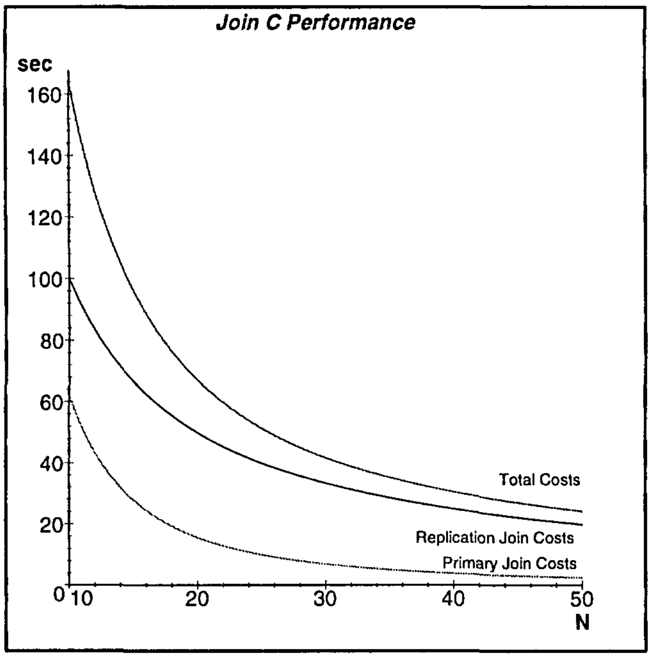


Figure 7: The components of the join C costs

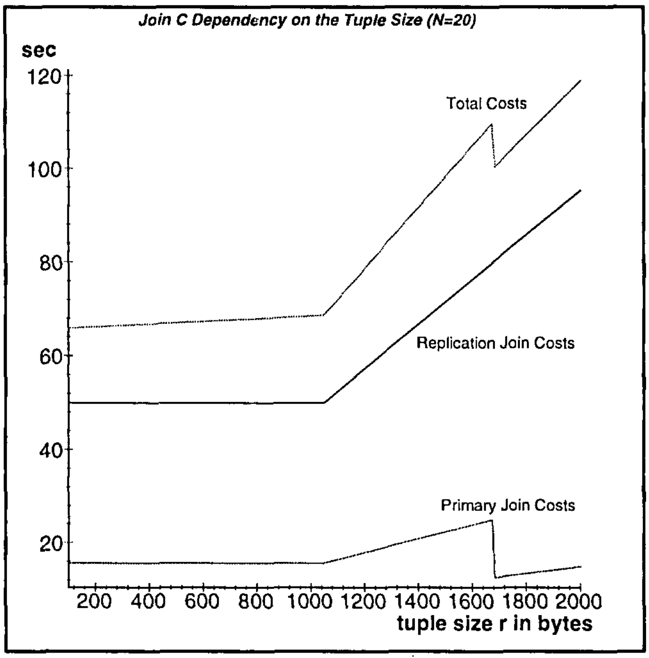


Figure 8: The costs of join C depending on the tuple size r

Z. Michalewicz, editor, *Proc. of the 5th Int. Conf. on Statistical and Scientific Database Management, Charlotte, NC, USA*, number 420 in Lecture Notes in Computer Science (LNCS), pp. 131–147. Springer, Apr. 1990.

[4] H. Gunadhi and A. Segev. Query Processing Algorithms for Temporal Intersection Joins. In *Proc. of the 7th Int. Conf. on Data Engineering, Kobe, Japan*, pp. 336–344, Apr. 1991.

[5] K. Hua, C. Lee, and J.-K. Peir. Interconnecting Shared-Everything Systems for Efficient Parallel Query Processing. In *Proceedings of the 1st Int. Conf. on Parallel Distributed Information Systems, Miami Beach, FL, USA*, pp. 262–270, Dec. 1991.

[6] T. Leung and R. Muntz. Query Processing for Temporal Databases. In *Proc. of the 6th Int. Conf. on Data Engineering, Los Angeles, CA, USA*, pp. 200–208, Feb. 1990.

[7] T. Leung and R. Muntz. Temporal Query Processing and Optimization in Multiprocessor Database Machines. In *Proc. of the 18th Int. Conf. on Very Large Data Bases, Vancouver, Canada*, pp. 383–394, Aug. 1992.

[8] H. Lu, B.-C. Ooi, and K.-L. Tan. On Spatially Partitioned Temporal Join. In *Proc. of the 20th Internat. Conf. on Very Large Data Bases (VLDB), Santiago de Chile*, pp. 546–557, Sept. 1994.

[9] P. Mishra and M. Eich. Join Processing in Relational Databases. *ACM Computing Surveys*, pp. 63–113, Mar. 1992.

[10] M. Norman and P. Thanisch. *Parallel Database Technology: An Evaluation and Comparison of Scalable Systems.* Bloor Research Group, UK, 1995. ISBN 1-874160-17-1.

[11] M. Norman, T. Zurek, and P. Thanisch. Much Ado about Shared-Nothing. *SIGMOD Record*, 25(3), Sept. 1996.

[12] G. Piatetsky-Shapiro and C. Connell. Accurate Estimation of the Number of Tuples Satisfying a Condition. In *Proceedings ACM SIGMOD 1984 Conf. on Management of Data*, pp. 256–276, 1984.

[13] S. Rana and F. Fotouhi. Efficient Processing of Time-Joins in Temporal Data Bases. In *Proc. of the 3rd Internat. Symposium on Database Systems for Advanced Applications*, pp. 427–432, Apr. 1993.

[14] M. Soo, R. Snodgrass, and C. Jensen. Efficient Evaluation of the Valid-Time Natural Join. In

Proc. of the 10th Int. Conf. on Data Engineering, Houston, Texas, USA, pp. 282–292, Feb. 1994.

[15] T. Zurek. Parallel Temporal Nested-Loop Joins. Technical Report ECS-CSG-20-96, Dept. of Computer Science, Edinburgh University, Jan. 1996.

[16] T. Zurek. Optimisation of Partitioned Temporal Joins. To appear in *Proc. of the 15th BNCOD Conf. London, UK*. Springer, July 1997.

A Performance Modelling

Stage	Disk I/O	Communication	CPU	Memory
1 (a)	$\frac{\lvert R\rvert}{N} \cdot \frac{r}{w_D}$		$\frac{\lvert R\rvert}{n \cdot N} \cdot \frac{r}{b} \cdot \frac{I_{sio}}{\mu}$	
1 (b)		$\frac{N-1}{N} \cdot \lvert R\rvert \cdot \frac{\delta_R}{n} \cdot \frac{r}{w_C}$	$\frac{N-1}{N} \cdot \frac{\lvert R\rvert}{n \cdot N} \cdot \frac{\delta_R}{n} \cdot$ $\left(\frac{r}{b} \cdot \frac{I_{scomm}}{\mu} + \frac{3 \cdot I_{exp}}{\mu}\right)$	
1 (c)				$\frac{\lvert R\rvert}{N} \cdot \frac{r}{w_M} \cdot \max\{\delta_R, n\}$
1 (d)	$\frac{\lvert R\rvert}{N} \cdot \delta_R \cdot \frac{r}{w_D}$		$\frac{\lvert R\rvert}{n \cdot N} \cdot \delta_R \cdot \frac{r}{b} \cdot \frac{I_{sio}}{\mu}$	

Table 2: Performance model for the partitioning stage (stage 1)

Stage	Disk I/O	CPU
2	$n \cdot \left(1 + \frac{\lvert S\rvert}{n \cdot N}\delta_S\right) \cdot \frac{\lvert R\rvert}{n \cdot N}\delta_R \cdot \frac{r}{w_D}$	$\left(1 + \frac{\lvert S\rvert}{n \cdot N}\delta_S\right) \cdot \frac{\lvert R\rvert}{n \cdot N}\delta_R \cdot \frac{r}{b} \cdot \frac{I_{sio}}{\mu}$ $+ \frac{\lvert R\rvert}{n \cdot N}\delta_R \cdot \frac{\lvert S\rvert}{n \cdot N}\delta_S \cdot \frac{I_{proc}}{\mu}$

Table 3: Performance model for the joining stage of join A (stage 2)

Stage	Disk I/O	CPU
2 (a)	$n \cdot \left(1 + \frac{\lvert S\rvert}{n \cdot N}\right) \cdot \frac{\lvert R\rvert}{n \cdot N} \cdot \frac{r}{w_D}$	$\left(1 + \frac{\lvert S\rvert}{n \cdot N}\right) \cdot \frac{\lvert R\rvert}{n \cdot N} \cdot \frac{r}{b} \cdot \frac{I_{sio}}{\mu}$ $+ \frac{\lvert R\rvert}{n \cdot N} \cdot \frac{\lvert S\rvert}{n \cdot N} \cdot \frac{I_{proc}}{\mu}$
2 (b)	$n \cdot \left(1 + \frac{\lvert S\rvert}{n \cdot N}(\delta_S - 1)\right) \cdot \frac{\lvert R\rvert}{n \cdot N} \cdot \frac{r}{w_D}$	$\left(1 + \frac{\lvert S\rvert}{n \cdot N}(\delta_S - 1)\right) \cdot \frac{\lvert R\rvert}{n \cdot N} \cdot \frac{r}{b} \cdot \frac{I_{sio}}{\mu}$ $+ \frac{\lvert R\rvert}{n \cdot N} \cdot \frac{\lvert S\rvert}{n \cdot N}(\delta_S - 1) \cdot \frac{I_{proc}}{\mu}$
2 (c)	$n \cdot \left(1 + \frac{\lvert R\rvert}{n \cdot N}(\delta_R - 1)\right) \cdot \frac{\lvert S\rvert}{n \cdot N} \cdot \frac{r}{w_D}$	$\left(1 + \frac{\lvert R\rvert}{n \cdot N}(\delta_R - 1)\right) \cdot \frac{\lvert S\rvert}{n \cdot N} \cdot \frac{r}{b} \cdot \frac{I_{sio}}{\mu}$ $+ \frac{\lvert R\rvert}{n \cdot N}(\delta_R - 1) \cdot \frac{\lvert S\rvert}{n \cdot N} \cdot \frac{I_{proc}}{\mu}$

Table 4: Performance model for the joining stage of join B (stage 2)

Stage	Disk I/O	CPU	Memory
2 (a)	$\lambda \cdot n \cdot \left(\frac{\lvert S\rvert}{m}(\delta_S - 1) + \frac{\lvert R\rvert}{m}\right) \cdot \frac{r}{w_D}$	$\lambda \cdot \left(\frac{\lvert S\rvert}{m}(\delta_S - 1) + \frac{\lvert R\rvert}{m}\right) \cdot \frac{r}{b} \cdot \frac{I_{sio}}{\mu}$ $+ \lambda \cdot \frac{\lvert R\rvert}{m} \cdot \frac{\lvert S\rvert}{m}(\delta_S - 1) \cdot \frac{I_{proc}}{\mu}$	$\lambda \cdot n \cdot \frac{\lvert R\rvert}{m} \cdot \frac{\lvert S\rvert}{m}(\delta_S - 1) \cdot \frac{r}{w_M}$
2 (b)	$\lambda \cdot n \cdot \frac{\lvert S\rvert}{m} \cdot \frac{r}{w_D}$	$\lambda \cdot \frac{\lvert S\rvert}{m} \cdot \frac{r}{b} \cdot \frac{I_{sio}}{\mu}$ $+ \lambda \cdot \frac{\lvert R\rvert}{m} \cdot \frac{\lvert S\rvert}{m} \cdot \frac{I_{proc}}{\mu}$	$\lambda \cdot n \cdot \frac{\lvert R\rvert}{m} \cdot \frac{\lvert S\rvert}{m} \cdot \frac{r}{w_M}$
2 (c)	$\lambda \cdot n \cdot \left(\frac{\lvert R\rvert}{m}(\delta_R - 1) + \frac{\lvert S\rvert}{m}\right) \cdot \frac{r}{w_D}$	$\lambda \cdot \left(\frac{\lvert R\rvert}{m}(\delta_R - 1) + \frac{\lvert S\rvert}{m}\right) \cdot \frac{r}{b} \cdot \frac{I_{sio}}{\mu}$ $+ \lambda \cdot \frac{\lvert R\rvert}{m}(\delta_R - 1) \cdot \frac{\lvert S\rvert}{m} \cdot \frac{I_{proc}}{\mu}$	$\lambda \cdot n \cdot \frac{\lvert R\rvert}{m}(\delta_R - 1) \cdot \frac{\lvert S\rvert}{m} \cdot \frac{r}{w_M}$

Table 5: Performance model for the joining stage of join C (stage 2)

Parameter	Description	Value in the Experiments
$\lvert R\rvert, \lvert S\rvert$	number of tuples in relations R, S	100000 tuples
r	size of a tuple in bytes	500 bytes
$\lvert T\rvert$	length of the joint relation spans $T = T_R \cup T_S$	5000 time units
τ_R, τ_S	*average lengths of the tuple timestamps in R, S*	*100 time units*
δ_R, δ_S	the average number of fragment-ranges that a tuple timestamp spans	derived from $\tau_R, \tau_S, \lvert T\rvert, m$

Table 6: The workload parameters

Parameter	Description	Value in the Experiments
N	number of nodes	varied
n	number of processors per node	4
μ	processor speed in MIPS	100 MIPS
w_D	disk I/O bandwidth per node	20 MB/sec
w_C	communication bandwidth	100 MB/sec
w_M	memory bandwidth per node	400 MB/sec
I_{proc}	number of CPU instructions for processing a tuple in each step	1000
I_{exp}	number of CPU instructions for computing arithmetic expressions *fragment*$_p(t)$	$\frac{I_{proc}}{10} = 100$
I_{scomm}	number of CPU instructions for initiating a data transfer	500
I_{sio}	number of CPU instructions for initiating a disk I/O	500
mem	amount of shared memory per node avalaible for data structures	8 MB
b	page size	4 kB

Table 7: The parameters describing the parallel architecture

Phasme: A High Performance Parallel Application-Oriented DBMS

Andres Frederic
Visiting researcher at National Center for Science Information Systems, Japan
email: andres@rd.nacsis.ac.jp
AND
Ono Kinji
Professor and Director of R&D at National Center for Science Information Systems, Japan
email: ono@rd.nacsis.ac.jp

Keywords: Application-oriented DBMS, Parallelism, Performance Evaluation

Edited by: Tadeusz Morzy
Received: October 31, 1996 **Revised:** May 9, 1997 **Accepted:** June 16, 1997

This paper presents the architecture of Phasme - a high performance application-oriented database system manager providing key facilities for the fast development of object-oriented, relational-based or other kinds of applications. Differing from conventional database systems, application-oriented servers are independent of a particular data model but cooperate with any, offer facilities to exploit new hardware architecture trend, are fully general to support efficiently wide range of heterogeneous objects, and offer facilities to enforce applications consistency of related objects. Phasme, a Parallel Application-Oriented Database System(AODMS) has been designed to meet the new information systems' requirements and to use the power and the trends of the new generation of hardware. The major contributions of Phasme are the application-oriented architecture, the data storage manager, the dynamic optimization of both inter and intra-operation parallelism, and the exploitation of operating system services such as multi-threading or memory mapping for efficient concurrent executions.

1 Introduction

To satisfy the next generation of heterogeneous information system's demands, the DBMSs have to support and to distribute a wide variety of complex and dynamic data types (e.g. audio, video, image, text) to a wide variety of machines in enterprisewide, heterogeneous environments. DBMSs have done an important step to support Client/Server architecture and non-standard applications as hypermedia systems[13], imaging database[24], on-line control systems[21], or CAD-CAM applications. However, none of the DBMSs is general enough to support effectively a large spectrum of different applications yet as it is shown in [31]. The DBMS overview proposed by [25] indicates that there is a room for further improvements and there is also a need for more effective DBMS implementation techniques. Even if improvements in term of customizability have been achieved as it is shown in [35], vertical customizability from the data definition to the execution model is not yet supported as far as we know. This paper presents, Phasme, a parallel Application-Oriented DBMS under development since 1994. It has already achieved successes in Delivery Information System[5], in WWW content-based document retrieval[6] and supporting video on demand([3], [4]). The design goals of Phasme are to achieve a high performance server, and to meet notably multimedia information systems' demands by developing customizable features. There is a need to provide application-oriented features to serve a broad specific application requirements. The application-oriented design also preserves both the data independence and the knowledge independence by using an extended graph storage structure.

To achieve these goals, the Phasme database server relies on the combination of the Decomposition Storage Model [38] and the DBGraph Storage Structure [37]. It uses a simple graph data storage structure along the lines of the Graph Data Model [26]. The resulting data structure is called Extended Binary Graph. The Phasme server largely uses the main memory with virtual memory management mechanisms.

As advocated for Cricket[33], ObjectStore[28], Texas[34], QuickStore[39], and other similar DBMS architectures, a DBMS using a virtual memory management based on a pointer swizzling mechanism[23] (called PS/VM) appears to be the only alternative to high-end systems according to new emerging hardware architecture. However, a DBMS using a virtual memory management without pointer swizzling mechanism(called NoPS/VM) is an interesting alternative which can provide similar or better query processing performance. Dali[22] and Monet[12] are some examples of DBMSs with no pointer swizzling mechanism included inside the memory management. Phasme is related to this last group. Intuitively, a DBMS architecture with no pointer swizzling-based virtual memory mechanism has less mapping overhead: the object format is the same on disk as in memory with no conversion overhead. One drawback could be a low reliability to access persistent objects by no deferencing standard virtual memory pointers, introducing the need for software checks. The use of NoPS/VM mechanism to support an Extended Binary Graph is a challenge to achieve high performance. Another motivation for Phasme is to support main memory algorithms for object parallel management. The rationale of this decision is that most of the main memory algorithms designed for PS/VM architecture are also relevant for NoPS/VM architecture.

The focus in Phasme is to support both non-query languages (SGML,HTML,Tcl/tk) and query languages such as SQL3 and ODMG-93 compilation with dynamic optimization and automatic parallelization. To achieve performance goals, Phasme exploits both inter and intra-operation parallelism. Overall, the contribution of Phasme is to show that a parallel application-oriented DBMS based on NoPS/VM mechanisms simplifies the object management and improves the performance of the database management using customizing and parallel capabilities while meeting new information system's requirements and end-users' needs. The optimizer integrates neural network technologies to customize query optimization strategies according to the target applications. The run-time system exploits much the underlying operating system low levels as memory mapping and multithreading to minimize the overhead of parallelism. In this paper, we concentrate on the design choices, on the architecture of Phasme and on report on the performance experiments of two major application domains: (1) object manipulation and (2) text retrieval. The remainder of the paper is organized as follows. Section 2 provides and discusses the design principles used inside Phasme. Section 3 describes the overall architecture of the system. Section 4 reports on performance evaluations and experiments made with the current implementation on a SPARC Station sun20. Section 5 concludes.

2 Design Overview

This section gives an overview of the major decisions which led to Phasme's architecture. Obviously, some of these decisions were made in order to support existing standards (UNIX, CORBA, OQL, SQL) and to provide high performance.

2.1 Application-oriented concept

The major improvement provided by the application-oriented concept is the vertical customizability from the data definition to the execution model to enable the application to tailor the DBMS according to its own requirements.

2.2 Design Principles

Since designing DBMS is becoming more and more complex, it was essential to adopt principles for the design of the DBMS architecture following the methodology presented in [17]. First, they stem from the teachings of previous research prototypes (DBS3, QuickStore, Dali, Monet, and others) and of previous commercial systems. Second, it is important to re-use strong state-of-the-art technologies which have already been thoroughly tested. Therefore, following principles have been established.

- **Customizable interoperability-based approach.** The advantages of the interoperability approach are simplicity and high performance to exchange heterogeneous information between servers and among client applications inside distributed and heterogeneous environment. Combining a similar approach to the customizability-orientation into Phasme leads to shift data models' constraints to libraries for user applications. Furthermore, this approach extends in some way the ODMG-standard CORBA with customizability mechanisms. This approach enables the management of heterogeneous data and it also eases the integration of the various requirements of complex applications as data mining, knowledge discovery or multimedia applications. Phasme is a satellite DBMS which can discuss and exchange data with already existing DBMSs.

- **Main memory assumption.** To support new hardware trends, we assume that the hot data set fit in virtual main memory. This is an important factor to get high performance.

- **Rely on memory mapped file mechanisms.** Most of the new object-oriented database systems and parallel database systems ([22], [12], [10]) rely on the memory mapped file concept whose major advantage is the same format between the in-memory data and the on-disk data.

- **Rely on operating system's functionality.** The functions mmlock, mmap, and madvise are integrated into the Phasme implementation. However the trends in microkernel operating systems as Chorus or real-time Mach ([11]) indicate that they will provide efficient features as real-time scheduler, time-driven prefetching policies, real-time threads, synchronization and real-time inter-process communication (IPC).

- **Micro-kernel system.** The implementation of the Phasme system has been optimized at any level to produce an efficient kernel with size of code. This enables to provide a information engine for embedded system.

2.3 Storage Structure and Execution Model

The data storage structure and the execution model respectively represent the way that the application sees the data and the way that the application's requests are executed. Storage structure and execution model design can be seen as the first step of the query optimization. The storage structure is mandatory to integrate rich modeling power to reach wide application requirements. The execution model should provide efficient low-level execution. To understand the way that the execution model will be mapped on an extended object storage structure, it is important to recall the characteristics of object storage models largely based on [38].

- **N-ary Storage Model (NSM).** A NSM-based architecture (Figure [1]) is composed of objects which are considered as N-ary tuples. Each tuple is constituted of all the items characterizing the related object. This architecture forbids clustering objectives.

- **Decomposition Storage Model (DSM).** A DSM-based architecture (Figure [2]) is composed of objects whose items are stored separately. This architecture consists of triple-based data representation (OID, Item, Value). This architecture facilitates fragmentation objectives.

- **Partial Decomposition Storage Model (PDSM).** A PDSM-based architecture (Figure [3]) is composed of objects which have been vertically fragmented on specific items. Replication could become a shortcoming of this data structure.

To achieve good caching hit ratios, good load balancing and high response time, data are partitioned in data clusters. Recent works ([27],[32]) have shown that partitioned-based algorithms increase the query processing performance. In NSM and PDM, data partitions become a bottleneck while DSM seems to be the most suitable architecture. To achieve high performance for intensive object manipulation-based applications (e.g. office multimedia application), data and code fragmentation should also be considered, otherwise data conflicts appear.

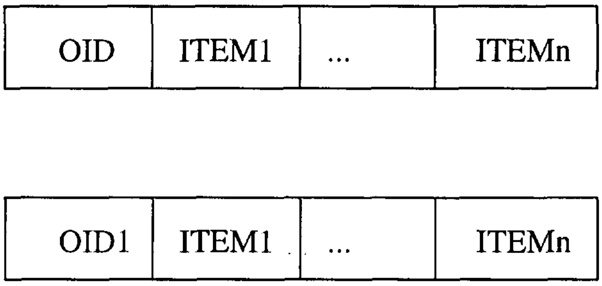


Figure 1: N-ary Storage Model

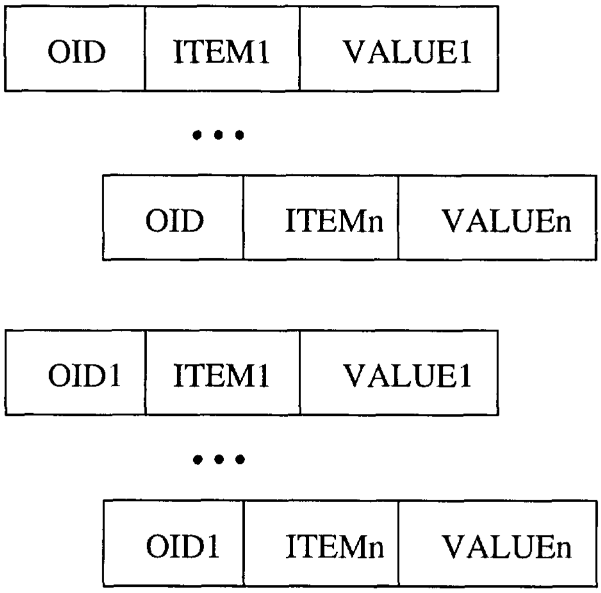


Figure 2: Decomposition Storage Model

2.4 The Storage Structure

The storage structure is an important performance factor for a database system as it has been shown in [36]. The major issue is to reduce the I/O costs for data retrievals according to the data structure and data placement on disks. As it has been pointed out in [18], the physical data independence is also a cornerstone of a new generation of DBMSs. The separation between the physical representation of the data and its semantic enables to take out data model logical aspects from the DBMS kernel.

One Virtual Memory-base Level: Unlike DBS3[8] which implemented a two-level store to distinguish between permanent and temporary data, we adopt one virtual memory-based level to store both kinds of data. The permanent data are shared among the users whereas temporary data are not shared.

DSM Model: The Decomposition Storage Model[38] has been chosen to implement the Data Storage Structure of Phasme because it enables to select independently each item and to change dynami-

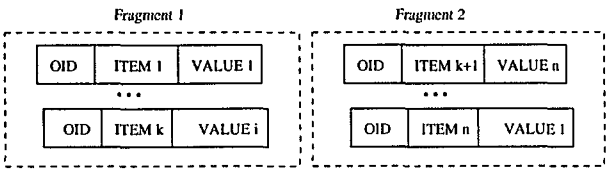


Figure 3: Partial Decomposition Storage Model

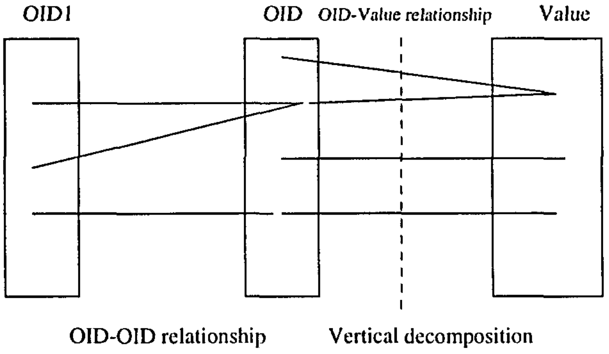


Figure 4: Phasme data storage structure

cally the object structure. Furthermore, this storage model is suitable with distributed environments.

Binary Graph: A Binary Graph structure following previous works ([38],[37], [26]) has been chosen to implement the data storage structure because it enables efficient data access methods [1] and assures a compact data structure to maximize the probability that the hot data set fits in main memory. This data storage structure enables to support wide different application domains.

Persistent Data: Persistent data are directly manipulated inside the binary graph in which they are stored, without incurring any in-memory copying cost.

At the storage level, Phasme manipulates Extended Binary Graphs (EBG) resulted in the combination of the binary graph approach, of the Decomposition Storage Model approach, and of the Graph Data Model approach. Phasme partitions all the information in Extended Binary Graphs (EBG) as it is shown in Figure [4] in order to support high performance access methods. The EBG data structure is based on no-oriented arcs, a set of arcs representing one object item. Each arc is composed by two extremities (source = OID, destination = value) which can be inversed according to access methods. The database consists of a number of EBG files which are UNIX files. Partition and replication are used to accelerate data retrieval when data are mainly on disk. Partitioning can be done on EBG data structure for optimal accesses in two ways: (1) on the EBG extremities, and (2) on search data structure. For example, hashing and index data structure such as G-TREE and BANG are used respectively in (1) and (2) cases. Other specific structures as signature file can be added to support specific data processing as text retrievals.

2.5 The Execution Model

The execution model is a parallel dataflow execution model which supports main memory data operators, transfer operators, and control operators. The vehicle of the query processing is DORLA[2], an algebraic language incorporating support for Deductive, Object and Relational capabilities. Following previous research work[20], DORLA is characterized as a many-sorted algebra which eases the customizability of the interface and internal languages as adding new types or new modules according to application's requirements.

The Phasme compiler transforms a user query into a parallel execution plan statically optimized. The parallel execution plan will be dynamically optimized at runtime according to Phasme's behavior. Each execution plan is represented as a directed graph of operators. The rationale of this approach is to perform compile-time optimization and run-time optimization while keeping the server simple and providing very efficient parallel data accesses. A major issue addressed by the run-time optimization is to maximize the query processing throughput and to minimize the response time of users. This is dynamically achieved through a mechanism based on the concept of matching the available resources of the environment and the operators of the dataflow in order to evaluate the degree of parallelism of the query execution plans. This mechanism is described in [7].

3 The Phasme Architecture

Phasme is based on a Client/Server organization. The users are viewed as processes. Phasme is fully customizable from the data types to the execution model as it is shown in Figure [5]. The kernel is based on main three subsystems: the Service manager, the Data Manager, and the Memory Manager as it is shown in Figure [6].

3.1 The Service Manager

The Service Manager, based on an ORB-like architecture (CORBA 1.2), provides the entry points to Phasme both for the application and the user front-ends. It receives request expressed in IDL (Interface Data Language) following the CORBA mechanism([30]). The interoperability feature provides an implementation independence to exchange data inside distributed and heterogeneous environments. Furthermore, the use of CORBA approach enables to encapsulate dynamically applications as a

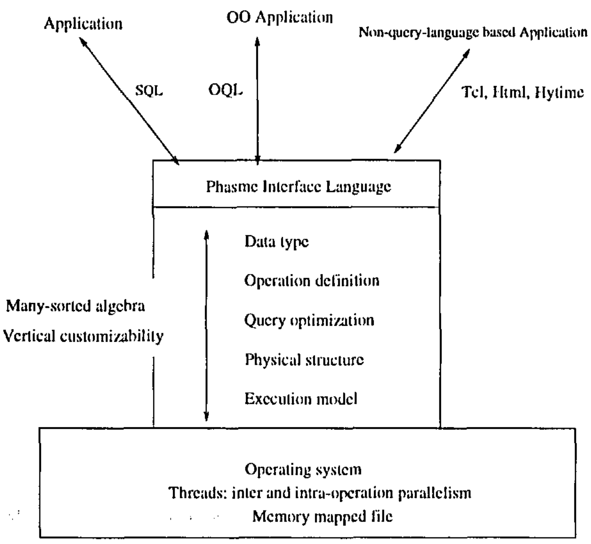


Figure 5: Phasme Architecture

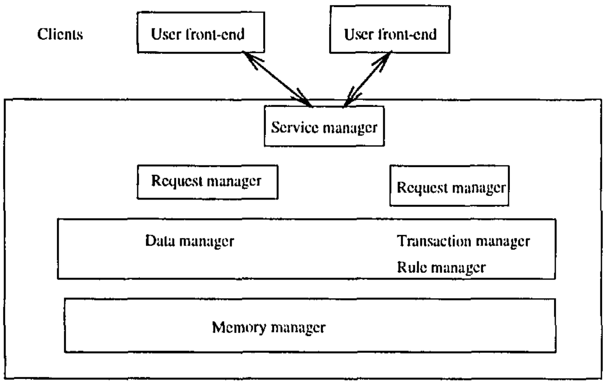


Figure 6: Phasme Component Architecture

set of distributed objects and their associated modules. This property enables to manage concurrently heterogeneous information and applications based on different data models.

3.2 The Data Manager

The Data Manager provides all the functions of the database system needed to support the execution of several parallel query execution plans. The Data Manager includes the transaction management and the rule manager (active mechanism of Phasme). Each compiled execution plan is dynamically optimized according to the behavior of the server and then the result is dynamically linked with a set of library functions. These libraries correspond to some sets of functions defined according to the application's data model (e.g. object-oriented, relational, or deductive) and its semantics.

3.3 The Memory Manager

The Memory Manager provides a direct support for access methods to persistent data and volatile data. It is based on standard techniques (e.g. strict 2 PL, 2PC) as well as operating system functions (mmap, madvise, mlock) for main memory management. We took the assumption that the hot-set of the database fits into the main memory for all the concurrent transactions and the other data may be swapped out to disks. To achieve the memory management, Phasme uses the operating system mechanism of memory mapped file. The database is mapped into the virtual memory as contiguous logical address space. This memory management mechanism is bounded by the size of the virtual memory space and it requires no pointer swizzling. The memory management includes three main modules : (1) the Memory Manager it-self, (2) the Lock Manager, and (3) the Recovery Manager. The **Memory Manager** manages all the transfers of memory mapped files between disk and memory. It supports a LRU caching policy for EBG manipulations. The **Lock Manager** serializes the concurrent transactions by a strict two phase locking mechanism([9]). The **Recovery Manager** ensures by a two phase commit algorithm that all the modifications of the database done by committed transactions are visible and persistent.

4 Performance Experiments

In this section, we report performance experiments of two kinds of workload made with the Phasme prototype: object manipulation and textual retrieval. The motivation of the choice of these two kinds of workload (Object manipulation and Content-based textual retrieval) was the increasing influence of those two aspects in major real applications. Furthermore, we concentrate on the performance of the kernel layer for both the two kinds of workload. The platform used in the result presented here is a bi-processor Sun SPARC Station 20/50 MHz running Solaris 2.4, with 96 MB main memory, 16 KB data-, 20 KB instruction and 1MB secondary cache, local disk and swap space.

4.1 OO7 Experiments

In order to get insights in the prototype's behavior, we have investigated a subset of the OO7 benchmark [14] in increasing order of complexity: exact lookup (Q1), scan (Q2,Q3,Q7), path lookup(Q4), single-level make(Q5), join (Q8), insert and delete. The metric used in the performance experiments is completion time. The experimental database was generated automatically following the specification of the OO7 benchmark. the size of the medium database (fanout 9) is equal to 69 MB. The parallelization of the operation

Table 1: Performance Results

queries	Cold time	Hot time
Q1	0.67 s	0.46 s
Q2 (1%)	0.52 s	0.51 s
Q3(10%)	0.57 s	0.53 s
Q4	0.74 s	0.53 s
Q5	1.7 s	1.56 s
Q7(100%)	1.80 s	1.62 s

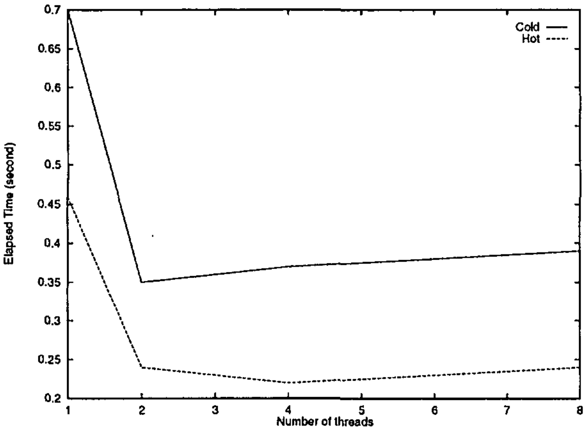


Figure 7: Exact match lookup query

relies on the Solaris thread management system. The Solaris system effectively distributed the work between the available processors, when several threads worked concurrently. We assumed a perfect parallelism resulting from a good distribution between threads and processors. The performance results of the queries for the medium/9 database are shown in Table 1.

Figure [7] compares the results of the cold and hot exact match lookup query execution varying the number of threads for the medium database (fanout 9). In each case, the system used the clustered index to provide high performance.

Figure [8] compares the results of the cold and the hot scan execution of the all atomic parts varying the number of threads for the medium database(fanout 9). The degree of intra-operator parallelism enables to improve the efficiency of the OO query processing.

It is also important to evaluate the performance behavior of structural operations as insert and delete to stress build ability. Figure[9] shows the results of insert and delete operations. The intra-operation parallelism between the insertion operations enables to decrease the response time by a factor 5 if the number of threads is equal to the number of new composite parts. The same result is obtained for the delete query.

Raw traversal speed has been also studied. Figure [10] shows that the cold times are maximal for a factor 1.6 slower that the hot times. The main memory approach is not a major cost factor.

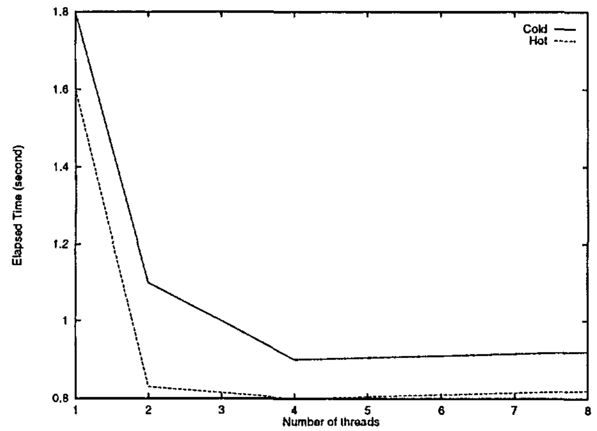


Figure 8: Scan query, medium db/9

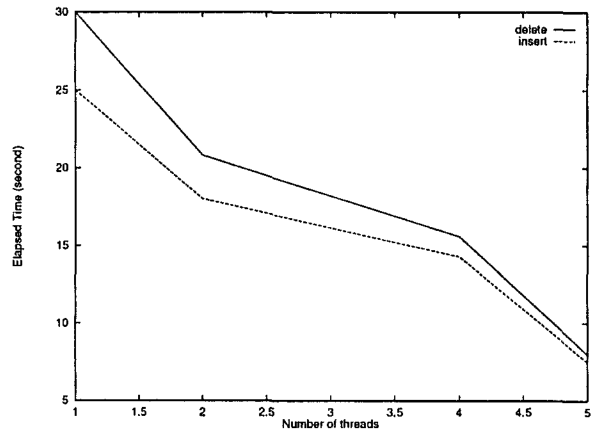


Figure 9: Delete and insert, cold, medium db/9

The association between EBGs and parallel query executions is the key factor to provide high performance.

4.2 Textual Retrieval Experiments

To demonstrate the feasibility and effectiveness of Phasme, we implemented the multilevel superimposed coding method([29]). Results on Signature Files have been published for both research and commercial text retrieval servers([15],[16]) in term of disk-oriented data accelerators. Our implementation of signature files sets a new mile-stone for further developments in this area in the field of main memory text retrieval. The implementation of the Phasme textual datatypes are based on the Phasme plug-ins interface.

4.2.1 Extension Plug-ins

To support large sets of documents, we have implemented document types and the multilevel superimposed coding method as index. Since new information

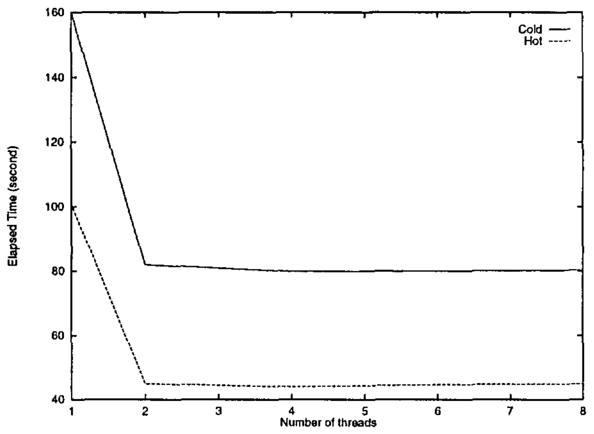


Figure 10: Traversal T1, medium size

systems need index which perform well in both main-memory as disk-based settings, we completed the signature file implementation with clustering operations.

All the operations related to the signature file plugins are shown below:

```
Plug-ins Document;
 ITEM Document;
  COMPARE = DOCUMENT_compare;
  ERASE= DOCUMENT_erase;
  HASH= DOCUMENT_hash;
  INSERT = DOCUMENT_insert;
  READ= DOCUMENT_read;
END Document;

Plug-ins SF;
 USE Document;

 INDEX SF;
  CREATE = SFcreate;
  ERASE = SFerase;
  INSERT=SFinsert;
  SAVE=SFsave;
 END;
END SF;
```

4.2.2 Performance Evaluation

– **With and without Signature File**

The first set of measurements shows how the introduction of the signature file access method based on the multilevel superimposed coding method contributes to the improvement of the text retrieval. Figure [11] shows the average elapsed time of text retrieval queries varying the number of documents from 10000 to 200000 when the kernel is a using a single thread.

The performance evaluation of the EBG data structure based on the memory-mapped file points out that the memory-mapped file mechanism is also efficient for selection queries.

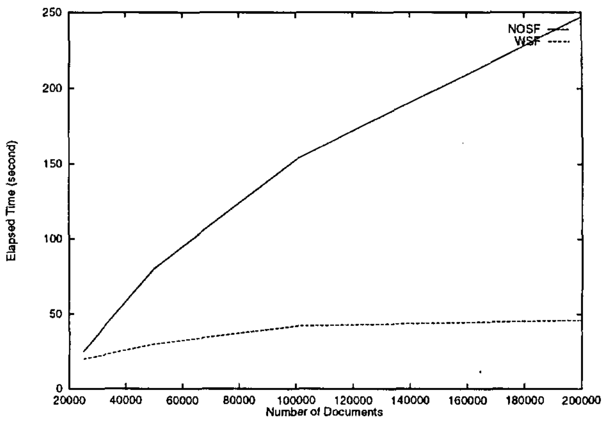


Figure 11: Text retrieval performance using signature index (WSF) and without using signature index (NOSF)

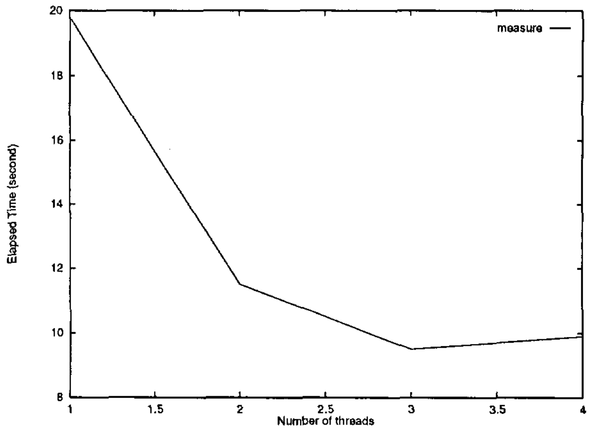


Figure 12: TR and intra-operation parallelism

– **Intra-operation Parallelism**

Figure[12] shows the influence of the intra-operation parallelism of Phasme on the text retrieval processing. Database size is set to 200 000 documents. This experience was run on the same kind of SUN workstation but in this case the machine contains 2 processors. We varied the number of threads from 1 to 4 threads. We have horizontally clustered the multilevel signature file structure in several buckets of signatures. In this way, the set of threads can be linked to the set of buckets (signature). The intra-operation parallelism mechanism of the execution model considerably improves the retrieval performance.

The creation of 4 threads introduces some overhead due to the small number of processors. There are some thread overlapping on the pro-

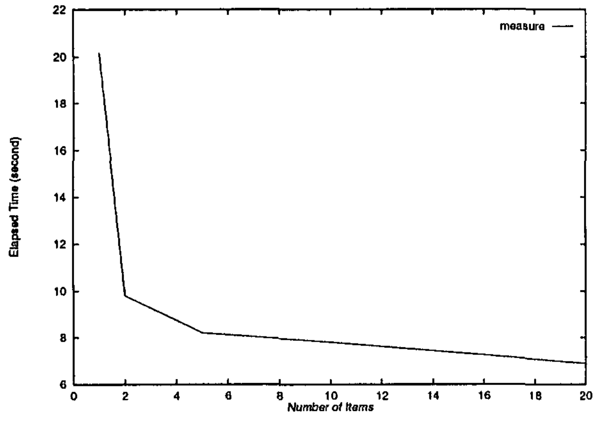


Figure 13: TR and variation of items

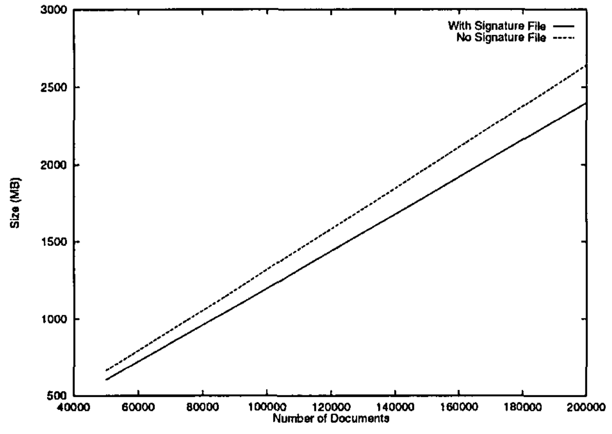


Figure 14: Influence of the signature file structure on the database size

cessors. Figure [13] combines the variation of items and inter-operation parallelism processing. The increase of the number of items improves the elapsed time. Figure [14] shows the influence of the signature data structure on the size of the database. The result points out that the signature file has nearly no impact on the total size of the databases.

- **Length of the Signature**

  We varied the length of the signature at the first level from 1000 bytes to 4000 bytes. This variation has been done for three different sets of documents (e.g. 10, 000; 50, 000; 100, 000). Figure [15] shows the influence of the length of the signature at the first level on the elapsed time. We have verified that the increment of the length of the signature at the first level decreases the false drop probability, and thus improves the retrieval performance.

- **Weight of Signatures**

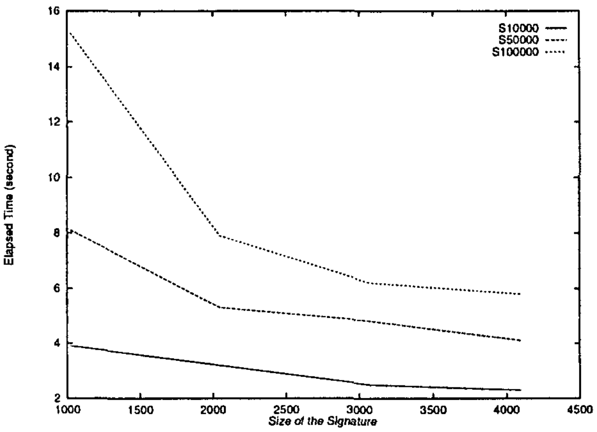


Figure 15: Influence of the lenght of the signature on the performance

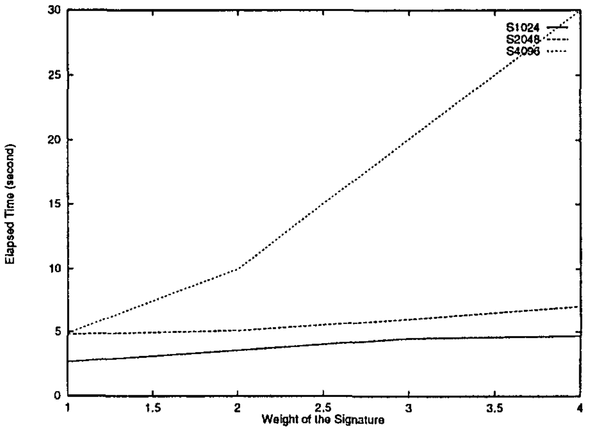


Figure 16: Influence of the weight of the signature on the Performance

We examined the effect of the weight of signature on the performance. We confirmed that the smallest weight gives the best configuration for high performance as it is shown in Figure [16].

5 Conclusion

Phasme is a parallel Application-Oriented DBMS whose goals are first to provide a new approach for information processing technology and second to satisfy both the requirements of the new generation of information systems and the hardware trends. Phasme supports a main memory data storage structure called Extended Binary Graph (EBG). In this paper, we have presented the design decisions, the execution model, and the architecture of Phasme. We also reported experiments with the current implementation of two major workload domains (object manipulation, textual retrieval). The alpha implementation of Phasme (V1) was complemented in June 1995 and the beta release

(V2) was completed in April 1996. Phasme DBMS is implemented in C. Phasme server is operational for Sparc/Sun Solaris and DEC alpha platforms. Phasme client is operational for Solaris, Macintosh Power PC and Windows 95. Also, we are planning on porting Phasme on Windows NT. An extension of Phasme to a distributed database system (DPhasme) is the purpose of the TOSDHIM project (cooperation between Kyushu University, NACSIS, and Paris VI/MASI University). We consider the following features to be the most significant contributions of the Phasme project:

- **Parallel execution model.** Phasme implements a parallel dataflow execution model based on the Extended Binary Graph structure which provides data model independence but also allows to collaborate with any. Compared to other main-memory systems such as Texas and Cricket, this leads to support efficiently intra-operation parallelism for object management.

- **Parallel optimization.** The Phasme supports several optimization strategies in order to produce parallel query plans. The optimizer can be customized according to the target environment and neural network-based cost models.

- **Run-time system.** The run-time system exploits the virtual memory functions and advanced features of the Solaris or DEC operating system to minimize the overhead of the parallelism. Furthermore, it implements a data structure which does not use any pointer swizzling to map data into memory. The data format on disk is the same as the in-memory one.

The performance evaluation of the Phasme DBMS against two different kinds of workload (object manipulation, textual retrieval) was useful to point out three things. First, the implementation of an Application-oriented DBMS based on memory mapped files and allowing intra-operation and inter-operation parallelisms improves the database management performance. Second, the customizability of Phasme implementation allows to meet with higher simplicity the OO applications' needs. Our performance experiment using the OO7 benchmark has been done in order to stress the response time for single-user queries. It shows a good parallel query processing. It also partly confirms our intuition about the need of a new generation of data storage server. Third, signature files which are usually disk-oriented data accelerators are also very efficient as memory-oriented data accelerators. The implementation of the Phasme's memory management based on the memory-mapped file concept and based on the Extended Binary Graph (EBG) data structure provide the opportunity to manage efficiently signature files. The memory mapped file concept avoids the overhead for data retrieval introduced by traditional DBMSs. It gives an uniform data format for memory and disk data storage and manipulations. As a customizable DBMS, Phasme *(a version for cooperation is available on request at phasme@rd.nacsis.ac.jp)* is being used inside the AHDS project (Active Hypermedia Delivery System) and inside the MODOS project (Museum On-Demand Open System) at NACSIS, experimental site for this emerging database system.

6 Acknowledgments

We would like to thank Dr. Boulos (NACSIS), Dr. Kunii (Ricoh), Dr. Fuluse (Ricoh), Professor Kerstern (CWI), and Dr. Robertson (CSIRO) for their valuable comment and helping us improve the architecture of Phasme. Thanks are also due to Dr. Lee (Hong Kong UST) for his valuable help to provide us the source of multi-level signature file algorithms and for his advises. We thanks also NACSIS members for their useful comments and reviews.

References

[1] Analyti A., & Pramanik S.(1992) Fast Search in Main Memory Databases *in Proc. ACM SIGMOD*, pp 215-224.

[2] Andres F.(1994) DORLA a Many-sorted Algebra Language to reach multiple Data Models' Requirements *Phasme Project*, report No 941105.

[3] Andres F., Ihara K., Boulos J., Ono K., & Yasuda Y. (1996) The OLVP (OnLine Video Processing) System *Proc. Int. Workshop DMS96*, Hong Kong.

[4] Andres F., Boulos J., Ihara K., Ono K., & Yasuda Y. (1996) Performance Evaluation of the OLVP System *in Proc. Int. COMPSAC*, Seoul, Korean.

[5] Andres F., & Boulos J.(1996) DDS The Data Delivery System, IFIP 1996, *Advanced Intelligent Tool*, Canberra, Australia.

[6] Andres F., Boulos J., Lee D. L., & Ono K.(1996) Providing Information Retrieval Mechanisms inside a WWW Database Server for structured Documents Management, *in Proc. of ADB96*, Japan.

[7] Andres F., & Ono K. (1996) A High Efficient Multimedia DBMS in Object Management, *in Proc. of NACSIS Bulletin*, Japan.

[8] Bergsten B., Couprie M., & Valduriez P.(1991) Prototyping DBS3, A Shared Memory Parallel Database System *in Proc. PDIS 1991.*

[9] Berstein P., Hadzilacos V., & Goodmann N.(1987) Concurrency Control and Recovery in Database Systems ,*Addison-Wesley,* Reading, Massachusetts.

[10] Biliris A., & Panagos E.(1995) A High Performance Configurable Storage Manager *in Procs ICDE 95,* pp35-43.

[11] Black D.L., et al.(1992) Microkernel Operating System Architecture and Mach *in Procs of the workshop on Microkernels and other Kernel Architectures.*

[12] Boncz P.A., & Kerstern M.L.(1995) Monet: An Impressionist Sketch of an Advanced Database System *In Proc. IEEE BITWIT Workshop,* San Sebastian (Spain).

[13] Buford J.L., & Rutledge L. (1997) Third Generation Distributed Hypermedia Systems *In Multimedia Information Management Handbook* (ed. W. Grozky),Prentice Hall.

[14] Carey M., DeWitt D.J., & Naughton J.F. (1993) The DEC OO7 Benchmark *in Proc. ACM SIGMOD 1993,* pp12.

[15] Chang W.W., & Schek H.J. (1989) A Signature Access Method for the Starburst Database System, *Proc. 15th Int'l Conf. Very Large Databases,* Amsterdam, The Netherlands, pp. 145-153.

[16] Furuse K., Asada K., & Iizawa A.(1995) Implementation and Performance Evaluation of Compressed Bit-Sliced Signature Files *in the proceeding of CISMOD95,* Bombay, India.

[17] Geppert A., & Dittrich K. R.(1994) Constructing the Next 100 Database Management Systems: Like the Handyman or Like the Engineer ? *in SIGMOD RECORD Vol. 23, No 1.*

[18] Graefe G.(1993) Options in Physical Database Design *in SIGMOD Record, Vol. 22, No 3,* pp 76-81.

[19] Gray J., & Reuter A. (1993) Transaction Processing Concepts and Techniques, *Morgan Kaufmann Publishers,* Inc., San Francisco, California.

[20] Guting R. H.(1993) Second Order Signature: A tool for specifying data models, query processing and optimization *in Proc. ACM SIGMOD.*

[21] Hatonen K., Klemettinen M., Mannila H., Ronkainen P., & Toivonen (1996) Knowledge Discovery from Telecommunication Network Alarm Databases *in IEEE ICDE,* pp 115-122.

[22] Jagadish H., Lieuwn D., Rastogi R., & Silberschatz A.(1994) Dali: A High Performance Main Memory Storage Manager *in Proc. of the 20th International Conference on VLDB. Santiago,* Chile, pp 48-59.

[23] Kemper A., & Kossman D.(1995) Adaptable Pointer Swizzling Strategies in Object Bases: Design, Realization, and Quantitative Analysis *in VLDB Journal,* pp 519-567.

[24] Khoshafian S., & Baker A.B. (1996) Multimedia and Imaging Databases, *Morgan Kaufmann Publishers,* Inc.

[25] Kim W.(1994) Modern Database Systems *Addison-Wesley,* ACM Press.

[26] Kunii H.S.(1990) Graph Data Model and its Data Language *Springer-Verlag,* 1990.

[27] Nyberg C., Barclay T., Cvetanovic Z., Gray J., & Lomet D.(1994) AlphaSort: A RISC Machine Sort *in Proc. ACM SIGMOD,* pp233-242.

[28] Lamb C., Landis G., Orenstein J., & Weireb D.(1991)The ObjectStore Database System *in Communication of the ACM,* 34(10):50-63.

[29] Lee D. L., Kim Y. M., & Patel G.(1995) Efficient Signature File methods for text retrieval, *in IEEE Transactions on Knowledge and Data Engineering,* Vol 7, No 3, pp 423-435.

[30] Object Management Group (1993) The Common Object Request Broker: Architecture and Specification, *Revision 1.2 ODMG Document No 93.12.1.*

[31] Rosenblatt B.(1994) Unix RDBMS: The Next Generation What are the Unix Relational database vendors doing to survive in the next generation of client/server environments*in SIGMOD RECORD,* vol. 23, No 4., pp 91-103.

[32] Shatdal A., Kant C., & Naughton J.F.(1994) Cache Conscious Algorithms for Relational Query Processing *in Proc. of the 20th VLDB Conference,* Santiago, Chile, pp 510-521.

[33] Shekita E.,& Zwilling M.(1990) Cricket: A Mapped Persistent Object Store *in Proc. on the 4th Int. Workshop on Persistent Object Systems,* Martha's Vineyard, MA, pp 89-92.

[34] Singhal V., Kakkad S.V., & Wilson P.R.(1992) Texas: An Efficient, Portable Persistent Store, *in Proc. 5th Workshop on Persistent Object Systems,* pp 1-33.

[35] Stonebraker V., & Moore D. (1996) Object-relational DBMSs The Next Great Wave, *Morgan Kaufmann Publishers,* Inc.

[36] Teeuw W.B., Rich C., Scholl M.H., & Blaken H.M.(1993) An Evaluation of Physical Disk I/Os for Complex Object Processing, *in Proc. ICDE,*Vienna, Austria, pp 363-372.

[37] Thevenin J.M.(1989) Architecture d'un Systeme de Gestion de Bases de Donnees Grande Memoire, Ph.D. Thesis of Paris VI University.

[38] Valduriez P., Khoshafian S., & Copeland G. (1986)Implementations techniques of Complex Objects, *in Proc. of the International Conference of VLDB*, Kyoto, Japan, pp 101-110.

[39] White S.J. , & DeWitt D.J.(1994) QuickStore: A High Performance Mapped Object Store, *in Proc. of the ACM SIGMOD 94*, Meneapolis, MN.

A Sliding-Window Approach to Supporting On-Line Interactive Display for Continuous Media

Chien-I Lee
Dept. of Computer and Information Science
National Chiao Tung University, Hsinchu, Taiwan, R.O.C
E-mail: leeci@dbsun1.cis.nctu.edu.tw
AND
Ye-In Chang
Dept. of Applied Mathematics
National Sun Yat-Sen University, Kaohsiung, Taiwan, R.O.C
E-mail: changyi@math.nsysu.edu.tw
AND
Wei-Pang Yang
Dept. of Computer and Information Science
National Chiao Tung University, Hsinchu, Taiwan, R.O.C

Keywords: data placement strategies, digital continuous media, interactive display, multi-disk drive, real-time database systems, striping

Edited by: Silvio Salza
Received: October 30, 1996 **Revised:** April 10, 1997 **Accepted:** May 7, 1997

To efficiently support continuous display for continuous media, many approaches based on the striping strategy that is implemented on a multi-disk drive have been proposed. However, the striping strategy can only support simultaneous display of continuous media which are predetermined before they are stored in the multi-disk drive. For an interactive display application, a system must support users to make any choice of objects for display even when display has started. Although Shahabi et. al. have proposed the replication and prefetching strategies for interactive display of continuous media, the combination of objects for display and the branch points for choices both have to be predetermined. Based on their strategies, they have to consider all the possible cases according to the given the combination of objects and the branch points of choices; it will require a lot of additional overhead of space and time. To reduce the overhead, in this paper, we will propose a sliding window approach to supporting interactive display for continuous media, in which we only record a little necessary information of retrieval of the following subobjects for display in a sliding window. For the way of interactive display described above, in which the combination of objects for display and the branch points for choices are predetermined, we call it off-line interactive display. As opposed to off-line, an on-line interactive display is the one, in which the combination of objects for display and the branch points for choices are dynamically determined. To support on-line interactive display, we will extend the sliding window approach to the dynamic sliding window approach. In this dynamic sliding window approach, the size of the sliding window can be changed according to the future requirements of data for display.

1 Introduction

Some media (such as audio and video) are classified as *continuous* because they consist of separate media *quanta* (such as audio samples or video frames), which convey meaning only when presented in time. Several multimedia types, video, in particular require high bandwidths and large storage space. For example, one hour and a half of video based on HDTV (High Definition Television) quality images has approximately 36 Gbits of data and requires approximately a 100 Mbps bandwidth while a current typical magnetic disk drive has only an 80 Gbit capacity and approximately a 20 Mbps bandwidth. In general, conventional file systems are unable to guarantee that clients can access continuous media in a way that permits delivery deadlines to be met under normal buffering conditions. Therefore, finding a way to support continuous retrieval of multimedia data at the required bandwidth and a way to store the multimedia data are challenging tasks [2, 5, 7, 13, 14, 21]. In this paper, for convenience, we use an object to denote an object of digital continuous media.

Previous approaches to supporting real-time applications of digital continuous media can be classified into three directions: *continuous retrieval*, *random access* and *interactive browsing*. To support *continuous retrieval*, some strategies clustered the data on a single disk to reduce the cost of disk head movement [6, 16, 17, 18, 22, 23], and some strategies tended to increase the bandwidth of disk device by using *parallelism*, which combines the bandwidths of multiple disks to provide a high data bandwidth requirement [1, 9, 12]. To support *random access*, like *editing operations*, since there is a trade-off between the flexible placement and the overhead of disk head movement in a disk drive, a straightforward idea is to find a compromise between them, which restricts a group of consecutive data to be stored consecutively in each cylinder on the disk while such a group of data can be randomly stored on any cylinder [11]. To support *interactive browsing* such as *fast forward* and *fast backward*, some strategies used the *scalable compression algorithms* to generate the multiresolution data [10], and some strategies supported browsing at any desired display speed by a predetermined *sampling* procedure [4].

Since future demands for high storage capacity and high bandwidth are expected, to efficiently support these three different directions for real-time applications, the *striping* strategy [1, 12] implemented on a multi-disk drive has been proposed. Basically, in the *striping* strategy [1, 12], the object is split up into subobjects and placed in various locations on the disks. Moreover, in the *simple striping* strategy [1], the striped subobjects are stored among the multi-disk drive in a predetermined sequence and must be read in this predetermined sequence to guarantee continuous retrieval. Furthermore, Berson et al. [1] further generalized the simple *striping* (called *staggered striping*) to support a database that consists of a combination of objects, each with a different bandwidth requirement.

Note that the *striping* strategy can only support simultaneous display of objects which are predetermined before they are stored in the multi-disk drive. For an interactive display application, a system must support users to make any choice even when display has started. Although in [3, 20], they have proposed the *replication* and *prefetching* strategies for interactive display, the combination of objects for display and the branch points for choices both have to be predetermined. Based on their strategies, they have to resolve all the possible *conflicts* before display starts, where a *conflict* means that a pair of two subobjects which are stored in the same disk must be retrieved simultaneously. Consequently, their strategies will require a lot of additional overhead of space and time. Moreover, display may be interrupted by the users at any time. When this current display is interrupted and is no longer needed, efforts for *prefetching* or *replication* are wasted because the following subobjects for display do not need to be retrieved. Therefore, to reduce the overhead, in this paper, we will propose a *sliding window* approach to supporting interactive display for continuous media, in which we only record a little necessary information of retrieval of the following subobjects for display in a *sliding window* and resolve the possible conflicts within the *sliding window*. From the simulation results, we[1] find that the larger the size of a *sliding window* is, the larger the waste of time and space once display is interrupted. Therefore, we prefer a *sliding window* with a *smaller* size. However, a *hiccup* can occur when the size of the *sliding window* is not large enough, where a *hiccup* means that the subobjects for being displayed has not been retrieved and will be ready in the next time interval. Therefore, we have to select a proper *sliding window* size for a predetermined interactive display. A mathematical analysis will be studied to speed up the selection of the value of a *sliding window* size.

Furthermore, for the way of interactive display described above, in which the combination of objects for display and the branch points for choices are predetermined, we call it *off-line* interactive display. As opposed to *off-line*, an *on-line* interactive display is the one, in which the combination of objects for display and the branch points for choices are dynamically determined. To support *on-line* interactive display, we will extend the *sliding window* approach to the *dynamic sliding window* approach. In this *dynamic sliding window* approach, the size of the *sliding window* can be changed according to the future requirements of data for display. Since the subobjects for future display are not predictable, we use some information of previous retrieval to guess the possible subobjects for future display. From the simulation results, we find that the probability of *hiccup* is decreased as the amount of information of previous retrieval is increased.

Basically, our proposed approach can be applied not only to the multi-disk drives, but also to the parallel database management systems, such as, parallel multimedia systems based on the *share-nothing* architecture [9]. A *share-nothing* architecture consists of a number of processors interconnected by a high speed communication network. Processors do not share disk drives or random access memory and can only communicate with one another by sending messages using an interconnection network. In this case, we assume that the bandwidth of both the network and the network device driver exceeds the bandwidth requirement of an object. The rest of the paper is organized as follows. Section 2 briefly describes the *striping* strategy that

[1] This research was supported in part by the National Science Council of Republic of China under Grant No. NSC-86-2213-E-110-004.

is applied in our approach. Section 3 presents the proposed *sliding-window* approach. Section 4 presents the simulation results of the proposed approach. In Section 5, we will present a mathematical analysis of the *sliding window* approach. In Section 6, we will extend the *sliding window* approach to support *on-line* interactive display. Section 7 contains a conclusion.

2 The Striping Strategy

In our approach, we apply the *striping* strategy [1] to arrange the objects on a multi-disk drive. Suppose the bandwidth of both the network and the network device driver exceeds the bandwidth requirement of an object. Assume that there are N disks which operate independently, called a *multi-disk* drive and each disk has a fixed bandwidth d, a worst seek time WS, and a worst latency time WL. The *simple striping* strategy uses the aggregate bandwidth of several disk drives by striping an object across multiple disks [1]. For example, an object X with bandwidth requirement C_X at least requires the aggregate bandwidth of $M_X = \lceil \frac{C_X}{d} \rceil$ disk drives to support continuous retrieval of X. (Note that the maximum aggregate bandwidth of a multi-disk drive with N disks is ($N \times d$) which must not be smaller than C_X.) Moreover, object X is organized as a sequence of equi-sized subobjects (X_0, X_1, X_2, ...), where the size of a subobject is s_X Mbits. Each subobject X_i represents a contiguous portion of X and is stored randomly in the disks. For the load balance for each disk, the subobjects of X are assigned to the N disks in a round-robin manner and the N disks are divided into R ($= \lfloor \frac{N}{M_X} \rfloor$) disk clusters, where each cluster is assigned to an object for retrieval of the M_X consecutive subobjects to guarantee the real-time transfer. The duration of retrieval of a subobject is fixed for all subobjects and is in terms of a *time interval I*. According to [16], concurrent pipelining of retrieval and display of an object requires prefetching and at least two buffers with size M_X subobjects. One buffer is for retrieval of the next M_X consecutive subobjects while the other one is to store the previous retrieved M_X consecutive subobjects which are currently displayed. The real-time retrieval (i.e., continuous retrieval) can be achieved by satisfying following equations:

$$\begin{aligned}
&M_X \geq \lceil \tfrac{C_X}{d} \rceil, \\
&M_X \leq N, \\
&R = \lfloor \tfrac{N}{M_X} \rfloor, \\
&I = \frac{s_X}{\frac{C_X}{M_X}}, \\
&WS + WL + \tfrac{s_X}{d} \leq \frac{s_X}{\frac{C_X}{M_X}}.
\end{aligned}$$

Hence, display of X employs only a single cluster at a time in a round-robin manner. In each cluster, consecutive subobjects of object X are stored on these M_X disks in a liner order. Figure 1 shows an example of simple striping for an object X with bandwidth requirement C_X = 80 Mbps, where N = 10, d = 20 Mbps, WS = 30 ms (ms = 10^{-3} seconds), WL = 10 ms and the value i denotes subobject X_i inside a disk. Suppose M_X = 5 ($\geq \frac{80}{20}$), then s_X ($= \frac{(WS+WL)\times d \times \frac{C_X}{M_X}}{d - \frac{C_X}{M_X}}$) = 3.2 Mbits (Mbits = 10^6 bits), I = 0.2 seconds and R = 2. Note that display of X first employs cluster 0 to read the subobjects X_0, X_1, X_2, X_3 and X_4 from disks 0, 1, 2, 3 and 4, respectively, into a buffer in the first time interval. In the second time interval, subobjects X_5, X_6, X_7, X_8 and X_9 are read into the other buffer from cluster 1. At the same time, subobjects X_0, X_1, X_2, X_3 and X_4 are displayed. Then, alternatively, the subsequent subobjects of X are read from cluster 0 or cluster 1 into two buffers and then are displayed.

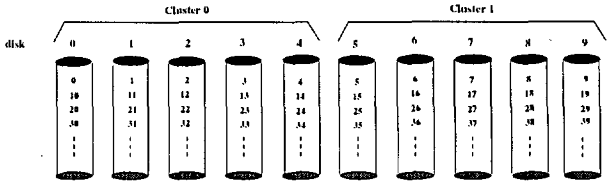


Figure 1: An example of object X striped on a multi-disk drive.

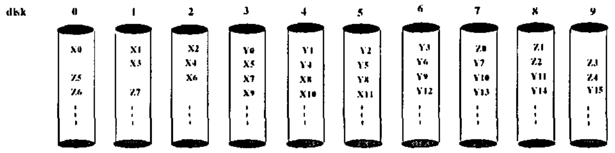


Figure 2: An example for *staggered striping* with a combination of 3 different objects.

Moreover, when the database consists of a combination of objects each with a different bandwidth requirement, the design of simple *striping* is extended to minimize the percentage of wasted disk bandwidth by constructing the disk clusters based on the media type that has the highest bandwidth requirement. The percentage of waste disk bandwidth can be large. Therefore, Berson et al. [1] proposed a generalization of simple *striping*, called *staggered striping*, which constructs the disk clusters *logically* (instead of physically). It assigns the subobjects such that the disk containing subobject X_{i+M_X} is g disks (modulo the total number of disks) apart from the disk drive that contains subobject X_i. The objects with different bandwidth requirements are assigned to disk drives independently but all with the same value of k. Figure 2 illustrates the assignment of objects X, Y and Z, where g = 1, M_X = 3, M_Y = 4, M_Z = 2 and N = 10.

3 The Sliding-Window Approach

In this section, we will describe the basic idea of the proposed *sliding-window* and the *retrieval* algorithm for the proposed approach.

3.1 Basic Idea

Suppose the desired subobjects of these 3 *striped* objects for display are changed to those shown in Figure 3. Obviously, two conflicts will occur. One conflict occurs between subobjects Z_3 and Y_{15} and the other one occurs between subobjects Z_5 and X_{30}. To resolve these conflicts, a straightforward method is to reorganize these 3 objects according to the *striping* strategy. However, it requires a lot of overhead. A better solution as the one in [3, 20] is to prefetch the conflicting subobjects. For example, subobjects Z_3 and Z_5 are prefetched to resolve the conflicts in Figure 3.

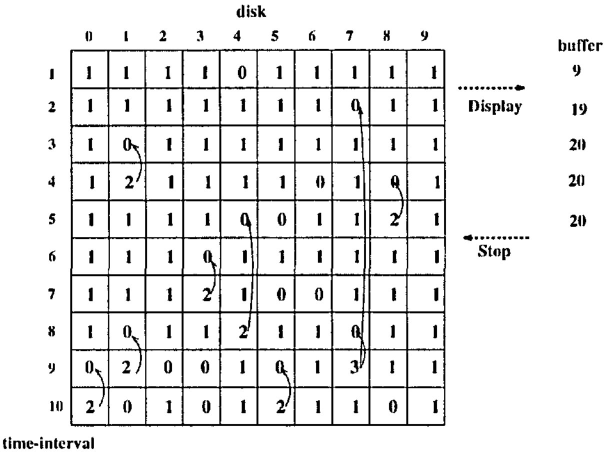
	0	1	2	3	4	5	6	7	8	9 disk
1	X0	X1	X2	Y0	Y1	Y2	Y3	Z0	Z1	
2		X3	X4	X5	Y4	Y5		Y7	Z2	Z3/Y15
3	Z5/X30			X7	X8	Y8	Y9	Y10	Y11	Z4
⋮										
time-interval										

Figure 3: An example of display.

Recall that based on the *striping* strategy, the duration (in terms of a *time interval I*) of retrieval of a subobject is fixed in each disk of the multi-disk drive. Logically, we can use a *time table of retrieval* to record the retrieved subobjects for each time interval. For example, the logical *time table of retrieval* (*TT*) of the display in Figure 3 is shown in Figure 4, where the value of each entry TT_{ij} denotes the number of subobjects that have to be retrieved from disk j in time interval i. For an entry TT_{ij} which value is greater than 1, a conflict will occur due to more than one subobjects that have to be retrieved in the same time interval i from disk j. To resolve the conflict, we have to prefetch (TT_{ij} - 1) subobjects of these TT_{ij} conflicting subobjects from disk j before time interval i. Logically, such *prefetching* operations can be viewed as a series of *replacement* operations, each of which is to find an entry TT_{kj} with value = 0 for such an entry TT_{ij} and then, TT_{kj} is set to 1 and TT_{ij} is decreased by one, where k is the maximum value such that $1 \leq k < i$. This *replacement* operation will be repeated until TT_{ij} is reduced to 1. Therefore, to guarantee continuous retrieval, we have to find (TT_{ij} - 1) entries with value = 0 for each such an entry TT_{ij} which value is greater than 1.

	disk 0	1	2	3	4	5	6	7	8	9
1	1	1	1	1	1	1	1	1	1	0
2	0	1	1	1	1	1	0	1	1	2
3	2	0	0	1	1	1	1	1	1	1
⋮						⋮				
time-interval										

Figure 4: The *time table of retrieval* of Figure 3.



	disk 0	1	2	3	4	5	6	7	8	9		buffer
1	1	1	1	1	0	1	1	1	1	1		9
2	1	1	1	1	1	1	1	0	1	1	Display	19
3	1	0	1	1	1	1	1	1	1	1		20
4	1	2	1	1	1	1	0	1	0	1		20
5	1	1	1	1	0	0	1	1	2	1		20
6	1	1	1	0	1	1	1	1	1	1	Stop	
7	1	1	1	2	1	0	0	1	1	1		
8	1	0	1	1	2	1	1	0	1	1		
9	0	2	0	0	1	0	1	3	1	1		
10	2	0	1	0	1	2	1	1	0	1		
time-interval												

Figure 5: An example in the *prefetching* approach.

Let us consider another example, where the *time table of retrieval* is shown in Figure 5. The total length of this display (*Len*) is 10 time intervals. For each TT_{ij} which value is greater than 1, we perform the *replacement* operation. For example, one of these 2 conflicting subobjects in entry TT_{41} has to be prefetched in entry TT_{31}. As shown in Figure 5, continuous display can be guaranteed after all the conflicts are resolved by a series of *replacement* operations. However, for an interactive display, users may stop this display at any time. In this example, suppose display is interrupted in time interval 5. That is, these retrieved subobjects in entry TT_{ij} for display after time interval 5 are no longer needed, where $6 \leq i \leq 10$ and $0 \leq j \leq 9$. In this case, there are one conflicting subobject of TT_{84} and one of TT_{97} which have been prefetched in time interval 5 and time interval 2, respectively. The disk bandwidths and buffers for retrieving these two subobjects are wasted. The corresponding sizes of the buffer (in terms of the number of subobjects) in each time interval for storing these retrieved subobjects are also shown in Figure 5.

To avoid the waste of time to prefetch and the waste of buffer space to store these unnecessary prefetched subobjects as the example shown in Figure 5, in this paper, we propose a *sliding window* approach. The basic concept of a *sliding window* in our proposed approach is to record a little necessary information of retrieval of the following subobjects for display in a *sliding window*. In this proposed approach, first, we use a window with size = SW (≥ 2) time intervals to record the first SW consecutive entries for each disk in the *time table of retrieval*, i.e, time intervals 1, 2, ..., SW. Second, we only perform the *replacement* operations within the *sliding window*. (Note that when $SW = 1$, no *replacement* operation can be done for any entry with value > 1. Therefore, the minimun size of SW is 2.) After the possible conflicts within the *sliding window* have been resolved by a series of *replacement* operations, these subobjects in time interval 1 are ready to be retrieved for display and the *window* is slid forward by including time interval (SW + 1) and excluding time interval 1. Third, we resolve the conflicts within the *sliding window* again and then, slide the window forward. At the same time, these subobjects in time interval 2 are ready for retrieval. These *replacement* and *sliding* operations will be repeated until display is finished or interrupted.

For illustrative purpose, let us consider a simple example shown in Figures 6, where the objects for display are the same as the ones in Figure 5 and SW is assumed to be 2. (Note that we use $SW(a,b)$ to denote that the current *sliding window* includes time intervals a and b.) As shown in Figure 6-(a), first, no *replacement* operation is needed since all the entries in the *sliding window* ≤ 1. Second, we slide the *window* forward as shown in Figure 6-(b). At the same time, these subobjects in time interval 1 are ready to be retrieved and no *replacement* operation is needed within the current *sliding window*. Third, we slide the *window* again and perform a *replacement* operation for entry TT_{41} by setting TT_{31} and TT_{41} to be 1 as shown in Figure 6-(c). At the same time, these subobjects in time interval 2 are being retrieved and these subobjects that have been retrieved in time interval 1 are being displayed. Forth, we set TT_{58} and TT_{48} to be 1 for $TT_{58} = 2$ after the *sliding window* is slid again. When the *sliding window* has been slid forward to include time intervals 6 and 7 as shown in Figure 6-(d), suppose display is interrupted. In this case, we set TT_{63} and TT_{73} to be 1 for $TT_{73} = 2$. Since the current time interval for retrieval is time interval 5, this *prefetching* subobject for TT_{73} in TT_{63} has not be retrieved yet. Moreover, the waste of time and space to prefetch the conflicting subobjects for TT_{84} and TT_{97} in Figure 5 can be avoided.

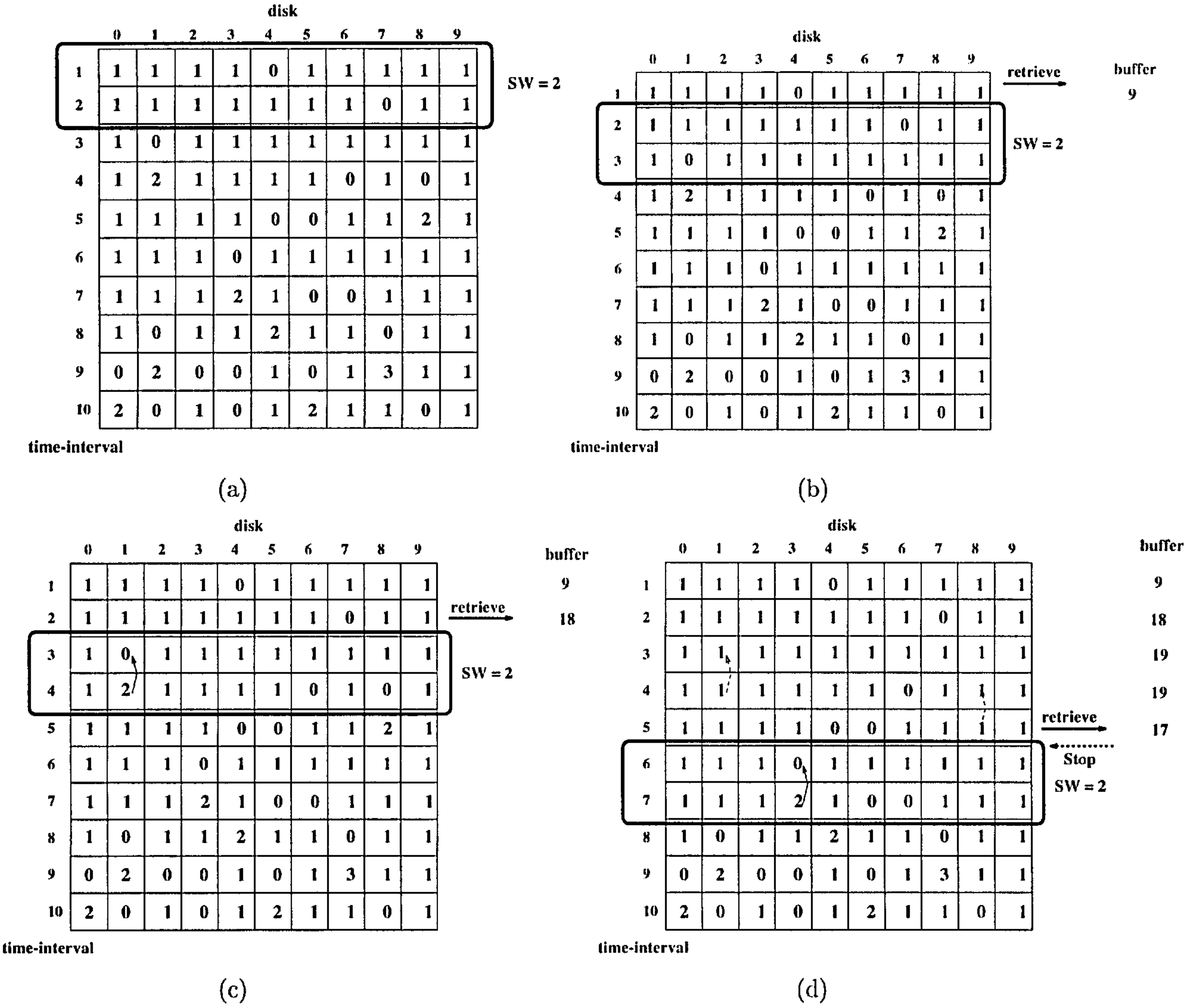
From the above example, we observe that continuous retrieval can also be guaranteed by using the *sliding window* approach with $SW = 2$. Obviously, the larger the window size SW is, the longer the waste of time and the higher the buffer space to prefetch the unnecessary subobjects once display is interrupted. However, a *hiccup* can occur in the proposed *sliding window* approach when the window size SW is not large enough, where a *hiccup* means that the subobjects for being displayed has not been retrieved and will be ready in the next time interval. Therefore, from the view of users, display will not be continuous when a *hiccup* occurs.

Such an example is shown in Figure 7, where $SW = 2$.

From Figure 7-(a), we observe that we can not find an entry $TT_{i1} = 0$ for TT_{41} to resolve the conflict within the *sliding window*. Note that even though $TT_{21} = 0$, we can not set TT_{21} ($= 0$) to be 1 to prefetch one conflicting subobjects for TT_{41} in time interval 2 because that these subobjects in time interval 2 are being retrieved. Therefore, TT_{41} is still 2 after the *replacement* operation. Then, a conflict will occur such that one of these 2 conflicting subobjects for retrieval in entry TT_{41} will be lost when these subobjects in time interval 4 are being retrieved. To resolve this problem, one proposed solution is called *time interval stealing*, which inserts a new time interval $3'$ with all $TT_{3'j} = 0$ ($0 \leq j < 10$) between time intervals 3 and 4 and slides the window forward as shown in Figure 7-(b). Now, we can set $TT_{3'1}$ to be 1 for TT_{41}. However, a *hiccup* will occur. Another better proposed solution is to select a large size of the *sliding window* $SW = 3$, instead of $SW = 2$, initially. As shown in Figure 7-(c), we can set TT_{21} to be 1 for TT_{41}. Moreover, by applying the *sliding window* approach with $SW = 3$, continuous display can be guaranteed. Therefore, to select a proper size of the *sliding window* for display is an important task and will be investigated in Sections 4 and 5.

3.2 The Algorithm

In this subsection, we present the *retrieval* algorithm based on the *sliding window* approach as shown in Figure 8. Suppose the execution time for the *replacement* process in each for-loop routine of the *retrieval* algorithm is negligible compared with the time interval for retrieval (e.g., 0.2 seconds in Figure 1). Moreover, the *retrieval* algorithm is *preemptive*. That is, the execution of the *retrieval* algorithm can be interrupted whenever display is interrupted. In the *retrieval* algorithm, after given the value of the *sliding window* size and the *time table* for display, we logically put the first SW time intervals of the *time table* into the *sliding window* and resolve the conflicts within the *sliding window*. Then, we slide the window forward by excluding time interval 1 and including time interval (SW + 1). Let *ptr* be 1, where *ptr* denotes the identification number of time interval, in which these subobjects are ready for retrieval. In the for-loop routine, such a *resolving-sliding* operation will be repeated until display is finished or interrupted.



(a)

time-interval	0	1	2	3	4	5	6	7	8	9
1	1	1	1	1	0	1	1	1	1	1
2	1	1	1	1	1	1	1	0	1	1
3	1	0	1	1	1	1	1	1	1	1
4	1	2	1	1	1	1	0	1	0	1
5	1	1	1	1	0	0	1	1	2	1
6	1	1	1	0	1	1	1	1	1	1
7	1	1	1	2	1	0	0	1	1	1
8	1	0	1	1	2	1	1	0	1	1
9	0	2	0	0	1	0	1	3	1	1
10	2	0	1	0	1	2	1	1	0	1

(b)

time-interval	0	1	2	3	4	5	6	7	8	9	buffer
1	1	1	1	1	0	1	1	1	1	1	9
2	1	1	1	1	1	1	1	0	1	1	
3	1	0	1	1	1	1	1	1	1	1	
4	1	2	1	1	1	1	0	1	0	1	
5	1	1	1	1	0	0	1	1	2	1	
6	1	1	1	0	1	1	1	1	1	1	
7	1	1	1	2	1	0	0	1	1	1	
8	1	0	1	1	2	1	1	0	1	1	
9	0	2	0	0	1	0	1	3	1	1	
10	2	0	1	0	1	2	1	1	0	1	

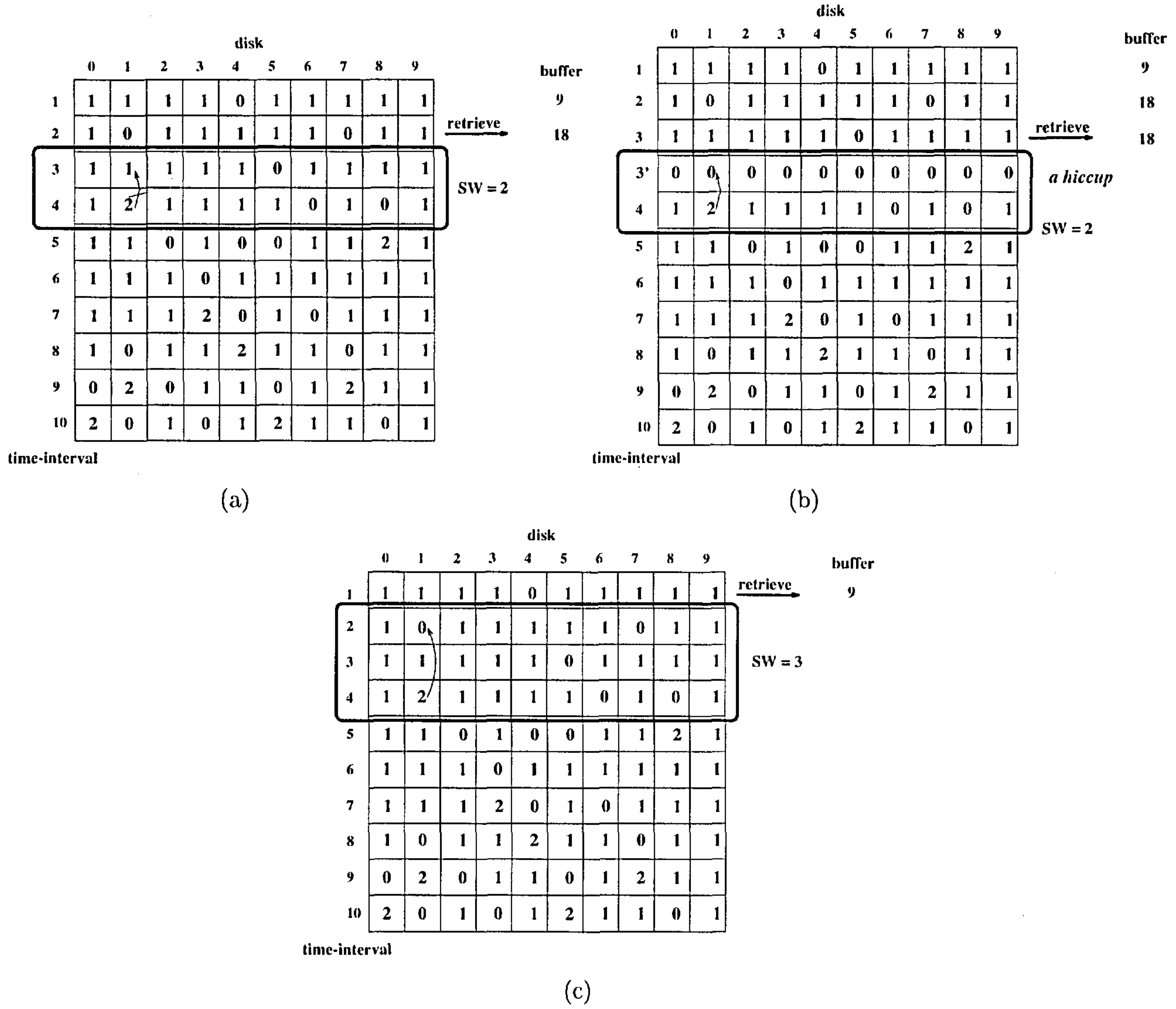
(c)

time-interval	0	1	2	3	4	5	6	7	8	9	buffer
1	1	1	1	1	0	1	1	1	1	1	9
2	1	1	1	1	1	1	1	0	1	1	18
3	1	0	1	1	1	1	1	1	1	1	
4	1	2	1	1	1	1	0	1	0	1	
5	1	1	1	1	0	0	1	1	2	1	
6	1	1	1	0	1	1	1	1	1	1	
7	1	1	1	2	1	0	0	1	1	1	
8	1	0	1	1	2	1	1	0	1	1	
9	0	2	0	0	1	0	1	3	1	1	
10	2	0	1	0	1	2	1	1	0	1	

(d)

time-interval	0	1	2	3	4	5	6	7	8	9	buffer
1	1	1	1	1	0	1	1	1	1	1	9
2	1	1	1	1	1	1	1	0	1	1	18
3	1	1	1	1	1	1	1	1	1	1	19
4	1	1	1	1	1	1	0	1	1	1	19
5	1	1	1	1	0	0	1	1	1	1	17
6	1	1	1	0	1	1	1	1	1	1	
7	1	1	1	2	1	0	0	1	1	1	
8	1	0	1	1	2	1	1	0	1	1	
9	0	2	0	0	1	0	1	3	1	1	
10	2	0	1	0	1	2	1	1	0	1	

Figure 6: An example in the *sliding window* approach with $SW = 2$: (a) $SW(1,2)$; (b) $SW(2,3)$; (c) $SW(3,4)$; (d) $SW(6,7)$.



time-interval \ disk	0	1	2	3	4	5	6	7	8	9	buffer
1	1	1	1	1	0	1	1	1	1	1	9
2	1	0	1	1	1	1	1	0	1	1	18
3	1	1	1	1	1	0	1	1	1	1	
4	1	2	1	1	1	1	0	1	0	1	
5	1	1	0	1	0	0	1	1	2	1	
6	1	1	1	0	1	1	1	1	1	1	
7	1	1	1	2	0	1	0	1	1	1	
8	1	0	1	1	2	1	1	0	1	1	
9	0	2	0	1	1	0	1	2	1	1	
10	2	0	1	0	1	2	1	1	0	1	

time-interval \ disk	0	1	2	3	4	5	6	7	8	9	buffer
1	1	1	1	1	0	1	1	1	1	1	9
2	1	0	1	1	1	1	1	0	1	1	18
3	1	1	1	1	1	0	1	1	1	1	18
3'	0	0	0	0	0	0	0	0	0	0	
4	1	2	1	1	1	1	0	1	0	1	
5	1	1	0	1	0	0	1	1	2	1	
6	1	1	1	0	1	1	1	1	1	1	
7	1	1	1	2	0	1	0	1	1	1	
8	1	0	1	1	2	1	1	0	1	1	
9	0	2	0	1	1	0	1	2	1	1	
10	2	0	1	0	1	2	1	1	0	1	

time-interval \ disk	0	1	2	3	4	5	6	7	8	9	buffer
1	1	1	1	1	0	1	1	1	1	1	9
2	1	0	1	1	1	1	1	0	1	1	
3	1	1	1	1	1	0	1	1	1	1	
4	1	2	1	1	1	1	0	1	0	1	
5	1	1	0	1	0	0	1	1	2	1	
6	1	1	1	0	1	1	1	1	1	1	
7	1	1	1	2	0	1	0	1	1	1	
8	1	0	1	1	2	1	1	0	1	1	
9	0	2	0	1	1	0	1	2	1	1	
10	2	0	1	0	1	2	1	1	0	1	

Figure 7: An example in the *sliding window* approach: (a) with $SW = 2$; (b) *time interval stealing*; (c) with $SW = 3$.

```
procedure retrieval (SW,TT[Len][N]);
var
          N : integer; /* the number of disks in a multi-disk drive*/
          Len : integer;
                    /* the length of display in terms of the number of time intervals */
          SW : integer; /* the size of a sliding window*/
          TT[Len][N] : integer; /* the time table of retrieval */
          Buf : a buffer; /* the buffer space for storing the retrieved subobjects */
          i,j,k,m,flag,ptr : integer;
begin
          resolve any conflict in the first SW time intervals by using the replacement approach;
          slide the window forward;
          ptr = 1;
          for (i=SW+1;i<=Len;i++) do
                    retrieve TT[ptr][*] for display;
                    ptr = ptr + 1;
                    for (j=0;j<N;j++) do
                         flag = 0;
                         while ((TT[i][j] > 1) and (flag == 0)) do
                            begin
                              for (k=i-1;k>i-SW;k- -) do
                                   if (TT[k][j] == 0)
                                      begin
                                        TT[k][j] = 1;
                                        TT[i][j] = TT[i][j] - 1;
                                        k = -1;
                                   end if;
                              end for;
                              if (k <> -1) flag = 1;
                         end while;
                    end for;

       /***** steal time intervals to resolve the unresolved conflicts *****/
                    if (flag == 1)
                       begin
                         m = max(TT[i][*] - 1);     /* function max returns the maximun value */
                         insert m new time intervals between time intervals (i - 1) and i
                         in the time table of retrieval;
                         Len = Len + m;
                         i = i + m - 1;
                    end if;
       /*****_______________________________________________*****/

                    slide the window forward;
          end for;
          for (i=ptr;i<=Len;i++) do retrieve TT[i][*] for display;
end;
```

Figure 8: Algorithm *retrieval*.

For each for-loop operation, first, these subobjects in time interval *ptr* are ready for retrieval, where time interval *ptr* is just removed from the *sliding window*. In other words, these entries TT_{ptrj} $(0 \leq j < N)$ in time interval *ptr* is no longer to be changed because that time interval *ptr* is not included in the *sliding window*. Second, for each $TT_{ij} > 1$ in time interval i, we find $(TT_{ij}$ - 1) entries with values = 0 within the *sliding window* and set them to be 1 to resolve the conflicts. However, there may not exist enough entries with values = 0 for each $TT_{ij} > 1$. In this case, we have to *steal* some time intervals. Third, after the conflicts in time interval i are resolved, we slide the window again.

4 Simulation Results

In this section, we will present simulation results for the proposed algorithm based on the *sliding window* approach. In this simulation model, we assume that each disk in a multi-disk drive operates independently. When an I/O request arrives, it may be decomposed into subrequests, each of which will be serviced independently on a different disk. Objects are stored on the multi-disk drive by applying the *striping* strategy and all the subobjects of each object have the same size. Moreover, the duration of retrieval of a subobject is fixed for all objects and is in terms of a *time interval I*. At any time interval i, the required bandwidth RB_i for display should not be larger than the aggregate bandwidth AB of the multi-disk drive. The required bandwidth RB_i can be varied according to the combination of objects for display. To describe the desired display, we propose a data model. In this data model, first, we use a *load_factor* to denote the average load of a multi-disk drive for a display with length *Len* (in terms of the number of time intervals) and let *load_factor* be $\frac{\sum_{i=1}^{Len} RB_i}{AB \times Len}$. Second, to describe the status of conflicts in a display, we use a series of probabilities P_k^n $(0 \leq k \leq n)$, each of which is to denote the probability of an entry with value = k when the desired display is combined with n objects. Consequently, $\sum_{k=0}^{n} P_k^n = 1$ and $\sum_{k=0}^{n} (k \times P_k^n) =$ *load_factor*. The performance measure is the average hiccup ratio *ave_HR*, which is the number of the number of *hiccups* *num_HR* divided by the length of display *Len*, that is, $ave\_HR = \frac{num\_HR}{Len}$. Another performance measure is the average size of buffer *ave_Buf* (in terms of the number of subobjects) that is used to store the retrieved subobjects during the duration of display.

Figures 9 show the relationship between the size of a *sliding window* (*SW*) for three different displays (D_1, D_2 and D_3) and the average hiccup ratio (*ave_HR*), where $N = 10$, $n = 3$ with $Len = 1000$ and *load_factor* is 0.7. The *time tables of retrieval* of three different displays are randomly generated, where (P_0^3, P_1^3, P_2^3, P_3^3) is (0.35, 0.6, 0.05, 0), (0.42, 0.47, 0.1, 0.01) and (0.44, 0.44, 0.1, 0.02), respectively. From this figure, we observe that *ave_HR* is decreased as the size of *sliding window SW* is increased. The reason is that the probability of a *hiccup* is decreased as *SW* is increased. Moreover, *ave_HR* is increased as the number of conflicts is increased (i.e., P_3^3 and P_2^3 is increased; while P_1^3 is decreased). To support continuous display with *hiccup*-free (i.e., $ave\_HR = 0$), the minimum sizes of the *sliding window* for these 3 display D_1, D_2 and D_3 are 42, 44 and 50, respectively. However, the users may choose a small *SW* with a tolerable average hiccup ratio to reduce to overhead once display is interrupted.

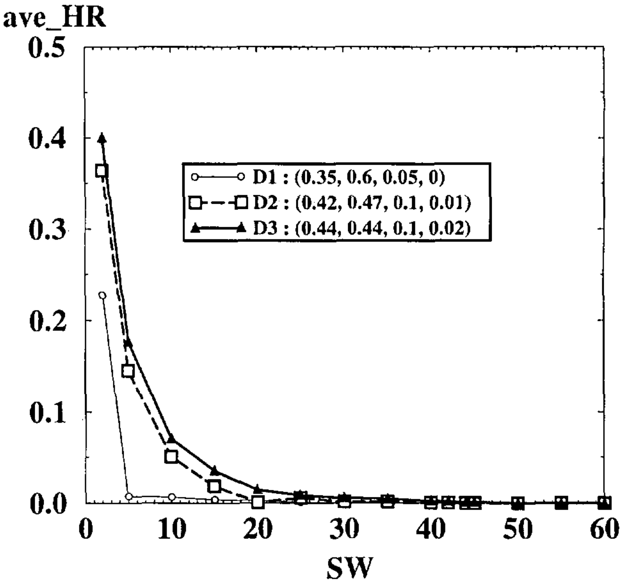


Figure 9: The relationship between the size of a *sliding window* (*SW*) and the average hiccup ratio (*ave_HR*).

Figure 10 shows the relationship between the size of a *sliding window* (*SW*) for three different displays (D_1, D_2 and D_3) and the average buffer size (*ave_Buf*) used in Figure 9. Obviously, the larger the value of *SW* is, the larger the value of *ave_Buf*. The reason is that it requires a larger buffer space to store these prefetched subobjects within a larger *sliding window* than a smaller one within a smaller *sliding window*. Even though *ave_Buf* is also increased as *load_factor* is increased, *ave_Buf* is still constant when *SW* is large enough such that *ave_HR* is reduced to 0.

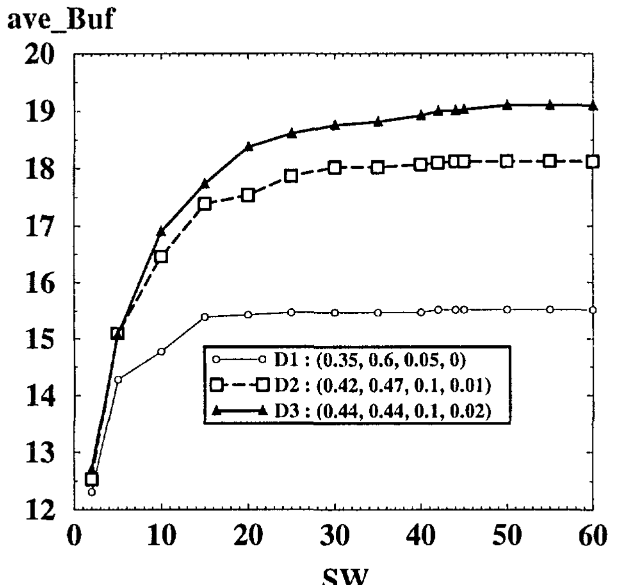


Figure 10: The relationship between the size of a *sliding window* (*SW*) and the average buffer size (*ave_Buf*).

5 Analysis of an Sliding Window Size

In the *sliding window* approach, we have to select a proper value of *SW* to support continuous display. However, such a selection will be made by repeating a series of imitations of the *retrieval* algorithm from an initial value of *SW* = 2 as described in Figures 9 and 10; it will waste much time when P_k^n ($1 < k \leq n$) is large. For example, it requires to perform 42 times imitations for D_1; while it requires to perform 50 times imitations for D_3 in Figure 9. Therefore, to speed up such a process, in this section, we will present the mathematical analysis of average hiccup ratio *ave_HR* with a given value of *SW* in the *sliding window* approach. Moreover, given a tolerable value of *ave_HR*, we can analyze the minimum value of *SW* for display of a combination of n objects. Consequently, instead of $SW = 2$, a good initial value of *SW* can be obtained from the mathematical analysis in order to speed up the selection of a proper value of *SW*.

Suppose there is a combination of n striped objects for display by using the *sliding window* approach with *SW* ($\geq$ 2). The values of *load_factor* and the series of P_k^n ($0 \leq k \leq n$) are given. Assume that the probability of k subobjects that will be retrieved from any disk j in any time interval i has the same value P_k^n, where $0 \leq k \leq n$. In other words, the probabilities of all TT_{ij} with value = k are P_k^n. The subobjects that will be retrieved in each time interval are independent. Therefore, there are (n + 1) cases of the value of an entry TT_{ij}. Since the conflict-resolution process (i.e., the *replacement* operation) is performed from the initial time interval to the last one in the *sliding window* approach, the subobjects in time interval e which are ready for retrieval implies that those in time interval m are also ready for retrieval, where $1 \leq m < e$. Consequently, the value of each entry in these time intervals that are ready for retrieval will be either 0 or 1. The probability of such an entry with value = 0 is (1 - *load_factor*); while the one of such an entry with value = 1 is *load_factor*. Therefore, for a time interval i that is in the conflict-resolution process, the probabilities for an entry TT_{ij} with values = 0 and 1 are P_0^n and P_1^n, respectively. In these two case, no conflict and hiccup can occur in entry TT_{ij}. When TT_{ij} = 2, a hiccup can occur if there does not exist any one entry $TT_{mj} = 0$, where $(i - SW + 1) \leq m \leq (i - 1)$. The probability of such a case is $\binom{SW-1}{SW-1} f^{SW-1}$. Otherwise, no hiccup can occur. To simplify the notations, in the following formulas, we use f to denote *load_factor*. The number of hiccup with TT_{ij} = 2 is obtained as

$$UH_2^n = P_2^n \times (1 \times \binom{SW-1}{SW-1} f^{SW-1}).$$

Similarly, the number of hiccup with TT_{ij} = 3 can be obtained as

$$UH_3^n = P_3^n \times (2 \times \binom{SW-1}{SW-1} f^{SW-1} + 1 \times \binom{SW-1}{SW-2} f^{SW-2}(1\text{-}f)).$$

In general, the number of hiccup with $TT_{ij} = k$ can be obtained as

$$UH_k^n = P_k^n \times ((k - 1) \times \binom{SW-1}{SW-1} f^{SW-1} + (k - 2) \times \binom{SW-1}{SW-2} f^{SW-2}(1\text{-}f) + \ldots + 1 \times \binom{SW-1}{SW-k+1} f^{SW-k+1}(1\text{-}f)^{k-2}).$$

Therefore, the average number of hiccup can be obtained as

$$ave\_HR = \textstyle\sum_{k=2}^{n} UH_k^n.$$

6 The Dynamic Sliding Window Approach

In the *sliding window* approach, the combination of objects for display and the branch points for choices are predetermined. That is, the *time table of retrieval* has to be predetermined. To support *on-line* interactive display, in this section, we will extend the *sliding window* approach to the *dynamic sliding window* approach, which can support *on-line* interactive display for any combination of objects by applying a *dynamic window size*. The size of the

```
procedure retrieval* (PI,TT[Len][N]);
var
    N,Len,PI,SW,CSW : integer;
    TT[Len][N] : integer; /* the time table of retrieval */
    WT[PI][N] : integer;
    Buf : a buffer; /* the buffer space for storing the retrieved subobjects */
    i,j,k,m,r,flag,ptr,I_flag,P_flag : integer;
begin
    SW = 2;        WT[*][*] = 0;
    resolve any conflict in the first SW time intervals;
    slide the window forward;
    for (r=1;r≤SW;r++) do WT[ptr-SW+r][*] = TT[r][*];
    ptr = 1;
    for (i=SW+1;i<=Len;i++) do
        for ( r=1;r<PI;r++) do WT[r][*] = WT[r+1][*];
        WT[PI][*] = TT[i][*];
        if (I_flag == 0)
          begin
            retrieve TT[ptr][*] for display;
            ptr = ptr + 1;
        end if;
        for (j=0;j<N;j++) do
            flag = 0;
            while ((TT[i][j] > 1) and (flag == 0)) do
              begin
                for (k=i-1;k>i-SW;k- -) do
                    if (TT[k][j] == 0)
                      begin
                        TT[k][j] = 1;
                        TT[i][j] = TT[i][j] - 1;
                        k = -1;
                    end if;
                end for;
                if (k <> -1) flag = 1;
            end while;
        end for;
        if (flag == 1)
          begin
            m = max(TT[i][*] - 1);
            insert m new time intervals between time intervals (i - 1)
            and i on the time table of retrieval;
            Len = Len + m;
            i = i + m - 1;
        end if;
        P_flag=0;
        for (r=2;r<=PI;r++) do
            imitate retrieval(r,WT[PI][N]);
/* perform the retrieval algorithm without the retrieval operations;
   we only want to get the number of hiccups after
   the retrieval algorithm is applied on WT */
            if (the number of hiccups in the imitation is 0)
              begin
                CSW = r;
                P_flag = 1;
            end if;
        end for;
        if (P_flag == 0) CSW = PI;
        if (CSW<SW)
              begin
            for (j=CSW;j<SW;j++) do
                retrieve TT[ptr][*] for display;
                ptr = ptr + 1;
            end for;
            SW = CSW;
        end if;
        if (CSW>SW) { I_flag = 1; SW = SW + 1;}
        if (CSW=SW) I_flag = 0;
        slide the window forward;
    end for;
    for (i=ptr;i<=Len;i++) do retrieve TT[i][*] for display;
end;
```

Figure 11: Algorithm *retrieval**.

sliding window is changed according to the future requirements of data for display. The basic concept is that display can be interrupted or may eventually be changed to another display by the users at any time. That is, the contents of the *time table of retrieval* can be dynamically changed. Therefore, we have to dynamically change the size of the *sliding window* according to the current status of subobjects for display in order to still support continuous retrieval with the a little overhead. Since the subobjects for future display are not predictable, in the *dynamic sliding window* approach, we use the PI (≥ 1) pervious time intervals of retrieval to guess the subobjects for future display. Then, the size of the *sliding window* SW for the next time interval i is chosen to be the minimum value of SW for these PI time intervals with $num\_HR = 0$ by applying the *sliding window approach*.

The retrieval algorithm based on the *dynamic sliding window* approach is called the *retrieval** algorithm as shown in Figure 11. This *retrieval** algorithm is also preemptive. The differences between the *retrieval** algorithm and the *retrieval* algorithm are printed in bold font as shown in Figure 11. In the *retrieval** algorithm, after given the value of PI and an initial value of SW, we logically put the first SW time intervals of the *time table* into the *sliding window* and resolve the conflicts within the *sliding window*. (Note that as opposed to the predetermined *time table* in the *retrieval* algorithm, the one in the *retrieval** algorithm is dynamically determined.) Then, we slide the window forward. In the for-loop routine, such a *resolving-sliding* operation will be repeated until retrieval for display is finished or interrupted. For each for-loop operation, first, these subobjects in time interval *ptr* are ready for retrieval as the case in the *retrieval* algorithm. Second, we put the information of retrieval of previous PI time intervals into the working table WT. Then, for each TT_{ij} > 1 in time interval i, we find (TT_{ij} - 1) entries with values = 0 within the *sliding window* and set them to be 1 to resolve the conflicts. However, there may not exist enough entries with values = 0 for each $TT_{ij} >$ 1. In this case, we have to *steal* some time intervals. Third, after the conflicts in time interval i are resolved, we have to predict a *sliding window* size CSW for the next time interval by imitating the *retrieval* algorithm with WT. In this imitation, we find the minimum value of *sliding window* size (CSW) for WT with *hiccup*-free. When the new *sliding window* size CSW is smaller than SW, we have to retrieve the subobjects in these time intervals *ptr*, (*ptr* + 1), ..., (*ptr* + SW - CSW - 1). The reason is that the size of *sliding window* will be reduced and these previous time intervals that will be removed from the *sliding window* are no longer to be changed. Therefore, the subobjects in these time intervals are ready for retrieval. On the other hand, when $CSW > SW$, we have to enlarge

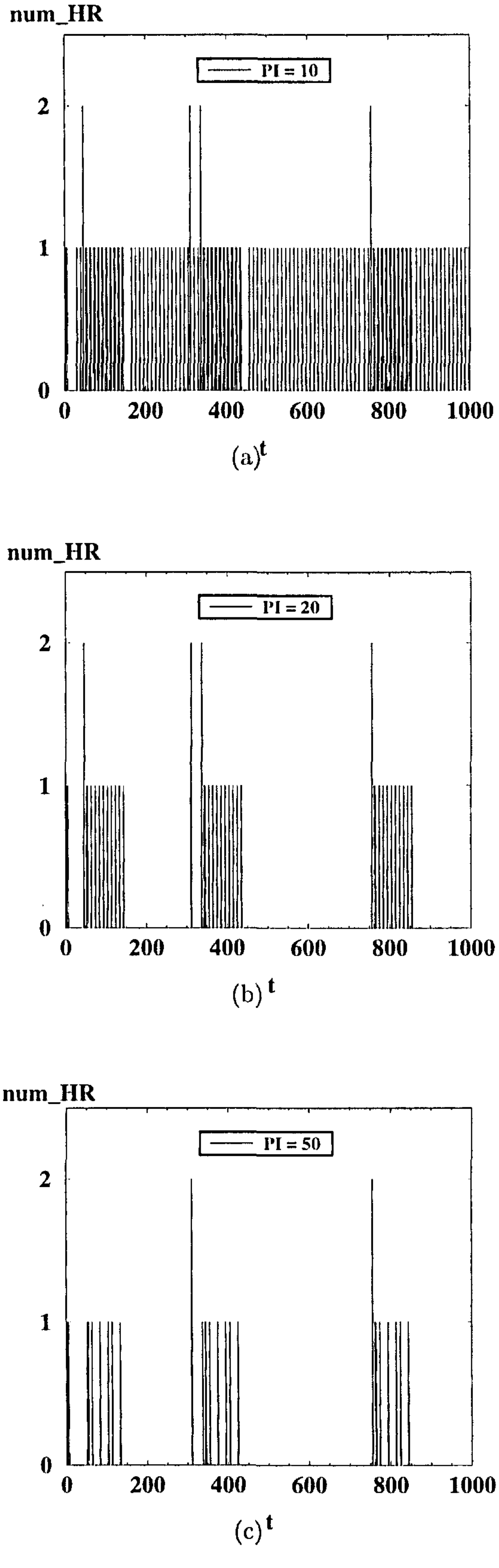


Figure 12: The relationship between the time interval t and the number of hiccups num_HR: (a) PI = 10; (b) PI = 20; (c) PI = 50.

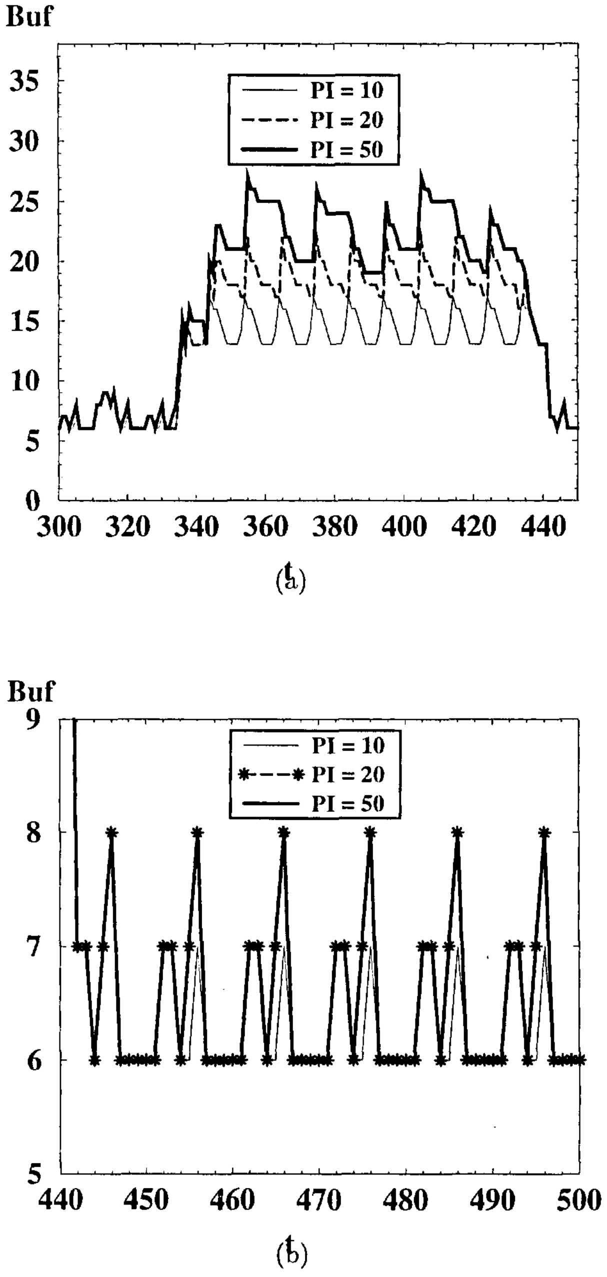
the *sliding window*. In this case, time intervals (*ptr* - 1), (*ptr* - 2), ..., (*ptr* - (*CSW* - *SW* - 1)) have to be included in the *sliding window*. However, since these subobjects in these time intervals before time interval *ptr* had been retrieved, we can not change the values of these entries in these time intervals to resolve any conflict. Therefore, in this case, the size of the *sliding window* is only increased by one, in which time interval *ptr* is not removed from the *sliding window* and time interval (i + 1) is included in the *sliding window*. That is, the *sliding window* size for the next time interval is (SW + 1). *I_flag* is set to 1 to denote that such a case occurs and to prohibit the retrieval operation of time interval *ptr*. Finally, we slide the window again.

Figures 12-(a), 12-(b) and 12-(c) show the relationship between the time interval t and the number of hiccups *num_HR*, where $N = 10$, $n = 3$ with $Len = 1000$ and PI is 10, 20 and 50, respectively. The *load_factor* and the series of P_k^3 $(0 \le k \le 3)$ are varied in each time interval. To simulate the unpredetermined subobjects for *on-line* interactive display, we use a random function of t to generate the values of the *load_factor* and the series of P_k^3 $(0 \le k \le 3)$ for each time interval. From these figures, we observe that *num_HR* is decreased as PI is increased due to that the more the information of retrieval are considered, the better the value of SW.

Figures 13-(a) and 13-(b) show the relationship between the time interval t and the size of buffer *Buf*, where the related parameters are the same as those in Figure 12. Since there are similar results in the other ranges of t, Figures 13-(a) and 13-(b) only show the range of t from 300 to 440 and from 440 to 500, respectively. From Figures 12 and 13, we observe that the size of buffer *Buf* is increased as *num_HR* is increased in all these 3 cases. The larger the value of PI is, the larger the size of buffer. Moreover, the corresponding value of SW for each time interval t in Figure 12 is also shown in Figures 14. Compared to Figure 12, we observe that the *sliding window* is changed according to the retrieval information of the previous PI time intervals. The larger the value of PI is, the better the value of SW. However, the overhead to decide a new value of SW is increased as the value of PI is increased. Therefore, based on the *dynamic sliding window* approach, users can choose a proper PI with a tolerable average hiccup ratio and overhead.

7 Conclusion

In this paper, we have proposed an efficient approach, called the *sliding window* approach, which can support interactive display for continuous media with a little overhead of *prefetching*. From the simulation results, we have observed that the smaller the size of a *sliding window* is, the smaller the waste of time and



(a)

(b)

Figure 13: The relationship between the time interval t and the size of buffer *Buf*: (a) t = (300, 440); (b) t = (440, 500).

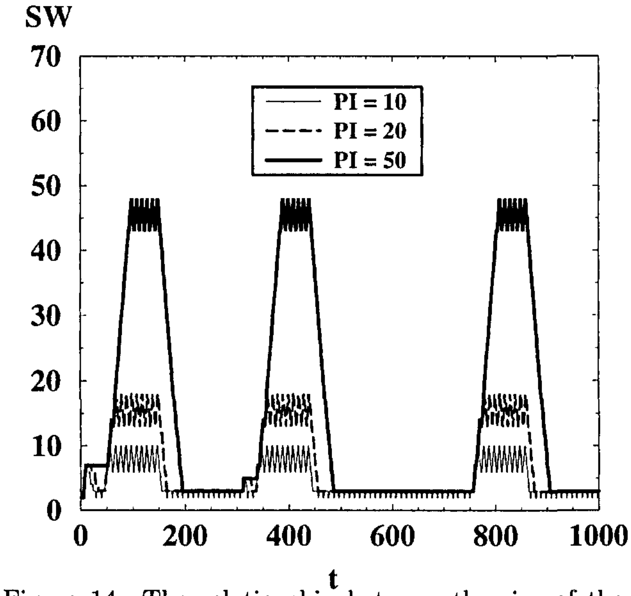


Figure 14: The relationship between the size of the time interval *t* and the *sliding window SW*.

space once display is interrupted. However, a *hiccup* can occur when the size of the *sliding window* is not large enough. Moreover, we have presented a mathematical analysis of the *sliding window* approach to speed up the selection of a *sliding window* size. To support *on-line* interactive display, we have extended the *sliding window* approach to the *dynamic sliding window* approach. From the simulation results, we have observed that the probability of a *hiccup* is decreased as the amount of information of previous retrieval is increased. How to support *on-line* interactive display for continuous media at any desired display speed rate is a future research direction.

References

[1] Berson, Steven, Ghandeharizadeh, Shahram, Muntz, Richard and Ju, Xiangyu, "Staggered Striping in Multimedia Information Systems," *ACM SIGMOD*, pp. 79-90, 1994.

[2] Buford, John F. K., "Multimedia File Systems and Information Models," *Multimedia Systems*, Buford, John F. K., ed., Addison-Wesley, 1994.

[3] Chaudhuri, Surajit, Ghandeharizadeh, Shahram, and Shahabi, Cyrus, "Avoiding Retrieval Contention for Composite Multimedia Objects," *Proc. of the 21st VLDB Conference*, pp. 287-298, 1995.

[4] Chen, Ming-Syan, Kandlur, Dilip D. and Yu, Philip S., "Storage and Retrieval Methods to Support Fully Interactive Playout in a Disk-Array-Based Video Server," *ACM Multimedia Systems*, Vol. 3, pp. 126-135, 1995.

[5] Christodoulakis, S. and Koveos, L., "Multimedia Information Systems: Issues and Approaches," in *Modern Database Systems: the Object Model, Interoperability and Beyond*, Kim, W., Editor, Addison-Wesley, 1994.

[6] Gemmell, Jim and Christodoulakis, Stavros, "Principles of Delay-Sensitive Multimedia Data Storage and Retrieval," *ACM Transactions on Information Systems*, Vol. 10, No. 1, pp 51-90, Jan. 1992.

[7] Gemmell, D. James, Vin, Harrick M., Kandlur, D. D., Rangan, P. Venkat and Rowe, L. A., "Multimedia Storage Servers: A Tutorial," *IEEE Computer*, pp. 40-49, May 1995.

[8] Ghandeharizadeh, Shahram and Dewitt, D., "A Multiuser Performance Analysis of Alternative Declustering Strategies," *Proc. of IEEE International Conference on Data Engineering*, pp. 466-475, 1990.

[9] Ghandeharizadeh, Shahram and Ramos, Luis, "Continuous Retrieval of Multimedia Data Using Parallelism," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 5, No. 4, pp. 658-669, August 1993.

[10] Keeton, Kimberly, and Katz, Randy H., "Evaluating Video Layout Strategies for a High-Performance Storage Server," *ACM Multimedia Systems*, Vol. 3, pp. 43-52, 1995.

[11] Liu, Jonathan C. L., Du, David H. C. and Schnepf, James A., "Supporting Random Access on Real-Time Retrieval of Digital Continuous Media," *Computer Communications*, Vol. 18, No. 3, pp. 145-159, March 1995.

[12] Lougher, P. and Shepherd, D., "The Design of a Storage Server for Continuous Media," *The Computer Journal*, Vol. 36, No. 1, pp. 33-42, 1993.

[13] Mourad, Antoine N., "Issues in the Design of a Storage Server for Video-On-Demand," *ACM Multimedia Systems*, Vol. 4, pp. 70-86, 1996.

[14] Ozden, Banu, Rastogi, Rajeev, and Silberschatz, Avi, "On the Design of a Low-Cost Video-On-Demand Storage System," *ACM Multimedia Systems*, Vol. 4, pp. 40-54, 1996.

[15] Patterson, D., Gibson, G. and Katz, R., "A Case for Redundant Arrays of Inexpensive Disks (RAID)," *ACM SIGMOD*, pp. 109-116, 1988.

[16] Rangan, P. Venkat, and Vin, Harrick M., "Designing File Systems for Digital Video and Audio," *Proc. 13th ACM Symposium on Operating System Principles*, pp. 81-94, 1991.

[17] Rangan, P. Venkat, Vin, Harrick M. and Ramanathan, Srinivas, "Designing an On-Demand Multimedia Service," *IEEE Communications Magazine*, pp. 56-64, July 1992.

[18] Rangan, P. Venkat and Vin, Harrick M., "Efficient Storage Techniques for Digital Continuous Multimedia," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 5, No. 4, pp. 564-573, August 1993.

[19] Salem, K. and Carcia-Molina, H., "Disk Striping," *Proc. of IEEE International Conference on Data Engineering*, pp. 336-342, 1986.

[20] Shahabi, Cyrus, and Ghandeharizadeh, Shahram, "Continuous Display of Presentations Sharing Clips," *ACM Multimedia Systems*, Vol. 3, pp. 76-90, 1995.

[21] Steinmetz, R., "Multimedia File Systems Survey: Approaches for Continuous Media Disk Scheduling," *Computer Communications*, Vol. 18, No. 3, pp. 133-144, March 1995.

[22] Vin, Harrick M. and Rangan, P. Venkat, "Designing a Multiuser HDTV Storage Server," *IEEE Journal on Selected Areas in Communications*, Vol. 11, No. 1, pp. 153-164, Jan. 1993.

[23] Yu, Clement, Sun, Wei, Bitton, Dina, Yang Qi, Bruno, Richard and Tullis, John, "Efficient Placement of Audio Data on Optimal Disks for Real-Time Applications," *Communications of ACM*, Vol. 32, No. 7, pp. 862-871, July 1989.

Parallel Database Architectures: A Comparison Study

Emad Mohamed, Computer Science Department, Old Dominion University
E-mail: mohamed@cs.odu.edu
AND
Hesham El-Rewini, Computer Science Department, University of Nebraska at Omaha
E-mail: rewini@csalpha.unomaha.edu
AND
Hussein Abdel-Wahab, Computer Science Department, Old Dominion University
E-mail: wahab@cs.odu.edu
AND
Abdelsalam Helal, MCC, Austin, Texas 78759
E-mail: helal@mcc.com

Keywords: parallel architectures

Edited by: Bogdan Czejdo
Received: October 30, 1996 **Revised:** April 21, 1997 **Accepted:** May 7, 1997

Parallel database systems are gaining popularity as a solution that provides high performance and scalability in large and growing databases. A parallel database system exploits multiprocessing to improve performance. Parallel database architectures can be broadly classified into three categories: shared memory, shared disk, and shared nothing. An important question, however, is which architecture should be used for a specific database application. Each architecture has its strengths and weaknesses. In this paper, simulation models for the three main architectures are presented. Using these models, a number of experiments were conducted to compare the system performance of these architectures under different workloads and transaction models. The goal of this work is to aid the decision making process of which architecture is better satisfying the requirements of a given database application.

1 Introduction

Indisputably, large and complicated databases are no longer unusual applications in our daily life, with the expectation of further growth of such databases. A direct result of having large and complicated databases is the need of efficient systems to handle the new requirements of the applications involving these databases. A recent trend in the computer industry, and in the database technology, is the use of parallel computers instead of large expensive mainframes. There are two reasons behind the movement towards parallelism. The first is related to the price of the system as a parallel computer, using commodity hardware devices such as cheap microprocessors, can give the same performance of a large mainframe but at a fraction of the cost [12]. Functionality is the second motivation behind parallelism. The technology of single processor computers is bounded by the speed of light, suggesting the use of several processors that work together in harmony to perform a large task [12]. Specifically for database systems, experiences with input/output (I/O) bottleneck found in large mainframes speed up the steps towards parallel systems. Scalability is another important issue in motivating parallel computers. Some database systems require more performance or get larger than what they were originally designed for. Parallel computers easily adapt for this by adding more components to the original system. Parallelism is taking place in several academic as well as commercial products. Examples of academic systems include Gamma and Bubba. Tandem NONStop SQL, Teradata, IBM's DB2, and Oracle parallel server are few examples of commercial products.

An important question, however, that is faced when using parallel computers to develop database systems is what architecture should be used. Architectures for parallel databases can be categorized into: shared memory, shared disk, and shared nothing computers [2, 4]. Several qualitative studies were introduced and recommended one architecture over the other [17]. There is a need for more concrete studies that quantitatively compare these architectures. This can be reached using analytical models, empirical analysis, or simulation. Among these methodologies, simulation has the advantage of a great deal of flexibility and provides more insights about the way the simulated systems may behave. However, an important point using simulators is that care must be taken in modeling the relevant aspects of the simulated object, otherwise incorrect and misleading results may arise.

In this paper, simulation models for the main parallel database systems are developed. Based on these models, a number of experiments were conducted to investigate the performance of the three main architectures under different workload and transaction models. It turns out that each architecture has proven some strengths for some specific workload and transaction model. This indicates that having a prior expectation of the transaction type and the amount of workload in a database application can help in choosing the most suitable architecture for the application.

The rest of this paper is organized as follows. Section 2 is an introduction to parallel database systems. An overview of the previous work in this field is given in Section 3. Section 4 is devoted to introducing the simulators for the various parallel database architectures. The experiments performed and results obtained are discussed in Section 5. Finally, Section 6 concludes this paper.

2 Parallel Database Systems: Alternative Architectures

Several parallel architectures have been developed. Shared memory, shared disk, and shared nothing are the main architectures of parallel database systems. Each architecture has some advantages and disadvantages. Recently, hybrid architectures are being investigated to combine the virtues of the three architectures and overcome their individual shortcomings. To compare parallel database systems, we need to define the criteria upon we base the comparison. In the following, we discuss the performance indexes along with the various architectures of parallel database systems.

2.1 Parallel Database Performance Indexes

The performance of a database system can be evaluated by several measurements. Response time and throughput are the most popular measures. This is because they are easy to measure and they directly affect the users of the database system. Response time can be loosely defined as the time required to perform an operation submitted by a user of the system. Throughput is how mush work can be performed in a unit time. Other performance measurements include system availability, and extensibility. System availability is the probability that the system is available during a time interval despite failure in some of the system components [4]. System extensibility is the ability of smooth growth of the system by adding more components to it to adapt for growth in the database size and/or the functionality required. Mainly two metrics can be used to measure the extensibility of a system: scaleup and speedup. Scaleup can be defined as the ability to grow the problem size by adding more components to the system, while maintaining the same performance. Speedup is the performance gain due to increasing the power of the system, while fixing the problem size [4].

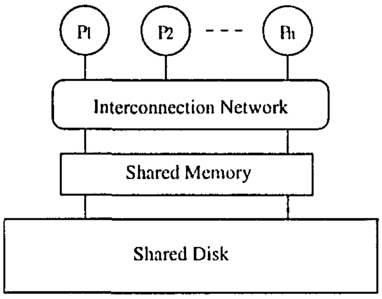


Figure 1: Shared Memory

Using parallelism within the database can increase system performance significantly. Many parallel database systems provide a near linear speedup and scaleup. Speedup can be gained by incorporating intra-query parallelism. This parallelism can be reached by partitioning the data and using the existing sequential routines, or by writing new special parallel routines for the queries. Complicated queries can be divided into subtasks that can work in parallel. Scaleup, on the other hand, can be gained by using inter-query parallelism in which several queries can be performed independently increasing the overall system throughput.

2.2 Shared Memory Architecture

In shared memory architecture, also named shared everything, the system consists of a number of processors all connected to one shared memory and one logical shared disk (Figure 1). Communication between processors comes for free by using the shared memory. Systems following this architecture are characterized by excellent load balance. The main disadvantage of these systems is the limited scalability and availability. Limited scalability is imposed by the fact that all of the processors compete to use the shared memory. This puts an upper limit on the number of processors. Availability may also be limited, for instance if the shared memory is failed the entire system is down.

2.3 Shared Disk Architecture

Shared disk architectures provide each processor by its own private memory, with one global (logical) shared disk (Figure 2). Processors no longer compete for a shared memory. They, however, compete to access the shared disk. This leads to a better processor scaleup, which is in the range of the hundreds. These systems

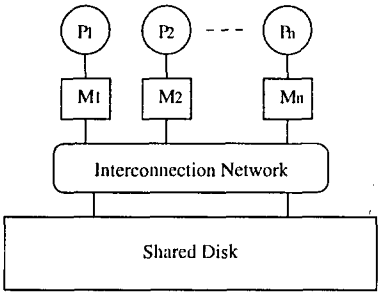


Figure 2: Shared Disk

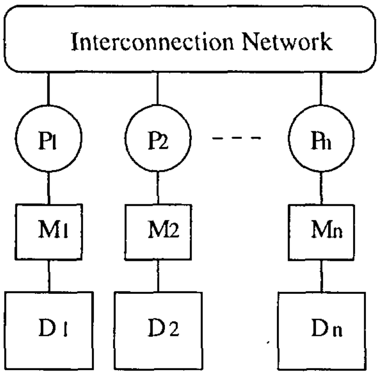


Figure 3: Shared Nothing

still have a good load balance. Availability is much better than shared memory, with the fact that failure of the shared disk means a failure to the entire system though. In practical systems, the shared disk is physically several disk modules connected through a network to all the processor elements. This provides a high reliability to the disk as a whole.

A shared nothing system can be thought of as a number of autonomous computers, each has its own private memory and disk, and are connected by an interconnection network (Figure 3). The main advantage of this architecture is the ability to scaleup to thousands of processors. Also it has high availability and reliability. In fact, the failure of one node should not affect the rest of the nodes. Availability can be reached by replicating data in the different nodes within the system. Load balance and skew (variance well exceeds the mean due to inappropriate data partitioning scheme) are the major problems faced in this architecture. Also, adding new nodes may require reorganizing the data within the entire system.

2.4 Hybrid Architecture

2.5 Shared Nothing Architecture

As it is apparent from the previous discussion, each of the various architectures has some advantages and some disadvantages. While shared memory and shared disk architectures have good load balance, they have limited scaleup and limited availability. Shared nothing has the advantages of high scalability and high availability but suffers from load balance problem. A hybrid architecture is to have a shared nothing system with nodes that are more powerful than a single computer. The nodes within the shared nothing system can be a shared memory or shared disk systems. This, while preserving the scalability and availability, adds the load balance that exists in the shared memory and shared disk systems. Figure 4 illustrates this idea.

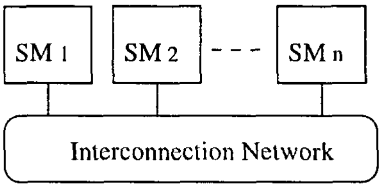


Figure 4: Hybrid (Shared Nothing with Nodes as Shared Memory)

3 Related Work

While there exit several commercial parallel database systems (such as Tandem NonStop SQL and Teradata system) and research ones (Gamma and Bubba for example), yet it is not clear which is the best architecture. Many researchers have investigated the parallel database architectures. Many of the researchers in this area investigated and implemented one specific architecture as in the case of Bubba and Gamma.

Bubba is a shared nothing parallel database [3]. The machine consists of several nodes connected through a message-passing interconnect. Parallelism is achieved mainly through data partitioning, where data are horizontally clustered across systems nodes using either hashing or range partitioning. Bubba incorporates function shipping instead of data shipping. In this strategy, an operation is sent to the node where the data reside, reducing the overhead of sending large amount of data between nodes.

Gamma is another example of shared nothing relational parallel database machine that operates on Intel iPSC/2 hypercube with 32 processors and 32 disk drives [5]. As in Bubba, parallelism is achieved through data partitioning. Data is partitioned horizontally among the system disk drivers using a round robin, hashing, or range strategy.

An instance of a commercial shared nothing parallel database machine is the NCR/Teradata DBC/1012 [14]. The machine is a dedicated relational database system. It consists of several nodes connected through a special interconnect named Y-net. The machine basically offloads the database work from a host computer by accepting database requests from the host computer, performing the database operation required, and returning the result back to the host computer. The machine can perform both online transactions and decision support systems (DSS) operations. Online transactions can work with one DSS on the background. Parallelism is achieved through data partitioning.

Other researchers gave a qualitative comparisons between the various architectures. In their paper, DeWitt and Cray gave an introduction to parallel database systems and described the three parallel architectures [4]. They described three data partitioning techniques (round-robin, hash, and range partitioning). A discussion of intra- and inter-query parallelism was introduced. They provided an overview of some industrial and academic parallel database systems including Tandem Non Stop SQL, Teradata, Bubba, and Gamma. Valduriez and Rodin in their papers classified the parallel database systems, again, into three architectures [16, 17]. They discussed the advantages and shortcomings for each architecture. Finally, they argued for a shared something architecture: a hybrid between shared nothing and shared disk systems.

A third group investigated specific aspects of one of the architectures. Helal et al. examined the performance of parallel database systems in shared nothing architecture [9]. They considered two different techniques for concurrency control (dynamic locking with general waiting and dynamic locking with no waiting). In their work, to solve the skew problem, they suggested a dynamic data reallocation technique to redistribute the most accessed data from a loaded node to a less loaded one, without blocking the entire system.

This work takes a global overview of the various parallel database systems. The study is a quantitative in contrast to the qualitative research introduced before. Simulation models are developed to provide a concrete and accurate comparisons among the various architectures taking into consideration the different aspects that impact the database system performance. Several experiments were conducted to measure the system performance under different workloads and transaction models. For a more comprehensive study and experimentation work refer to [13].

4 Simulation Models

This section introduces the simulators we developed to compare the different parallel database architectures. As discussed earlier, there are three main configurations for parallel database systems: shared memory, shared disk, and shared nothing. Since every architecture has its own characteristics, it is more efficient to provide a separate simulator for each architecture. Simulators for hybrid architectures can be easily developed by tuning the shared nothing simulator.

4.1 Shared Memory Simulation Model

Figure 5 gives the model for the shared memory architecture. As shown in the figure, there is only one logical I/O unit that is shared among all processors. The I/O system may be more than one unit as in the case of distributed shared memory (DSM) and redundant arrays of inexpensive disks (RAID), but with the condition that all are shared among all system processors. This condition suggests the use of centralized deadlock detection technique, which is conveyed in the figure through the use of one shared lockup table. For details on deadlock detection, and database concept in general, refer to [6]. The scheduling of transaction operations to processors is based on the load of the processors.

Initially, a number of transactions are generated and are input to the transaction scheduler. The transaction scheduler schedules a transaction, from those waiting in its queue, to the least busy processor. Every processor originally is assigned a unique sequence number. When several processors have the same load, the tie is broken by choosing the processor with the lowest sequence number. If the scheduled operation is a computation only, i.e. no I/O, the processor executes it. If the scheduled operation is an I/O, the transaction has to go through the lockup table manager to request a lock for the data item to be accessed. If the lock is granted, the transaction proceeds to the I/O server to access the required data item. Otherwise (lock is failed) a deadlock detection routine is invoked to examine if the transaction should be aborted or not. In the case there is a deadlock, all of the locks previously given to the transaction are freed and the transaction itself restarts again. If there is no deadlock, the transaction is blocked till it can gain the lock for the required data. Upon reactivating a blocked transaction, it is scheduled again by the transaction scheduler to a processor that may be different from the one it was previously scheduled to. When a transaction finishes its computation or I/O operation, it is examined to see if it is entirely finished, so it can be committed. If not, the transaction goes through the transaction scheduler again for further processing. If the transaction is to be committed, it frees all of the locks it has, reactivating all of the blocked transactions waiting for those locks. The transaction then exits the system and another transaction is generated in its place. This procedure continues till the simulation time finishes.

4.2 Shared Disk Simulation Model

Figure 6 gives the model for shared disk parallel database. Processors of the system are assumed to be fully connected to each other through a reliable interconnect. As shown in the figure, there is only one logical shared disk, even though this shared disk can be physically a collection of several disks that can

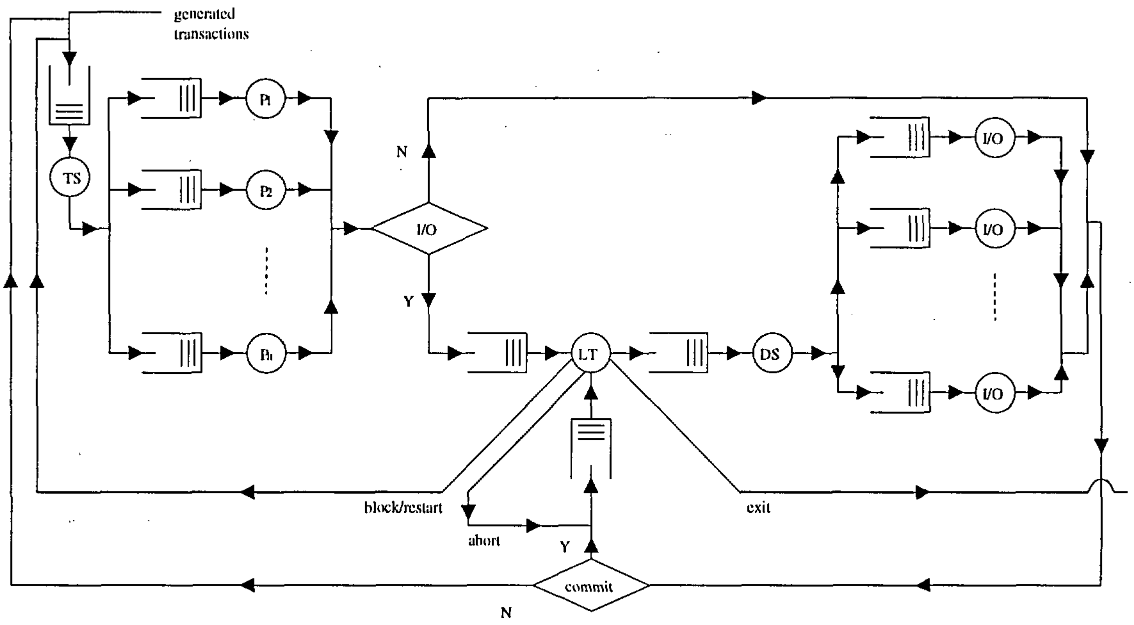


Figure 5: Shared Memory Model
(TS: Transaction Scheduler, Pn: Processor number n - assuming n processors in the system, LT: Lockup Table, DS: Disk Scheduler)

work in parallel as in the case of RAID. In the case of RAID, a disk scheduler is required to determine which disk should be operated to access a specific data item. Information can be exchanged between processors through message passing. The shared disk suggests a centralized lockup table. Each processor has its own memory module that is not shared with other processors, and a lockup table which is a copy of the centralized one. Any update to the processor local lockup table should be followed by updating the centralized one in order to convey the locking information to the other processors.

The simulation process is almost the same as in the shared memory case. Like the shared memory model, processors are scheduled to transactions based on the load of the processor: the least busy processor is chosen. Communication overhead is encountered to ship an operation from the scheduler (which is assumed to be centralized) to the scheduled processor.

4.3 Shared Nothing Simulation Model

Figure 7 models the shared nothing parallel database systems. Every processor has its own lockup table and its I/O module as shown in the figure. Processors of the system are assumed to be fully connected to each other through a reliable interconnect. Distributed deadlock detection is implemented.

The simulation process is almost the same as in the shared memory and shred disk systems with one major difference. In scheduling processors to transactions, the most suitable processor for the transaction (based on the location of the data needed for the transaction operations) is chosen. If a transaction is reactivated after being blocked, it does not go through the transaction scheduler. Instead, the transaction continues at the same processor (as the scheduling process is based in the location of data). As in the shared disk, communication overhead is encountered to ship a transaction to the scheduled processor.

5 Experimentation

This section presents some of the results of the experiments conducted to compare the three architectures. Unless otherwise stated, table 1 gives the parameters used to conduct the experiments. Without loss of generality, the database is assumed to consist of one large table. On having several disk units, the database table is range partitioned among them. Two models of transactions were used in the experiments: short-term debit/credit and long-term DSS models. Workload is modeled as the number of transactions running concurrently on the system. The main measurement taken out of the experiments is the average response time. The simulators are implemented using C language under UNIX. This choice is made based on the popularity of that environment compared to other specialized simulation languages.

Most efforts have been made to ensure that the implemented simulators behave correctly. Toward this end, the random number generator is disabled and carefully selected data are used to test and debug the

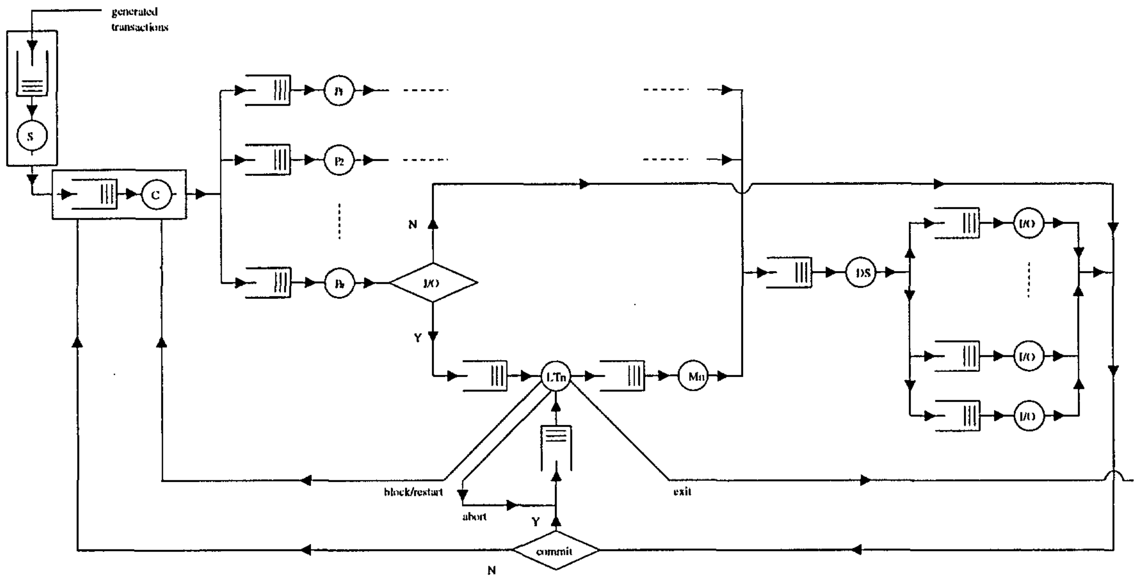


Figure 6: Shared Disk Model
(S: Transaction Scheduler, C: Communication link, Pn: Processor number n,
LTn: Lockup Table that belongs to processor n, Mn: Memory module that belongs to processor n, DS: Disk Scheduler)

programs. While not closely compared to real systems, the results obtained agree to a large extent with our intuition about how the simulated systems behave. Figure 5.3 gives the results obtained from experimenting the simulators for the three architectures along with comparisons among them. The experiments are conducted for number of processors ranging from 1 to 10 because we did not find much enhancement in performance if the number of processors increases beyond this limit. For the same reason, we selected the number of disk units (d in the figure) to range from 1 to 10. Before going into the details of these experiments, the following section briefly discusses the transaction models used.

5.1 Transaction Models

An important factor that affects the system performance is the nature of the transactions performed on the database. A straightforward classification of database transactions is the duration of the transactions (how many operations within a transaction). Some transactions are fine grained, debit/credit in nature. Others are coarse grained and may require locking the entire database such as those involved in decision support applications. Some deductive databases can be categorized as medium grained.

There exist several benchmarks (such as TPC-A, TPC-B, and TPC-C [8]) to measure transactional database performance. To adapt with our abstracted simulation models, we have decided to develop two transaction models. The first models the update, fine grained transactions. The records accessed by a transaction following this type are few and they are related to each other. This type of transactions is typical in debit/credit applications. The model developed takes the form:

- operation1: read recordID
- operation2: process
- operation3: write recordID
- operation4: read (recordID+1)

⋮

recordID (record identification number) in operation 1 is generated at random. The record accessed by operation 4 is simply calculated by adding 1 to the one generated for operation 1.

The second type of transactions is typical in DSS. In this type, transactions are characterized by being long and the records accessed by a transaction are not related. As in the above model, the database is assumed to consist of one table. The model developed for this type is:

- operation1: R / P (recordID / computation amount)
- operation2: R / P (recordID / computation amount)

⋮

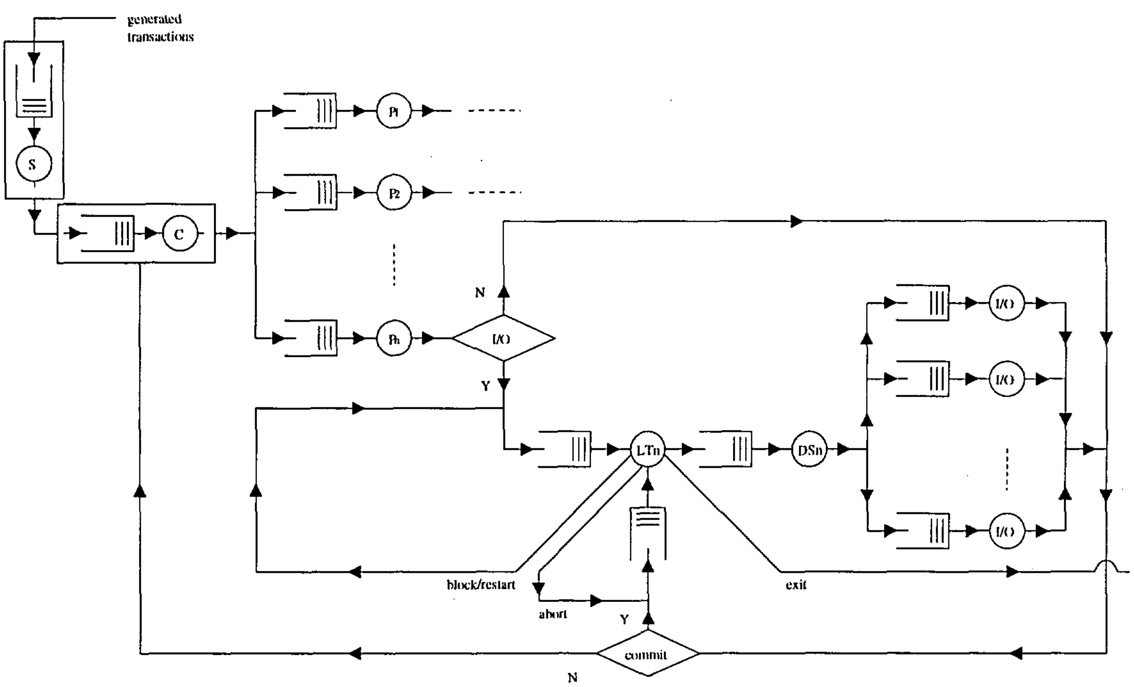


Figure 7: Shared Nothing Model
(S: Transaction Scheduler, C: Communication link, Pn: Processor number n,
LTn: Lockup Table that belongs to processor n, DSn: Disk Scheduler that belongs to processor n)

Table 1: system parameters

eprocessor speed	100 MIPS
memory access time	0.1 microseconds
disk access time	10000 microseconds
processing overhead encountered in an I/O operation	25000 instructions
communication link latency	5000 microseconds

where R / P denotes either read or process operation (which is selected at random) and recordID / computation amount is the record ID to be accessed (in case of read operation) or the amount of computation (in case of process operation). recordID in operation 2 has no relation with recordID in operation 1 and is selected at random.

5.2 Shared Memory, Shared Disk, and Shared Nothing Experimentation

Charts (a, b) in Figure 8 give the results obtained in experimenting the shared memory system with transactions following the debit/credit model under two different workloads. Low workload is modeled by running 10 concurrent transactions: Chart (a), and high workload is modeled by running 1000 concurrent transactions: Chart (b). While increasing number of processors in this case does not significantly affect the system, parallelism in I/O decreases the average response time (hence, enhances the system performance). That is because this type of system is I/O bound, and the shared disk imposes the main bottleneck in the system.

Chart (c) examines systems running transactions following the DSS model. In this model, processing time exceeds the I/O time. From the chart, parallelism is taking place in processing units rather than in I/O units. However, due to the bottleneck encountered in the shared memory, the system performance saturates at some point regardless of how many processing or I/O units are used.

Charts (d-f) in Figure 8 give the results obtained in experimenting the shared disk system. The behavior of the system is similar to what we have in the shared memory. While there is no shared memory, communication delays have an impact on the system performance.

Charts (g-i) in Figure 8 examine shared nothing systems. The behavior of the system is essentially the same as in the cases of shared memory and shared disk. While every processor has its own memory and disk modules, communication overhead is the main suffering in the system.

5.3 Parallel Database Architectures: Comparison

This section concludes the experimentation study by comparing the performance of the three architectures. Charts (j-l) in Figure 8 give such comparison for systems running short-term transactions following the debit/credit model for the two workloads, and the DSS model.

In Chart (j), the transaction model is debit/credit and the number of concurrent transactions running in the systems is 10. In the shared nothing system, a processor is scheduled for an operation based on the location of the data record to be accessed by the operation. For the debit/credit of transaction model, records to be accessed within a transaction are related. This leads to limiting the scheduling overhead for a transaction to only one scheduling operation most of the time. This way, a very low communication and scheduling overhead is encountered within the system. On the other hand, a transaction needs to be scheduled for very operation in shared disk and shared memory, as the criteria used is the load of the processor not the location of data. This leads to a high scheduling overhead. Beside this overhead, the shared disk suffers from its need to communication within the scheduling operations to ship transaction operations to processors. This explains why shared nothing performs the best among the three systems in such transaction type and workload.

Chart (k) compares the three systems for heavy load debit/credit transactions. The above arguments hold here again with an exception in the shared memory system which suffers from a contention in the shared memory under the heavy load. The shared disk does not have such a problem as every processor has its own memory module. This gives the reasons why the shared nothing system gives the best performance followed by the shared disk.

Chart (l) compares the three system for DSS transactions. It is found out that the shared memory and shared disk give a much better performance than the shared nothing, with the shared memory is the leading system. In the DSS model, a transaction is long and the data accessed by its operations are not related. For the shared nothing system, as transaction operations are scheduled based on data location, a transaction needs to be scheduled almost for every operation. This leads to a high overhead in both scheduling and communication. In shared memory and shared disk, scheduling is based on the processor workload and this evenly distributes the load on the processors leading to a better performance. The communication overhead encountered in the shared disk system, however, puts it in the second place after the shared memory system which has the communication comes for free through the shared memory.

6 Conclusion

Based on simulation, this study compares the performance of the main architectures of parallel database systems. We developed three simulators for the shared memory, shared disk, and shared nothing parallel databases. A large number of experiments were conducted using these simulators. Two transaction models were used throughout the study: short-term transactions with related operations, and long-term transactions with unrelated operations.

For the three systems, we found out that for fine grain transactions where the I/O time dominates the processing time, the parallelism in the disk units is more effective than it is in processing units. For medium and course grained transactions where the processing time is comparable to or exceeds the I/O time, parallelism in processing units can improve system performance.

Deciding on the best architecture for a database system depends heavily on the application requirements and the nature of operations involved in the application. When the transactions follow the debit/credit transaction model, the shared nothing provides the best performance among the three architectures (especially for heavy loaded systems). Through function shipping and as the operations of a transaction are related, no much communication is required. This is why shared nothing provides almost a linear speedup. For decision support system workload, transactions are long and the operations are not related. This raises a communication overhead. Through a good load balance, shared disk and shared memory systems provide better performance than shared nothing. The memory contention found in the shared memory is offset by the communication overhead involved in the shared disk. This gives the reason why the shared memory still provides the best performance in such transaction model.

References

[1] F. Barlos and O. Frieder, "A Load Balanced Multicomputer Relational Database System for Highly Skewed Data," Parallel computing, vol. 21, no. 9, 1995, pp. 1451-1483.

[2] B. Bergsten, M. Couprie, and P. Valduriez, "Overview of Parallel Architectures for

Databases," The Computer Journal, vol. 36, no. 8, 1993, pp. 734-740.

[3] H. Boral, W. Alexander, L. Clay, G. Copeland, S. Danforth, M. Franklin, B. Hart, M. Smith, and P. Valduriez, "Prototyping Bubba, A Highly Parallel Database System," IEEE Transactions on Knowledge and Data Engineering, vol. 2, no. 1, 1990, pp. 4-24.

[4] D. Dewitt and J. Gray, "Parallel Database Systems: the Future of the High Performance Database Systems," Comm of ACM, vol. 35, no. 6, 1992, pp. 85-98.

[5] D. DeWitt, S. Ghandeharizadeh, D. Schneider, A. Bricker, H. Hsiao, and R. Rasmussen, "The Gamma Database Machine Project," IEEE Transactions on Knowledge and Data Engineering, vol. 2, no. 1, 1990, pp. 44-62.

[6] R. Elmasri and S. Navathe, "Fundamentals of Database Systems," Benjamin/Cummings, 1994.

[7] G. Graefe and S. Thakkar, "Tuning a Parallel Database Algorithm on a Shared-memory Multiprocessor," Software, practice and experience, vol. 22, no. 7, 1992, pp. 495-517.

[8] J. Gray, Editor, "The Benchmark Handbook: for Database and Transaction Processing Systems," Morgan Kaufmann, San Mateo, CA, 1991.

[9] A. Helal, D. Yuan, and H. El-Rewini, "Dynamic Data Reallocation for Skew Management in Shared-Nothing Parallel Databases," International Journal of Distributed and Parallel Databases, Kluwer Academic Publishers, Volume 5, Number 3, 1997. (to appear)

[10] D. Hsiao, "A Parallel, Scalable, Microprocessor-Based Database Computer for Performance Gains and Capacity Growth: Using a variable number of processors to produce an experimental computer," Ieee micro, vol. 11, no. 6, 1991, pp. 44-60.

[11] B. Jenq, B. Twichell, and T. Keller, "Locking Performance in a Shared Nothing Parallel Database Machine," IEEE transactions on knowledge and data engineering, vol. 1, no. 4, 1989, pp. 530-543.

[12] T. Lewis and H. El-Rewini, "Introduction to Parallel Computing," Prentice-Hall, 1992.

[13] E. Mohamed, "Parallel Database Architectures: A Simulation Study," MS Thesis, Computer Science Department, University of Nebraska at Omaha, 1996.

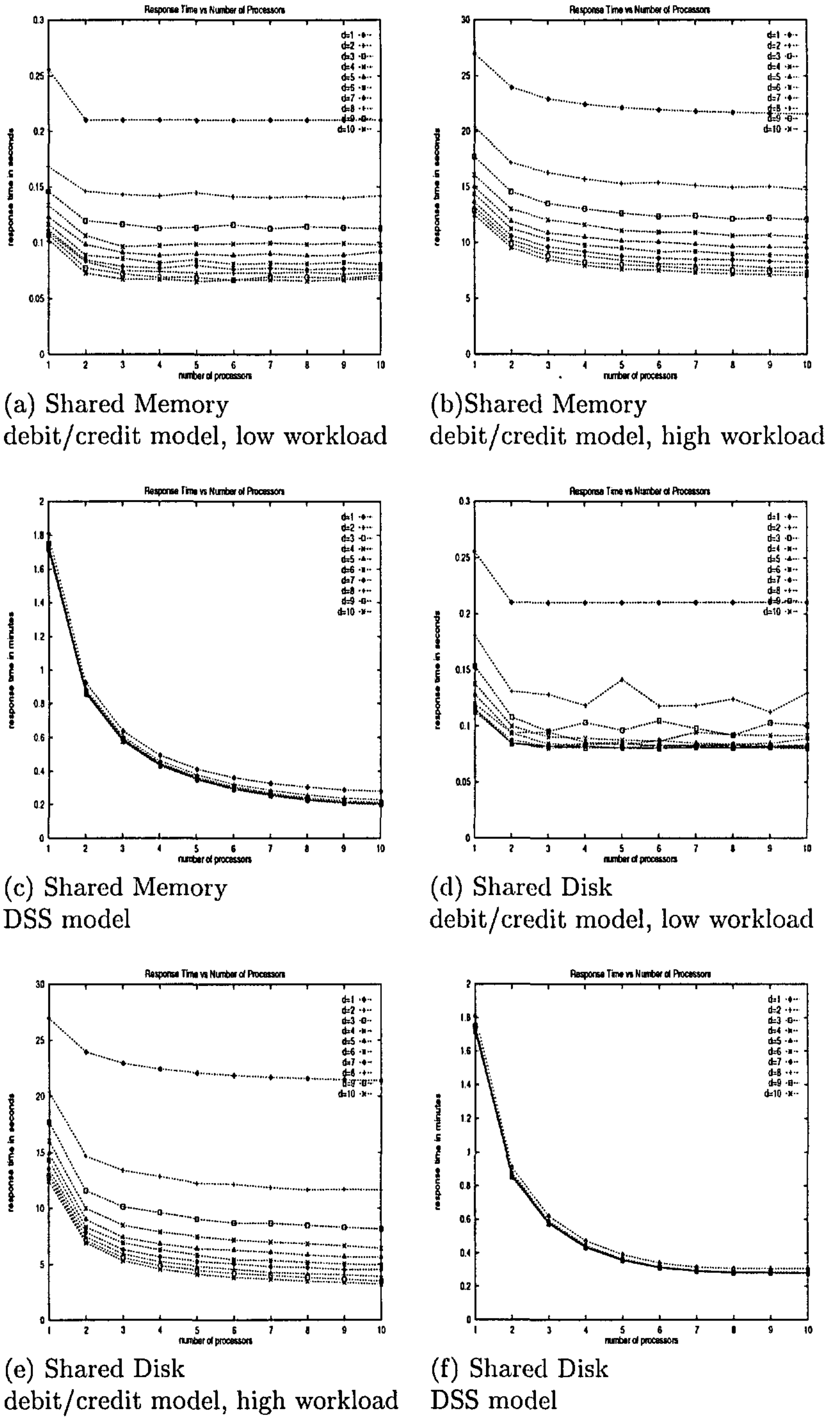
[14] J. Page, "A Study of A Parallel Database Machine and its Performance - The NCR/Teradata DBC/1012," Lecture Notes in Computer Science, Springer-Verlag, vol. 618, 1992, pp. 115-137.

[15] E. Rahm, "Parallel Query Processing in Shared Disk Database Systems," Sigmod record, vol. 22, no. 4, 1993, pp. 32-37.

[16] P. Valduriez and P. Rodin, "Parallel Database Systems: Open Problems and New Issues," Distributed and Parallel Databases vol. 1, 1993, pp. 137-165.

[17] P. Valduriez and P. Rodin, "Parallel Database Systems: the case for shared-something," Proceedings of the Ninth International Conference on Data Engineering , Vienna, Austria, IEEE Computer Society, April 1993, pp. 460-465.

[18] S. Zhou, M. Williams, and H. Taylor, "Practical Throughput Estimation for Parallel Databases," Software Engineering Journal, pp. 255-263, vol. 11, no. 4, 1996, pp. 255-263. 1996.



(a) Shared Memory
debit/credit model, low workload

(b)Shared Memory
debit/credit model, high workload

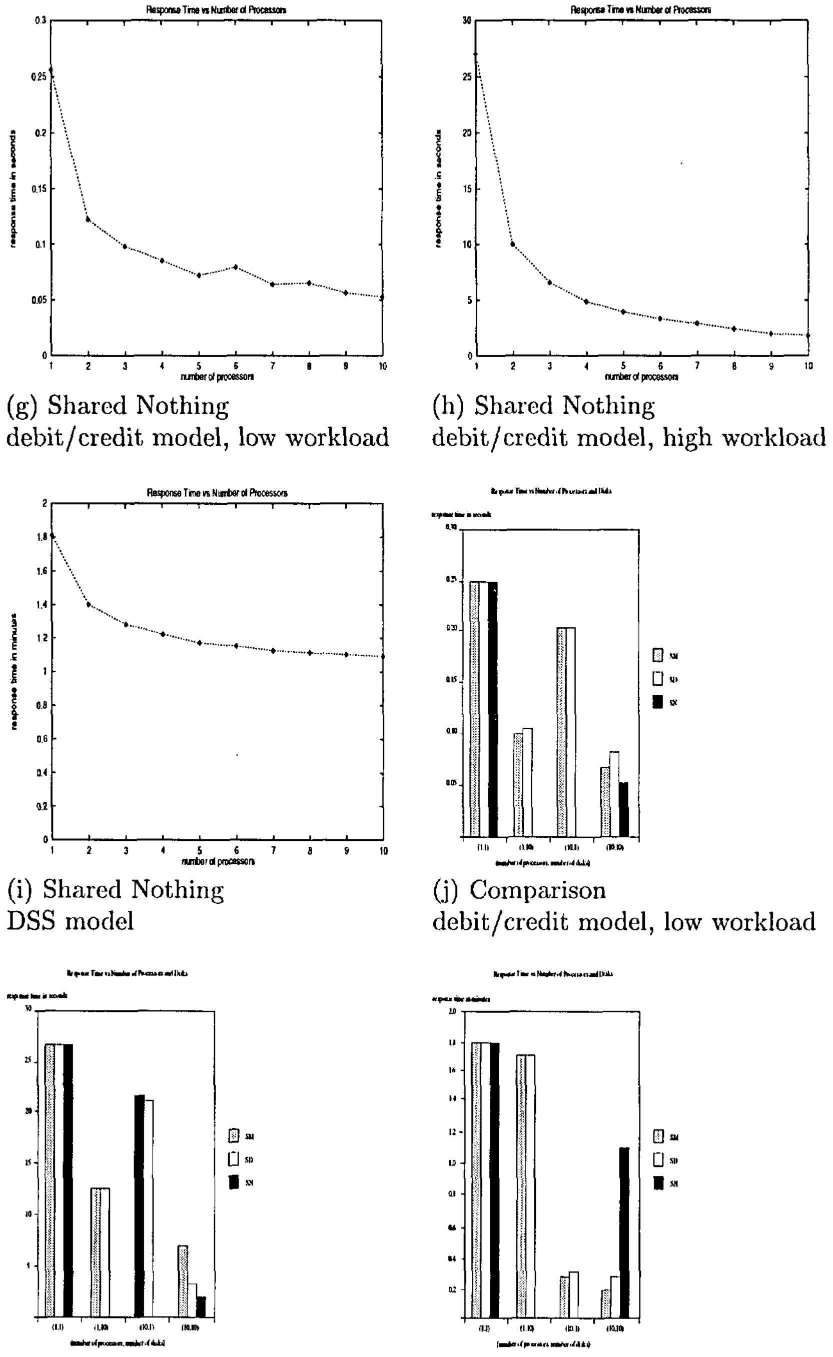
(c) Shared Memory
DSS model

(d) Shared Disk
debit/credit model, low workload

(e) Shared Disk
debit/credit model, high workload

(f) Shared Disk
DSS model

Figure 8: Experimentation



(g) Shared Nothing
debit/credit model, low workload

(h) Shared Nothing
debit/credit model, high workload

(i) Shared Nothing
DSS model

(j) Comparison
debit/credit model, low workload

(k) Comparison
debit/credit model, high workload

(l) Comparison
DSS model

Figure 8: Experimentation (continued)

Experimental evaluation of three partition selection criteria for decision table decomposition

Blaž Zupan and Marko Bohanec
Jožef Stefan Institute,
Jamova 39, Ljubljana, Slovenia
Phone: +386 61 177 3900
Fax: +386 61 125 1038
E-mail: blaz.zupan@ijs.si, marko.bohanec@ijs.si

Keywords: decision table decomposition, partition selection criteria, intermediate concepts, concept hierarchy, knowledge discovery

Edited by: Rudi Murn

Received: May 20, 1997 **Revised:** March 7, 1998 **Accepted:** April 18, 1998

Decision table decomposition is a machine learning approach that decomposes a given decision table into an equivalent hierarchy of decision tables. The approach aims to discover decision tables that are overall less complex than the initial one, potentially easier to interpret, and introduce new and meaningful intermediate concepts. Since an exhaustive search for an optimal hierarchy of decision tables is prohibitively complex, the decomposition uses a suboptimal iterative algorithm that requires the so-called partition selection criterion to decide among possible candidates for decomposition. This article introduces two such criteria and experimentally compares their performance with a criterion originally used for the decomposition of Boolean functions. The experiments highlight the differences between the criteria, but also show that in all three cases the decomposition may discover meaningful intermediate concepts and relatively compact decision tables.

1 Introduction

A decision table provides a simple means for concept representation. It represents a concept with labeled instances, each relating a set of attribute values to a class. Decision table *decomposition* is a method based on the "divide and conquer" approach: given a decision table, it decomposes it to a hierarchy of decision tables. The method aims to construct the hierarchy so that the new decision tables are less complex and easier to interpret than the original decision table.

The decision table decomposition method is based on function decomposition, an approach originally developed for the design of digital circuits [2]. The method iteratively applies a *single decomposition step*, whose goal is to decompose a function $y = F(X)$ into $y = G(A, H(B))$, where X is a set of input attributes $x_1, \ldots, x_n$, and y is the class variable. F, G and H are functions represented by *decision tables*, i.e., possibly incomplete sets of attribute-value vectors with assigned classes. A and B are nonempty subsets of input attributes such that $A \cup B = X$. The functions G and H are developed by decomposition and are not predefined in any way. Such a decomposition also discovers a new intermediate concept $c = H(B)$. Since the decomposition can be applied recursively on G and H, the result in general is a *hierarchy* of decision tables. As each decision table represents a concept, the result of decomposition can be regarded also as a *concept hierarchy*.

Each single decomposition step aims to minimize the joint complexity of G and H and executes the decomposition only if this is lower than the complexity of F. Moreover, it is of crucial importance for the algorithm to find such partition of attributes X into sets A and B that yields G and H of the lowest complexity. The criteria that guide the selection of such partition are called *partition selection criteria*.

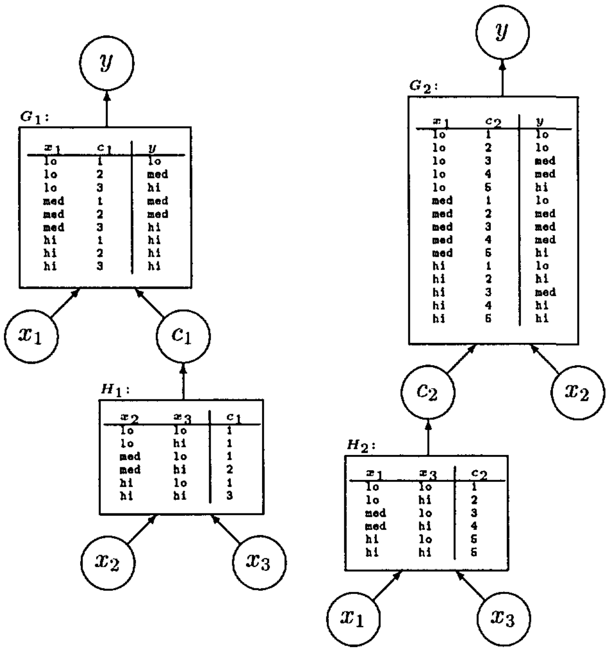
Let us illustrate the decomposition by a simple example (Table 1). The decision table relates the input attributes x_1, x_2, and x_3 to the class y, such that $y = F(x_1, x_2, x_3)$. There are three possible partitions of attributes that yield three different decompositions $y = G_1(x_1, H_1(x_2, x_3))$, $y = G_2(x_2, H_2(x_1, x_3))$, $y = G_3(x_3, H_3(x_1, x_2))$. The first two are given in Figure 1, and the comparison shows that:

- decision tables in the decomposition $y = G_1(x_1, H_1(x_2, x_3))$ are overall smaller than those for $y = G_2(x_2, H_2(x_1, x_3))$,
- the new concept $c_1 = H_1(x_2, x_3)$ uses only three values, whereas that for $H_2(x_1, x_3)$ uses five,
- it is hard to interpret decision tables G_2 and H_2,

x_1	x_2	x_3	y
lo	lo	lo	lo
lo	lo	hi	lo
lo	med	lo	lo
lo	med	hi	med
lo	hi	lo	lo
lo	hi	hi	hi
med	lo	lo	med
med	lo	hi	med
med	med	lo	med
med	med	hi	med
med	hi	lo	med
med	hi	hi	hi
hi	lo	lo	hi
hi	lo	hi	hi
hi	med	lo	hi
hi	med	hi	hi
hi	hi	lo	hi
hi	hi	hi	hi

Table 1: An example decision table.

whereas by inspecting G_1 and H_1 it can be easy to see that $c_1 = \text{MIN}(x_2, x_3)$ and $y = \text{MAX}(x_1, c_1)$. This can be even more evident with the reassignment of c_1's values: 1 to lo, 2 to med, and 3 to hi.



G_1:

x_1	c_1	y
lo	1	lo
lo	2	med
lo	3	hi
med	1	med
med	2	med
med	3	hi
hi	1	hi
hi	2	hi
hi	3	hi

H_1:

x_2	x_3	c_1
lo	lo	1
lo	hi	1
med	lo	1
med	hi	2
hi	lo	1
hi	hi	3

G_2:

x_1	c_2	y
lo	1	lo
lo	2	lo
lo	3	med
lo	4	med
lo	5	hi
med	1	lo
med	2	med
med	3	med
med	4	med
med	5	hi
hi	1	lo
hi	2	hi
hi	3	med
hi	4	hi
hi	5	hi

H_2:

x_1	x_3	c_2
lo	lo	1
lo	hi	2
med	lo	3
med	hi	4
hi	lo	5
hi	hi	5

Figure 1: Two different decompositions of the decision table from Table 1.

The above comparison indicates that the decomposition $y = G_2(x_2, H_2(x_1, x_3))$ yields more complex and less interpretable decision tables than the decomposition $y = G_1(x_1, H_1(x_2, x_3))$. The questions of interest are thus:

1. How do we measure the overall complexity of original decision table and of the decomposed system?

2. Which are the criteria that can guide the single decomposition step to chose among possible decompositions?

3. How much information is contained within the hierarchical structure itself?

4. How does interpretability relate to the overall complexity of decision tables in the decomposed system? Is a less complex system also easier to interpret?

Some of these questions were already addressed in the area of computer aided circuit design where decomposition is used to find a circuit of minimal complexity that implements a specific tabulated Boolean function. There, the methods mostly rely on the complexity and partition selection criterion known as Decomposed Function Cardinality (DFC, see [21]). However, a question is whether this criterion can be used for the decomposition of decision tables of interest to machine learning, where attributes and classes usually take more than two values. Moreover, the main concern of Boolean function decomposition is the minimization of digital circuit, leaving aside the question of comprehensibility and interpretability of the resulting hierarchy.

This article is organized as follows. The next section reviews related work on decision table decomposition with the emphasis on its use for machine learning. The decomposition algorithm to be used throughout the article is presented in section 3. Section 4 introduces two new partition selection criteria that are based on the information content of decision tables (DTIC) and on the cardinality of newly discovered concepts (CM). That section also discusses how DFC and DTIC may be used to estimate the overall complexity of derived decision tables, and shows how DTIC may be used to assess the information content of the discovered hierarchical structure itself. Section 5 experimentally evaluates the different criteria and complexity measures. Section 6 summarizes the results and concludes the article.

2 Related work

The decomposition approach to machine learning was used early by a pioneer of artificial intelligence, A. Samuel. He proposed a method based on a *signature table system* [22] and successfully used it as an evaluation mechanism for checkers playing programs. This approach was later improved by Biermann et al. [3]. Their method, however, did not address the problem of deriving the hierarchy of concepts, which was supposed to be given by a domain expert.

A similar approach had been defined even earlier within the area of switching circuit design. In 1956, R.L. Ashenhurst reported on a unified theory of *decomposition of switching functions* [2]. The decomposition method proposed by Ashenhurst was used to decompose a completely specified truth table of a Boolean function to be then realized with standard binary gates. Thus, the method could construct concept hierarchies as well as their corresponding decision tables. Most of other related work of those times is reported and reprinted by Curtis [8].

Recently, the Ashenhurst-Curtis approach was substantially improved by research groups of M. A. Perkowski, T. Luba, and T. D. Ross. In [18], Perkowski et al. report on the *decomposition approach for incompletely specified switching functions.* Luba [12] proposed a method for the decomposition of *multi-valued switching functions* in which each multivalued variable is encoded by a set of Boolean variables. A decomposition of k-valued functions was proposed by Files et al. [10]. The authors identify the potential usefulness of function decomposition for machine learning, and Goldman [11] indicates that the decomposition approach to switching function design might be termed *knowledge discovery,* since a function not previously foreseen might be discovered. From the viewpoint of machine learning, however, the main drawbacks of these methods are that they are mostly limited to Boolean functions and incapable of dealing with noise.

Feature discovery has been at large investigated by *constructive induction* [14]. Perhaps closest to function decomposition are the constructive induction systems that use a set of existing attributes and a set of constructive operators to derive new attributes. Several such systems are presented in [13, 19, 20].

Within machine learning, there are other approaches that are based on problem decomposition, but where the problem is decomposed by the expert and not by a machine. A well-known example is *structured induction,* developed by Shapiro [23]. His approach is based on a manual decomposition of the problem. For every intermediate concept either a special set of learning examples is used or an expert is consulted to build a corresponding decision tree. In comparison with standard decision tree induction techniques, Shapiro's approach exhibits about the same classification accuracy with the increased transparency and lower complexity of the developed models. Michie [15] emphasizes the important role the structured induction will have in the future development of machine learning and lists several real problems that were solved in this way.

The work presented here is based on our own decomposition algorithm [25] in which we took the approach of Curtis [8] and Perkowski et al. [18], and extended it to handle multi-valued categorical attributes and functions. The algorithm was demonstrated to perform well in terms of generalization [26], discovery of relevant concept hierarchies [7], and feature construction [27] in fairly complex problem domains.

3 Decomposition algorithm

Let F be a decision table consisting of attribute-value vectors that map the attributes $X = \{x_1, \ldots, x_n\}$ to the class y, so that $y = F(X)$. A single decomposition step searches through all the partitions of attributes X into a *free* set A and *bound* set B, such that $A \cap B = \emptyset$, $A \cup B = X$, and A and B each contain at least one attribute. Let us denote such a partition with $A|B$ and assume that a *partition selection criterion* $\psi(A|B)$ exists that measures the appropriateness of this partition for decomposition (partitions with lower ψ are more appropriate). The partition with the lowest ψ is selected and F is decomposed to G and H, so that $y = G(A, c)$ and $c = H(B)$. Provided there exists a *complexity measure* θ for F, G, and H, F is decomposed only if the *complexity condition* $\theta(F) > \theta(G) + \theta(H)$ is satisfied. Several partition selection (ψ) and complexity (θ) measures are introduced in the next section.

The algorithm that implements the single decomposition step and decomposes a decision table F to G and H is described in detail in [25]. Here, we illustrate it informally using the decision table from Table 1. For every attribute partition, the method constructs a *partition matrix* with the attributes of bound set in columns and of free set in rows. Each column in the partition matrix denotes the behavior of F for a specific combination of values of bound attributes. The same columns can be represented with the same value of c. The number of different columns is equal to the minimal number of values for c to be used for decomposition. In this way, every column is assigned a value of c, and G and H are straightforwardly derived from such an annotated partition matrix. For each of three partitions for our example decision table F, the partition matrices with the corresponding values of c are given in Figure 2.

The assignment of c's values is trivial when decision table instances completely cover the attribute space. When this is not the case, Wan and Perkowski [24] proposed an approach that treats missing decision table entries as "don't cares". Each partition matrix can then have several assignments of values for c. The problem of finding the assignment that uses the fewest values is then equivalent to optimal graph coloring. Graph coloring is an NP-hard problem and the computation time of an exhaustive search algorithm is prohibitive even for small graphs. Instead, Wan and Perkowski suggested a heuristic Color Influence Method of polynomial complexity and showed that the method performed well compared to the optimal algorithm. Although the examples used in this article use decision tables that completely cover the attribute space, the complexity and partition measures

	x_2	lo	lo	med	med	hi	hi
x_1	x_3	lo	hi	lo	hi	lo	hi
lo		lo	lo	lo	med	lo	hi
med		med	med	med	med	med	hi
hi		hi	hi	hi	hi	hi	hi
c		1	1	1	2	1	3

	x_1	lo	lo	med	med	hi	hi
x_2	x_3	lo	hi	lo	hi	lo	hi
lo		lo	lo	med	med	hi	hi
med		lo	med	med	med	hi	hi
hi		lo	hi	med	hi	hi	hi
c		1	2	3	4	5	5

	x_1	lo	lo	lo	med	med	med	hi	hi	hi
x_3	x_2	lo	med	hi	lo	med	hi	lo	med	hi
lo		lo	lo	lo	med	med	med	hi	hi	hi
hi		lo	med	hi	med	med	hi	hi	hi	hi
c		1	2	3	4	5	5	6	6	6

Figure 2: Partition matrices for Table 1 using three different partitions of attributes x_1, x_2, and x_3.

introduced apply with no difference to incompletely covered cases as well.

The decomposition algorithm examines all decision tables in the evolving concept hierarchy and then applies a single decomposition step to the decision table and its partition that was evaluated as the most appropriate by ψ and that satisfies the complexity condition $\theta(F) > \theta(G) + \theta(H)$. If several partitions are scored equal, the algorithm arbitrarily selects one among those with the lowest number of elements in the bound set. The process is repeated until no decomposition is found that would satisfy the complexity condition.

We illustrate this stepwise decomposition using the CAR domain that is described in section 5. Figure 3 shows a possible evolving concept hierarchy obtained by decomposition. Each consecutive hirarchy is a result of a single decomposition step. Only the hierarchical structure without decision tables is shown.

The overall time complexity of decision table decomposition algorithm is polynomial in the number of examples, number of attributes, and maximal number of columns in partition matrices [26]. As the latter grows exponentially with the number of bound attributes, it is advantageous to limit the size of the bound set. In the experiments presented in Section 5, however, the problems were sufficiently small to examine all possible bound sets.

The above decomposition algorithm was implemented in the C language as a part of the system called HINT (Hierarchy INduction Tool) [25]. HINT runs on several UNIX platforms, including HP/UX and SGI Iris.

4 Partition selection criteria and complexity measures

This section reviews one and introduces two new partition selection criteria. For each, it also defines the complexity measure and corresponding complexity condition. Furthermore, two overall complexity measures for the hierarchy of decision tables are defined, and, finally, a measure for estimating the information content of the hierarchy itself is presented.

4.1 Partition selection criteria

4.1.1 Decomposed function cardinality

Decomposed function cardinality (DFC) was originally proposed by Abu-Mostafa [1] as a general measure of complexity and used in decomposition of Boolean functions [21]. DFC is based on the cardinality of the function. Given a decision table $F(X)$, DFC-based complexity is defined as:

$$\theta_{\mathrm{DFC}}(F) = ||X|| = \prod_i |x_i|, \ x_i \in X \qquad (1)$$

where $|x_i|$ represents the cardinality of attribute x_i, i.e., the number of values it uses.

The DFC partition selection criterion for decomposition $F(X) = G(A, c)$ and $c = H(B)$ is then:

$$\begin{aligned} \psi_{\mathrm{DFC}}(A|B) &= \theta_{\mathrm{DFC}}(G) + \theta_{\mathrm{DFC}}(H) \\ &= |c| \ ||A|| + ||B|| \end{aligned} \qquad (2)$$

The complexity condition using the above definitions is $\theta_{\mathrm{DFC}}(F) > \theta_{\mathrm{DFC}}(G) + \theta_{\mathrm{DFC}}(H)$, or equivalently $||X|| > |c| \ ||A|| + ||B||$.

For our example decision table (Table 1) and the corresponding partition matrices (Figure 2), the partition selection criteria are: $\psi_{\mathrm{DFC}}(x_1|x_2x_3) = 9+6 = 15$, $\psi_{\mathrm{DFC}}(x_2|x_1x_3) = 15 + 6 = 21$, and $\psi_{\mathrm{DFC}}(x_3|x_1x_2) = 12 + 9 = 21$. $\theta_{\mathrm{DFC}}(F)$ is 18. The only partition that satisfies the DFC decomposition criterion is $x_1|x_2x_3$.

DFC's ability to guide the decomposition of Boolean functions has been illustrated in several references including [21, 11]. For multi-valued logic synthesis, a DFC-guided decomposition was proposed in [10].

4.1.2 Information content of decision tables

Decision table information content (DTIC) is based on the idea of Biermann et al. [3] who counted the num-

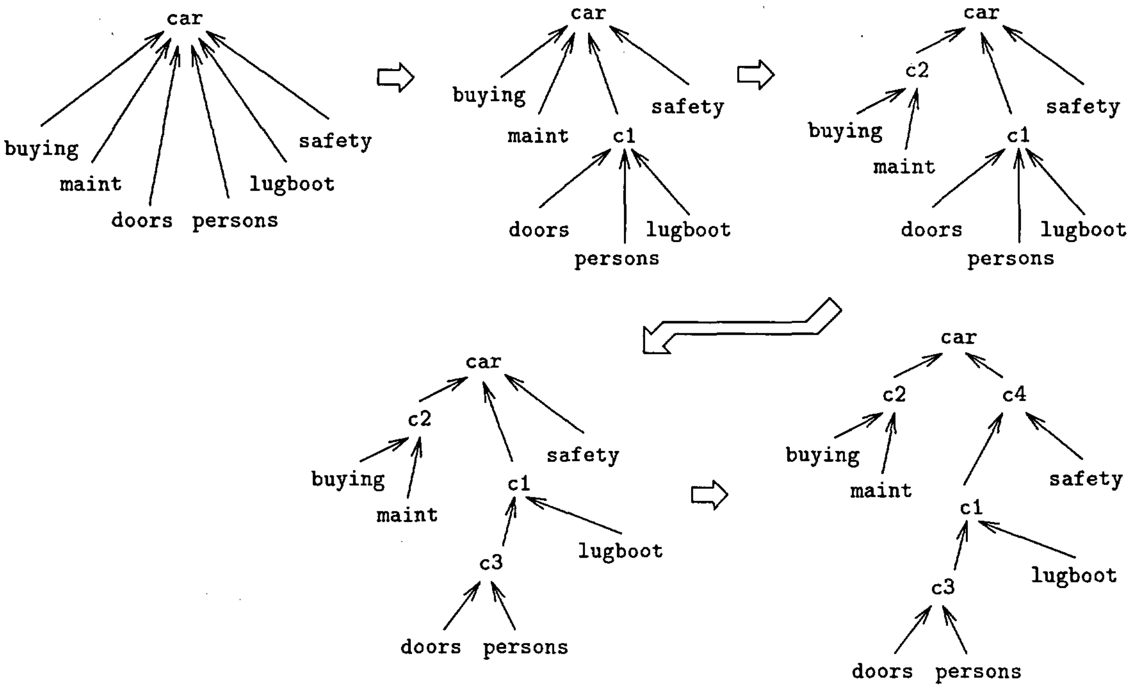


Figure 3: Evolving concept hierarchy discovered by decomposition of the CAR decision table. Each consecutive hierarchy results from a single-step decomposition of its predecessor.

ber of different functions that can be represented by a given *signature table schema*, i.e., a tree of concepts whose cardinality is predefined.

A decision table $y = F(X)$ can represent $|y|^{||X||}$ different functions. Assuming the uniform distribution of functions, the number of bits to encode such a decision table is then

$$\theta_{\mathrm{DTIC}}(F) = ||X|| \log_2 |y| \text{ bits} \tag{3}$$

Note that for binary functions where $|y| = 2$, this is equal to $\theta_{\mathrm{DFC}}(F)$.

When decomposing $y = F(X)$ to $y = G(A, c)$ and $c = H(B)$, we assign a single value from the set $\{1, 2, \ldots, |c|\}$ to each of the columns of partition matrix. But, each of the values has to be assigned to at least one instance. In other words, from $|y|^{||B||}$ different functions we have to subtract all those that use less than $|c|$ values. The number of different functions with exactly $|c|$ possible values is therefore $N(|c|)$, where N is defined as:

$$\begin{aligned} N(x) = & \; x^{||B||} - \sum_{i=1}^{x-1} \binom{x}{i} N(i) \\ N(1) = & \; 1 \end{aligned} \tag{4}$$

Furthermore, since the actual label (value of c) of the column is not important, there are $|c|!$ such equivalent assignments and therefore $|c|!$ equivalent decision tables H. A specific H therefore uniquely represents $N(|c|)/|c|!$ functions with exactly $|c|$ values, and the corresponding information content is:

$$\theta'_{\mathrm{DTIC}}(H) = \log_2 N(|c|) - \log_2(|c|!) \text{ bits} \tag{5}$$

The DTIC partition selection criterion prefers the decompositions with simple decision tables G and H and low information content, so that:

$$\psi_{\mathrm{DTIC}}(A|B) = \theta_{\mathrm{DTIC}}(G) + \theta'_{\mathrm{DTIC}}(H) \tag{6}$$

The DTIC-based complexity condition is:

$$\theta_{\mathrm{DTIC}}(F) > \theta_{\mathrm{DTIC}}(G) + \theta'_{\mathrm{DTIC}}(H) \tag{7}$$

For Table 1, DTIC evaluates to: $\psi_{\mathrm{DTIC}}(x_1|x_2x_3) =$ 20.76 bits, $\psi_{\mathrm{DTIC}}(x_2|x_1x_3) =$ 27.68 bits, and $\psi_{\mathrm{DTIC}}(x_3|x_1x_2) =$ 30.39 bits. $\theta_{\mathrm{DTIC}}(F)$ is 28.53 bits, and, in contrast to DFC, two partitions qualify for decomposition. Among these, as with DFC, the partition $x_1|x_2x_3$ is preferred.

4.1.3 Column multiplicity

Column multiplicity (CM) is the simplest complexity measure introduced in this article and equals to the cardinality of c ($|c|$), also referred to by Ashenhurst and Curtis as *column multiplicity* number of partition matrix [2, 8]. Formally,

$$\psi_{\mathrm{CM}}(A|B) = |c| \tag{8}$$

The idea for this measure came from practical experience with DEX decision support system [5].

There, the hierarchical system of decision tables is constructed manually and it has been found that decision tables with small number of output values are easier to construct and interpret.

For our example and similarly to DFC and DTIC, CM also selects the partition $x_1|x_2x_3$ with $\psi_{\mathrm{CM}} = 3$. The remaining two partitions have $\psi_{\mathrm{CM}}(x_2|x_1x_3) = 5$ and $\psi_{\mathrm{CM}}(x_3|x_1x_2) = 6$.

Unlike DTIC and DFC, CM can not be simply summed up to determine the joint complexity of a set of decision tables, which is needed to determine the complexity condition. Consequently, when we employ CM to guide the partition selection, we use DTIC to determine the decomposability.

4.2 Complexity estimation for decision table hierarchy

Using DFC, the overall complexity of decision tables in the concept hierarchy is the sum of θ_{DFC} for each decision table. Similarly, for DTIC, the complexity estimation is again the sum of DTIC complexities of each of the decision tables, with the distinction that θ_{DTIC} is used for the decision table at the root of the hierarchy and θ'_{DTIC} for all other decision tables.

For example, consider the two concept hierarchies from Figure 1. Their overall complexities as measured by DFC are 15 and 21, respectively, and 20.76 bits and 27.68 bits as measured by DTIC. These measures confirm that the first decomposition is less complex and thus preferred to the second one. The original undecomposed decision table had DFC equal to 18 and DTIC equal to 28.53 bits. Therefore, in terms of DTIC both decompositions reduced the complexity, while using DFC this happened only with the first one.

Note that the so-obtained DTIC complexity estimation is just an approximation of the exact complexity that would take into account the actual number of functions representable by a multi-level hierarchy. This is because DTIC is designed for a single table only and does not consider the reducibility [3] that occurs in multi-level hierarchies and effectively decreases the number of representable functions. Therefore, the estimated overall DTIC is the upper bound of the actual complexity.

4.3 Structure information content

Using DTIC we can assess both the amount of information contained in the original decision table and contained in the resulting decision tables that were constructed by decomposition. The difference of the two is the information contained in the hierarchical structure itself. We call this measure *structure information content* (SIC). The more informative the hierarchy, the overall less complex the resulting decision tables.

For the two decompositions in Figure 1, the corresponding structure information contents are 7.77 bits and 0.85 bits, respectively. Since the first SIC is considerably greater than the second one, the first structure is more informative and its decision tables more compact.

5 Experimental evaluation

To evaluate the proposed partition selection criteria and complexity measures, we used three artificial and three real-world domains that were selected so that their concept hierarchies were either known in advance or could have been easily anticipated. For each domain, the decomposition aimed to discover this hierarchy. For evaluation, we qualitatively assess the similarity of the two hierarchies and quantitatively compare them by using the proposed complexity measures.

Each of six domains is represented with the initial decision table containing instances that completely cover the attribute space. Although the experiments could as well be done with sparser decision tables (see [25]), we wanted to focus in this article only on the discovery of concept hierarchies. Note that the proposed partition selection measures depend only on cardinalities of attributes and concepts, and not on the actual number of instances in decision tables. Furthermore, we have shown in [26] that by increasing the problem space coverage by training instances, the discovered concepts converge to those from complete training sets.

The results of decompositions are shown as concept hierarchy structures, where, unless otherwise noted, the labels of intermediate concepts indicate the order in which they were discovered.

5.1 Artificial domains

Three artificial domains were investigated:

1. a Boolean function
   $y = (x_1 \text{ OR } x_2) \text{ AND } x_3 \text{ AND } (x_4 \text{ XOR } x_5)$,
2. a six-attribute palindrome function,
3. a three-valued function
   $y = \mathrm{MIN}(x_1, \mathrm{AVG}(x_2, \mathrm{MAX}(x_3, x_4), x_5))$.

For the first function, the initial decision table has $2^5 = 32$ instances, $\theta_{\mathrm{DFC}} = 32$ and $\theta_{\mathrm{DTIC}} = 32$ bits. While decomposition with DTIC and CM discovered the anticipated hierarchy, the DFC-guided decomposition terminated too soon because the complexity condition did not allow to decompose the decision tables any further (see Figure 4). Note that the overall DFC is the same for all discovered hierarchies, while the structure information content is higher for those discovered by DTIC and CM. The decision tables (not

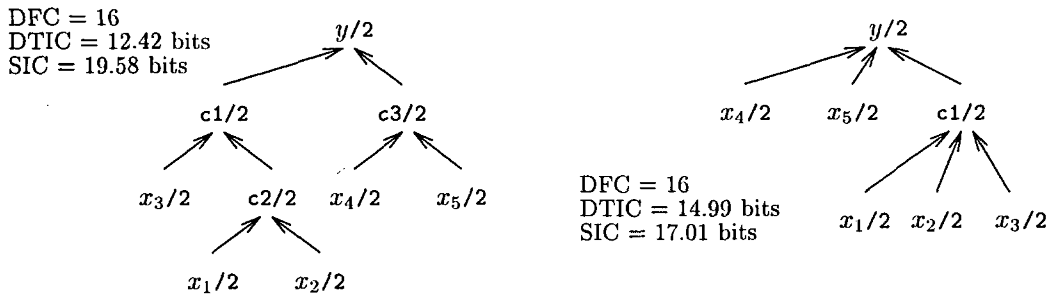


Figure 4: Decomposition of decision table representing the function $y = (x_1 \text{ OR } x_2) \text{ AND } x_3 \text{ AND } (x_4 \text{ XOR } x_5)$ guided by DTIC and CM (left), and DFC (right).

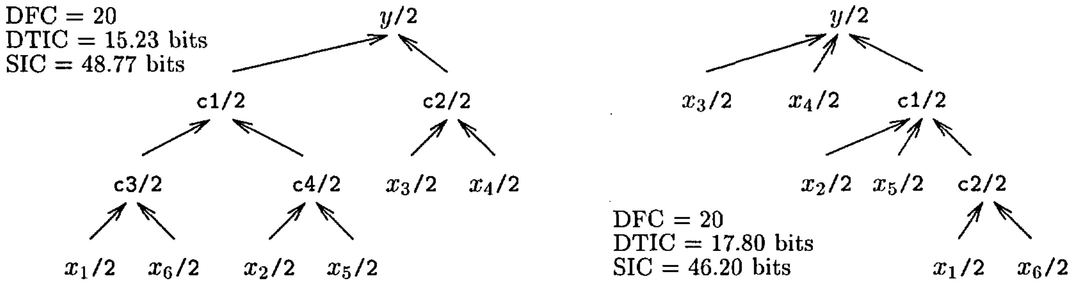


Figure 5: Decomposition of decision table representing the palindrome function guided by DTIC and CM (left), and DFC (right).

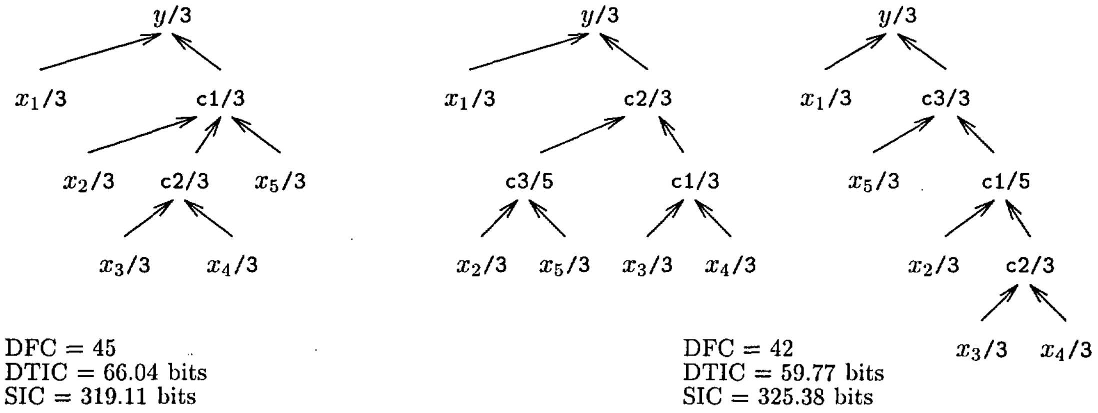


Figure 6: Decompositions of the function $y = \text{MIN}(x_1, \text{AVG}(x_2, \text{MAX}(x_3, x_4), x_5))$: the anticipated hierarchy (left), the hierarchy discovered using CM (middle), and DFC and DTIC (right). The complexity and information measures for the latter two decompositions are the same.

shown in the figure) were checked for interpretability and were found to represent the expected functions.

The second function $y = \mathrm{PAL}(x_1, x_2, \ldots, x_6)$ returns 1 if the string $x_1 \ldots x_6$ is a palindrome and returns 0 otherwise, i.e., $y = (x_1 = x_6)$ AND $(x_2 = x_5)$ AND $(x_3 = x_4)$. In the first experiment, six Boolean attributes $x_1 \ldots x_6$ were used. The initial decision table has $\theta_{\mathrm{DFC}} = 64$ and $\theta_{\mathrm{DTIC}} = 64$ bits. Again, the decomposition with DFC stops sooner and the domain favors the decomposition using CM and DTIC. However, for both this and previous case a DFC-guided decomposition could discover the expected hierarchy if the corresponding complexity condition would be changed to $\theta_{\mathrm{DFC}}(F) \geq \theta_{\mathrm{DFC}}(G) + \theta_{\mathrm{DFC}}(H)$. The same experiment was repeated with three-valued attributes $x_1 \ldots x_6$. This time, however, all three criteria lead to the same and anticipated concept hierarchy.

The third function $y = \mathrm{MIN}(x_1, \mathrm{AVG}(x_2, \mathrm{MAX}(x_3, x_4), x_5))$ uses ordinal attributes $x_1 \ldots x_5$ that can take the values 1, 2, and 3. While MIN and MAX have the standard interpretation, AVG computes the average of its arguments and rounds it to the closest integer. The initial decision table has $\theta_{\mathrm{DFC}} = 243$ and $\theta_{\mathrm{DTIC}} = 385.15$ bits. The anticipated and discovered hierarchies are shown in Figure 6. Quite surprisingly, in all three cases the decomposition yields a hierarchy with a higher structure information content than expected by introducing an additional five-valued intermediate concept. If this were removed, the discovered hirarchy and decision tables would have been the same as anticipated. It is also interesting to note that the hierarchy discovered using CM on one side and DFC or DTIC on the other are different but of the same complexity. This example illustrates that for a specific domain there may exist several optimal concept hierarchies with regard to complexity.

5.2 DEX models

An area where concept hierarchies have been used extensively is decision support. There, the problem is to select an option from a set of given options so that it best satisfies the aims or goals of the decision maker. DEX [5] is a multi-attribute decision support system that has been extensively used to solve real-world decision making problems. DEX uses categorical attributes and expects the concept structure and corresponding decision tables to be defined by the expert. The formalism used to describe the DEX model and its interpretation are essentially the same as with concept hierarchies studied in this article. This makes decision models developed by DEX ideal benchmarks for the evaluation of decision table decomposition. In this article, we use the following three DEX models:

CAR: A model for evaluating cars based on their price and technical characteristics. This simple model was developed for educational purposes and is described in [4].

EMPLOY: This is a simplified version of the models that were developed with DEX for a common problem of personnel management: selecting the best candidate for a particular job. While the realistic models that were practically used in several mid- to large-size companies in Ljubljana and Sarajevo consisted of more than 40 attributes, the simplified version uses only 7 attributes and 3 intermediate concepts and was presented in [6].

NURSERY: This model was developed in 1985 to rank applications for nursery schools [17]. It was used during several years when there was excessive enrollment to these schools in Ljubljana, and the rejected applications frequently needed an objective explanation. The final decision depended on three subproblems: (1) occupation of parents and child's nursery, (2) family structure and financial standing, and (3) social and health picture of the family.

The CAR and NURSERY datasets are available from the UCI Machine Learning Repository [16].

The goal of this experiment was to reconstruct these DEX models from examples. The learning instances were derived from the original models, where for all combinations of input attributes the class was determined by the corresponding model. The examples were stated as attribute-value vectors, hiding from the decomposition method any underlying conceptual structure of the domain.

The discovered hierarchies are given in Figures 7, 8, and 9. In all cases, the decomposition guided by DFC, DTIC, and CM found the same hierarchical structures and corresponding decision tables. Using DFC and DTIC, the order in which new intermediate concepts were found was the same but different to the one using CM. For example, in EMPLOY, DFC and DTIC-guided decomposition discovered `c1` first, while, using CM, this concept was discovered as the last one.

All the discovered hierarchies have higher information content than the original ones. Also, the overall complexity of decision tables is lower according to both DFC and DTIC. Most importantly, the discovered concept hierarchies are very similar to the original ones. In fact, if `c3` would be removed from CAR (making `c4` directly dependent on `lugboot`, `doors`, and `persons`), the two hierarchies would be the same. The same applies to EMPLOY and NURSERY if `c1` and `c2` are removed, respectively. In other words, the decomposition found the same concept hierarchies as the original ones but additionally decomposed the decision tables for `comfort` (CAR), `employ` (EMPLOY), and `struct+finan` (NURSERY). In this way it obtained less complex decision tables.

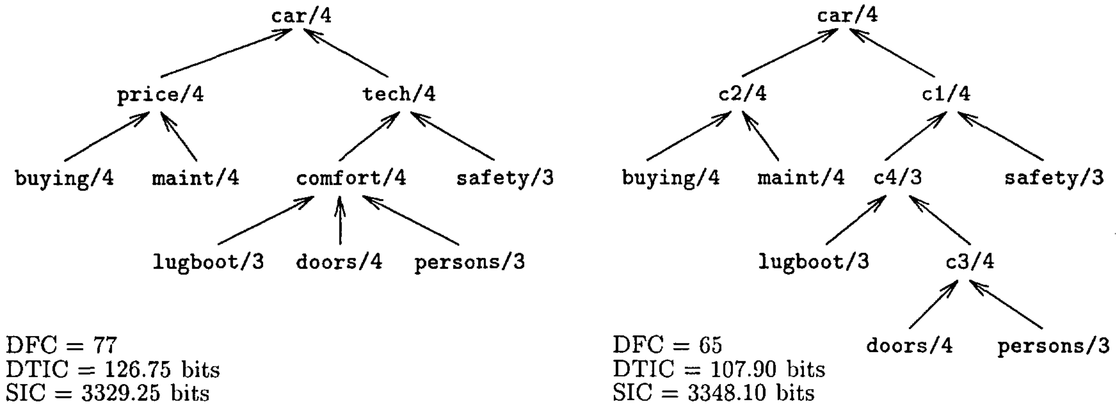


Figure 7: The original concept hierarchy of CAR (left) and the decompositions based on CM, DFC and DTIC (right).

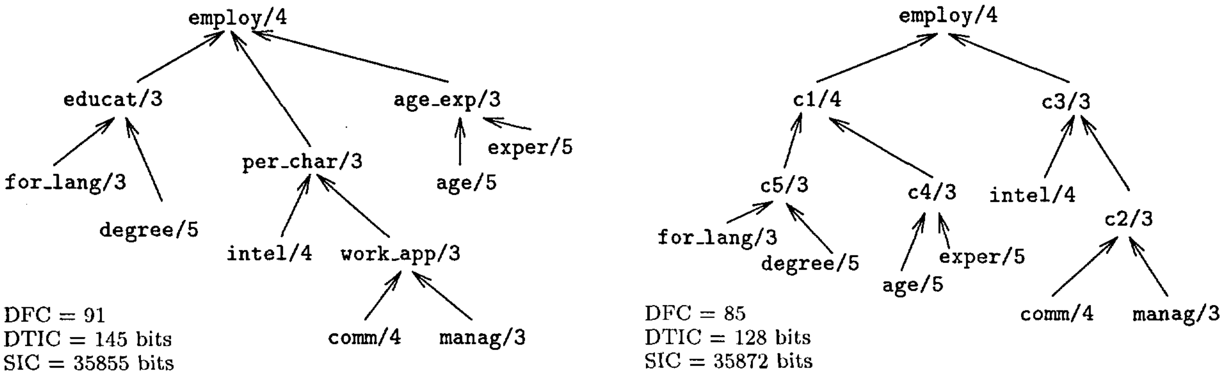


Figure 8: The original concept hierarchy of EMPLOY (left) compared to the hierarchy discovered by CM, DFC, and DTIC-guided decomposition (right).

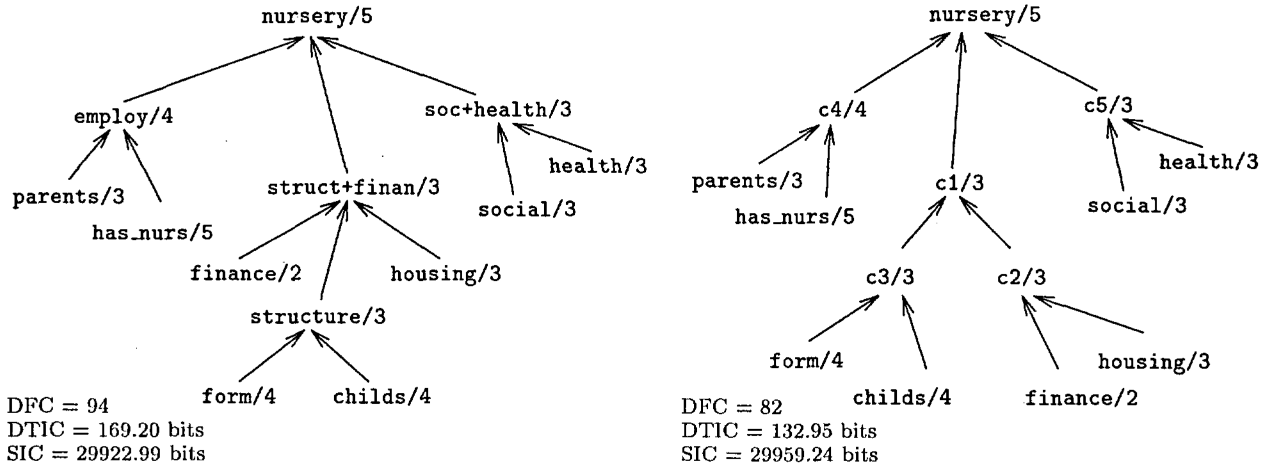


Figure 9: The original (left) and discovered concept hierarchy using CM, DFC and DTIC criteria (right) for NURSERY.

The derived decision tables were compared to the original ones and found to be the same but in the names used for instance labels (the decomposition uses abstract labels while the original decision tables use meaningful names). The only exception are decision tables for `tech` and `comfort` in the CAR domain, where the decomposition succeeded to find a more compact representation.

6 Conclusion

We investigated the appropriateness of three partition selection measures for decision table decomposition: decision table information content (DTIC) and column multiplicity (CM) introduced in this article, and decomposed function cardinality (DFC) that has already been used primarily for the decomposition of Boolean functions.

The experimental evaluation exposed the deficiency of DFC when decomposing a decision table that expresses a Boolean function. This may be alleviated by relaxing the DFC complexity condition. In more complex domains with multi-valued attributes, the decomposition guided by any of the proposed criteria discovered concept hierarchies that were very similar to those expected. Furthermore, the discovered hierarchies were equal to or even better than the anticipated ones in terms of the complexity of decision tables and structure information content. The order under which the intermediate concepts were discovered was the same for DFC and DTIC, but different for CM. A qualitative evaluation of derived hierarchies reveals that, in general, the discovered decision tables represent meaningful and interpretable concepts.

Although less complex in definition and easier to compute, DFC and CM both stand well in comparison with a more complex partition selection measure DTIC. Also comparable is the utility of DFC and DTIC to assess the complexity of the original and derived decision tables, although we have shown that DFC-based measure performed worse on two Boolean functions. Overall, while DFC and DTIC have better theoretical foundations than an intuitive partition selection measure CM, the experimental evaluation does not indicate that any of these is to be strictly preferred over the other.

The decision table decomposition was primarily developed for switching circuit design. However, experiments in non-trivial domains like DEX's strongly encourage further research and development of this method for machine learning and knowledge discovery. As the method has recently been extended to deal with continuous attributes [9] and noise [25], further research is needed to assess the quality of corresponding partition selection criteria under these extensions.

References

[1] Y. S. Abu-Mostafa. *Complexity in Information Theory.* Springer-Verlag, New York, 1988.

[2] R. L. Ashenhurst. The decomposition of switching functions. Technical report, Bell Laboratories BL-1(11), pages 541–602, 1952.

[3] A. W. Biermann, J. Fairfield, and T. Beres. Signature table systems and learning. *IEEE Trans. Syst. Man Cybern.*, 12(5):635–648, 1982.

[4] M. Bohanec and V. Rajkovič. Knowledge acquisition and explanation for multi-attribute decision making. In *8th Intl Workshop on Expert Systems and their Applications*, pages 59–78, Avignon, France, 1988.

[5] M. Bohanec and V. Rajkovič. DEX: An expert system shell for decision support. *Sistemica,* 1(1):145–157, 1990.

[6] M. Bohanec, B. Urh, and V. Rajkovič. Evaluating options by combined qualitative and quantitative methods. *Acta Psychologica,* 80:67–89, 1992.

[7] M. Bohanec, B. Zupan, I. Bratko, and B. Cestnik. A function decomposition method for development of hierarchical multi-attribute decision models. In *Proc. 4th Conference of the International Society for Decision Support Systems (ISDSS-97),* pages 503–514, Lausanne, Switzerland, July 1997.

[8] H. A. Curtis. *A New Approach to the Design of Switching Functions.* Van Nostrand, Princeton, N.J., 1962.

[9] J. Demšar, B. Zupan, M. Bohanec, and I. Bratko. Constructing intermediate concepts by decomposition of real functions. In M. van Someren and G. Widmer, editors, *Proc. European Conference on Machine Learning, ECML-97,* pages 93–107, Prague, April 1997. Springer.

[10] C. Files, R. Drechsler, and M. Perkowski. Functional decomposition of MVL functions using multi-valued decision diagrams. In *International Symposium on Multi-Valued Logic,* may 1997.

[11] J. A. Goldman. Pattern theoretic knowledge discovery. In *Proc. the Sixth Int'l IEEE Conference on Tools with AI,* 1994.

[12] T. Luba. Decomposition of multiple-valued functions. In *25th Intl. Symposium on Multiple-Valued Logic,* pages 256–261, Bloomigton, Indiana, May 1995.

[13] R. S. Michalski. A theory and methodology of inductive learning. In R. Michalski, J. Carbonnel, and T. Mitchell, editors, *Machine Learning:*

An Artificial Intelligence Approach, pages 83–134. Kaufmann, Paolo Alto, CA, 1983.

[14] R. S. Michalski. Understanding the nature of learning: Issues and research directions. In R. Michalski, J. Carbonnel, and T. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, pages 3–25. Kaufmann, Los Atlos, CA, 1986.

[15] D. Michie. Problem decomposition and the learning of skills. In N. Lavrač and S. Wrobel, editors, *Machine Learning: ECML-95*, Notes in Artificial Intelligence 912, pages 17–31. Springer-Verlag, 1995.

[16] P. M. Murphy and D. W. Aha. UCI Repository of machine learning databases [http://www.ics.uci.edu/~mlearn/mlrepository.html]. Irvine, CA: University of California, Department of Information and Computer Science, 1994.

[17] M. Olave, V. Rajkovič, and M. Bohanec. An application for admission in public school systems. In I. Th. M. Snellen, W. B. H. J. van de Donk, and J.-P. Baquiast, editors, *Expert Systems in Public Administration*, pages 145–160. Elsevier Science Publishers (North Holland), 1989.

[18] M. A. Perkowski et al. Unified approach to functional decompositions of switching functions. Technical report, Warsaw University of Technology and Eindhoven University of Technology, 1995.

[19] B. Pfahringer. Controlling constructive induction in CiPF. In F. Bergadano and L. De Raedt, editors, *Machine Learning: ECML-94*, pages 242–256. Springer-Verlag, 1994.

[20] H. Ragavan and L. Rendell. Lookahead feature construction for learning hard concepts. In *Proc. Tenth International Machine Learning Conference*, pages 252–259. Morgan Kaufman, 1993.

[21] T. D. Ross, M. J. Noviskey, D. A. Gadd, and J. A. Goldman. Pattern theoretic feature extraction and constructive induction. In *Proc. ML-COLT '94 Workshop on Constructive Induction and Change of Representation*, New Brunswick, New Jersey, July 1994.

[22] A. Samuel. Some studies in machine learning using the game of checkers II: Recent progress. *IBM J. Res. Develop.*, 11:601–617, 1967.

[23] A. D. Shapiro. *Structured induction in expert systems.* Turing Institute Press in association with Addison-Wesley Publishing Company, 1987.

[24] W. Wan and M. A. Perkowski. A new approach to the decomposition of incompletely specified functions based on graph-coloring and local transformations and its application to FPGA mapping. In *Proc. of the IEEE EURO-DAC '92*, pages 230–235, Hamburg, September 1992.

[25] B. Zupan. *Machine learning based on function decomposition.* PhD thesis, University of Ljubljana, April 1997. Available at http://www-ai.ijs.si/BlazZupan/papers.html.

[26] B. Zupan, M. Bohanec, I. Bratko, and J. Demšar. Machine learning by function decomposition. In Jr. D. H. Fisher, editor, *Proc. Fourteenth International Conference on Machine Learning (ICML-97)*, pages 421–429, San Mateo, CA, 1997. Morgan Kaufmann.

[27] B. Zupan, M. Bohanec, J. Demšar, and I. Bratko. Feature transformation by function decomposition. *IEEE Intelligent Systems & Their Applications*, 13(2):38–43, March/April 1998.

Dynamic Load Balancing for Object-Based Parallel Computations

Michele Di Santo
Università del Sannio, Benevento, Italy
Email: disanto@acm.org
AND
Franco Frattolillo
Università di Salerno, Fisciano (SA), Italy
AND
Wilma Russo
Università della Calabria, Rende (CS), Italy
AND
Eugenio Zimeo
Università di Napoli "Federico II", Napoli, Italy

Keywords: load balancing, object-oriented programming, parallelism, actors, Transputers

Edited by: Rudi Murn
Received: October 23, 1997 **Revised:** February 23, 1998 **Accepted:** March 3, 1998

Object-based parallel programming allows for the expression of ideal programs, which do not specify the mapping of objects to machine nodes. A parallel machine can efficiently execute ideal programs only if a runtime tool dynamically takes the appropriate placement decisions. This paper presents a new distributed adaptive load balancing algorithm, called PWF (Probabilistic Wave Front). It uses simple heuristics that guide the dynamic allocation of objects on the nodes of a parallel machine and their migration among the nodes. Experimental results show that PWF constantly outperforms both the random algorithm and the ACWN (Adaptive Contracting Within Neighborhood) one and therefore succeeds in accurately placing objects on the nodes of a parallel system.

1 Introduction

While technological factors are making parallel computers more and more cost-effective and are imposing a common architectural organization made by a collection of *nodes* (processor-memory pairs) connected by a communication network [6], developing efficient and portable parallel programs is still hard [4]. In fact, programmers are still forced to use rather low-level programming models and languages and to explicitly manage computational resources.

In the search for a solution to these problems, the use of the *object-based paradigm* [17] has stirred the interest of the parallel computing community. In fact, it combines well with parallelism, since their logical autonomy makes objects a natural unit for parallel execution [2, 11], and allows for the expression of *ideal*, i.e. architecture-independent, parallel algorithms.

In order to automatically and efficiently map an ideal algorithm onto an architecture, an appropriate object placement policy is needed. It must specify both where to allocate each new object and if and how to redistribute the already allocated ones.[1] Because of the extreme flexibility offered by dynamic creation and interconnection of objects, it is very difficult to statically predict the shapes and the extents of the structures to which the computation will give rise at runtime, and so to give an automatic, static solution to the problem of devising an efficient object placement policy aimed at minimizing the total execution time of a parallel program. Therefore, two possible approaches are:

- To explicitly program, for each group of objects in the application, a partition and distribution strategy (PDS). In this case, reusability and scalability are greatly enhanced by adopting methodologies for modular specification of PDSs. [12].
- To use an automatic *placement tool* that dynamically decides where to allocate each new object and if and how to redistribute the already allocated ones.

This paper examines the problem of the automatic

[1] This research was supported in part by the Italian Organization for University and for Scientific and Technological Research (M.U.R.S.T.) under grants "60%" and "40%".

[1] Generally, efficiency depends also on the scheduling policy that each node adopts in selecting the next object ready to run, but we disregard this dependency here.

placement of objects and proposes the PWF (*Probabilistic Wave Front*) algorithm, a new distributed adaptive load balancing algorithm based on some simple heuristics that guide the dynamic allocation of objects on the nodes of a parallel machine and their migration among the nodes. In order to verify the effectiveness of the proposal, we included the PWF, the ACWN [14, 15] and the random load balancing algorithms in an Actor [1] programming environment running on a Transputer network. Experimental results show that the PWF algorithm constantly outperforms the other two and therefore succeeds in accurately placing objects on the nodes of our parallel system.

In the programming model adopted, objects, which unify both data and code in a local state, are dynamically created and referred through system-wide identifiers. They manifest a pure reactive nature and interact with other objects only via message passing. The communication mechanism is point-to-point, asynchronous and one-directional. Messages are eventually delivered to their destinations, but transmission order is not necessarily preserved at delivery. Unbounded queues associated to receiving objects buffer incoming messages, before they are serially processed. Functional interactions among objects are modeled with the use of continuations.

The structure of the article is as follows. In the following two sections, we describe a framework for dynamic placement of objects and then present and discuss the PWF algorithm. In the last three sections, we describe how to tune the algorithm for obtaining the best performances, illustrate a set of experimental results that prove the effectiveness of our proposal and present the conclusions.

2 A framework for dynamic placement of objects

The study of provably efficient on line scheduling algorithms for parallel programs whose computations are revealed only at runtime is still in its infancy and some theoretical results are available only for a few kinds of applications and for specific computational models [3].

Therefore, in order to achieve good speedups, existing object-based programming environments pragmatically adopt *dynamic placement algorithms* based on heuristics that essentially try to satisfy the two goals of load balance and locality. *Load balance* guarantees that, at each moment during the computation, all the nodes of the machine have sufficient work to do. Instead, *locality* reduces network traffic, by decreasing the distance between data and the node where it is needed. Unfortunately, these goals are in conflict, in that load balance benefits from the uniform distribution of objects across the network, while locality is favored by the concentration of objects on a few nearby nodes.[2]

Information collected by dynamic placement algorithms is very often limited to load information and so these algorithms are generally referred to as *dynamic load balancing algorithms* (DLBAs), even if locality concerns are to some extent taken into account.

In general, among all the balancing algorithms cited in the literature, the *distributed adaptive* ones (DALBAs) give better chances of achieving good performance and scalability [5, 10, 13, 9]. These algorithms, in the context of object-based computations, run on each machine node in order to execute the following activities:

1. updating local load and state;
2. exchanging with other nodes load balancing messages (*LBM*s) derived from local states;
3. choosing the node where to allocate a new object;
4. deciding if and how to redistribute some of the already allocated objects.

In order to set up a framework for DALBAs in the context of object-based programming models, we spend a few words on each activity and on the main strategies that each one can adopt.

Updating local load and state. The basic activity performed by a DALBA running on a node is to evaluate the local load. Because messages exchanged among objects are the driving force, a good measure of the current load may be the number of "serviceable" messages waiting to be processed on the node. This measure is sufficiently accurate when all the messages in the computation have near equal elaboration times, as it is often the case in applications characterized by rather small-grained objects.

Another DALBA activity is to handle local state, which often includes both data derived from received *LBM*s and some adaptive indices and thresholds. Therefore, local state allows each DALBA to relate local load with the load of other nodes and to adapt its activities and strategies to the changing load conditions in the system.

Exchanging information. Nodes exchange information in the form of *LBM*s which can be communicated either periodically or when load changes by prefixed amounts. The latter solution avoids exchanging useless information, but, in any case, the amounts should be tuned for the specific application in order to realize a trade-off between communication costs and accuracy of exchanged information.

[2] Even if in modern interconnection networks latencies are relatively insensitive to distance, many long distance messages may result in link contention and consequently degrade network throughput.

Load in a *LBM* can be specified either as an absolute value or as an estimate expressed as a value in a finite number of alternatives, such as *light*, *moderate*, or *heavy*. The latter solution is to be preferred because it lets each sending node evaluate its own load condition.

In order to assure scalability to DALBAs, each node has to exchange information only with a subset of the other ones. In the case of multicomputers and geographically or hierarchically distributed systems, this subset can coincide with the physical neighbors of the node, so contributing to reduce network traffic. Instead, in the case of a fully connected distributed system, it is necessary to adopt a *node-grouping strategy*, in order to determine for each node both a receiving set and a sending one. Each node sends *LBM*s only to the members of its *sending set* and receives *LBM*s only from the members of its *receiving set*. The node-grouping strategy must satisfy some minimal requirements [16]:

- sets must be "reasonably" small and of similar size;

- for each pair of nodes a and b, b must be "reachable" from a, i.e. either a and b must belong to the same receiving set or a node x must exist, such as x is reachable from a and b is reachable from x.

This last requirement guarantees that, if necessary, a creation or migration request that originates from a node can reach any other node.

In the following, we say that node a is a "neighbor" of node b if either a is a physical neighbor of b or a belongs to the receiving set of b.

Taking the allocation decision. Whatever the allocation strategy adopted, allocation must be guaranteed to take place in a finite time. This can be assured by limiting to some maximum value the number of "hops" traveled by the allocation request. In particular, a small maximum value promotes locality together with fast and low-cost object creation, but may prevent quick load spreading and so cause a loss of efficiency at the beginning of the computation.

Reducing the maximum number of hops to zero corresponds to adopting a purely local allocation strategy and so to using the only redistribution mechanism in order to gain load balancing. This is generally inappropriate, in that redistribution induces high overheads. Instead, a strategy which allocates objects on the basis of local state of nodes should be adopted, in order to reduce the probability of redistributing objects and so increasing the efficiency of balancing.

Taking the migration decision. Object redistribution is necessary when, despite a good initial allocation, some nodes move towards light load conditions. In migrating objects, a DALBA can adopt a *sender initiated* strategy or a *receiver initiated* one or a mix of them [13, 18]. Anyway, in order to preserve locality, an object should not be migrated many times and, in order to reduce communication overhead, the amount of load to move should be obtained from a small number of objects and by minimizing the total number of transferred bytes.

3 The PWF Algorithm

The PWF (*Probabilistic Wave Front*) algorithm is a new DALBA, based on the framework set up in the previous section, and applicable to object-based, parallel programming environments running on systems made up of a number of nodes communicating only by means of message passing. The only assumptions about the communication network are that message passing is reliable and, for each node, a set of neighboring nodes is defined in such a way as to satisfy the minimal requirements stated in the previous section. Messages need not arrive in the same order they are sent, but, if this happens among neighbors, the performance of PWF improves.

The name PWF has been chosen in order to make explicit the two main characteristics of the algorithm:

- Objects diffuse through the system according to the simple rule that the number of hops traveled by each request of creation or migration is at most one.

- The candidate node on which to create a new object, is firstly selected by a simple round-robin strategy among the neighbors of the node where the request occurs, and then passes a validation step, based on a probability value that depends on its load level.

The PWF algorithm is based on the following main assumptions:

- Each node "sees" only a subset of the other nodes (its neighbors) and so stores load information only relative to these nodes and exchanges *LBM*s and requests of creation and migration only with these nodes.

- Local load is measured as the number of serviceable messages waiting to be processed on the node.

- Load in *LBM*s is expressed as a value in a finite number of alternatives, derived from the actual load value by using some appropriate thresholds.

- Some of the thresholds are adaptively modified.

- The load of a node is known to each of its neighbors only indirectly through a probability value.

Symbols	Meanings
NS	Number of nodes in the system
0..NS-1	Identifiers of nodes in the system
thisNode	Identifier of the node running the PWF algorithm
N	Set of identifiers of thisNode neighbors, totally ordered
N'	Set N ∪ {thisNode}, totally ordered
load	Current load value of thisNode (initialized to 0)
idle, light, medium, heavy	Load levels of a node
level	Current load level of thisNode (initialized to light)
IT	idle level threshold (level = idle iff load $\leq$ IT)

LT, HT	Current light and heavy level thresholds
	(level = light iff IT < load $\leq$ LT
	level = medium iff LT < load $\leq$ HT
	level = heavy iff load > HT)

LT_0, HT_0	Initial values of LT and HT (received by PWF at the beginning of computation)
ΔT	Adaptive increase of LT and HT (received by PWF at the beginning of computation)
last	The node in N' where the last creation was made (initialized to anyone of N values)
succ(n)	$\forall n \in N'$, returns the node immediately following n in N', if it exists; the first node in N', otherwise
MP	Probability value for a medium load level ($0 < MP < 1$) (received by PWF at the beginning of computation)
P_n	$\forall n \in N$, current probability value of node n, equal to: 0 if n is known to have a heavy load level; 1 if n is known to have a light or idle load level; MP if n is known to have a medium load level
$P_{thisNode}$	Current probability value of thisNode, equal to: 0 if $P_n = 1, \forall n \in N$; 1 otherwise
f	A real value in [0, 1], which determines the maximum fraction of neighbors to involve in a migration (received by PWF at the beginning of computation)
M	Set of neighbors to involve in a migration $(M \subseteq \{n : n \in N \wedge P_n \neq 1\}) \wedge (\mid M \mid \leq \lfloor f \times \mid N \mid \rfloor)$
random()	Returns a random real value in [0, 1[
send msg to S	Sends the message msg to all the nodes in the set S
migrate ΔL to n	Migrates to node n a load at most equal to ΔL

Table 1: Symbols and their meanings.

Moreover, on each node, a further probability value synthesizes the aggregate load of neighbors.

```
updateLoad(dl):
    $load \leftarrow load + dl$
    if $load > HT$ then
        if $level \neq heavy$ then
            $level \leftarrow heavy$
            send $(thisNode, heavy)$ to $N$
    elsif $load > LT$ then
        if $level \neq medium$ then
            $level \leftarrow medium$
            send $(thisNode, medium)$ to $N$
    elsif $load > IT$ then
        if $level > light$ then
            $level \leftarrow light$
            send $(thisNode, light)$ to $N$
    elsif $level = light$ then
        $level \leftarrow idle$
        send $(thisNode, idle, |M|)$ to $M$
```

Figure 1: The updateLoad component.

In detail, the PWF algorithm consists of the three components updateLoad, handleLBM, and selectedNode, which are executed on each node of the system. They are respectively described in Figures 1, 2, and 3. The symbols adopted and their meanings are described in Table 1.

updateLoad(dl) runs each time the node changes its load by a quantity dl, which can be either positive (a new message is received by an object residing on the node or an old message becomes serviceable) or negative (a message is processed by an object on the node or some object is migrated or a message becomes temporarily unserviceable). It is up to this component to notify each load level variation to all the neighbors, by sending them appropriate *LBMs* consisting of two fields: the identity of the sender node and its new load level. Only in the case of a light to idle transition, when migrations are to be activated, the *LBMs* include, as a third field, the cardinality of M, to be used by the target nodes in order to evaluate the amount ΔL of

```
handleLBM(sender, newLevel, m):
  case $newLevel$ of
    $heavy$:   $P_{sender} \leftarrow 0$
               $P_{thisNode} \leftarrow 1$
               if $P_n = 0 \ \forall n \in N$ then
                   $HT \leftarrow HT_0 + \Delta T$
                   $LT \leftarrow LT_0 + \Delta T$
    $medium$:  $P_{sender} \leftarrow MP$
               $P_{thisNode} \leftarrow 1$
               $HT \leftarrow HT_0$
               $LT \leftarrow LT_0$
    $light$:   $P_{sender} \leftarrow 1$
               if $P_n = 1 \ \forall n \in N$ then
                   $P_{thisNode} \leftarrow 0$
               $HT \leftarrow HT_0$
               $LT \leftarrow LT_0$
    $idle$:    $P_{sender} \leftarrow 1$
               if $P_n = 1 \ \forall n \in N$ then
                   $P_{thisNode} \leftarrow 0$
               $HT \leftarrow HT_0$
               $LT \leftarrow LT_0$
               if $load > LT_0$ then
                   $\Delta L \leftarrow min(\lceil (LT_0 - IT)/m \rceil,$
                                   $load - LT_0)$
                   $migrate\ \Delta L\ to\ sender$
```

Figure 2: The handleLBM component.

```
selectedNode():
    repeat
        $last \leftarrow succ(last)$
    until $random() < P_{last}$
    returns $last$
```

Figure 3: The selectedNode component.

load to be migrated. In this case, notification is limited to a set M containing at most a fraction f of heavy or medium load neighbors; obviously, in forming M, heavily loaded neighbors are to be preferred.

handleLBM(sender, newLevel, m) runs each time the node receives an *LBM* from one of its neighbors. This component updates probability values and the thresholds LT and HT; moreover, when required, it commands object migrations. As migrations tend to elevate the load of the idle node to LT_0, by knowing that m neighbors are involved in the migration, the amount of load ΔL to be migrated is evaluated as the m-th part of LT_0-IT. Obviously, if necessary, this amount is reduced in order to guarantee a load level on the node at least equal to medium. Therefore, a migration can never cause a node to move to a light or idle load level.

selectedNode() runs each time a request to create a new object rises on the node and consists of two possibly repeated steps:

- a candidate node in the set N' is selected, according to a round-robin strategy;
- the candidate node is validated if a randomly generated real number in the range [0, 1[is lesser than the current probability of the node.

It must be noted that, according to the probability values assigned to nodes:

- a node is selected in a number of steps at most equal to the cardinality of N';
- a heavily loaded neighbor is never validated;
- a lightly loaded neighbor is always validated;
- a medium loaded neighbor is validated with probability MP;
- a creation is always remote, as long as all the neighbors are lightly loaded;
- a creation is always local, as long as all the neighbors are heavily loaded.

The strategies adopted in the PWF algorithm are at the same time simple, and therefore efficiently implementable, and effective in guaranteeing the accuracy of balancing.

As to simplicity, it is worth noting that:

- Measuring the load as the number of serviceable messages waiting to be processed on the node requires only counting.
- Nodes involved in communications are always neighbors.
- *LBM*s are sent only when transitions in load levels occur. The many *LBM*s generated by the oscillations of load around a threshold can be avoided by an "hysteresis mechanism" that splits each threshold into two, with the lower one to be used when load decreases and the upper one when load increases.
- Allocation of a new object is guaranteed to take place after at most one delegation, so that each object is created either locally or onto a neighbor of the node where the creation request arises.

Instead, accuracy of balancing is assured by the following algorithm behaviors:

- Both non-local allocation and redistribution of objects are pursued. The adopted allocation strategy significantly reduces the probability of expensive migrations, which remain however necessary to contrast both residual load imbalances, typical of highly dynamic computations, and structural ones occurring in the final phases of computations.
- Initially, when nodes are not loaded yet, allocation is mainly controlled by the round-robin selection strategy, so guaranteeing a quick load spreading. Later, when nodes tend to be more heavily loaded, allocation becomes mainly controlled by the validation step, so assuring a more accurate selection of the destination.
- Adaptively incrementing the thresholds LT and HT of a node, when all its neighbors are heavily loaded, delays the transition of the node towards higher load levels, so enabling it to receive further work from its neighbors.

4 Choosing parameters

The performance of the PWF algorithm depends on the values chosen for the parameters $\mathsf{LT_0}$, $\mathsf{HT_0}$, MP, IT, ΔT, and f.

Load distribution is essentially controlled by the values of the thresholds $\mathsf{LT_0}$ and $\mathsf{HT_0}$ that should be chosen according to the expected maximum load per node *EML*. Practically, this value can be approximated by MML_{RANDOM}, the average of the maximum loads measured on each node during a preliminary run which makes use of the random balancing strategy, chosen in order to eliminate any dependence from balancing parameters. Our experience shows that the best performances are obtained when $\mathsf{LT_0}$ and $\mathsf{HT_0}$ are respectively set to the 40% and 80% of *EML*. In particular:

- lower $\mathsf{LT_0}$ values may prevent a quick load spreading, so determining imbalances and consequently inducing expensive load redistribution;
- higher values may cause objects to be spread out even when it is not actually necessary, by giving rise to both a wasteful communication overhead and a loss of computation locality.

Analogous considerations can be made for the threshold $\mathsf{HT_0}$. In fact, lower values may induce load imbalances, because objects tend to be locally created, while higher values may cause object creations on heavily loaded nodes.

Even if MP may assume values in the range]0, 1[, its value should be about $\frac{1}{2}$. In fact, the choice of an extreme value in the range corresponds to reduce the two thresholds $\mathsf{LT_0}$ and $\mathsf{HT_0}$ to only one. In particular, MP=0 corresponds to set $\mathsf{HT_0}$ equal to $\mathsf{LT_0}$, while MP=1 corresponds to set $\mathsf{LT_0}$ equal to $\mathsf{HT_0}$. Moreover, in order to choose an accurate value of MP, both the kind of computation and the connectivity level of the network should be taken into account. In particular, if the object creation activity is rather evenly distributed among nodes and the average number of neighbors in the system is high, it is likely that a medium loaded node will move towards a heavy load condition. In this case, in order to exploit computation locality and limit the overhead due to object spreading, MP should assume a value lesser than $\frac{1}{2}$. Conversely, if the object creation activity is limited to a few nodes and the average number of neighbors in the system is low, it is likely that a medium loaded node will move towards a light load condition. In this case, in order to favor object spreading, MP should be set to a value higher than $\frac{1}{2}$. Anyway, our experience shows that the best results are obtained with MP values ranging from 0.4 and 0.6.

The value of the IT threshold determines when a node that is going to become idle requests work from some of its neighbors. Therefore, too low a value could excessively delay redistribution, so as not to impede the node from becoming idle. On the contrary, too high a value could cause an anticipated and useless redistribution, so inducing a wasteful overhead.

As already said, the adaptive increase of the thresholds LT and HT by the quantity ΔT, when neighbors are all heavily loaded, aims at delaying node transition towards higher load levels, so enabling it to receive further work from its neighbors. Therefore, if ΔT is set to a too low value, the delay effect is negligible. On the contrary, too high a value may cause imbalances because of the out to date load information kept on neighbors.

The value of f determines the number of neighbors to involve in a migration. It should be set according to the kind of computation. In fact, in a highly dynamic computation, there may be load imbalances among nodes and a node is likely to be surrounded by neighbors in very different load conditions. In such a situation, f should be set to a low value, so that the amount of load to be migrated to an idle node tends to be provided only by the most loaded neighbors. On the contrary, in a computation characterized by rather regular load conditions, an idle node is likely to be surrounded by lightly loaded neighbors. Therefore, f should be set to a high value, so that the amount of load to be migrated to an idle node tends to be provided by a greater number of lightly loaded neighbors.

5 Experimental results

In order to prove the effectiveness of our proposal, we have compared the PWF algorithm with the random and the ACWN (*Adaptive Contracting Within Neighborhood*) ones [14, 15]. These algorithms were chosen for the following reasons:

- The random algorithm achieves quite a uniform load distribution with a minimum exploitation of system resources, but neither it assures any locality to the computation, nor is it adaptive.
- The ACWN algorithm gets a good load distribution, assures a high locality to the computation, and is adaptive, but it induces some communication and computing overheads.

The three load balancing algorithms have been integrated into ASK (Actor System Kernel), the runtime support of AL++ [7], a semantic extension of C++, implemented through a class library which provides an object-oriented interface for Actor programming. The prototype implementation of ASK has been developed in the 3L Parallel C programming language. It runs on an INMOS system which consists of a network of sixteen T800 Transputers, clocked at 20 MHz, with links at 20 Mbits/s, and each equipped with 1 Mbyte of RAM with two wait states. A PC acts as a host system and I/O server. ASK runs at the top of

Algorithm **N-QUEENS** *Board size:* 9 × 9	Number of columns searched by each object			
	1	2	4	6
Average number n of objects created on each node	503	244	102	65
Random	9	10.8	16.1	22.2
Standard deviation σ_n *ACWN*	13.5	15.2	16.7	18.1
PWF	11.4	12.3	13	13.9
Average number m of messages processed on each node	525	266	124	87
Average processing time *grain* for each message g (*msec*)	1.1	1.8	3.6	6.0
Standard deviation σ_g (*msec*)	0.8	0.9	2.1	3.7
Minimum grain g_{min} (*msec*)	1.1	1.3	1.3	1.3
Maximum grain g_{max} (*msec*)	1.9	2.8	6.1	10.5

Table 3: Main characteristics of *n-queens*.

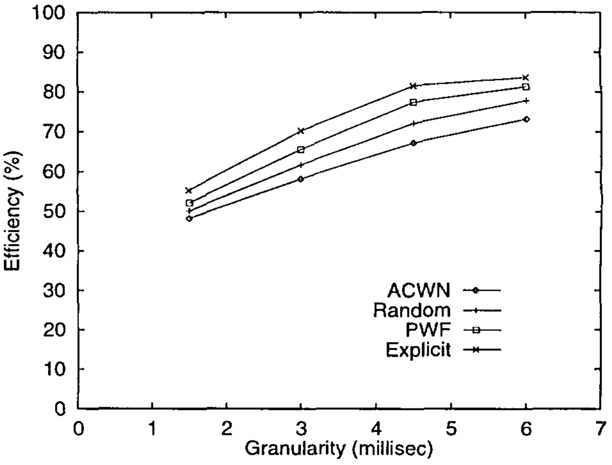


Figure 4: Efficiency of *range-add*.

a low-level network environment that provides node-to-node asynchronous communication and routing between non-adjacent nodes, for both ring and 2D Torus topologies.

Here we show the experimental results obtained on three sample programs, characterized by different computation structures and communication patterns, which stress in different ways the dynamic properties of the balancing algorithms tested:

- *Range-add*, which uses a "divide-and-conquer" strategy in order to compute in parallel the sum of all the integers in the range between 0 and 10 millions. The computation is characterized by a binary tree structure, where each leaf object adds the numbers in the received range and passes on the sum to its parent, while each internal object splits the received range into two, passes on them to two new created objects, receives back the two sums, combines them and passes on the result to its parent.

- *N-queens*, which realizes a concurrent search of all the solutions to the problem of placing n queens on an $n \times n$ chessboard in such a way that no queen may be taken by any other queen. The computation is characterized by a highly dynamic structure whose shape cannot be predicted at compile-time. Each search object receives a chessboard with a partial solution, i.e. i queens safely placed on the first i columns, and tries to extend it by finding all the safe positions on the $(i+1)$-th column. Whenever a safe position is found, the new partial solution is passed to a new search object that tries to extend it further.

- *Tsp*, which generates a solution of the *traveling salesman problem*, by finding a "minimum-distance" route a sales representative may follow in order to visit each city in a given list exactly once. The computation has a tree structure whose size cannot be predicted at compile-time. Searching starts by creating *n-1* objects and proceeds in parallel according to a "branch and bound" scheme. Each object receives a different partial route, extends it by adding one city, evaluates the new partial distance and, only if it is less than the one stored in a *minimum* object, passes it to a new object. This action continues repeatedly, until the number of cities to add to partial routes is equal to a value fixed at computation start up. Then, objects complete the received partial routes with all the remaining cities.

Tables 2, 3, and 4 summarize the main characteristics concerning the executions of these programs. In particular, each table reports, for a given problem size and different grain sizes, the following features:

- object distribution, i.e. the average number (n) of objects created on each node and, for each balancing algorithm, the standard deviation of n (σ_n);

- the average number (m) of messages processed on each node;

- the minimum (g_{min}), maximum (g_{max}), and average (g) message processing times (grain), as well as the standard deviation of g (σ_g).

Instead, Figures 4, 5, and 6 show, for each balancing strategy, the efficiencies, expressed in percentages, gained by the tested algorithms for different grain sizes. These results are the best ones we were able to obtain by setting, for each application, the parameters of the PWF and ACWN algorithms.

Efficiency is computed as the ratio of the real speedup to the number of processing nodes. The real speedup is the ratio of the time needed by the best serial algorithm running on a single node of our machine to the time needed by the parallel algorithm running

Algorithm	Number of sums carried out by each leaf object			
RANGE-ADD *Total number of operations: 10,000,000*	2,500	5,000	7,500	10,000
Average number n of objects created on each node	500	250	166	125
Standard deviation σ_n *Random*	8.9	10.3	12.8	14.2
ACWN	17.2	18.3	19.7	21.1
PWF	13	14.4	15.8	16.5
Average number m of messages processed on each node	1000	500	332	250
Average processing time *grain* for each message g (*msec*)	1.5	3.0	4.5	6.0
Standard deviation σ_g (*msec*)	2.6	5.2	7.8	10.4
Minimum grain g_{min} (*msec*)	0.001	0.001	0.001	0.001
Maximum grain g_{max} (*msec*)	6	12	18	24

Table 2: Main characteristics of *range-add*.

Algorithm	Number of cities a leaf object tries to add to a partial route			
TSP *Number of cities: 8*	1	2	3	4
Average number n of objects created on each node	428	271	113	34
Standard deviation σ_n *Random*	9.3	10.1	14.4	23.1
ACWN	16.1	16.9	17.3	18.8
PWF	12.3	13.9	14.4	15.2
Average number m of messages processed on each node	1,284	813	339	102
Average processing time *grain* for each message g (*msec*)	0.1	0.2	0.3	0.6
Standard deviation σ_g (*msec*)	0.2	0.3	0.4	1.6
Minimum grain g_{min} (*msec*)	0.01	0.01	0.01	0.01
Maximum grain g_{max} (*msec*)	0.4	0.8	1.6	6.0

Table 4: Main characteristics of *tsp*.

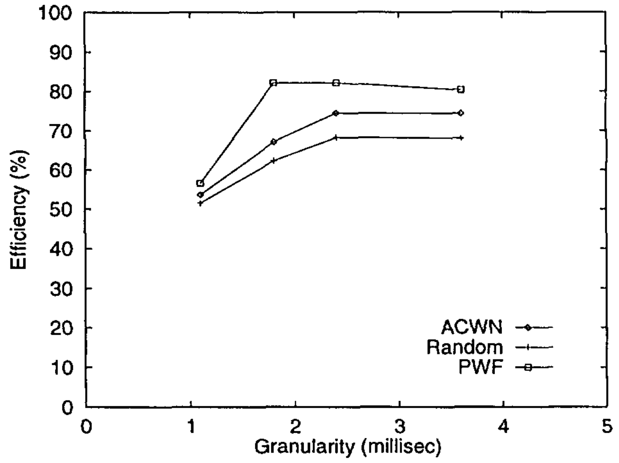


Figure 5: Efficiency of *n-queens*.

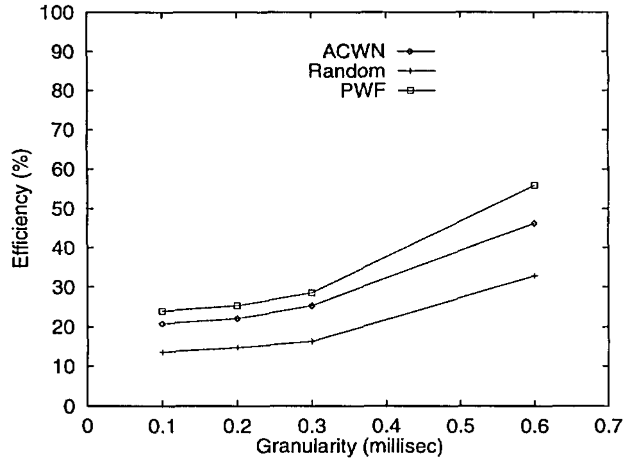


Figure 6: Efficiency of *tsp*.

on all the available nodes. Figure 4 also shows the efficiency gained by the *range-add* algorithm when a programmed placement strategy, which is both optimized for the specific problem and parametric with respect to the input size, is adopted. The results achieved by employing this strategy can be regarded as an upper limit to efficiency, since they are obtained with optimal balancing conditions and without any overhead. The low levels of efficiency shown in Figure 6 are to be attributed to the characteristics of the runtime kernel that is not able to adequately support fine-grained computations.

Moreover, Figures 7, 8 and 9 show, for a particular execution of each algorithm, how efficiency depends on the adopted values of LT_0, HT_0 and MP. The other balancing parameters have the following values: IT=2, ΔT=0.1×HT_0 and f=1.

Tables 5, 6 and 7 also report the *MML* values and their standard deviations for the three balancing algorithms used.

The experimental results show that PWF constantly outperforms both the random algorithm and the ACWN one.

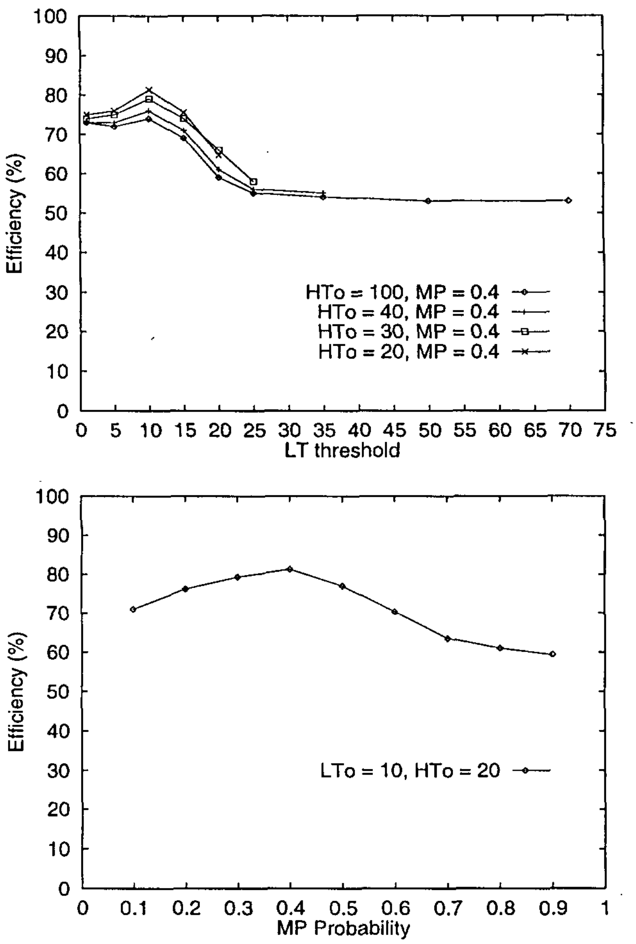


Figure 7: Efficiency of *range-add* versus LT_0, HT_0 and MP.

10,000 sums for each leaf object	*MML*	σ_{MML}
RANDOM	24.8	7.4
ACWN	36.1	11.3
PWF (LT_0=10,HT_0=20,MP=0.6)	20.9	3.6

Table 5: *MML* values and their standard deviations for the execution referred in Figure 7.

3 columns searched by each object	*MML*	σ_{MML}
RANDOM	12.4	6.1
ACWN	10.9	3.3
PWF (LT_0=5,HT_0=10,MP=0.5)	8.7	2.1

Table 6: *MML* values and their standard deviations for the execution referred in Figure 8.

For the *range-add* program, the results can be explained by observing that:

- Computation is characterized by few data communications, so that locality is not very impor-

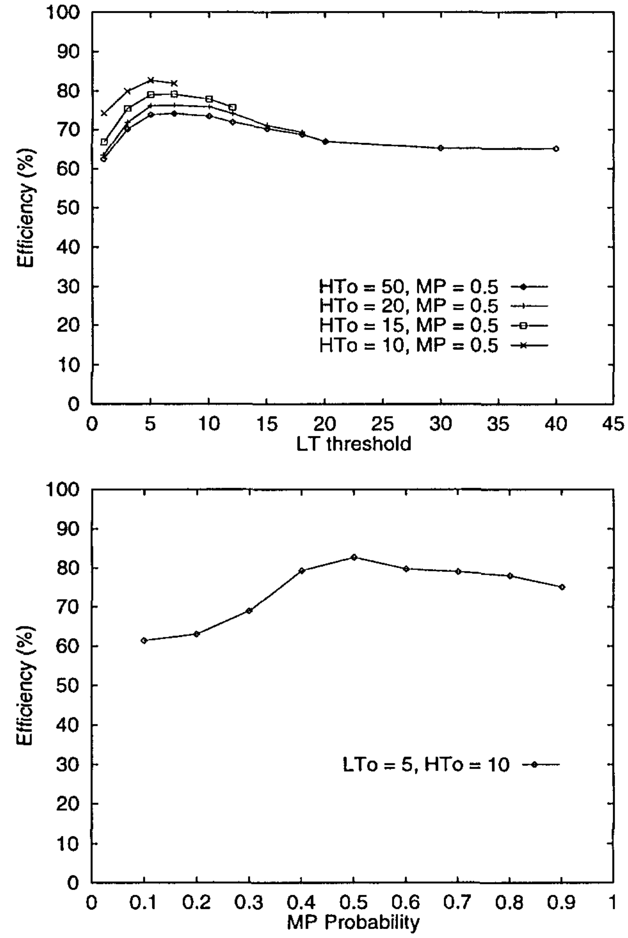


Figure 8: Efficiency of *n-queens* versus LT_0, HT_0 and MP.

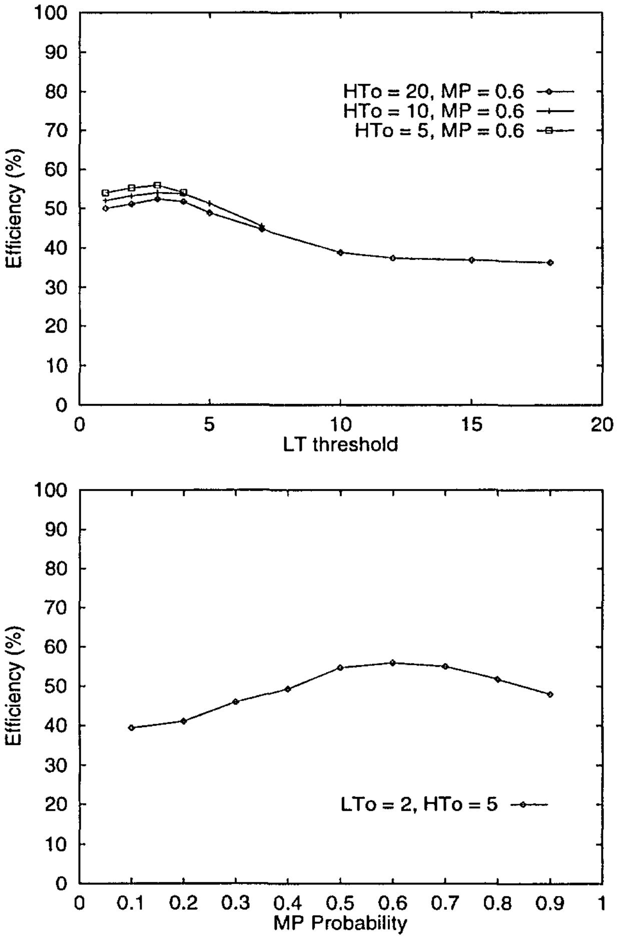


Figure 9: Efficiency of *tsp* versus LT_0, HT_0 and MP.

tant and therefore even the random strategy can obtain satisfactory results.

- An even load distribution is important, particularly in the final phase of the computation when the partial results are collected towards the root of the computation tree. Such a distribution should be preferably obtained by a shrewd initial placement of objects, because redistribution can induce a high overhead owing to the fine grain of the computation. Therefore, in the end, PWF outperforms both its two competitors thanks to its more accurate load distribution.

4 cities for each leaf object	*MML*	σ_{MML}
RANDOM	7.6	3.2
ACWN	6.9	1.8
PWF (LT_0=2,HT_0=5,MP=0.6)	4.5	0.8

Table 7: *MML* values and their standard deviations for the execution referred in Figure 9.

For the *n-queens* program, the results can be explained by observing that:

- Computation is highly dynamic, so that keeping well-balanced load conditions during the whole execution is strategic. Therefore, the random strategy, which is not able to react to load unbalances, does not achieve satisfactory results, even if computation prevails over communication and so locality is not very important.

- The ACWN algorithm gets a poorer performance than the PWF one because it is penalized by higher computation and communication overheads essentially due to the traveling of object creation requests before being accepted for execution.

In the *tsp* program, the computation is fine-grained and is characterized by many communications and by a quickly variable structure at runtime. Therefore, during the whole computation, it is important to maintain both computation locality and an even load distribution. Consequently:

- The random algorithm performs worse than the other two because it is neither adaptive, nor does it assure any computation locality.

- The PWF algorithm prevails over the ACWN one because it induces a lower overhead and carries out cheaper object redistribution.

6 Conclusion

The proposed PWF algorithm is based on simple heuristics and is applicable to object-based, parallel programming environments running on distributed memory architectures. The main characteristics of the algorithm are:

1. It is fully distributed and scalable with the number of nodes in the system.

2. It exploits limited load information on each node that comes only from neighbors.

3. It is able both to promote an even load distribution and to exploit computation locality, without inducing a great use of computing resources.

The experimental results are encouraging in that they show that the PWF algorithm behaves better than the ACWN algorithm, which is one of the few DALBAs explicitly designed for object-based environments.

From our experimental tests, we also deduced that the PWF algorithm exhibits a rather stable behavior with the varying of the thresholds that characterize the algorithm. Therefore, it is generally simple and quick to find out the best values of the thresholds for a given user application.

References

[1] G. A. Agha (1986) Actors: a Model of Concurrent Computation in Distributed Systems, The MIT Press, Cambridge.

[2] G. A. Agha (1990) Concurrent Object-Oriented Programming, *Communications of the ACM*, 33(9), p. 125-141.

[3] G. E. Blelloch, P. B. Gibbons & Y. Matias (1995) Provably Efficient Scheduling for Languages with Fine-Grained Parallelism, *7th Annual ACM Symposium on Parallel Algorithms and Architectures*, ACM Press, New York, p. 1-12.

[4] G. E. Blelloch, (1996) Programming Parallel Algorithms,*Communications of the ACM*, 39(3), p. 85-97.

[5] T. L. Casavant & J. G. Kuhl (1988) A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems, *IEEE Transactions on Software Engineering* 14(2), p. 141-154.

[6] D. E. Culler, R. M. Karp, D. A. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian & T. von Eicken (1993) LogP: Towards a Realistic Model of Parallel Computation, *4th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, p. 1-12.

[7] M. Di Santo, F. Frattolillo, W. Russo & E. Zimeo (1996) The AL++ Project: Object-Oriented Parallel Programming on Multicomputers, *1st International Workshop on Parallel and Distributed Software Engineering*, Chapman & Hall, London, p. 277-282.

[8] M. Di Santo, F. Frattolillo & G. Iannello (1995) Experiences in Dynamic Placement of Actors on Multicomputer Systems, *3rd Euromicro Workshop on Parallel and Distributed Processing*, IEEE Computer Society Press, Los Alamitos, p. 130-137.

[9] A. Dubrovski, R. Friedman & A. Schuster (1997) Load Balancing in Distributed Shared memory Systems, to appear on *International Journal on Applied Software Technology.*

[10] D. L. Eager, E. D. Lazowska & E. D.Zahorjan (1986) Adaptive Load Sharing in Homogeneous Distributed Systems, *IEEE Transactions on Software Engineering*, 12(5), p. 662-675.

[11] D. G. Kafura & R. G. Lavender (1996) Concurrent Object-Oriented Languages and the Inheritance Anomaly in Parallel Computers: Theory and Practice (Eds. T. L. Casavant, P. Tvrdík and F. Plásil), IEEE Computer Society Press, Los Alamitos, p. 221-264.

[12] R. Panvar & G. Agha (1994) A Methodology for Programming Scalable Architectures, *Journal of Parallel and Distributed Computing*, 22(3), p. 479-487.

[13] N. G. Shivaratri, P. Krueger & M. Singhal (1992) Load Distributing for Locally Distributed Systems, *IEEE Transactions on Computers*, 41(12), p. 33-44.

[14] W. Shu & L. V. Kalè (1989) Dynamic Scheduling of Medium-Grained Processes on Multicomputers, TR-89-1528, Computer Science Department, University of Illinois at Urbana-Champaign.

[15] A. Sinha & L. V. Kalè (1993)A Load Balancing Strategy for Prioritized Execution of Tasks, *International Parallel Processing Symposium*, p. 230-237.

[16] T. T. Y. Suen & J. S. K. Wong (1992) Efficient Task Migration Algorithm for Distributed Systems, *IEEE Transactions on Parallel and Distributed Systems*, 3(4), p. 488-499.

[17] P. Wegner (1990) Concepts and Paradigms of Object-oriented Programming, *OOPS Messenger*, 1(1).

[18] M. H. Willebeek-LeMair & A. P. Reeves (1993) Strategies for Dynamic Load Balancing on Highly Parallel Computers, *IEEE Transactions on Parallel and Distributed Systems*, 4(9), p. 979-993.

Advances in Computer Assisted Image Interpretation

Wim Mees and Christiaan Perneel
Royal Military Academy — Signal & Image Centre
Renaissancelaan 30, B-1000 Brussels (Belgium)
Phone: +32 2 737 6121, Fax: +32 2 737 6163
E-mail: wim@elec.rma.ac.be

Keywords: scene analysis, artificial intelligence, fuzzy expert system, blackboard system

Edited by: Rudi Murn
Received: July 12, 1997 **Revised:** November 26, 1997 **Accepted:** January 18, 1998

In this paper we present a semi-automatic aerial image interpretation system. The knowledge, to be incorporated into the system in order to identify the best solution for the otherwise underconstrained problem of image interpretation, is structured in a logical way and distributed over a set of knowledge sources in a blackboard system. Each knowledge source applies the knowledge representation method which is best suited for its type of knowledge. In the instances where the knowledge represented in the system is insufficient, the system will suffer from an optical illusion and the human supervisor will have to correct the solution stored on the blackboard. The main goal of the system is to free the human image interpreter from the routine object vectorization part of the work and allow him/her to focus on the interpretation of the presence of specific objects at certain positions.

1 Introduction

For decades photo-interpreters have been looking at photographs, using a magnifying glass to observe smaller details. Some computation aids were applied to perform elementary conversions from distances measured on the photograph to approximate real world distances, based on an average scale for the photograph. Nowadays they use a computer for visualizing scanned images, performing resolution changes and some simple computations but apart from that not a lot has changed. Instead of drawing on the paper copies of the photographs and preparing a report on a typewriter, a photo-interpreter now vectorizes on-screen and uses some word processor, but he/she is essentially still performing the same basic operations.

Most commercial software packages which combine aerial image manipulation with the management of vector-style geographical information still require the user to perform the passage from raster to vector information manually. Some have limited automatic line-following capabilities but these are most often not useful in an operational environment. What these software packages excel in however, is their huge libraries of image processing techniques. The image can be filtered in order to obtain a smoother, noise-filtered image, to make the image sharper and eliminate some unwanted blurring effect, to detect edges, etc.

Unfortunately, experience shows that in practice most photo-interpreters don't use these features. They claim that the human visual system is the best possible adaptive filter there is and that whichever type of filtering can never result in an image with a higher information content than the original image. Therefore they prefer to perform their interpretation on the original, unfiltered image.

When one looks at the evolution in the field of earth observation data acquisition systems, it becomes obvious that the flux of aerial and satellite imagery will grow exponentially, whereas the number of trained photo-interpreters does not. Therefore the real challenge for computer system developers in the field of scene analysis is to build a semi-automatic system which relieves the human photo-interpreter from the routine part of his work. The photo-interpreter can then focus on those parts of the job for which his/her human intelligence and intuition are indispensable. In this paper we will present a possible layout for such a system. First we will discuss in section 2 the fact that scene analysis is essentially an ill-posed problem and that therefore a priori knowledge has to be integrated in the system in order to regularize the problem and obtain a single, unique solution. In section 3 we briefly sketch the classic image analysis paradigm and we explain why this doesn't suit our needs. This knowledge will be analyzed in further detail in section 4. We will show that each type of knowledge ideally requires a different type of representation method. In section 5 a system design will be presented which allows us to realize this requirement and furthermore offers us an interesting problem solving method. A prototype of this system has been developed and will be described in section 7. Finally, in section 8 we will present our conclusions.

2 Ill-posed problem

Hadamard introduced the notion of ill-posedness in the field of partial differential equations [?]. A problem is *well-posed* when a solution *exists*, is *unique* and depends *continuously* on the initial data. It is *ill-posed* when it fails to satisfy at least one of these criteria.

Ill-posed problems have been a mathematical curiosity for many years. Nowadays they arouse great interest since many problems of practical interest turned out to be ill-posed, such as the under-constrained inverse problem of scene analysis where one tries to obtain information about the 3D world from a 2D image. This is illustrated in example 2.1.

Example 2.1 In figure 1 it is impossible to tell whether the object, shown in the image, is in reality a cube, a piece of cardboard with a cube drawn upon it or some sort of a deformed cube.

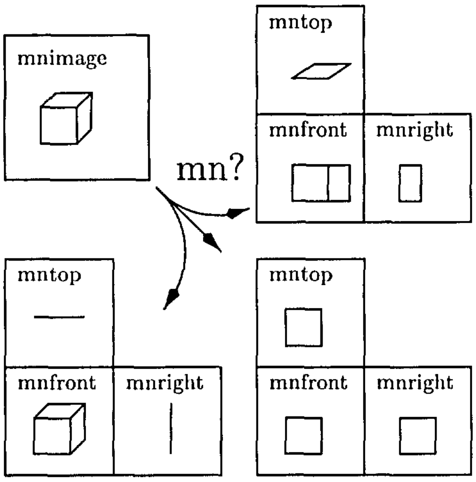
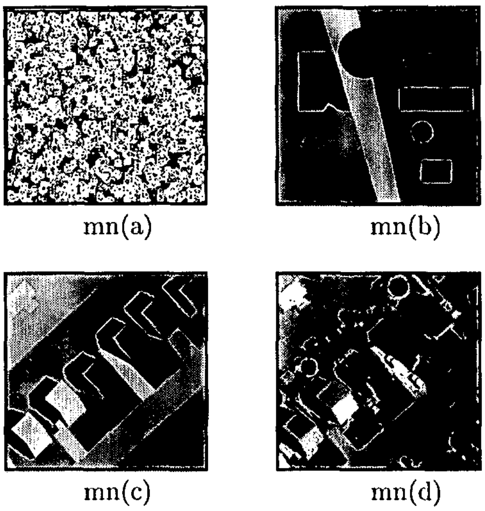


Figure 1: Extract 3D information from 2D data

In order to solve an ill-posed problem, well-posedness must be restored by restricting the class of admissible solutions using *a priori knowledge*. Several techniques exist, such as variational regularization using a quadratic stabilizer. Often realistic a priori knowledge requires non-quadratic functionals in order to be implemented, resulting in a no longer convex solution space and thus the need to use stochastic methods to escape from local minima [?].

Another regularization method consists of choosing a *discrete solution space* with finite dimensions and imposing *generic constraints*. This may seem like a harsh restriction at first but this isn't really so. Indeed, our real world is highly structured and is constrained by physical laws to a number of basic patterns. The apparent complexity of our environment is produced from this limited vocabulary by compounding these basic forms in different combinations. If the intrinsic complexity of our environment were approximately the same as its apparent complexity, there would be no lawful relations and no intelligent prediction. This is illustrated in example 2.2.



mn(a) mn(b)

mn(c) mn(d)

Figure 2: Complexity of real world images

Example 2.2 In figure 2, image (a) shows a matrix with random pixel values. It is clear that this image doesn't correspond with a real world scene. Image (b) already has a certain structure since the gray-values are distributed in a limited number of homogeneous regions. Nevertheless it is immediately clear to the human observer that this image does not represent a real world scene. Image (c) uses the same gray-values as (b), with similarly sized regions. Yet, a human observer will interpret this image as an aerial image or a sketch thereof. This is because the structure of the regions is compatible with the a priori knowledge a human observer has acquired over the years about his real world environment. Image (d) shows the aerial photograph from which the sketch in (c) was derived.

It is the internal structuring of our environment that allows us to reason successfully using the simplified descriptions we typically employ [?]. Different techniques have been described for determining these generic constraints, based on engineering, statistical, biological, and physical approaches [?].

In example 2.1, restricting the solution space to cubes and ellipsoids would regularize the problem since only one of the proposed solutions would then be compatible with this a priori knowledge.

An example of the use of constraints based on physical laws is given in example 2.3.

Example 2.3 When the front view is

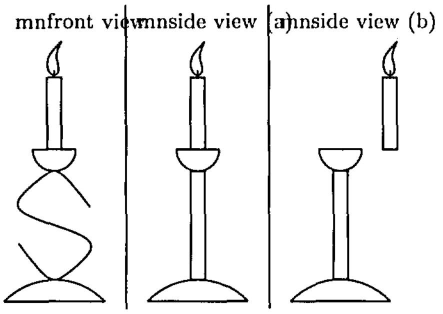


Figure 3: Candle example

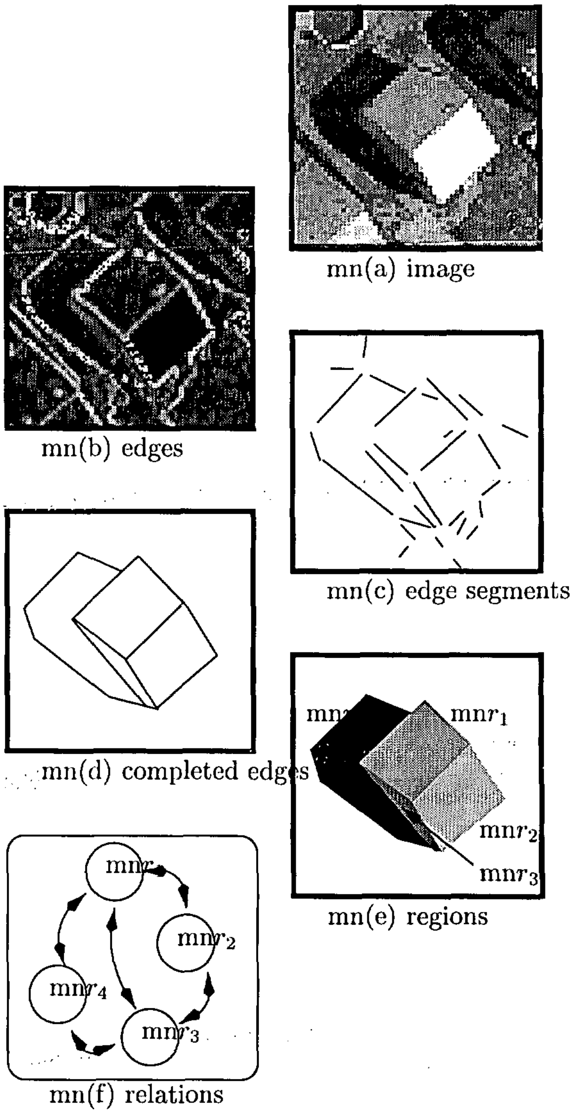
given, a human observer will immediately think of the situation depicted in side view (a), namely a candle put on top of a pedestal. The solution shown in side view (b) is immediately discarded even though it would result in the same front view. This is because we use our a priori knowledge of the law of gravity which teaches us that a situation as depicted in side view (b) is impossible in the real world.

When the a priori assumptions are violated in specific instances, the obtained solution may not correspond to the real world situation. The algorithm then suffers from an optical illusion.

3 Classic Paradigm

The aim of scene analysis is to render explicit the information which is implicit in the image. A very popular approach is to build a bottom-up hierarchy of ever more abstract higher-level information layers from the image pixel level up to the level of real world objects [?]. In a first phase low-level features, such as contours and regions with certain attributes, are extracted from the image, resulting in a map-like representation, which Marr called "primal sketch" [?]. These features are then grouped into more complex elements at a higher level of abstraction. Finally the highest-level structures are matched against stored models, which are generalized relational structures representing prototype elements that correspond to general types of images. This is illustrated in example 3.1.

Example 3.1 In figure 4 image (a) shows the lowest layer, the input aerial image. A Sobel edge detector is applied to this image, resulting in image (b). The raster information in (b) is converted to line segments, shown in (c). The segments are joined and gaps are filled in order to obtain the completed edge representation, shown in (d). These edges enclose a series of regions, $r_1 \dots r_4$ shown in (e). The combination of these regions, taking into account their characteristics as well their neighborhood relations, shown in (f), will then be matched with a series of reference models in


mn(a) image

mn(b) edges

mn(c) edge segments

mn(d) completed edges

mn(e) regions

mn(f) relations

Figure 4: Classic paradigm

order to deduce that we're actually seeing a house.

Due to the fact that scene structure is underdetermined by local image data [?], *unverifiable* large-scale assumptions (eg. isotropy, smoothness, ...) have to be imposed with this classic approach. As a result, techniques based on these assumptions are often fragile and error-prone. We therefore have to organize real world scene analysis knowledge in an efficient way and incorporate it into the automatic interpretation system.

4 Adding scene analysis knowledge

Scene analysis knowledge is a *qualitative knowledge*. Reichgelt describes qualitative knowledge as any kind of knowledge that doesn't always allow a correct and consistent match between the represented objects and the real world but can nevertheless be used to get an approximate characterization of the behavior of the modeled domain [?].

A complex, real-world problem such as scene analysis cannot be formalized in a nice and neat way. Most often not all the information needed is available. The available information will furthermore not be 100% correct and consistent and may thus prove and disprove a fact with the same theory. Therefore, inference techniques which deal with partially incorrect, incomplete and/or inconsistent knowledge have to be used.

A human expert solves common problems within his/her field of expertise without much reasoning. When he is faced with an unfamiliar situation, he relies on a more profound domain knowledge and will solve the problem using more elaborate models. This is illustrated in example 4.1.

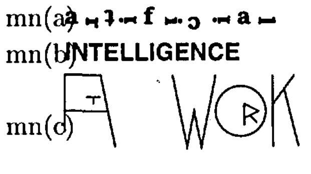


Figure 5: Simple versus elaborate reading model

Example 4.1 In figure 5 some words are shown. The human observer will immediately recognize the word in (b), whereas the words in (a) and (c) require a more important effort since the structure used to combine individual characters into a single word is a bit unusual. The simple left-to-right all-characters-straight-up model we typically use, doesn't apply here so the reader has to discover the words using a more elaborate and therefore slower reading-model.

We will thus have to implement at least two levels of knowledge representation. The first level contains very efficient knowledge that immediately produces approximate results whereas the second level consists of a more complicated model with more expensive problem solving methods. Both forms of knowledge are closely connected and it would be hard to assign a certain piece of knowledge to either of the two. Both probably represent the same kind of knowledge but simply at a different level of detail. Neither level can be omitted. Alongside a set of detailed, correct rules, a set of simple rules should be known. Although incorrect in some exceptional cases, they combine high predictivity with efficiency. All other things being equal, they are furthermore more likely to be predictive for future data.

We propose to classify the scene analysis knowledge according to the following scheme [?, ?]:

- scene description

  - *geographical database:* when a geographical database of a sufficiently large scale (e.g. 1/10.000), so as not to undergo displacements due to generalization, is available, it will be consulted by the scene analysis system in order to collect a priori knowledge.

  - *input by a human operator:* a human operator could prepare a scene by marking already certain key elements in the scene serving as a priori knowledge for the automatic system. He may also follow the evolution of the semi-automatic interpretation process using the graphical user interface and intervene by adding or removing objects in order to steer the system or correct the intermediate partial solution.

- scene independent knowledge

  - *interpretation strategy:* it is well known that our eyes don't move in a smooth and continuous manner when viewing an image. They go briefly over numerous fixation points, separated by jumps, and concentrate on those features conveying salient information [?]. The human mind doesn't scan from left to right and top to bottom like most image processing algorithms do. A trained image interpreter will steer his/her focus of attention based on hypotheses generated by previously interpreted objects as well as on a set of standard operating procedures. For instance in a suburban area first look for roads in a down-sampled copy of the image and after that look for buildings alongside the roads in the full-resolution image. For the image in figure 6 the attention will first be focused on the houses and the road and only thereafter on smaller details such as the driveways, cars, swimming pools, ...

- *generic constraints:* this knowledge implements general physics laws (e.g. relationships between a building and its shadow), administrative regulations regarding land-use as well as the experience a human operator has acquired after years of practice, expressed as series of rules of thumb. It can best be represented by a set of global rules acting on clusters of objects, of the same or of different types. The rules will judge the geometric relationships between the objects within a cluster [?]. In a multi-sensor system this knowledge will be sensor-independent.

- *object-type specific knowledge:* this is the knowledge which allows one to distinguish a certain type of object from all other types. It is most often expressed as a list of conditions on image features such as contours, texture and the gray-value histogram (e.g. a building is rectangular or L-shaped with certain limits on its dimensions). This knowledge will in general be sensor-dependent.

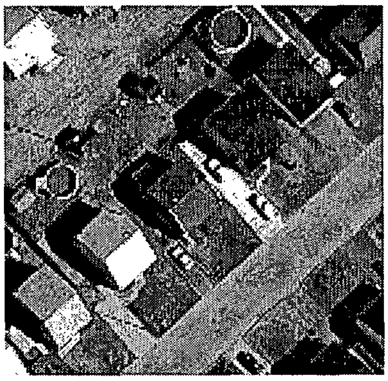


Figure 6: Aerial photo

The interaction between the local and global part of the scene analysis knowledge is very important since neither can do the job alone. This is illustrated in examples 4.2 and 4.3.

Example 4.2 In figure 7 we see that using local information about the features alone, as is given in window (a), we cannot uniquely identify their type. However, when we take into account the relative positions and orientations of the features, information which is available to the observer in window (b), we immediately observe that they are nothing else but the characteristic features of a face.

Example 4.3 In figure 8 five parts of an image are shown which were cut out of an aerial image. Given only the local information of the objects themselves and a very limited part of their immediate surroundings, it is clearly difficult to identify them. This might be our first guess:

(a) truck with tractor and trailer
(b) factory building
(c) car
(d) cooling tower of power plant or a silo
(e) flight of stairs

However, when we look at these objects in the original image, from their position relative to the houses and roads we learn that we had it completely wrong. These are the correct answers:

(a) small building in backyard
(b) low greenhouse
(c) car
(d) swimming pool
(e) driveway

This goes to show that the relative position with respect to other objects is indispensable when trying to recognize an object in an image.

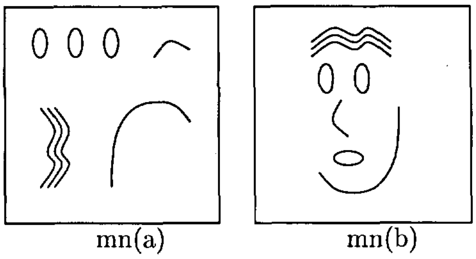


Figure 7: Local versus global knowledge (I)

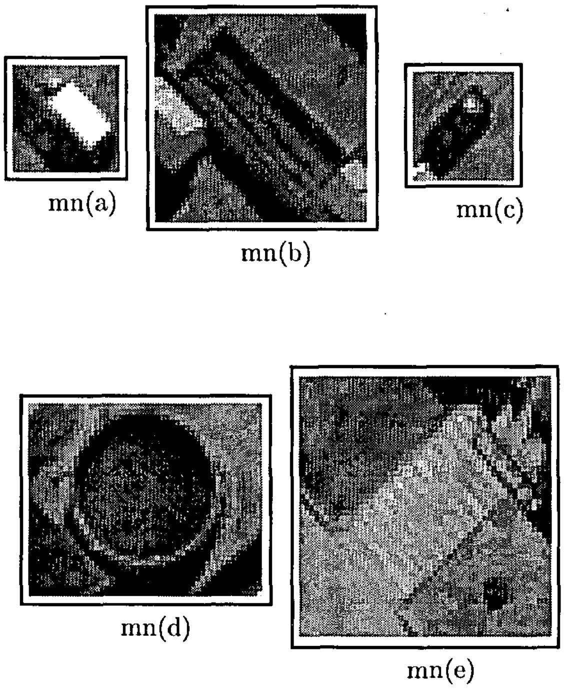


Figure 8: Local versus global knowledge (IIa)

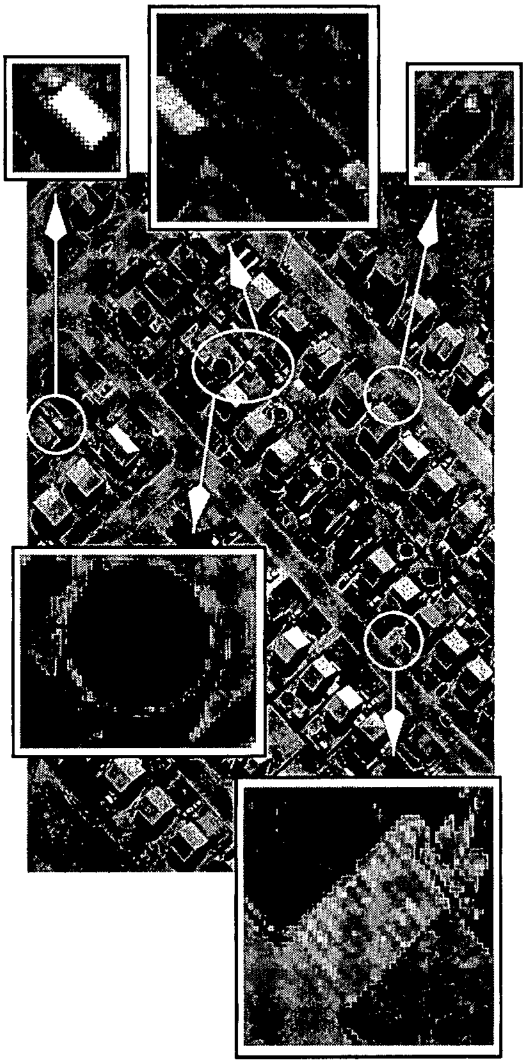


Figure 9: Local versus global knowledge (IIb)

5 General system lay-out

One could consider tackling the image interpretation problem using a "generate-and-test" approach. Unfortunately, a complex problem like scene analysis is combinatorially explosive so that generating all possible solutions would be a sheer impossible task. It would furthermore be impossible to build a generator which produces only a limited set of possible solutions yet always containing the correct solution.

We will therefore base our problem solving strategy on the technique of generating and fusing uncertain and partial solutions to construct solutions, using an *island-driven* approach. A relatively reliable partial hypothesis is designated as an island of certainty and the hypothesis building process pushes out from this island in a number of directions into the ocean of uncertainty surrounding it. Image processing modules will generate low-level features which will be combined into candidate objects, forming partial solutions. This bottom-up hypothesis generation will be linked to a model-driven top-down analysis. The top-down analysis will verify the presence of the expected characteristic features for a specific type of object. The system will try to extract from the images any features that were for some reason missed during the bottom-up pass.

Fitting all the above-mentioned types of knowledge in a single knowledge representation scheme would involve compromising and would thus result in a suboptimal solution. The general strategy can be well represented using a a *goal-reduction* scheme [?]. The generic constraints will be expressed by human photo-interpretation experts. They will use natural language rules, based on vague terms. This set of rules will be imposed using a *fuzzy production rule system.* For the *composition* of the fuzzy relations several techniques are mentioned in the literature. We use the *max-min method* as it is used by Zadeh in his approximate reasoning based on linguistic IF-THEN rules. It is claimed that this method correctly reflects the approximate and interpolative reasoning used by human beings when using natural language propositions for deductive reasoning [?] [?]. The knowledge related to each object type is necessary when evaluating or extracting a single object without taking into account other objects in the scene. This knowledge varies strongly from one type of object to another. It will be integrated in the implementation of local detectors, each one dedicated to a specific type of object. Different sensors will require different local detectors for specific object-types.

The *blackboard* problem-solving model is particularly well suited for this type of complex problems. It supports the incremental development of solutions, can apply different types of knowledge and can adapt its strategy to a particular problem situation [?, ?].

It allows the use of independent *knowledge sources* in order to represent the different types of knowledge as shown in figure 10.

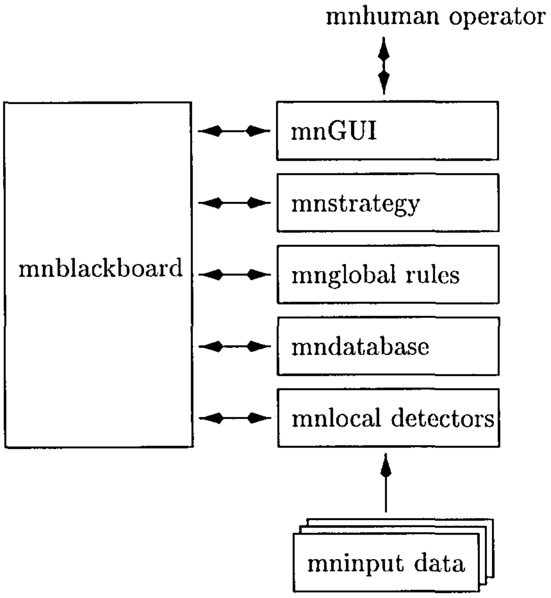


Figure 10: System design

The incorporation of knowledge at two levels of detail within the system, which was mentioned earlier, will be implemented at the level of the "generic constraints" rule base by combining specific rules together with more general ones, as well as at the level of the local detectors by placing "quick and dirty" detectors together with "slow and dedicated" ones at the disposal of the local detector manager [?].

The blackboard model is quite popular with scene analysis systems (e.g. Sacap [?], Messie [?], ...). There is, however, no consensus on the way the scene analysis knowledge is to be distributed over a set of knowledge sources.

6 Knowledge sources

6.1 World-model

With the basic system design, shown in figure 10, different types of information will be stored in the same shared memory, namely the central blackboard. This will result in a heterogeneous mixture of data, the objects which form the partial solution together with the transactions between knowledge sources, all posted on the same blackboard.

In order to structure all these data elements we could have used different panels within the blackboard. We have however decided to split the blackboard in a knowledge source, called "world-model" and containing the partial solution, and the blackboard, containing the transactions. The resulting system design is shown in figure 11. This solution allow to add a series of methods which can be applied to the solution on demand, such as geographical queries on the objects in the partial solution. It is indeed very useful to be able to perform queries like "obtain a list of all buildings around a certain road segment". This wouldn't have been possible when using panels within the blackboard.

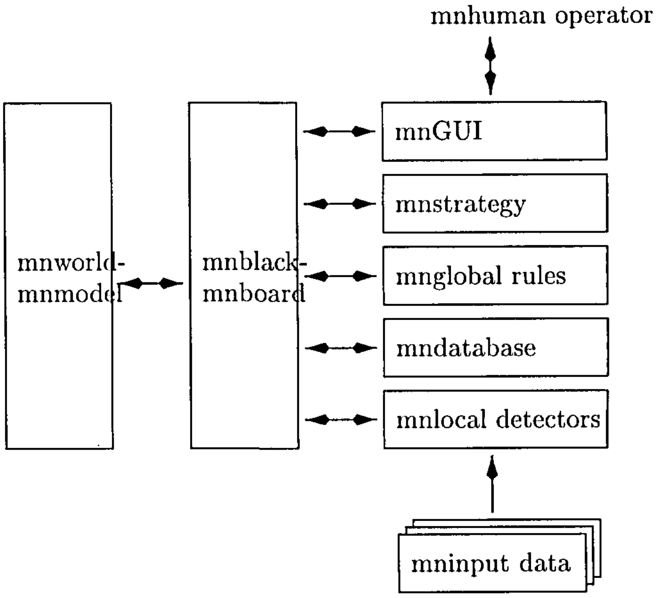


Figure 11: System design with world-model

6.2 Local knowledge

The local information about object types, defined in section 4, will be stored at two places.

For each type of information element an object prototype will be pre-defined. Whenever the system detects a possible instance of an information element, a candidate object will be created as a realization of the appropriate object prototype. These object prototypes store information using a frame based knowledge representation method [?]. Each object contains a series of attributes as well as a number of methods, respectively storing and applying local knowledge.

Another part of the scene independent object-type specific knowledge is implemented by the local detectors. They use it to look for specific features or information elements in the input data.

An example hereof is shown in figure 12. A local detector, specialized in the extraction of houses, is presented with (part of) an aerial image and produces a series of candidate house objects as its output. The local model of a house, used by the detector, is that of a rectangle with a smooth interior. This results in four correct candidates (hs_1, hs_2, hs_4, hs_5) and two instances where the local knowledge was not sufficient

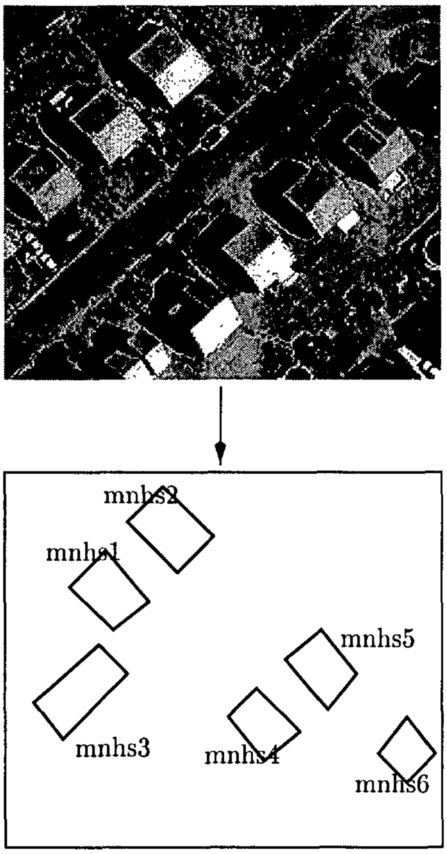


Figure 12: Local detector for the extraction of houses

to describe the complex real world environment and false candidates were generated (hs_3, hs_6). One can furthermore observe that some of the houses, present in the scene, were not extracted by the local detector.

object	house	hs1
attributes	(x_1, y_1)	$(100, 100)$
	(x_2, y_2)	$(150, 100)$
	(x_3, y_3)	$(150, 150)$
	(x_4, y_4)	$(100, 150)$
	surface smoothness	5.7
	confidence	0.9
methods	wall length $l_a()$	
	wall length $l_b()$	
	wall length $l_c()$	
	wall length $l_d()$	
	surface $S()$	
	angle $\alpha_1()$	
	angle $\alpha_2()$	
	angle $\alpha_3()$	
	angle $\alpha_4()$	

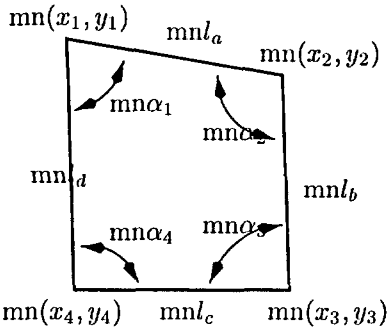


Figure 13: House object stored in world-model

Each of these houses is stored in the world-model as an object with attributes and methods, as shown in figure 13.

6.3 Global knowledge

In a real world environment it is impossible to define the spatial data analysis global constraints in an exact mathematical way. On the one hand our understanding of this problem is at a qualitative and declarative level, based on vague linguistic terms. On the other hand we need a method for applying this knowledge in a numerical way to the instances of the pre-defined object prototypes that were extracted from the input aerial images. We will therefore use a fuzzy production rule system to represent the scene independent inter-object knowledge in the global rules knowledge source since it allows us to acquire knowledge symbolically yet process the data numerically [?].

R_1	restriction S_1	
R_2	restriction S_2	
...	...	
R_r	restriction S_r	
R_{r+1}	IF condition C_{r+1}	THEN restriction S_{r+1}
R_{r+2}	IF condition C_{r+2}	THEN restriction S_{r+2}
...	...	
R_{N_r}	IF condition C_{N_r}	THEN restriction S_{N_r}

Figure 14: Canonical form of a fuzzy production rule system

A fuzzy production rule system can be written in canonical form, shown in figure 14. The unconditional restrictions $R_1 \ldots R_r$ can be expressed as conditional restrictions with a conditional part which is always true in order to obtain a uniform formalism. In the thus resulting set of conditional restrictions $R_1 \ldots R_{N_r}$ both the condition and the restriction are a concatenation of atomic terms, as there are:

primary terms:	labels of fuzzy subsets of the universe of discourse,
negation:	"not",
connectives:	"and", "or",
hedges:	linguistic modifiers ("very", "much", "slightly", ...), operating on primary terms,
markers:	parentheses.

The rules will thus be expressed in the following form; for rule $r = 1 \ldots N_r$:

$$\begin{array}{ll} \text{IF} & (x_1 \text{ is } A_{1r}) \text{ AND } (x_2 \text{ is } A_{2r}) \ldots \\ & \qquad \text{AND } (x_{N_i} \text{ is } A_{N_i r}) \\ \text{THEN} & (y_1 \text{ is } B_{1r}) \text{ AND } (y_2 \text{ is } B_{2r}) \ldots \\ & \qquad \text{AND } (y_{N_o} \text{ is } B_{N_o r}) \end{array}$$

with input fuzzy sets A_{ir} defined on universes $x_i \in X_i$ and output fuzzy sets B_{jr} defined on universes $y_j \in Y_j$. The variables x_i and y_j actually refer to the attributes of the objects in the world-model. The fuzzy sets A_{ir} and B_{jr} are expressed using following membership functions (with $F(z)$ denoting the family of fuzzy sets defined on z).

$$\begin{array}{ll} A_{ir} \in F(x_i) \; ; & \mu_{A_{ir}}(x_i) : X_i \to [0,1] \\ B_{jr} \in F(y_j) \; ; & \mu_{B_{jr}}(y_j) : Y_j \to [0,1] \end{array}$$

Consider once more the situation in figure 12. Let's suppose that a road detector would have detected the road rd_1 shown in figure 15. As an example we will then consider following rules, expressed on the distance d between a house and the nearest road and on the confidence c we have in this candidate house,

IF	d is TOOSHORT	THEN	c is LOW
IF	d is TYPICAL	THEN	c is HIGH
IF	d is TOOLONG	THEN	c is LOW

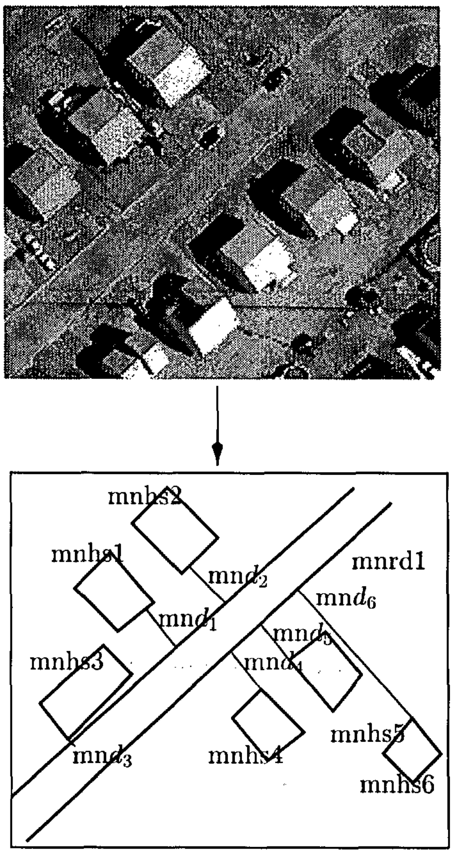


Figure 15: Result of road detector

with the membership functions as shown in figures 16 and 17. In the example of figure 15, this will result in a high confidence in the objects (hs_1, hs_2, hs_4, hs_5), whereas the candidates (hs_3, hs_6) will be assigned a low confidence. If no other objects support these last houses through rules which increase their confidence, they will be eliminated from the world-model in the long run.

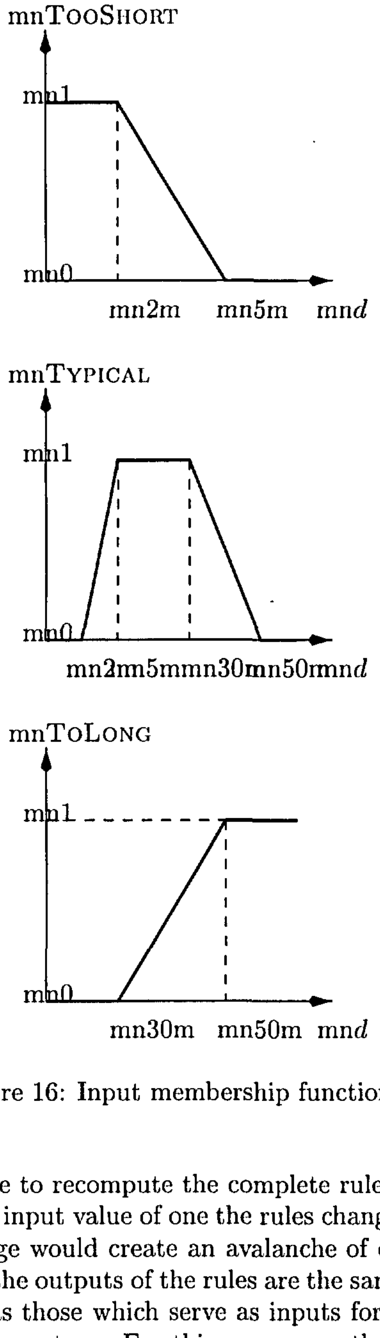


Figure 16: Input membership functions

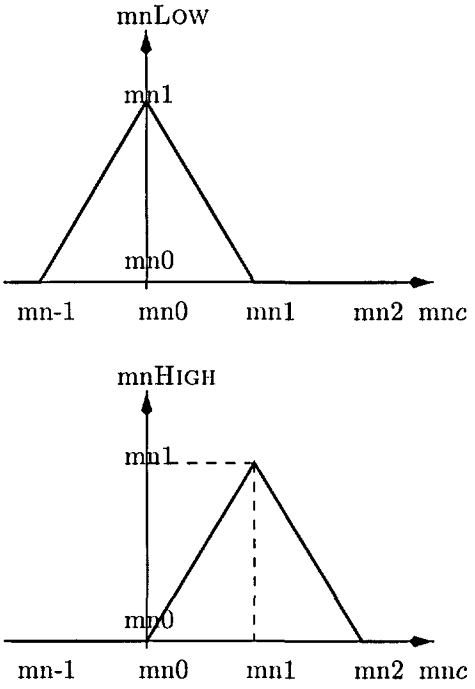


Figure 17: Output membership functions

If we were to recompute the complete rule base every time an input value of one the rules changes, every single change would create an avalanche of computations since the outputs of the rules are the same object attributes as those which serve as inputs for the production rule system. For this reason another way of managing the processing to be performed by the production rule system has been developed.

The rules are stored as prototypes in the global rules knowledge source. Whenever a new combination of objects, corresponding to the antecedents required by a specific rule, appears on the blackboard and is added to the world-model, an instance of this rule is created and posted on the blackboard by the global rules knowledge source. For this reason, whenever a new object is posted on the blackboard, for each relevant rule prototype all possible combinations of its antecedent objects containing this new object are requested from the world-model. For each of these combinations a rule instance will be derived from the prototype, its input and output variables pointed towards the attributes of the objects in the combination and posted on the blackboard. It is then added to the world-model, where it will be linked to its antecedent and consequent objects. This results in a high-level network, representing the solution hypothesis together with the local and global knowledge supporting is. An example hereof is shown in figure 18.

6.4 Strategy

When a human expert analyzes spatial data, his focus of attention doesn't move in a smooth and continuous manner. His eyes jump rapidly from one fixation point to another, concentrating on those features which convey salient information [?]. The identification of specific objects at certain positions in the scene will give rise to hypotheses which then influence the path the focus of attention follows across the scene.

It is the task of the strategy knowledge source to steer the focus of attention of the semi-automatic data analysis system based on the types of input data available as well as on the state of the partial solution in such a way that the best possible solution is obtained

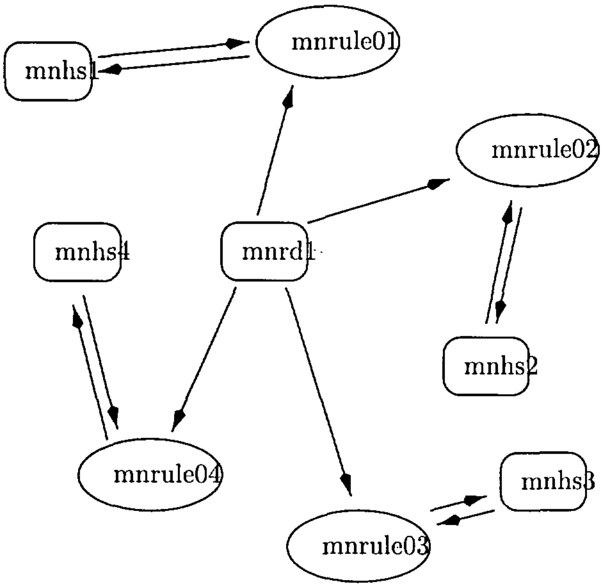


Figure 18: High-level network

in a minimum of time.

The strategy knowledge source uses a goal reduction scheme to represent its knowledge. In such a scheme every goal is iteratively split into a conjunctive or disjunctive set of sub-goals at a lower level until these subgoals themselves correspond to simple actions which must be undertaken. As an example a small part of such a goal reduction tree applied to scene analysis is given in figure 19. In this example different strategies are defined as a function of the available image-types (visual VIS, thermal infrared THIR or synthetic aperture radar SAR) at the input.

The user can also define a series of different strategies for specific tasks he has to perform.

7 Prototype

7.1 Prototype definition

At present a prototype has been developed which implements the system described in section 5. The central blackboard, the different knowledge sources and the local detectors are all built as independent executables. The blackboard and the different knowledge sources communicate using TCP/IP sockets as interprocess communication technique. The local detectors are actually scripts which call a series of executables with specific parameters and in a well-defined order. They are launched by the local detector manager knowledge source.

The blackboard module checks at startup its host's services database for a "blackboard" service definition. If this is found, the central blackboard will listen on the corresponding port number for incoming requests from

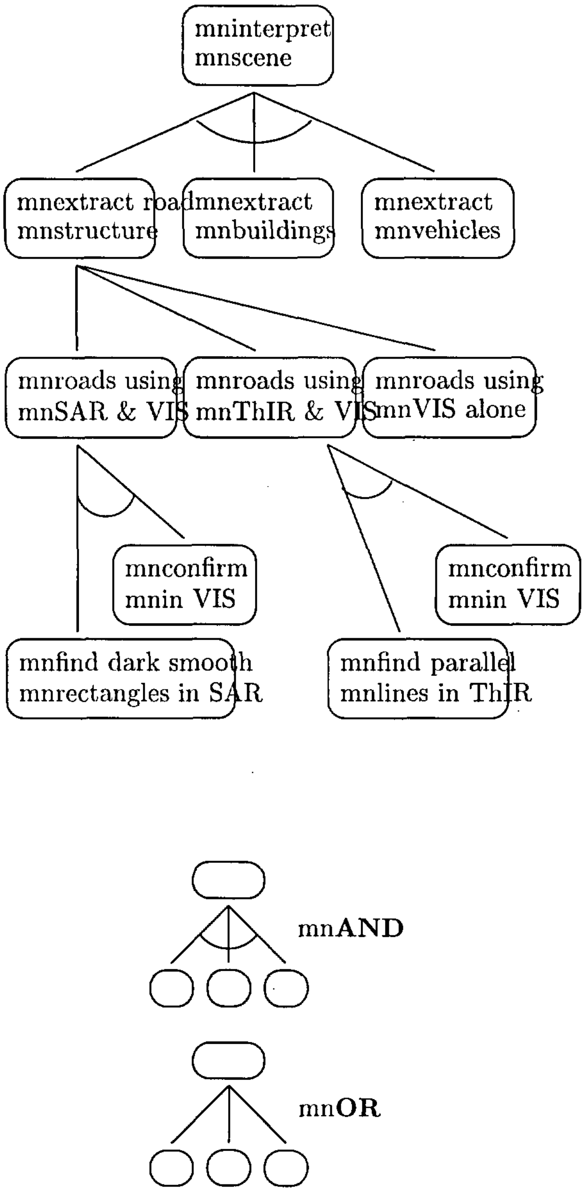

mninterpret mnscene
mnextract roads mnstructure
mnextract mnbuildings
mnextract mnvehicles
mnroads using mnSAR & VIS
mnroads using mnThIR & VIS
mnroads using mnVIS alone
mnconfirm mnin VIS
mnfind dark smooth mnrectangles in SAR
mnconfirm mnin VIS
mnfind parallel mnlines in ThIR
mnAND
mnOR

Figure 19: Strategy goal reduction scheme

the knowledge sources. When no appropriate service definition is found the blackboard module looks for and uses the first free port within the user-assigned range of port numbers. It then writes the name of its host as well as the port number on which it listens to a configuration file.

The knowledge sources read this configuration file and connect to the blackboard server on the specified port number of the specified host. The knowledge source clients communicate with the central blackboard server through atomic read and write operations in order to preserve the integrity of the blackboard data.

When the directory, where the executables are located and the configuration file is stored, has been mounted using NFS on different workstations, the different knowledge sources can be run in whichever combination on this cluster of workstations.

At present a prototype of the different knowledge sources, as shown in section 6, have been developed as well as a limited number of local detectors. Where local detectors are missing and this would hinder the testing of the system, these have been simulated. In this way the expert-system part of the scene analysis system could nevertheless be evaluated.

7.2 Results & Future work

Our preliminary tests have shown that the system functions well for the typical configurations of input data which were considered when designing the local detectors and when writing the global rules and the strategy. It is quite clear however that in order to test the validity of the global rules a much larger amount of test images is needed. This will also show the shortcomings of the already developed local detectors in certain specific circumstances and thus lead to the development of some extra, complementary local detectors.

The system has a distinct advantage over other systems which only incorporate the typical local information handling routines. Indeed, it will itself operate these local routines, with varying parameters, allowing for a certain number of false candidates since it will thereafter weed out most of the false candidates using the knowledge contained in the global rules knowledge source. The high-level, expert system part here plays the role of a filter, eliminating the "noise peaks" corresponding to false candidates. Simulations with noisy detectors have shown that this will only function up to a certain level of false candidates. Beyond that level the system will start discovering fake structures in the set of false candidates, which will reinforce its confidence in these false candidates and may even suppress correct candidates in the same region.

Because of the fact that the rules can themselves initiate specific requests for the detection of new objects without any control by the strategy knowledge source, the risk is very real that an explosion of requests occurs. Indeed, for each request a local detector will be launched, potentially generating a number of objects which themselves may give rise to even more detection requests via the global rules which will be applied to them. For this reason we have decided to add a local detector manager knowledge source to the system, as shown in figure 20, which will sort the detection requests, monitor the system load and launch new detectors whenever this is possible without saturating or over-loading the system.

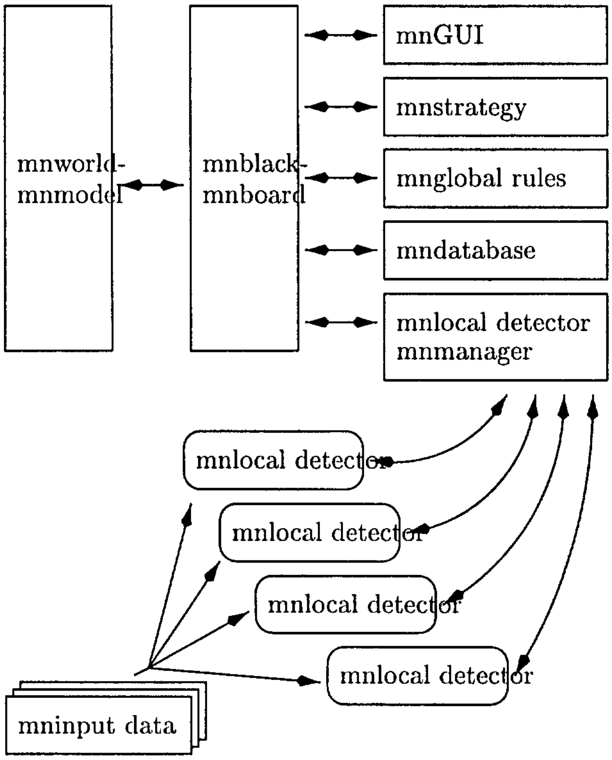


Figure 20: Adapted system design with local detector manager

We have furthermore found the need for rules which disappear after being fired as well as for rules which are computed only once and then disappear whether they're fired or not. An example of the first type would be a rule which applies to the cluster of a road and a perpendicular road segment connecting to it and checks whether a prolongation of this side-road exists at the opposite side of the main road. Once this rule was fired, a detection request will have been posted on the blackboard and a local detector will have been launched to perform the verification. There's no need to re-verify this several times using the same rule and therefore the same parameters for the local detector. An example of the second case is a rule which checks whether the positions of a house and a nearby road are compatible. Whichever decision is taken by the rule, the house and road won't start walking around in the world-model so their relative position won't change and therefore the inputs of the rule won't change either.

In all, we can say that the way in which the knowledge is structured in the system is the major advantage of the approach we've presented. The user has a good grip on the knowledge the system applies thanks to the clear logic as to which knowledge goes where. The way in which the objects, representing the solution, are combined with the rules, raising or lowering their confidence, in a single network in the world-model has shown to be very useful. The network allows the user to examine immediately what the influence of every single one of the global rules is on each of the objects to which it is applied. What we have ascertained however is the fact that some ideas we had where hard to implement with the structure as it is at present. We will therefore add in the next version of the system some extra knowledge sources, one of which will realize perceptual grouping at the level of the objects in order to detect different types of structures of objects, such as buildings lying on a parabola, cars equally spaced in a parking lot, etc.

8 Conclusion

Due to the ever increasing amount of available earth observation information, the need for semi-automatic systems, which aid the human expert in his analysis, will continue to grow. Aside from the well-known point and raster data manipulation techniques with which actual systems are already equipped, these semi-automatic aids will furthermore allow a user to integrate a part of his knowledge into the system in the form of a general strategy, global rules and local descriptions.

These systems will then be able to relieve the human expert from the routine part of his work and allow him to focus on "special cases" or on the interpretation of *why* certain features occur at certain positions without first having to extract them manually from the input data.

It is obvious that an experienced human data analyst will always outperform any artificial system when it comes down to the completeness of the analysis or the handling of exceptional cases. A human operator on the other hand has the disadvantages of a higher operating cost and a dislike for routine duties. Therefore if we combine both systems, we will obtain an increase in productivity with the same quality as when compared to the human expert alone.

Call for Paper
International Multi-Conference
Information Society - IS'98

6 – 9 October, 1998
Slovenian Science Festival
Cankarjev dom, Ljubljana, Slovenia

Programme committee:

dr. Cene Bavec, chairperson,
prof. dr. Ivan Bratko, co-chair,
prof. dr. Matjaž Gams, co-chair,
prof. dr. Tadej Bajd,
mag. Jaroslav Berce,
dr. Dušan Caf,
prof. dr. Saša Divjak,
dr. Tomaž Erjavec,
prof. dr. Nikola Guid,
prof. dr. Borka Jerman Blažič Džonova,
doc. dr. Gorazd Kandus,
doc. dr. Marjan Krisper,
mag. Andrej Kuščer,
prof. dr. Jadran Lenarčič,
dr. Franc Novak,
prof. dr. Marjan Pivka,
prof. dr. Vladislav Rajkovič,
prof. dr. Ivan Rozman,
dr. Niko Schlamberger,
prof. dr. Franc Solina,
prof. dr. Stanko Strmčnik,
prof. dr. Jurij Tasič,
prof. dr. Andrej Ule,
dr. Tanja Urbančič,
prof. dr. Baldomir Zajc,
dr. Blaž Zupan

Invitation

You are kindly invited to participate in the "New Information Society - (IS'98)" multi-conference to be held in Ljubljana, Slovenia, Europe, from 6-9 October, 1998. The multi-conference will consist of seven carefully selected conferences.

Basic information

The concepts of information society, information era, infosphere and infostress have by now been widely accepted. But, what does it really mean for societies, sciences, technology, education, governments, our lives? What are current and future trends? How should we adopt and change to succeed in the new world?

IS'98 will serve as a forum for the world-wide and national community to explore further directions, business opportunities, governmental European and American policies. The main objective is the exchange of ideas and developing visions for the future of information society. IS'98 is a standard high-quality scientific conference covering major recent achievements. Besides, it will provide maximum exchange of ideas in discussions, and concrete proposals in final reports of each conference.

The multi-conference will he held in Slovenia, a small European country bordering Italy and Austria. It is a land of thousand natural beauties from the Adriatic sea to high mountains. In addition, its Central European position enables visits to most European countries in a radius of just a few hours drive by car. The social programme will include trips by desire and organised trips to Skocjan or Postojna caves. Coffee breaks, the conference cocktail and dinner will contribute to a nice working atmosphere.

Call for Papers

Deadline for paper submission: 15 June, 1998

Registration fee is 100 US $ for regular participants (6.000 SIT for participants from Slovenia) and 50 US $ for students (3.500 SIT for Slovenian students). The fee covers conference materials and refreshments during coffee-breaks.

More information

For more information visit `http://turing.ijs.si/is/indexa.html` or contact `milica.remetic@ijs.si`.

The multi-conference consists of the following conferences:

Information Society

6–7 October, 1998
Chairs: dr. Cene Bavec, prof. dr. Matjaž Gams
Contact person: prof. dr. Matjaž Gams
Phone: (+386 61) 1773 644
E-mail: `matjaz.gams@ijs.si`
Jozef Stefan Institute, Jamova 39, 1000 Ljubljana, Slovenia, Europe

Language Technologies

Date: 6–7 October, 1998
Chair: dr. Tomaž Erjavec
Contact person: dr. Tomaž Erjavec
Phone: (+386 61) 1773 644
E-mail: tomaz.erjavec@ijs.si
Address: Jozef Stefan Institute, Jamova 39, 1000 Ljubljana, Slovenia, Europe
Submission deadline: 15 June, 1998

Manufacturing Systems and Technologies

Date: 7 October, 1998
Chair: prof. dr. Jadran Lenarčič
Contact person: prof. dr. Jadran Lenarčič
Phone: (+386 61) 1773 378
E-mail: jadran.lenarcic@ijs.si
Address: Jozef Stefan Institute, Jamova 39, 1000 Ljubljana, Slovenia, Europe
Submission deadline: 31 August, 1998

Education and Information Society

Date: 8 October, 1998
Chair: prof. dr. Vladislav Rajkovič
Contact person: Mojca Florjančič
Phone: (+386 064) 22 10 61
E-mail: mojca.florjancic@fov.uni-mb.si
Address: Faculty of Organizational Sciences, Kidričeva 55a, 4000 Kranj, Slovenia, Europe
Submission deadline: 15 June, 1998

Development and Reingeneering of Informaton Systems

Date: 8 October, 1998
Chair: prof. dr. Ivan Rozman
Contact person: dr. Ivan Rozman
Phone: (386 62) 2207 410
E-mail: i.rozman@uni-mb.si
Address: FERI, Smetanova 17, 2000 Maribor, Slovenia, Europe
Submission deadline: 15 June, 1998

Cognitive Sciences

Date: 9 October, 1998
Chair: prof. dr. Andrej Ule
Contact person: prof. dr. Andrej Ule
Phone: (061) 1769 200
E-mail: andrej.ule@guest.arnes.si
Address: Jozef Stefan Institute, Jamova 39, 1000 Ljubljana, Slovenia, Europe
Submission deadline: 15 June, 1998

Computer Analysis of Medical Data

Date: 9 oktober 1998
Chair: dr. Blaž Zupan
Contact person: dr. Blaž Zupan
Phone: (061) 1773 380
E-mail: blaz.zupan@ijs.si
Address: Jozef Stefan Institute, Jamova 39, 1000 Ljubljana, Slovenia, Europe
Submission deadline: 15 June, 1998

Machine Learning List

The Machine Learning List is moderated. Contributions should be relevant to the scientific study of machine learning. Mail contributions to ml@ics.uci.edu. Mail requests to be added or deleted to ml-request@ics.uci.edu. Back issues may be FTP'd from ics.uci.edu in pub/ml-list/V<X>/<N> or N.Z where X and N are the volume and number of the issue; ID: anonymous PASSWORD: <your mail address> URL- http://www.ics.uci.edu/AI/ML/Machine-Learning.html

---

CC AI

The journal for the integrated study
of Artificial Intelligence, Cognitive Science
and Applied Epistemology.

CC-AI publishes articles and book reviews relating to the evolving principles and techniques of Artificial Intelligence as enriched by research in such fields as mathematics, linquistics, logic, epistemology, the cognitive sciences and biology.
CC-AI is also concerned with development in the areas of hard- and software and their applications within AI.

Editorial Board and Subscriptions

CC-AI, Blandijnberg 2, B-9000 Ghent, Belgium.
Tel.: (32) (9) 264.39.52,
Telex RUGENT 12.754
Telefax: (32) (9) 264.41.97
e-mail: Carine.Vanbelleghem@RUG.AC.BE

Call for Papers

Advances in the Theory and Practice of Natural Language Processing

Special Issue of Informatica 22 (1998) No. 4

Informatica, an International Journal for Computing and Informatics, announces the Call for Papers for the issue of an interdisciplinary volume dedicated to the theoretical and practical aspects of natural language (NL) analysis and generation.

This special issue is intended to be a forum for presenting, first of all, the theoretical ideas proved to be effective in the design of NL processing systems or promising to considerably extend the sphere of successful NLP applications.

TOPICS: Original papers are invited in all subareas and on all aspects of NLP, especially on:

1. The current state and advancements in the last five years in particular subfields of NLP.
2. Natural-language-like knowledge and meaning representation formal systems.
3. Formal approaches to describing conceptual structures of complicated real discourses (pertaining, e.g., to medicine, technology, law, business, etc.).
4. New logics for NLP.
5. Semantics-oriented methods of natural language analysis, conceptual information retrieval in textual data bases.
6. Computational lexical semantics, ontologies for NLP.
7. Understanding of metaphors and metonymy.
8. Anaphora resolution.
9. Generation of natural language discourses.
10. Parallel conceptual processing of natural language texts.
11. Intelligent text summarization.
12. New directions in NLP.

Informatica 22 (1998) No. 4, in an enlarged volume, is fixed as the special issue.

Time Table and Contacts

The deadline for the paper submission in four copies is July 30, 1998.

Printed-paper mail address:
Prof. A.P.Zeleznikar, Jozef Stefan Institute,
Jamova c. 39, SI-1111 Ljubljana, Slovenia.

Correspondence (e-mail addresses):

— anton.p.zeleznikar@ijs.si
Prof. Anton P. Železnikar, Slovenia
— vaf@nw.math.msu.su
Prof. Vladimir A. Fomichov, Russia
— kitano@csl.sony.co.jp
Prof. Hiroaki Kitano, Japan

Format and Reviewing Process

As a rule, papers should not exceed 8,000 words (including figures and tables but excluding references. A full page figure should be counted as 500 words).
Ideally 5,000 words are desirable.

Each paper will be reviewed by at least two anonymous referees outside the author's country and by the appropriate editors.

In case a paper is accepted, its author (authors) will be asked to transform the manuscript into the Informatica LaTeX style (available from ftp.arnes.si; directory: /magazines/informatica).

For more information about the Informatica and the Special Issue see
FTP: ftp.arnes.si
with anonymous login or
URL:
http://turing.ijs.si/Mezi/informat.htm.

First Call for Papers

International Conference on Systems, Signals, Control, Computers (Sscc'98)

International Association for the Advancement of Methods for System Analysis and Design (Iaamsad) and Academy of Nonlinear Sciences (Ans)

Announce the International Conference on
Systems, Signals, Control, Computers (Sscc'98)
Durban, South Africa (September 22-24, 1998)

and Invite Potential Authors for Submission of Papers

A preliminary WEB home page can be accessed on
`http://nsys.ntech.ac.za/iaamsad/SSCC98test.html`
This home page will become public when International Programme Committee membership become confirmed.

Honorary Chairman: Academician V.M.Matrosov (Russia)
Conference Chairman: V.B.Bajic (South Africa)

Advisory Board:

V.B.Bajic (South Africa), J.Brzobohaty (Czech Republic), P.Daoutidis (USA), W.Hide (South Africa), C.Morabito (Italy), V.V.Kozlov (Russia), P.Leach (South Africa), P.C.Muller (Germany), L.Shaikhet (Ukraine), E.Rogers (UK), H.Szu (USA).

International Programme Committee:

V.Apanasovich (Belarus), V.B.Bajic (South Africa), C.Berger-Vachon (France), J.Brzobohaty (Czech Republic), M.Campolo (Italy), P.Daoutidis(USA), T.Fukuda(Japan), Z.Gajic (USA), M.Gams (Slovenia), J.Gil Aluja (Spain), Ly.T.Gruyitch (France), H.Hahn (Germany), M.Hajek (South Africa), R.Harley (South Africa), W.Hide (South Africa), M.Jamshidi (USA), V.Kecman (New Zealand), B.Kovacevic (Yugoslavia), V.Krasnoporoshin (Belarus), V.V.Kozlov (Russia), P.Leach (South Africa), L.K.Kuzmina (Russia), V.Milutinovic (Yugoslavia), C.Morabito (Italy), P.C.Muller (Germany), H.Nijmeijer (The Netherlands), D.H.Owens (UK), D.Petkov (South Africa), K.M.Przyluski (Poland), E.S.Pyatnitskii (Russia), E.Rogers (UK), L.Shaikhet (Ukraine), A.V.Savkin (Australia) H.Szu (USA), E.I.Verriest (USA), R.Vrba (Czech Republic), J.Ziska (Czech Republic).

Local Organizing Committee:

V.Bajic, P.Govender, R.Hacking, M.Hajek, M.McLeod, K.S.Moodley, R.Papa, C.Radhakishun, A.Singh.

Address Of The Conference Office:

Sacan, P.O.Box 1428, Link Hills 3652, Durban, South Africa Tel./Fax: (+27 31) 204-2560 e-mail: `bajic.v@umfolozi.ntech.ac.za`

Supporting Organizations:

SANBI - South African National Institute for Bioinformatics (South Africa)
SAICSIT - South African Institute for Computer Scientists and Information Technologists (South Africa)
CER - Centre for Engineering Research, Technikon Natal (South Africa)
M L Sultan Technikon (South Africa)

General Information

1998 year is the year of Science and Technology in South Africa. The intention of the Department of Arts, Culture, Science and Technology of South Africa is to make South Africans more aware of how Science and Technology affects them in every-day life. Such a national initiative is in a way a very good environment for a conference like this: one that has a broad scope and spans many different fields. At the same time an opportunity is given to the research community of

South Africa to interact more directly with overseas peers.

Aims And Scope

The Conference is broad in scope and will provide a forum for the exchange of the latest research results as applied to different branches of science and technology. The areas of interest include concepts, techniques and paradigms associated with systems, signals, control and/or computers.

Domains of application include informatics, biomedical technology, economics, management, diverse engineering and science fields and applied mathematics. Artificial intelligence techniques are of particular interest, as well as reports on industrial applications.

The conference will include several plenary and invited lectures from world renowned scientists and regular papers. A number of special and invited sessions will also be organised, dealing with focussed areas of interest. The proposals for these special sessions should be submitted at the same time as the abstracts. A special session cannot have less than three papers or more than six.

The official language of the conference is English.

Manuscript Submission And Review Process

Three copies of the extended abstract (at least two pages) should be sent to the Conference Office at the address given below. Full papers are preferred. Papers in Microsoft Word can be sent by e-mail. All submissions will be reviewed by members of the International Programme Committee; additional reviewers will be consulted if necessary. The submissions will be reviewed as soon as they arrive; the average review time is about four weeks. Authors of accepted papers will thereafter be informed (by e-mail if available) of the required format for camera-ready paper submissions. In order for reviewers to be able to assess the submissions, the extended abstract has to provide sufficient information about the background to the problem, the novelty of the obtained results and the results achieved, the conclusions drawn and some references. Up to five keywords should be provided. All submitted papers have to be original, unpublished and not submitted for publication elsewhere.

Proceedings

All accepted papers will be published in the Conference Proceedings, which will be issued by a renowned international publisher.

Important Notice

Although we expect that the authors of accepted papers will present the papers at this Conference, we recognize that circumstances may prevent authors from participation at the Conference. In such cases the accepted papers will be published if the authors inform organizers of their non-attendance at the Conference by 15th May 1998. However, conference fees according to established rules have to be pre-paid in order that papers appear in the Proceedings.

Conference Fees

The conference fee for one participant covers the publication of two papers (each with a maximum of five A4 pages in length) according to the required format; one volume of the Proceedings in which the paper(s) appear(s); refreshment during the conference; one lunch and a banquet. Additional volumes of the Proceedings can be purchased for US$ 55.00. Authors of multiple papers are to pay additional fees for extra papers according to the specified rule. Social programme and tourist visits will be provided at extra cost.

Reduced registration fee of US$ 280.00 (South Africans R 1120.00) is applicable for early received, reviewed and accepted papers for which fee is paid by February 25, 1998 - prospective authors are encouraged to take advantige of this convenience; otherwise the following rates apply:

Early registration fee: US$ 350.00 (South Africans R 1400.00)

Late and on-site registration fee: US$ 400.00 (South Africans R 1600.00)

Student fee: US$ 200.00 (South Africans R 800.00) - to qualify for the student scale of fees, all authors mentioned on the paper have to be current students; written proof has to be provided at the time of payment

Payment in South African rands is possible only when all authors of the papers are South African residents; written proof has to be provided at the time of payment.

Deadlines

Extended Abstracts and Special Session Proposals:
- submission by mail (15th February, 1998)
- submissions by e-mail (15th January, 1998)
Notification of acceptance (15th April, 1998)
Submission of papers in camera ready form (15th May, 1998)
Early payment of conference fees (15th May, 1998)
Late payment of conference fees (31 June, 1998)

THE MINISTRY OF SCIENCE AND TECHNOLOGY OF THE REPUBLIC OF SLOVENIA

Address: Slovenska 50, 1000 Ljubljana, Tel.: +386 61 1311 107, Fax: +386 61 1324 140.
WWW:http://www.mzt.si
Minister: Lojze Marinček, Ph.D.

The Ministry also includes:

The Standards and Metrology Institute of the Republic of Slovenia
Address: Kotnikova 6, 61000 Ljubljana, Tel.: +386 61 1312 322, Fax: +386 61 314 882.

Slovenian Intellectual Property Office
Address: Kotnikova 6, 61000 Ljubljana, Tel.: +386 61 1312 322, Fax: +386 61 318 983.

Office of the Slovenian National Commission for UNESCO
Address: Slovenska 50, 1000 Ljubljana, Tel.: +386 61 1311 107, Fax: +386 61 302 951.

Scientific, Research and Development Potential:

The Ministry of Science and Technology is responsible for the R&D policy in Slovenia, and for controlling the government R&D budget in compliance with the National Research Program and Law on Research Activities in Slovenia. The Ministry finances or co-finance research projects through public bidding, while it directly finance some fixed cost of the national research institutes.

According to the statistics, based on OECD (Frascati) standards, national expenditures on R&D raised from 1,6 % of GDP in 1994 to 1,71 % in 1995. Table 2 shows an income of R&D organisation in million USD.

Objectives of R&D policy in Slovenia:

- maintaining the high level and quality of scientific technological research activities;
- stimulation and support to collaboration between research organisations and business, public, and other sectors;
- *stimulating and supporting of scientific and research disciplines that are relevant to Slovenian national authenticity;*
- co-financing and tax exemption to enterprises engaged in technical development and other applied research projects;
- support to human resources development with emphasis on young researchers; involvement in international research and development projects;
- transfer of knowledge, technology and research achievements into all spheres of Slovenian society.

Total investments in R&D (% of GDP)	1,71
Number of R&D Organisations	297
Total number of employees in R&D	12.416
Number of researchers	6.094
Number of Ph.D.	2.155
Number of M.Sc.	1.527

Table 1: Some R&D indicators for 1995

		Ph.D.			M.Sc.	
	1993	1994	1995	1993	1994	1995
Bus. Ent.	51	93	102	196	327	330
Gov. Inst.	482	574	568	395	471	463
Priv. np Org.	10	14	24	12	25	23
High. Edu.	1022	1307	1461	426	772	711
TOTAL	1565	1988	2155	1029	1595	1527

Table 2: Number of employees with Ph.D. and M.Sc.

Table source: Slovene Statistical Office.

	Basic Research		Applied Research		Exp. Devel.		Total	
	1994	1995	1994	1995	1994	1995	1994	1995
Business Enterprises	6,6	9,7	48,8	62,4	45,8	49,6	101,3	121,7
Government Institutes	22,4	18,6	13,7	14,3	9.9	6,7	46,1	39,6
Private non-profit Organisations	0,3	0,7	0,9	0,8	0,2	0,2	1,4	1,7
Higher Education	17,4	24,4	13,7	17,4	8,0	5,7	39,1	47,5
TOTAL	46,9	53,4	77,1	94,9	63.9	62,2	187,9	210,5

Table 3: Incomes of R&D organisations by sectors in 1995 (in million USD)

JOŽEF STEFAN INSTITUTE

Jožef Stefan (1835-1893) was one of the most prominent physicists of the 19th century. Born to Slovene parents, he obtained his Ph.D. at Vienna University, where he was later Director of the Physics Institute, Vice-President of the Vienna Academy of Sciences and a member of several scientific institutions in Europe. Stefan explored many areas in hydrodynamics, optics, acoustics, electricity, magnetism and the kinetic theory of gases. Among other things, he originated the law that the total radiation from a black body is proportional to the 4th power of its absolute temperature, known as the Stefan–Boltzmann law.

The Jožef Stefan Institute (JSI) is the leading independent scientific research institution in Slovenia, covering a broad spectrum of fundamental and applied research in the fields of physics, chemistry and biochemistry, electronics and information science, nuclear science technology, energy research and environmental science.

The Jožef Stefan Institute (JSI) is a research organisation for pure and applied research in the natural sciences and technology. Both are closely interconnected in research departments composed of different task teams. Emphasis in basic research is given to the development and education of young scientists, while applied research and development serve for the transfer of advanced knowledge, contributing to the development of the national economy and society in general.

At present the Institute, with a total of about 700 staff, has 500 researchers, about 250 of whom are postgraduates, over 200 of whom have doctorates (Ph.D.), and around 150 of whom have permanent professorships or temporary teaching assignments at the Universities.

In view of its activities and status, the JSI plays the role of a national institute, complementing the role of the universities and bridging the gap between basic science and applications.

Research at the JSI includes the following major fields: physics; chemistry; electronics, informatics and computer sciences; biochemistry; ecology; reactor technology; applied mathematics. Most of the activities are more or less closely connected to information sciences, in particular computer sciences, artificial intelligence, language and speech technologies, computer-aided design, computer architectures, biocybernetics and robotics, computer automation and control, professional electronics, digital communications and networks, and applied mathematics.

The Institute is located in Ljubljana, the capital of the independent state of **Slovenia** (or S♡nia). The capital today is considered a crossroad between East, West and Mediterranean Europe, offering excellent productive capabilities and solid business opportunities, with strong international connections. Ljubljana is connected to important centers such as Prague, Budapest, Vienna, Zagreb, Milan, Rome, Monaco, Nice, Bern and Munich, all within a radius of 600 km.

In the last year on the site of the Jožef Stefan Institute, the Technology park "Ljubljana" has been proposed as part of the national strategy for technological development to foster synergies between research and industry, to promote joint ventures between university bodies, research institutes and innovative industry, to act as an incubator for high-tech initiatives and to accelerate the development cycle of innovative products.

At the present time, part of the Institute is being reorganized into several high-tech units supported by and connected within the Technology park at the Jožef Stefan Institute, established as the beginning of a regional Technology park "Ljubljana". The project is being developed at a particularly historical moment, characterized by the process of state reorganisation, privatisation and private initiative. The national Technology Park will take the form of a shareholding company and will host an independent venture-capital institution.

The promoters and operational entities of the project are the Republic of Slovenia, Ministry of Science and Technology and the Jožef Stefan Institute. The framework of the operation also includes the University of Ljubljana, the National Institute of Chemistry, the Institute for Electronics and Vacuum Technology and the Institute for Materials and Construction Research among others. In addition, the project is supported by the Ministry of Economic Relations and Development, the National Chamber of Economy and the City of Ljubljana.

Jožef Stefan Institute
Jamova 39, 61000 Ljubljana, Slovenia
Tel.:+386 61 1773 900, Fax.:+386 61 219 385
Tlx.:31 296 JOSTIN SI
WWW: http://www.ijs.si
E-mail: matjaz.gams@ijs.si
Contact person for the Park: Iztok Lesjak, M.Sc.
Public relations: Natalija Polenec

INFORMATICA
AN INTERNATIONAL JOURNAL OF COMPUTING AND INFORMATICS
INVITATION, COOPERATION

Submissions and Refereeing

Please submit three copies of the manuscript with good copies of the figures and photographs to one of the editors from the Editorial Board or to the Contact Person. At least two referees outside the author's country will examine it, and they are invited to make as many remarks as possible directly on the manuscript, from typing errors to global philosophical disagreements. The chosen editor will send the author copies with remarks. If the paper is accepted, the editor will also send copies to the Contact Person. The Executive Board will inform the author that the paper has been accepted, in which case it will be published within one year of receipt of e-mails with the text in Informatica LaTeX format and figures in .eps format. The original figures can also be sent on separate sheets. Style and examples of papers can be obtained by e-mail from the Contact Person or from FTP or WWW (see the last page of Informatica).

Opinions, news, calls for conferences, calls for papers, etc. should be sent directly to the Contact Person.

QUESTIONNAIRE

□ Send Informatica free of charge

□ Yes, we subscribe

Please, complete the order form and send it to Dr. Rudi Murn, Informatica, Institut Jožef Stefan, Jamova 39, 61111 Ljubljana, Slovenia.

Since 1977, Informatica has been a major Slovenian scientific journal of computing and informatics, including telecommunications, automation and other related areas. In its 16th year (more than five years ago) it became truly international, although it still remains connected to Central Europe. The basic aim of Informatica is to impose intellectual values (science, engineering) in a distributed organisation.

Informatica is a journal primarily covering the European computer science and informatics community - scientific and educational as well as technical, commercial and industrial. Its basic aim is to enhance communications between different European structures on the basis of equal rights and international refereeing. It publishes scientific papers accepted by at least two referees outside the author's country. In addition, it contains information about conferences, opinions, critical examinations of existing publications and news. Finally, major practical achievements and innovations in the computer and information industry are presented through commercial publications as well as through independent evaluations.

Editing and refereeing are distributed. Each editor can conduct the refereeing process by appointing two new referees or referees from the Board of Referees or Editorial Board. Referees should not be from the author's country. If new referees are appointed, their names will appear in the Refereeing Board.

Informatica is free of charge for major scientific, educational and governmental institutions. Others should subscribe (see the last page of Informatica).

ORDER FORM – INFORMATICA

Name: ....

Title and Profession (optional): ....

....

Home Address and Telephone (optional): ....

....

Office Address and Telephone (optional): ....

....

E-mail Address (optional): ....

Signature and Date: ....

Referees:

Witold Abramowicz, David Abramson, Kenneth Aizawa, Suad Alagić, Alan Aliu, Richard Amoroso, John Anderson, Hans-Jurgen Appelrath, Grzegorz Bartoszewicz, Catriel Beeri, Daniel Beech, Fevzi Belli, Istvan Berkeley, Azer Bestavros, Balaji Bharadwaj, Jacek Blazewicz, Laszlo Boeszoermenyi, Damjan Bojadžijev, Jeff Bone, Ivan Bratko, Jerzy Brzezinski, Marian Bubak, Leslie Burkholder, Frada Burstein, Wojciech Buszkowski, Netiva Caftori, Jason Ceddia, Ryszard Choras, Wojciech Cellary, Wojciech Chybowski, Andrzej Ciepielewski, Vic Ciesielski, David Cliff, Travis Craig, Noel Craske, Tadeusz Czachorski, Milan Češka, Andrej Dobnikar, Sait Dogru, Georg Dorfner, Ludoslaw Drelichowski, Matija Drobnič, Maciej Drozdowski, Marek Druzdzel, Jozo Dujmović, Pavol Ďuriš, Hesham El-Rewini, Pierre Flener, Wojciech Fliegner, Terrence Forgarty, Hans Fraaije, Hugo de Garis, Eugeniusz Gatnar, James Geller, Michael Georgiopolus, Jan Goliński, Janusz Gorski, Georg Gottlob, David Green, Herbert Groiss, Inman Harvey, Elke Hochmueller, Rod Howell, Tomáš Hruška, Alexey Ippa, Ryszard Jakubowski, Piotr Jedrzejowicz, Eric Johnson, Polina Jordanova, Djani Juričič, Sabhash Kak, Li-Shan Kang, Roland Kaschek, Jan Kniat, Stavros Kokkotos, Kevin Korb, Gilad Koren, Henryk Krawczyk, Ben Kroese, Zbyszko Krolikowski, Benjamin Kuipers, Matjaž Kukar, Aarre Laakso, Phil Laplante, Bud Lawson, Ulrike Leopold-Wildburger, Joseph Y-T. Leung, Alexander Linkevich, Raymond Lister, Doug Locke, Peter Lockeman, Matija Lokar, Jason Lowder, Andrzej Małachowski, Peter Marcer, Andrzej Marciniak, Witold Marciszewski, Vladimir Marik, Jacek Martinek, Tomasz Maruszewski, Florian Matthes, Timothy Menzies, Dieter Merkl, Zbigniew Michalewicz, Roland Mittermeir, Madhav Moganti, Tadeusz Morzy, Daniel Mossé, John Mueller, Hari Narayanan, Elzbieta Niedzielska, Marian Niedq'zwiedziński, Jaroslav Nieplocha, Jerzy Nogieć, Stefano Nolfi, Franc Novak, Antoni Nowakowski, Adam Nowicki, Tadeusz Nowicki, Hubert Österle, Wojciech Olejniczak, Jerzy Olszewski, Cherry Owen, Mieczyslaw Owoc, Tadeusz Pankowski, Mitja Peruš, Warren Persons, Stephen Pike, Niki Pissinou, Ullin Place, Gustav Pomberger, James Pomykalski, Gary Preckshot, Dejan Rakovič, Cveta Razdevšek Pučko, Ke Qiu, Michael Quinn, Gerald Quirchmayer, Luc de Raedt, Ewaryst Rafajlowicz, Sita Ramakrishnan, Wolf Rauch, Peter Rechenberg, Felix Redmill, David Robertson, Marko Robnik, Ingrid Russel, A.S.M. Sajeev, Bo Sanden, Vivek Sarin, Iztok Savnik, Walter Schempp, Wolfgang Schreiner, Guenter Schmidt, Heinz Schmidt, Denis Sever, William Spears, Hartmut Stadtler, Janusz Stokłosa, Przemysław Stpiczyński, Andrej Stritar, Maciej Stroinski, Tomasz Szmuc, Zdzislaw Szyjewski, Jure Šilc, Metod Škarja, Jiři Šlechta, Zahir Tari, Jurij Tasič, Piotr Teczynski, Stephanie Teufel, Ken Tindell, A Min Tjoa, Wieslaw Traczyk, Roman Trobec, Marek Tudruj, Andrej Ule, Amjad Umar, Andrzej Urbanski, Marko Uršič, Tadeusz Usowicz, Elisabeth Valentine, Kanonkluk Vanapipat, Alexander P. Vazhenin, Zygmunt Vetulani, Olivier de Vel, John Weckert, Gerhard Widmer, Stefan Wrobel, Stanislaw Wrycza, Janusz Zalewski, Damir Zazula, Yanchun Zhang, Robert Zorc

EDITORIAL BOARDS, PUBLISHING COUNCIL

Informatica is a journal primarily covering the European computer science and informatics community; scientific and educational as well as technical, commercial and industrial. Its basic aim is to enhance communications between different European structures on the basis of equal rights and international refereeing. It publishes scientific papers accepted by at least two referees outside the author's country. In addition, it contains information about conferences, opinions, critical examinations of existing publications and news. Finally, major practical achievements and innovations in the computer and information industry are presented through commercial publications as well as through independent evaluations.

Editing and refereeing are distributed. Each editor from the Editorial Board can conduct the refereeing process by appointing two new referees or referees from the Board of Referees or Editorial Board. Referees should not be from the author's country. If new referees are appointed, their names will appear in the list of referees. Each paper bears the name of the editor who appointed the referees. Each editor can propose new members for the Editorial Board or referees. Editors and referees inactive for a longer period can be automatically replaced. Changes in the Editorial Board are confirmed by the Executive Editors.

The coordination necessary is made through the Executive Editors who examine the reviews, sort the accepted articles and maintain appropriate international distribution. The Executive Board is appointed by the Society Informatika. Informatica is partially supported by the Slovenian Ministry of Science and Technology.

Each author is guaranteed to receive the reviews of his article. When accepted, publication in Informatica is guaranteed in less than one year after the Executive Editors receive the corrected version of the article.

Volume 22 Number 2 May 1998 ISSN 0350-5596

Informatica

An International Journal of Computing and Informatics

Contents: